Statistics • Visualization • Data manipulation • Reporting

USER'S GUIDE

日本語マニュアル

Statistical software for data science

STATA USER'S GUIDE
リリース 18

A Stata Press Publication
StataCorp LLC
College Station, Texas

Published by Stata Press, 4905 Lakeway Drive, College Station, Texas 77845

ISBN-10:1-59718-403-9
ISBN-13:978-1-59718-403-8

ソフトウェアに関する引用は次のように行ってください。

StataCorp. 2023. *Stata: Release 18.* Statistical Software. College Station, TX: StataCorp LLC.

目次

Stataの基本

Stataの個別機能

アドバイス

Stataの基本

1 便利な情報-まずはご一読ください

目次

Stataマニュアル文書全セットは以下のマニュアルを含みます。

[GSM]	*Getting Started with Stata for Mac*
[GSU]	*Getting Started with Stata for Unix*
[GSW]	*Getting Started with Stata for Windows*
[U]	*Stata User's Guide*
[R]	*Stata Base Reference Manual*
[ADAPT]	*Stata Adaptive Designs: Group Sequential Trials Reference Manual*
[BAYES]	*Stata Bayesian Analysis Reference Manual*
[BMA]	*Stata Bayesian Model Averaging Reference Manual*
[CASUAL]	*Stata Causal Inference and Treatment-Effects Estimation Reference Manual*
[CM]	*Stata Choice Models Reference Manual*
[D]	*Stata Data Management Reference Manual*
[DSGE]	*Stata Linearized Dynamic Stochastic General Equilibrium Reference Manual*
[ERM]	*Stata Extended Regression Models Reference Manual*
[FMM]	*Stata Finite Mixture Models Reference Manual*
[FN]	*Stata Functions Reference Manual*
[G]	*Stata Graphics Reference Manual*
[IRT]	*Stata Item Response Theory Reference Manual*
[LASSO]	*Stata Lasso Reference Manual*
[XT]	*Stata Longitudinal-Data/Panel-Data Reference Manual*
[META]	*Stata Meta-Analysis Reference Manual*
[ME]	*Stata Multilevel Mixed-Effects Reference Manual*
[MI]	*Stata Multiple-Imputation Reference Manual*
[MV]	*Stata Multivariate Statistics Reference Manual*
[PSS]	*Stata Power, Precision and Sample-Size Reference Manual*
[P]	*Stata Programming Reference Manual*
[RPT]	*Stata Reporting Reference Manual*
[SP]	*Stata Spatial Autoregressive Models Reference Manual*
[SEM]	*Stata Structural Equation Modeling Reference Manual*
[SVY]	*Stata Survey Data Reference Manual*
[ST]	*Stata Survival Analysis Reference Manual*
[TABLES]	*Stata Customizable Tables and Collected Results Reference Manual*
[TS]	*Stata Time-Series Reference Manual*
[I]	*Stata Index*
[M]	*Mata Reference Manual*

これらの情報に追加して、インストールに関する情報が*Installation Guide*に記載されています。

1.1 STATA Getting Started ガイド

Getting Startedガイドは全部で3冊あります。

[GSM] *Stata Getting Started ガイド Mac版*
[GSU] *Stata Getting Started ガイド Unix版*
[GSW] *Stata Getting Started ガイド Windows版*

1. Stataの使用法を学ぶには、Getting Started ガイド(GSM、GSU、GSW)をご一読ください。
2. その他のマニュアルに関する情報は[U] 1.2 User's Guideとリファレンスマニュアルを参照してください。

1.2 User's Guideとリファレンスマニュアル

StataのUser's Guideは3つのセクションからなります。各セクションのタイトルはStataの基本、Stataの機能、データ処理です。セクションの内容はいくつもの章に分かれています。章を閲覧するには、目次の章タイトルをクリックします。

User's GuideはStataには便利な情報が豊富に用意されています。時間があまり無い方は、 **[U] 11 言語構文**と**[U] 12 データ**だけでもご一読ください。

本書以外のマニュアルはリファレンスマニュアルという位置付けです。百科事典と同じく、アルファベット順に項目が並んでいます。例えば、*Base Reference Manual*をご覧ください。そして、ご存じのコマンドを探してみてください。見出しにコマンドがない場合は、[I] *Stata Index*のsubject indexを利用します。密接に関連するコマンド、例えば、ranksumとmedianなどは一緒に記載してあります。実際、この2つのコマンドは [R] **ranksum**に説明があります。

*Base Reference Manual*の項目がすべてコマンドとは限りません。例えば、繰り返し計算により最大化を行う[R] **Maximize**や、エラーメッセージやリターンコードを取得する[R] **error messages** などは、コマンドではなく技術情報です。

リファレンスマニュアルはすべてのページを精読するというものではありません。あるコマンドの機能をリファレンスマニュアルで詳しく調べる場合は、制限事項や機能的な短所も確認できます。また、コマンドの実行結果が想像と異なる場合などは、解説を丁寧に読んでください。各項目は基本的にはコマンドを解説するものです。例えば、データの読み込みや、保存などの簡単なコマンドについて説明する場合、基本的にStataを初めて利用する場面を想定して解説しています。逆に、高度なコマンドの場合は、当該のコマンド以外の機能については既知であるとして説明しています。

目的のコマンドが*Base Reference Manual*で見つからない時は、その他のリファレンスマニュアルを参照してください。マニュアルのタイトルをご覧いただけば、解説されているコマンドについて想像できるはずです。ただし、*Programming Reference Manual*は少し異なり、プログラミングに必要な情報だけでなく、行列の操作方法に関しても解説しています。なお、行列プログラミング言語については、*Mata Reference Manual*で解説していますので、混同しないようご注意ください。

1.2.1 PDFマニュアル

Stataをインストールすると、PDFマニュアルも共にコピーされます。

PDFマニュアルの一覧を参照する場合は**ヘルプ ＞ 英文PDF マニュアル** と操作します。これ以外にも任意のヘルプ画面から、PDFマニュアルの解説ページにジャンプできます。例えば、help regressとしてヘルプの画面を表示し、タイトル部にある[R] **regress**という文字をクリックすると、自動的にPDFマニュアルの該当ページを表示します。

Stataの各マニュアルをPDF画面でより快適に表示する方法をhttps://www.stata.com/support/faqs/res/documentation.htmlにまとめておきましたので、ご参照ください。

1.2.1.1 ビデオ例題

PDF documentation in Stata(StataのPDF文書)

1.2.2 サンプルデータセット

このマニュアル内の多くの例題は自動車データセット、auto.dtaを使用します。このauto.dtaというデータセットは74種の自動車に関する価格、燃費、車重、その他の要素などを組み合わせたデータセットです。（これらの元データは1979年4月に発表されたConsumer Reportsと米国政府の燃料消費におけるEPA統計から構成されています。出典はChambers et al. [1983]にまとめられて発表されました。）

マニュアル内の例題では、しばしば次のような入力でサンプルデータを取得します。

```
. use https://www.stata-press.com/data/r18/auto
```

しかし、Stataのパッケージにはauto.dtaが予め含まれています。そこで、コンピュータ上のファイルを使用する場合は、以下のように入力します。

```
. sysuse auto
```

詳しくは[D] **sysuse**をご覧ください。

また、auto.dtaファイルは**ファイル > 例題データセット...**と操作して、ウィンドウから開く事もできます。ウィンドウ内の「Example datasets installed with Stata」をクリックして、表示されたページでauto.dtaのファイル名の隣にある「use」をクリックします。

サンプルとして使用できるデータセットは、Web上あるいはコンピュータ上にあります。以下はStata内に含まれているデータセットの一例です。

auto.dta	1978年 自動車データ
bplong.dta	ロング形式の架空の血圧データ
bpwide.dta	ワイド形式の架空の血圧データ
cancer.dta	薬物トライアル中の患者の生存率
census.dta	1980年州ごとの米国国勢調査データ
citytemp.dta	都市の気温データ
educ99gdp.dta	GDPと教育
gnp96.dta	U.S. GNP, 1967-2002
lifeexp.dta	1998年の平均寿命
network1.dta	架空のネットワークダイアグラムデータ
nlsw88.dta	U.S. National Longitudinal Survey of Young Women (NLSW, 1988抜粋)
pop2000.dta	米国国勢調査, 2000, 抜粋
sandstone.dta	オハイオ州内におけるLamont砂岩の海抜
sp500.dta	S&P 500
surface.dta	NOAA 海水面温度
tsline1.dta	シミュレートした時系列データ
uslifeexp.dta	1900-1999年における、米国平均寿命
voter.dta	1992年大統領選の有権者データ

これらのデータセットは**例題データセット**メニューから開く事ができます。また、データに関する概要を画面に表示することもできます。

さらに多くのサンプルデータセット、リファレンスマニュアル内で使用されているサンプルのほとんどは、Stata Pressのウェブサイト(https://www.stata-press.com/data/)で取得できます。データセットはコンピュータのブラウザからダウンロードするか、次のようにStataのコマンドで直接ダウンロードします。

```
. use https://www.stata-press.com/data/r18/nlswork
```

インターネット上のサンプルデータを取り込む際はuseコマンドの代わりにwebuseコマンドを使います。例えば、以下のように入力します。

```
. webuse nlswork
```

これは先のuseコマンドと同じ動作を行うものです。詳細は [D] **webuse**をご覧ください。

1.2.2.1 ビデオ例題

Example datasets included with Stata

1.2.3 相互参照

Getting Startedマニュアル、User's Guide、リファレンスマニュアルにはクロスリファレンスを設定してあります。

[R] **regress**
[D] **reshape**
[XT] **xtreg**

最初のregressコマンドに対するリファレンスは、*Base Reference Manual*に対応するエントリです。2番目のreshapeコマンドに対するエントリは*Data Management Reference Manual*を参照しています。そして、3番目のxtregコマンドに対するエントリは*Longitudinal-Data/Panel-Data Reference Manual*に記載されています。

[GSW] B Advanced Stata usage
[GSM] B Advanced Stata usage
[GSU] B Advanced Stata usage

上記の項目は、Stata Getting StartedガイドWindows版、Stata Getting StartedガイドMac版、Stata Getting StartedガイドUnix版の該当するセクションを示しています。

1.2.4 索引

Indexは全てのマニュアルを網羅する、共通の索引 です。

情報やコマンドを素早く検索するには、Stataのsearchコマンドを使用してください。詳細は[R] **search**を参照してください。例えば、Stata のコマンドウィンドウで「search geometric mean」と入力すると、searchコマンドはStataのキーワードデータベースとインターネットを参照してStataコマンドとadoファイルの検索を実行します。

1.2.5 項目別目次

User's Guideや他のリファレンスマニュアル(Mata Reference Manualを除く)の項目別目次は*Index*に記載されています。この項目別テーブルはPDFブックマークからも移動できます。

1.2.6 表示形式

本文中の解説においてStataのコマンドやオプションを表記する場合、タイプライター体(次のフォント：that look like this)と呼ばれる文字を利用します。英文字の書体であっても、それがタイプライター体の場合は、Stataのコマンドウィンドウに入力すべきコマンドやオプションである事を示しています。したがって、書体には十分、気をつけてください。

例えば、describeコマンドでデータの形式を調べ、「listコマンドで表示させることができる」とあったら、この場合、describeとlistはStataコマンドですので、ただちに実行できます。すでにお分かりかと思いますが、コマンドを文頭に記述する場合、先頭の文字を大文字で記述することはありません。本書ではコマンドをどの位置に記述しても、すべて小文字で表記します。

コマンドの説明において、例えばランクサム検定を実行する場合は、ranksum varname, by(groupvar)のように斜体を利用することがあります。もちろん、Stataのコマンドウィンドウに斜体で入力せよ、という意図ではありません。斜体部分にはそれに相当する変数名やオプションなどの情報を入力します。

引用符と句読点の用法について説明します。本書では数学の書籍や英語の文学書における用法をそのまま用います。例えば、句読点は引用符内の情報である場合に限り、その内部に記述します。例えば、Stataをとても気に入っているユーザが、とてもすばらしいプログラムだと感じている場合は、"very powerful program"。一方、Stataが気に入っています！という場合は、"I love Stata."のように引用符の中に句読点を含めます。

ただ、本書のなかで会話文を紹介することはほとんどありません。実際の用法は「"cd c:"と入力してください」というような形です。この場合、引用符の中に句読点は利用しません。句読点を利用するケースでは、次に示すように、引用符の外と内で2つの句読点を利用します。"the orthogonal polynomial operator, p.".

基本的に英文の記述方式に則って表記をおこなっており、おかしな表現、表示が存在するとすれば、それは当社の誤りです。ご助言、ご指摘いただければ幸いです。

英語版マニュアルにおいては英国Durham大学地理学部のNicholas J.Cox氏からは多くの助言をいただきました。ここに感謝の意を表します。

1.2.7 ビネット

例えば、PDFマニュアルの[R] **brier**の解説の最後に、気象統計学者として活躍したGlenn Wilson Brier(1913-1998)の生涯を簡単に紹介した研究者点描のコラム(ビネット)を用意してあります。この種のコラムはStata 8のマニュアルから登場し、バージョンアップの度に、その数を増やしています。コラムはその研究者と関連の深い統計機能のページに配置してあります。例えば、George E.P.Boxは[TS] **arima**や[R] **boxcox**のページに、それぞれに掲載したいところですが、紙面の都合上、[TS] arimaの方に載せてあります。ビネットの索引はIndexの項に用意してありますので、関心のある方は、是非、ご覧ください。

ビネットの内容の大部分はDurham大学Nicholas J.Cox氏によるものです。氏の著述に様々な書籍、記事、インターネット上の情報、そして個人的に提供された情報を追加、整理したものをコラムとしてまとめました。特に、Upton and Cook (2008)とEveritt and Skrondal (2010)そしてHeyde and Seneta (2001)とJohnson and Kotz (1997)らによる統計学者の伝記はとても有用であることを、ここに記しておきます。Upton and Cool(2008)は発表時点で存命する研究者だけを紹介したものです。

1.3 新着情報

Stata 18では数多くの新機能が追加されました。主な新機能は以下のウェブページでご覧いただけます。

https://www.stata.com/new-in-stata/

Stataからは次のコマンドで新機能をご覧ください。

```
.help whatsnew18
```

1.4 参考文献

Chambers, J. M., W. S. Cleveland, B. Kleiner, and P. A. Tukey. 1983. *Graphical Methods for Data Analysis.* Belmont, CA: Wadsworth.

Everitt, B. S., and A. Skrondal. 2010. *The Cambridge Dictionary of Statistics.* 4th ed. Cambridge: Cambridge University Press.

Gould, W. W. 2014. Putting the Stata Manuals on your iPad. *The Stata Blog: Not Elsewhere Classified.* http://blog.stata.com/2014/10/28/putting-the-stata-manuals-on-your-ipad/.

Heyde, C. C., and E. Seneta, ed. 2001. *Statisticians of the Centuries.* New York: Springer.

Johnson, N. L., and S. Kotz, ed. 1997. *Leading Personalities in Statistical Sciences: From the Seventeenth Century to the Present.* New York: Wiley.

Pinzon, E., ed. 2015. *Thirty Years with Stata: A Retrospective.* College Station, TX: Stata Press.

Upton, G. J. G., and I. T. Cook. 2014. *A Dictionary of Statistics.* 3rd ed. Oxford: Oxford University Press.

2 Stataの簡単な紹介

Stataはデータの管理、分析、グラフ化ができる統計パッケージです。

Stataは多くのプラットフォームで使用できます。Stataはマウスのクリックによる操作、あるいはコマンドの入力での制御が可能です。

StataのGUIは、初めてStataを使用する初心者や、熟練のStataユーザでもあまり使用しないコマンドを利用する際に、簡単に操作できるよう設計されています。

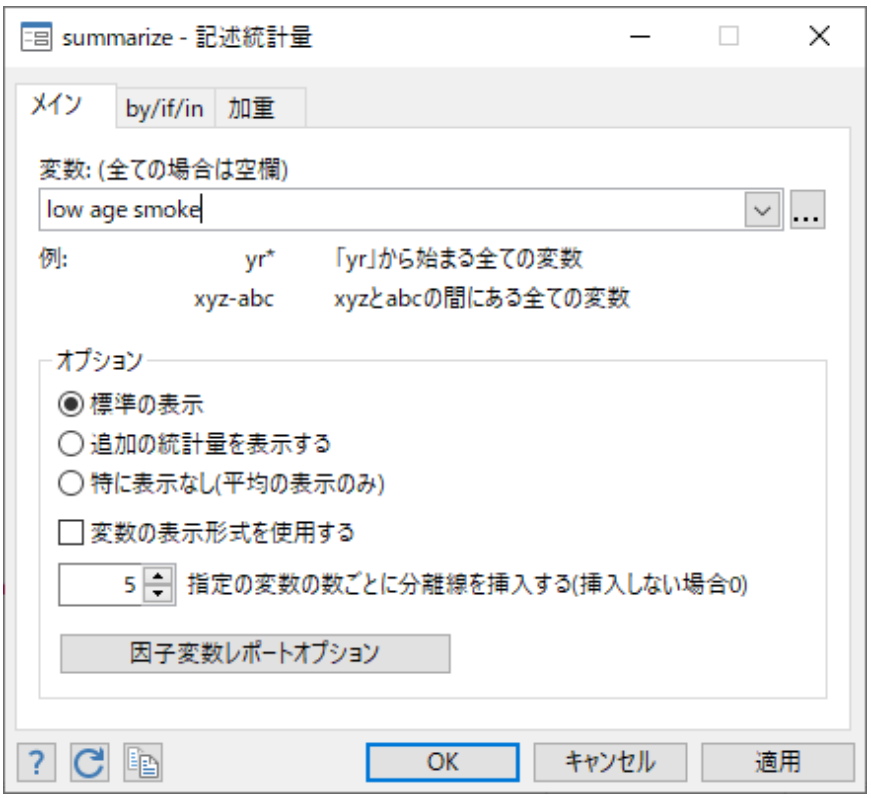
コマンドはStataの素早い操作に適しており、さらに複雑な指示を出す際にも便利です。

GUIを使用したStataのセッション例は次のとおりです。

（Stataのマニュアルは全体を通して、多くのデータセットを参照します。これらのデータセットには、https://www.stata-press.com/data/r18/から全てアクセスできます。Stata内から簡単にアクセスするには、「webuse データセット名」と入力するか **ファイル** > **例題データセット...** と操作して*Stata 17 manual datasets*をクリックします。

```
. webuse lbw
(Hosmer & Lemeshow data)
```

データ > **データの内容表示** > **記述統計量** と操作して、変数ウィンドウから入手した変数low, age, smokeについて要約統計量を計算します。OKをクリックします。

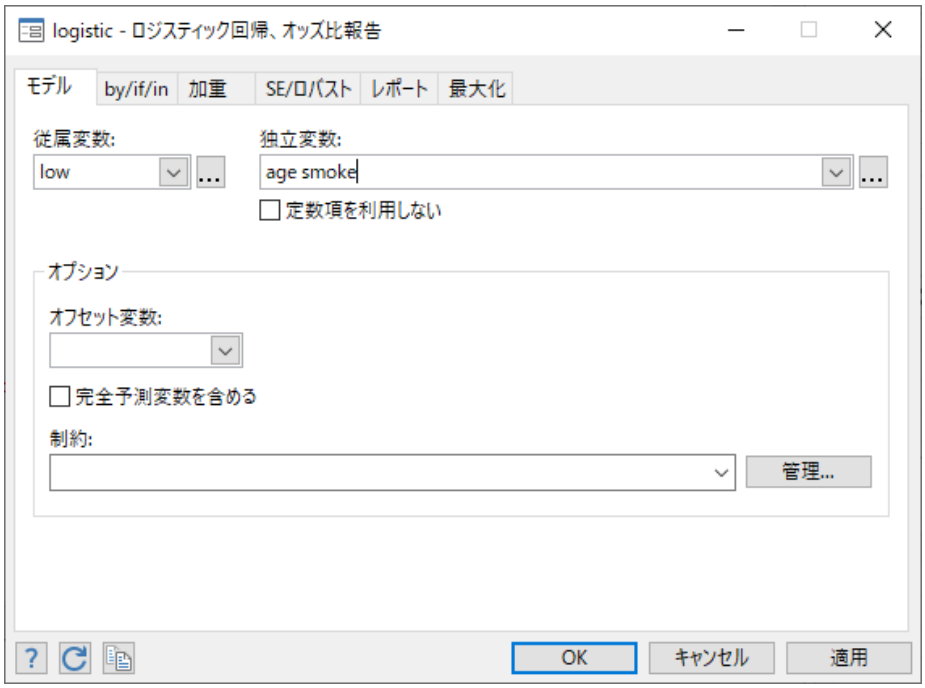


```
. summarize low age smoke
```

Variable	Obs	Mean	Std. dev.	Min	Max
low	189	.3121693	.4646093	0	1
age	189	23.2381	5.298678	14	45
smoke	189	.3915344	.4893898	0	1

Stataはメニューで操作しても、入力コマンド「summarize low age smoke」が出力結果の上に表示されます。

次に変数lowをageとsmokeに回帰させるロジスティック回帰モデルを推定します。メニューから**統計 > アウトカム(二値) > ロジスティック回帰**と操作し、項目を入力してからOKをクリックします。




```
. logistic low age smoke
Logistic regression                             Number of obs     =        189
                                                LR chi2(2)        =       7.40
                                                Prob > chi2       =     0.0248
Log likelihood = -113.63815                     Pseudo R2         =     0.0315
```

low	Odds ratio	Std. err.	z	P>|z|	[95% conf.	interval]
age	.9514394	.0304194	-1.56	0.119	.8936482	1.012968
smoke	1.997405	.642777	2.15	0.032	1.063027	3.753081
_cons	1.062798	.8048781	0.08	0.936	.2408901	4.689025

Note: _cons estimates baseline odds.

コマンド言語を使用したStataのセッション例は次のとおりです。

```
. use https://www.stata-press.com/data/r18/auto

(1978 automobile data)

. summarize mpg weight
```

Variable	Obs	Mean	Std. dev.	Min	Max
mpg	74	21.2973	5.785503	12	41
weight	74	3019.459	777.1936	1760	4840

ユーザが「summarize mpg weight」と入力すると、Stataは要約統計量の表を出力します。コマンドが違えば、表示される結果も異なります。

```
. generate gp100m = 100/mpg

. label var gp100m "Gallons per 100 miles"

. format gp100m %5.2f

. correlate gp100m weight (obs=74)
```

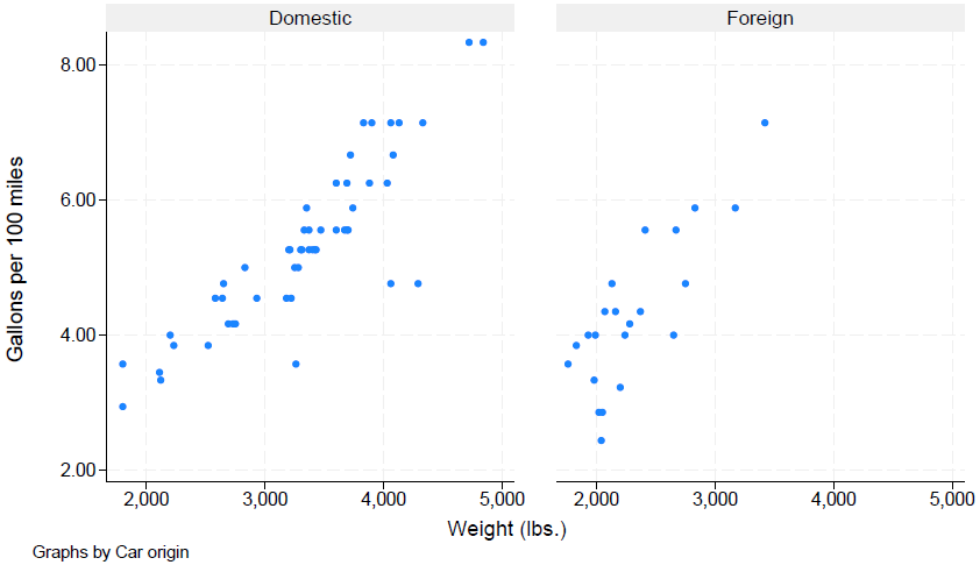
	gp100m	weight
gp100m	1.0000	
weight	0.8544	1.0000

```
. regress gp100m weight gear_ratio
```

Source	SS	df	MS		
Model	87.4543721	2	43.7271861	Number of obs	= 74
				F(2, 71)	= 96.65
Residual	32.1218886	71	.452420967	Prob > F	= 0.0000
				R-squared	= 0.7314
				Adj R-squared	= 0.7238
Total	119.576261	73	1.63803097	Root MSE	= .67262

gp100m	Coefficient	Std. err.	t	P>|t|	[95% conf.	interval]
weight	.0014769	.0001556	9.49	0.000	.0011665	.0017872
gear_ratio	.1566091	.2651131	0.59	0.557	-.3720115	.6852297
_cons	.0878243	1.198434	0.07	0.942	-2.301786	2.477435

```
.scatter gp100m weight, by(foreign)
```



メニュー操作は入力する量は少なくて済みますが出力結果は制限されます。目的に応じて使い分けてください。

Stataはデータセットを表形式で管理します。行は観測値で、列は変数を表します。

```
. list mpg weight gp100m in 1/10
```

	mpg	weight	gp100m
1.	22	2.930	4.55
2.	17	3.350	5.88
3.	22	2.640	4.55
4.	20	3.250	5.00
5.	15	4,080	6.67
6.	18	3.670	5.56
7.	26	2.230	3.85
8.	20	3,280	5.00
9.	16	3,880	6.25
10.	19	3,400	5.26

観測値には番号が付き、変数には名前がつけられます。

Stataは高速にデータを処理します。その高速処理を実現する1つの理由は丁寧なプログラミングにあります。また、Stataがデータをメモリに確保することも一役買っています。Stataのファイルはワープロのファイルをモデルにしています。データセットはディスクに存在しますが、メモリ内にデータセットのコピーを作成します。データセットはメモリ内にロードされ、操作、分析、変更されてから最後にディスクに保存されます。

メモリ内のデータのコピーを操作するということは、インタラクティブな使用に対して安全であるといえます。保存しているデータを壊す唯一の方法は上書き保存をすることです。

メモリ内にデータがあるので、データセットのサイズはコンピュータのメモリに依存します。Stataはメモリ内にデータを保存する時に効率の良い形式で保存します。メモリ内には驚くほど多くのデータが入ります。それでも、非常に大きなデータセットで操作をする場合は、メモリの許容量制限を超える可能性があります。データセットの効率的な保存方法については[D] compressを参照してください。

3 Stataを学び、使用する際のリソース

目次

3.1 概要

*Getting Started*や*User's Guide*、*Reference*マニュアルはStataを学ぶ一番の資料ですが、他にも多くの資料に情報がまとめられています。以下はそのうちのいくつかです。

- Stataそのもの。Stataにはsearchコマンドがあり、目的のトピックを簡単に検索してコマンドを実行できます。詳細は[U] **4 Stataのヘルプと検索機能**をご覧ください。
- Stataのウェブサイト。https://www.stata.comを覗いてください。大部分がユーザサポートページです。詳細は[U] **3.2.1 Stataのウェブサイト(www.stata.com)**をご覧ください。
- StataのYouTubeチャンネル。詳細はhttps://www.youtube.com/user/statacorpをご覧ください。定期的にStataの動画を更新しています。
- Stataブログ、Twitter、Facebook。https://blog.stata.com/、https://twitter.com/stata、https://facebook.com/statacorpをご確認ください。詳細は[U] **3.2.3 Stataの公式ブログ—Not Elsewhere Classified**と[U] **3.2.5 ソーシャルメディア上でのStata**をご覧ください。

- Stata Pressのウェブサイト。https://www.stata-press.comでは、マニュアル内で使用しているデータセットを掲載しています。詳細は[U] 3.3 Stata Pressをご覧ください。
- Stata掲示板。Stataユーザたちがインターネット上の掲示板で活発に議論を行っています。詳細は[U] 3.2.4 Stataの掲示板をご覧ください。
- *Stata Journal*と *Stata Technical Bulletin*。様々な分野で統計を使用する研究者に有益な情報を、論文、コラム、本のレビュー等の形で掲載するのが*Stata Journal* です。*Stata Technical Bulletin*は*Stata Journal*の前の媒体で、論文やユーザ定義コマンドを掲載します。詳細は[U] 3.4 The Stata Journalをご覧ください。
- Stataのソフトまたはユーザ提供のソフト配布サイト。Stataの更新・追加情報を入手・インストールできます。Stata社の公式なアップデートの提供があります。update queryを実行するか、**ヘルプ** > **アップデートのチェック** とメニュー操作します。一般のユーザが作成した追加ファイルや、ユーザが管理するサイトへのリンクも提供しています。netを実行するか、**ヘルプ** > **Stataジャーナル/コミュニティ投稿コマンド** とメニュー操作してください。詳細は[U] 3.5 Webから機能を更新する/追加する をご覧ください。
- NetCourses。インターネットを通じたトレーニングを提供します。詳細は[U] 3.6.2 NetCoursesを確認してください。
- 授業型トレーニングコース。米国各地のサードパーティの教室で行われるトレーニングの情報を提供します。詳細は[U] 3.6.3 授業型トレーニングコースを確認してください。
- ウェブ型レーニングコース。授業型トレーニングコースの内容をウェブ上で提供します。詳細は[U] 3.6.4 ウェブ型トレーニングコースを確認してください。
- 出張トレーニングコース。Stata社担当者がお客様の元へ出向き、カスタマイズしたトレーニングを提供します。詳細は[U] 3.6.5 出張トレーニングコースを確認してください。
- ウェビナー。専門家による、短時間のオンラインウェビナーを提供します。詳細は[U] 3.6.6 ウェビナーを確認してください。
- 書籍やサポート資料。他にもStataには資料があり、使用法を学ぶ際の補完資料として使用してください。詳細は[U] 3.7 書籍やその他のサポート資料をご覧ください。
- テクニカルサポート。Stata社へのテクニカルサポートの問い合わせはメール、電話で受け付けています。[U] 3.8 テクニカルサポートをご覧ください。

3.2 インターネットでのStata (www.stata.com または他のリソース)

3.2.1 Stataのウェブサイト(www.stata.com)

ブラウザでhttps://www.stata.comを開き、**Support**をクリックします。Stataのウェブサイトの半分以上はユーザにサポートを提供するためのウェブページになっています。

- ウェブサイトはWindows、Mac、Unix、統計、プログラミング、Mata、インターネット状況、データ管理などのFAQs(よく聞かれる質問)に対する回答を掲載します。これらのFAQsは「ファイルを保存/開くことができません」から「ロジスティック回帰の出力結果にある“completely determined”はどういう意味ですか」のような質問に対する回答まで、全体的な情報を提供します。多くのユーザが役立つ情報を発見できるでしょう。
- ウェブサイトは現在のスケジュールをはじめとした、NetCoursesに関する詳細を提供しています。詳しくは[U] 3.6.2 NetCoursesをご覧ください。
- ウェブサイトはまた主に米国で行われるStataのトレーニングコースやミーティングの情報も提供します。詳細は[U] 3.6.1 カンファレンスとユーザグループミーティング、[U] 3.6.3 一般トレーニングコース、[U] 3.6.4 ウェブ型トレーニングコース、[U] 3.6.5 出張トレーニングコースをご覧ください。
- また、ウェブサイトではStataの関連書籍や他のサポート製品を購入できる、オンラインストアを用意しています。詳細は[U] 3.7 書籍やその他のサポート資料をご覧ください。
- ウェブサイトでは統計に関する情報も提供しています。他の統計ソフトの提供元、本の出版社、統計専門誌、統計学会、統計に関するリストサーバなどの情報があります。

- ウェブサイト上ではStataを学ぶ上で重要な資料に関する情報を、https://www.stata.com/links/resources-for-learning-stata/のページで確認できます。このページには数多くのStataに関する有益な資料のリストがあります。

このように、ウェブサイトでは最新のサポート情報、そして可能な場合はその資料そのものを提供しています。時間に余裕があるときにでもhttps://www.stata.comをご覧ください。

3.2.2 StataのYouTubeチャンネル

StataのYouTubeチャンネルは https://www.youtube.com/user/statacorpにあります。ここでは、幅広い情報のトピック（基礎的なデータ管理とグラフ作成から少し上級者向けの統計分析、例えばANOVA、回帰、SEMなど）について、動画で紹介しています。このチャンネルには定期的に新しい動画が追加されます。

3.2.3 Stataの公式ブログ—Not Elsewhere Classified

Stataの公式ブログはhttps://blog.stata.comにあり、Stataのニュースや使用法に関するアドバイスの記事を掲載しています。ブログに投稿する記事はそれぞれ記名式になっており、Stataの開発・サポート・販売を実際に行うStata社の社員が書いています。

また、Stata Blogには世界中のStataユーザが作成したStataに関するブログへのリンクもあります。

3.2.4 Stata掲示板

StatalistはStata専門の掲示板です。何千人ものStataユーザがStataや統計に関する議論をしています。同掲示板は、Stataユーザにより運営、進行され、Stata社により保守されています。Statalistは1994年まで遡る質の高い議論の長い歴史があります。

多くの知識を持ったユーザ、さらにStata社のテクニカルスタッフも参加しているので、助言を得ることができるでしょう。だれでも参加が可能であり、Stata初心者も歓迎です。参加するための手順はhttps://www.statalist.orgで確認できます。登録を済ませて議論に参加したり、または単純に議論を読み進めてみてください。

Statalistに投稿する場合、事前に https://www.statalist.org/forums/help/にあるStatalist FAQをご一読ください。

3.2.5 ソーシャルメディア上でのStata

StataCorpは公式Twitterアカウント、Facebookページ、Instagram、LinkedInアカウントを開設しています。https://twitter.com/stataではTwitterのフォローができます。また、https://www.facebook.com/statacorpおよびhttps://www.instagram.com/statacorp/は、StataのFacebookおよびInstagramのページです。https://www.linkedin.com/company/statacorpからは、LinkedInでつながることができます。上記からは最新のStataに関する情報も提供されます。

3.2.6 Stataに関する他のインターネット上のリソース

非常に多くの人々がStataのチュートリアル、例題、データセットなどの情報をインターネット上で公開しています。Stataと統計の資料をインターネットで確認するにはhttps://www.stata.com/links/をご確認ください。

3.3 Stata Press

Stata PressはStataCorpLPの出版窓口で、あらゆる分野におけるStataや統計の一般的なトピックに関する専門的な書籍、マニュアル、ジャーナルを提供しています。

ブラウザでhttps://www.stata-press.comのページを開きます。このサイトはStata Pressの出版物とその関連する活動にのみ割り当てられています。

- Stataのリファレンスマニュアルで使用しているデータセットや他のStata Pressの本で使用しているデータセットはこのサイトからダウンロードできます。詳しくはhttps://www.stata-press.com/data/ページを確認してください。これらのデータセットをStata内で使用するには、シンプルに「use https://www.stata-press.com/data/r18/*dataset_name*」と入力します。例えば「use https://www.stata-press.com/data/r18/auto」と入力すれば、autoデータセットが開きます。または、「webuse auto」と入力することもできます。詳細は[D] **web use**をご覧ください。
- Stata社が提供している全ての書籍やマルチメディア商品のオンラインカタログはhttps://www.stata-press.com/books/を確認してください。可能な限り目次、序文等の必要な情報は掲載しているので、その書籍が必要かどうか判断できるようになっています。
- これから出版される書籍の情報は、https://www.stata-press.com/forthcoming/のページで確認できます。

3.4 The Stata Journal

Stata Journal (SJ) は印刷および電子版の機関誌で、年4回発行しています。内容は統計、データ分析、教授方法、Stata言語の有効活用法などにわたります。SJは査読済みの論文ですが、短い注釈やコメント、コラム、ヒント、書評など、統計を様々な分野で利用する人に有益な情報も合わせて掲載しています。SJは初心者から熟練者までのすべてのStataユーザを対象にした機関誌です。Stataの熟練度は問いません。統計、実験計画、データ管理、グラフ作成、結果発表などを、特にStataで行う研究者のための論文誌です。

SJはSAGE Publishingから出版・購入できます。過去に出版された冊子や論文のアブストラクトはhttps://www.stata-journal.com/archives/で確認できます。3年以上前に出版された記事のPDFはSJのウェブサイトから無料でご利用いただけます。

SJを購読することをお勧めします。購読のお申し込みやSJの詳細についてはhttps://www.stata-journal.comをご確認ください。

SJに掲載されている記事に関係するプログラムを入手するには、次のように入力します。

```
. net from https://www.stata-journal.com/software
```

または

- メニューから ヘルプ > Stataジャーナル/コミュニティ投稿コマンド と選択する
- *Stata Journal*をクリックする

3.5 Webから機能を更新する/追加する

Stataの内部からインターネット上のファイルを直接開くことができます。Stataはhttp, https, ftpプロトコルを理解できます。まず、以下を試してください。

```
. use https://www.stata.com/manual/oddeven, clear
```

このコマンドは特に意味の無いデータセットをウェブサイト上からコンピュータにロードします。ホームページを持っている場合、この方法を使用するとデータセットを共同研究者や同僚と共有できます。データセットをホームページに保存すれば、世界中の研究者が使用できます。詳細は[R] **net**をご覧ください。

3.5.1 公式アップデート

アップデートのリリースに関しては特にスケジュールがあるわけではありませんが、通常、Stataはアップデートを月1回程度の頻度で提供します。アップデートの入力は簡単です。次のように入力します。

```
. update query
```

あるいは、**ヘルプ > アップデートのチェック** と操作します。ユーザが指示を出すまで、何もインストールされません。アップデートをインストールすると、以下を確認できます。

```
. help whatsnew
```

あるいは、**ヘルプ > 更新情報** と選択して更新された内容について確認します。公式アップデートはバグを修正したり新機能を追加するために提供します。

3.5.2 非公式アップデート

また、Stataには“非公式”アップデート、つまりStataユーザにより作成されたStataへの機能追加もあります。この“非公式”アップデートの提供者にはStata者のテクニカルスタッフも含まれます。Stataにはプログラミング機能がありますが、この機能を利用しないという方にとっても、これらの追加機能は有益ですので、ぜひご利用ください。では、次のように入力するところからはじめましょう。

```
. net from https://www.stata.com
```

または、メニューから**ヘルプ** > **Stataジャーナル/コミュニティ投稿コマンド** と選択します。

Statistical Software Components (SSC)のアーカイブは必ず見るようにしてください。このアーカイブではStataへの無料の追加機能を提供しています。sscコマンドはSSCアーカイブから必要としているパッケージを簡単に検索、インストールやアンインストールができます。次のようにコマンドウィンドウに打ちましょう。

```
. ssc whatsnew
```

このコマンドでサイトの更新事項を確認できます。興味を引くもの、気になるものがあれば次のように入力してみましょう（pkgnameはパッケージ名です）。

```
. ssc describe pkgname
```

これで詳細をご覧いただけます。既にパッケージをインストール済みの場合、オプションとしてインストールしたアップデートを確認するには次のように入力します。

```
. ado update pkgname
```

オプションとしてインストールされたアップデートや今までにインストールしたパッケージを確認するには、次のように入力します。

```
. ad oupdate all
```

詳細は[U] 28 インターネットによるプログラムの更新をご覧ください。

3.6 カンファレンスとトレーニング

3.6.1 カンファレンスとユーザグループミーティング

毎年、世界各国のさまざまな場所でカンファレンスやユーザグループミーティングが開催されています。

これらのミーティングでは、Stata社の技術担当やStataユーザが新機能、新たな使用方法、ベストプラクティスを共有しています。Stata社は、米国とカナダの両方でStata会議を開催および主催し、世界中の会議をサポートしています。

今後開催されるカンファレンスやミーティングに関してはhttps://www.stata.com/meeting/をご覧ください。

3.6.2 NetCourses

Stataのトレーニングコースを入門と上級者向けの2つのレベルで提供しています。ソフトウェアのトレーニングコースは一般的に費用と時間を必要とします。トレーニングコースの費用そのものに加えて、開催場所まで移動する費用および時間を消費するためです。インターネット上で受講できるコースは参加者の時間とお金を節約できます。

インターネット上のコースはStata NetCourses™と呼ばれています。

- **NetCourseとは何ですか。**
  NetCourseはStataのウェブサイトを通じて提供されるコースで、7〜8週間の期間に渡って行われます。メールアドレスとウェブブラウザがあれば、誰でも参加できます。
- **どのような仕組みですか。**
  毎週金曜日にパスワードで保護されたウェブサイトにレッスンが公開されます。週末、または月曜日にこのレッスンを読んでいただき、参加者は質問やコメントをメッセージボードに記入します。コースの講師は通常、これらの質問やコメントには掲載日と同じ日に返事をします。他の参加者はその質問やコメントに追加をしたり、返事をすることが可能です。そして、次のレッスンが金曜日に公開され、同じことが繰り返されます。
- **どれぐらいの時間がかかりますか。**
  コースに依存しますが、入門編のコースはおおよそ1週間に3時間程になるように設計されています。
- **3人のチームです。そのうちの1人がコースに登録し、他のメンバーにNetCourseの資料などを配っていいですか。**
  再配布はご遠慮ください。NetCoursesはコースの講師たちが費やす多大な時間に見合うような価格設定になっています。さらに、受講者数はディスカッションが管理できる程度の人数に抑えられています。NetCourseの価値は、普通のトレーニングコースのように、受講者同士あるいはコースの講師とのやり取りにあります。

- **インターネット上でコースを受講したことがありません。うまくいくと思いますが、うまくいかない可能性も考えられます。どのように自分への利益や価値を確認すればよいでしょうか。**
  全てのStataのNetCoursesは30日間の満足度保障が付いてきます。この30日間は最後の講座が終わってから開始します。

現在、NetCoursesで提供しているレッスンについては、https://www.stata.com/netcourseを参照してください。

NetCourseNow

NetCourseNowはNetCoursesと同じ内容を提供していますが、その講座の時間や速度を選択でき、NetCourseを教える専属の講師がつきます。

- **NetCourseNowとは何ですか。**
  NetCourseNowはNetCourseと同じ内容の講座ですが、自分のペースで進めることができ、自分の都合の良い日に開始できます。また、NetCourseNowでは、専属のNetCourse講師がつくので、直接メールで講座や練習問題に関する質問をすることができます。メールアドレスとウェブブラウザを持っていることが必須条件です。
- **どのような仕組みですか。**
  コースのレッスンは一度に提供されるので、自分の空き時間に自分のペースで勉強できます。また、専属講師のメールアドレスが提供されるので、何か質問があれば連絡をする事ができます。
- **どれぐらいの時間がかかりますか。**
  NetCourseNowでは自分のペースで講座の勉強をできます。どれだけの時間がかかるかは毎週どれほどの時間をかけるかに依ります。

3.6.3 授業型トレーニングコース

授業型トレーニングコースは、Stataの使い方、より正確にはStataの提供する高度な統計処理手順の活用法の習得を目指した強化トレーニングコースです。Stata社の社員が講師となり、外部のトレーニングセンターなどで集中的に開催されます。

- **授業型トレーニングコースの概要を教えてください。**
  これはインタラクティブでハンズオンな講座です。受講者は講師と共に実際に操作を行うので、Stataの使用方法を間近で見て学ぶことができます。質問も随時受け付けます。
- **自分のコンピュータを準備する必要がありますか。**
  これらの講座は最新版のStataを実装しているコンピュータ室で行われるので、自分のコンピュータを持ってくる必要はありません。もちろん、使用できるStataをインストールしているコンピュータがあるなら、お持ちいただいて問題ありません。
- **テキストはありますか。**
  各講座に関して、テキストセットを受け取れます。これは講座の内容を網羅すると共に、サンプルコマンドの出力結果も含んでいます。

  提供している講座に関しては、https://www.stata.com/training/classroom-and-web/を確認してください。

3.6.4 ウェブ型トレーニングコース

ウェブ型トレーニングコースは、Stataの使い方、より正確にはStataの提供する高度な統計処理手順の活用法の習得を目指した強化トレーニングコースです。Stata社の社員が講師となり、自宅やオフィスからオンラインでコースに参加できます。

- **ウェブ型トレーニングコースの概要を教えてください。**
  これはインタラクティブでハンズオンな講座です。受講者は講師と共に実際に操作を行うので、Stataの使用方法を間近で見て学ぶことができます。質問も随時受け付けます。
- **自分のコンピュータを準備する必要がありますか。**
  オンラインコースに参加するための高速インターネットに接続環境を持ち、且つStataを実装しているパソコン１台が必要です。最新版のStataをお持ちでない場合、期限付きのライセンスの提供を受けることができます。
- **テキストはありますか。**
  各講座に関して、テキストセットを受け取れます。これは講座の内容を網羅すると共に、サンプルコマンドの出力結果も含んでいます。

  提供している講座に関しては、https://www.stata.com/training/classroom-and-web/を確認してください。

3.6.5 出張トレーニングコース

出張トレーニングコースはそれぞれの出張先の要望でカスタマイズされます。Stata社の担当が出張し、要望に応じ、例えば新しいユーザにStataの使用法を教えたりStataの特殊な操作を紹介したりします。

- **出張トレーニングコースの概要を教えてください。**
  これらは授業型トレーニングのようにインタラクティブでハンズオンな環境で学ぶことができます。各出席者のためのコンピュータが必要です。
- **どのトピックを学ぶことができるのですか。**
  Stataに関することでしたらなんでもトレーニングで提供します。必要に応じてカリキュラムを作成するので、ご要望をお伝えください。

- **ライセンスはどのような仕組みになりますか。**
  トレーニングセッション中に必要なライセンスを提供します。このライセンスはコンピュータルームで行う場合でも参加者各自のノートパソコンで行う場合でも大丈夫です。ライセンスやインストールの情報はあらかじめ提供させていただくので、トレーニングが始まる前に起動することを確認できます。

詳細はhttps://www.stata.com/training/onsite-trainingをご覧ください。

3.6.6 ウェビナー

無料のウェビナーでは、Stataの既存と新機能の両方のデモンストレーションを行っています。*Ready, Set, Go Stata*は、Stataに初めて触れる方にデータの操作、グラフ化、そして解析について簡単に解説します。すでにStataのユーザであれば、*Tips and tricks*で開発者達のおすすめの機能をご覧ください。この1時間のウェビナーでは、Stataの統計、グラフ、データ管理やレポート作成機能について、詳細な解説がされています。

- **ウェビナーにアクセスするには？**
  ウェビナーにはAdobe Connectソフトウェアまたは Zoomによってライブ配信されます。
- **コンピュータとStataのライセンスは必要ですか？**
  Adobe ConnectまたはZoomを利用してウェビナーに参加するには、高速なインターネット回線に接続したコンピュータが必要です。
- **参加料は？**
  ウェビナーは無料です、ただし、参加するには事前登録が必要となっています。登録数は限定されていますので、早めに登録されることをお勧めします。

詳細は、https://www.stata.com/training/webinar/をご覧ください。

3.7 書籍やその他のサポート資料

3.7.1 読者向け

Stata社により出版されたStataに関しての本、あるいは他の著者が書いた出版物があります。興味のある人はStata Bookstore(https://www.stata.com/bookstore/)を開いてみてください。Stata社が提供する書籍に関しては、目次とStata社のテクニカルスタッフによるお勧めコメントを掲載しています。

3.7.2 著者向け

Stataに関連する本を執筆し、Stata社のbookstoreで提供したい場合は、bookstore@stata.comまでメールしてください。

もし本を執筆しているなら、無料で提供しているAuthor Support Programに参加できます。StataのプロがStataのコードを確認するので、そのコードが効果的で最新の用法であるかチェックできます。また、製作担当がStataの出力形式について助言でき、編集者と統計学者がStataに関する内容の正確さを確認します。詳細はhttps://www.stata.com/authorsupport/をご覧ください。

Stata関連の書籍を執筆することをお考えなら、StataPressから出版することもご一考ください。submissions@statapress.comまでメールしてください。
興味のある方はsubmissions@statapress.comまでメールしてください。

3.7.3 編集者向け

Stataの操作や解析を扱っている本を編集している場合は、無料のEditor Supportプログラムに参加することができます。Stataの専門家が書籍内のStataに関する記述やコードをレビュー、解析結果が有効であることを保証します。さらにStataの結果(グラフを含む)のフォーマットや関連項目に関するアドバイスを提供します。詳細は、https://www.stata.com/publications/editor-support-program/をご覧ください。

3.7.4 講師向け

講師向けのウェブページでは、教育リソースをチュートリアルビデオ、*Ready, Set, Go Stata*、ウェビナー、Stataチートシートなどが利用できます。詳細は、https://www.stata.com/teaching-with-stata/をご覧ください。

3.8 テクニカルサポート

Stataソフトに関して、Stata社は優れたテクニカルサポートを提供するようお約束します。効率的にお手伝いするために、以下の手順にしたがってください。

3.8.1 ソフトウェアの登録

Stata社が提供するソフトウェアのテクニカルサポート、アップデート、特別なセールス情報、その他の特典を利用するには登録が必要です。登録をすると*Stata News*を受け取ることができ、Stata社のスタッフに無料で質問をすることができます。Stata社に登録するには電子的な方法で登録できます。

電子登録：StataをインストールしてLicenseとActivation Keyを無事に入力できたら、デフォルトでWebブラウザが起動してStata社のWebサイトのオンライン登録ページが開きます。あるいは、後から手動で登録するには、Webブラウザで https://www.stata.com/register/ページを開いてStataを登録してください。

3.8.2 テクニカルサポートに連絡を入れる前に

テクニカルサポート部署に連絡をするのに必要な情報を集め始める前に、質問への回答がヘルプファイルに無いことを確認してください。「help」と「search」コマンドを使用してStata内にあるヘルプ項目を検索できます。また、次のように選択してみてください。
ヘルプ > **詳細** でマニュアルを確認して特定のコマンドを確認してください。コマンドの項目内に良くある質問や注意点が掲載されています。その他にも、便利な情報がStata社のWebサイトに用意されています。よくある質問（FAQ）ページ(https://www.stata.com/support/faqs/)はブックマークをして、必要に応じて情報の有無を確認してください。

テクニカルサポートに連絡を取る必要がある場合、https://www.stata.com/support/tech-support/のページを参照してください。

3.8.3 電子メールからのテクニカルサポート

テクニカルサポートに質問をする際に推奨される方法です。この方法は以下のような点で有利です。

- メールを送ると自動返信でメールを受け取った旨を伝えるメールが届きます。この連絡は、送ったメールが*Technical Services*に回答してもらうために転送されたことを意味します。
- 送られた質問を関連分野のスペシャリストに転送されます。
- メールで送られた質問は通常の営業時間はもちろん、週末や祝日でも回答を得ることも可能です。上記が必ずしも可能であるとは約束できませんが、その可能性もあり、また、電話の留守番電話に録音を残すよりも電子メールのほうが早く返信を得られます。
- 何らかのエラーメッセージや想定外の結果が出てきたときは、問題を確認するためのログファイルを添付ファイルで送ることが簡単にできます。

テクニカルサポートに連絡を取ることに関する情報は、https://www.stata.com/support/tech-support/を確認してください。

3.8.4 電話またはFAXからのテクニカルサポート

インストールに関するサポートの電話番号は03-3864-5211(ライトストーンのテクニカルサポートの番号)です。シリアル番号をお手元に準備してからお電話ください。また、コンピュータを操作できる状態で電話をいただけるのが最適です。Stata社への電話質問はインストールに関する質問のみ受け付けています。インストール以外のお問い合わせは、電子メールでお問い合わせください。

テクニカルサポートに連絡を取ることに関する情報は、https://www.stata.com/support/tech-support/を確認してください。

3.8.5 テクニカルスタッフ向けのコメントや提案

コメントや提案がございましたら、どうぞお送りください。皆様からのご意見が新しいStataのリリースに追加される機能に結びつきます。つまり、何もご意見をいただけない場合、最も必要としている機能が追加されないかもしれません。ご連絡いただくリクエストの情報を残しておくため、電子メールで情報をいただければ幸いです。新しい機能をリクエストする場合は、機能の追加を検討する際に参照すべき参考文献などがあれば、共にご連絡ください。ご意見・ご感想はservice@stata.comまでご連絡ください。

3.9 参考文献

Haghish, E. F. 2019. On the importance of syntax coloring for teaching statistics. Stata Journal 19: 83-86.

4 Stataのヘルプと検索機能

目次

4.1 イントロダクション

Stataのヘルプにアクセスするには、次のいずれかの操作を行います。

1. メニューから **ヘルプ** を選択する
2. helpとsearchコマンドを使用する

どちらの方法を選んでも、結果はビューワまたは結果ウィンドウに表示されます。青い文字はハイパーリンクテキストで、クリックをすると関連するエントリに移動できます。

4.2 使い始めてみましょう

初めてヘルプを使用する場合は、以下のどちらかをお試しください。

1. メニューバーから **ヘルプ** > **アドバイス** を選択する
2. コマンドウィンドウに「help advice」と入力する

どちらのステップでも、ビューワウィンドウでhelp_adviceというヘルプファイルが開きます。このアドバイスファイルはStata内で目的のトピック及びコマンドを検索するときの手順について説明しています。この例題ではStata内の“ロジスティック回帰(logistic regression)”を調べる手順を説明しています。

4.3 helpコマンド：Stataのヘルプシステム

次のように操作をしてヘルプファイルを開きます。

1. **ヘルプ** > **Stataのコマンド...** とメニューから操作し、コマンド編集エリアにコマンド名を入力してOKをクリック

または

2. 「help」の後にコマンド名を続けて入力

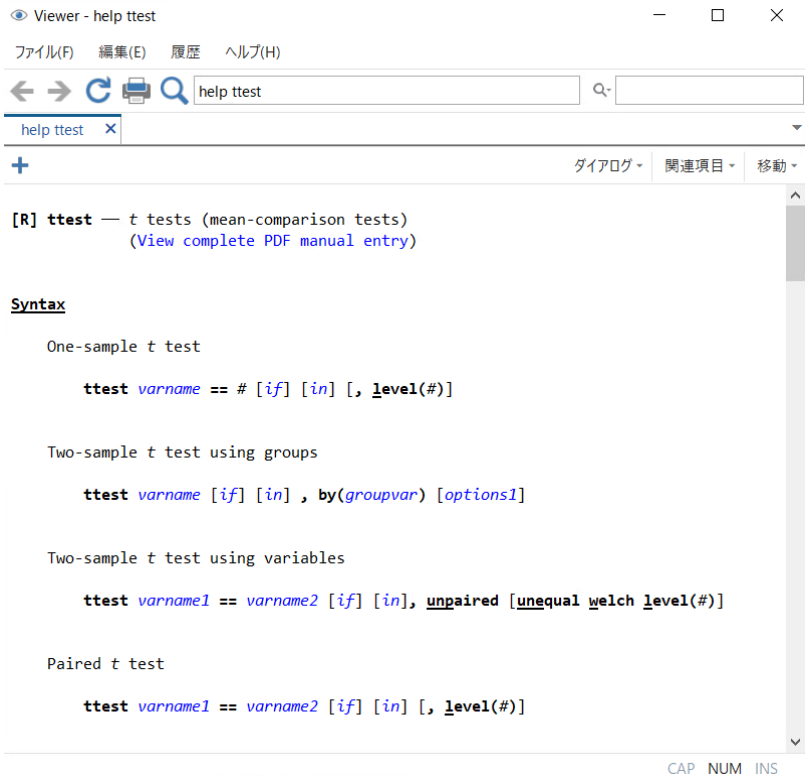
すると、Stataのヘルプファイルが開きます。これらのヘルプファイルは印刷(PDF)マニュアルにある内容を短くまとめたものです。では、Stataの「ttest」コマンドに関するヘルプファイルを開いてみましょう。次の操作を行ってください。

1. **ヘルプ** > **Stataのコマンド...** とメニューから操作し、コマンド編集エリアに「ttest」と入力してOKをクリック

または

2. コマンドウィンドウに「help ttest」と入力する

どちらの操作でも、結果は次の画像のようになります。



このヘルプ検索のポイントは、平均の均等性を検定するコマンドは「ttest」であると分かっていること、つまり「meanstest」ではないと分かっているということです。コマンドが分からない時は検索機能を利用します。

4.4 ヘルプ項目からPDFマニュアルにアクセスする

全てのStata内のヘルプファイルは対応するマニュアルへリンクがあります。例えば、help ttestの項目について読んでいる場合、ヘルプファイルのTitleセクションにある[R] ttestをクリックすると、直接マニュアル内の[R] ttest項目に移動できます。

Stata の文書を表示するのに最適なPDF ビューアの設定は、https://www.stata.com/support/faqs/resources/pdf-documentation-tips.htmlに記載しています。

4.5 検索

目的の操作を行うStataコマンドが分からない場合、コマンドをキーワード検索により見つけることができます。

1. メニューから **ヘルプ** > **検索...** と選択し編集エリアにキーワードを入力してOKをクリックする
2. 「search」の後にキーワードをつけて入力

searchコマンドは入力したキーワードでデータベースを検索し、ヒットしたStataコマンド、 www.stata.comに掲載しているFAQの回答、公式ブログ、Stata Journal内の項目を一覧で表示します。また、ウェブ上からダウンロードして使用可能になるユーザ定義の追加プログラムも検索します。

searchコマンドは一般的に使用する言葉で検索したいときや、探しているもの(特にユーザ定義の追加プログラム)がコンピュータにインストールされていない場合などに特に便利です。

4.6 searchコマンドについて

コマンドとメニュー、どちらの方法でsearchコマンドにアクセスしても、同じ結果を表示します。search コマンドには欲しい情報について入力すれば、それに関する情報を検索します。デフォルトでsearchは全ての資料を検索します。この資料にはシステムヘルプ、StataウェブサイトのFAQs、Stata Journal、あるいはユーザ定義の追加を含むStata関連のインターネット資料が含まれます。

searchコマンドは検索範囲を広げたり狭めたりすることも可能です。例えば、Kolmogorov-Smirnov検定を行い、分布の相等性を確認したい場合は、次のように入力します。

```
. search Kolmogorov-Smirnov test of equality of distributions
[R] ksmirnov . . . . . Kolmogorov-Smirnov equality of distributions test
(help ksmirnov)
```

実際には、ここまで完全に入力する必要はありません。「search Kolmogorov-Smirnov」と入力すれば十分です。検索の範囲を広げたい場合、「equality of distributions」と検索するとさらに長い検索結果のリストを確認でき、それにはksmirnovも含まれます。

searchコマンドを使用する際のガイドラインは次の通りです。

- 先頭の大文字は結果に影響しません。「Kolmogorov-Smirnov」と「kolmogorov-smirnov」を入力してみてください。
- 記号は結果に影響しません。「kolmogorov smirnov」を入力してみてください。
- 単語の順番は結果に影響しません。「smirnov kolmogorov」を入力してみてください。
- 単語を省略することはできますが、どれだけ省略できるかはその時々の状況により異なります。その際、音節で区切るようにしてください。「kol smir」を入力してみてください。search searchコマンドは多くの省略形を使うことができます。スペルミスを犯すより、省略するほうが良いでしょう。
- 「a, an, and, are, for, into, of, on, to, the, with」の単語は無視されます。使用する場合（「equality of distributions」と入力）と使用しない場合（「equality distributions」と入力）では、検索結果に違いはありません。
- search コマンドは複数形を理解します。特に最後に“s”が追加されるだけの場合は問題ありません。しかし、単数形で入力することをお勧めします。「normal distributions」ではなく、可能なら「normal distribution」と入力してください。
- 検索の項目は英語で入力してください。コンピュータの専門用語や他言語で入力しないでください。
- アメリカ式のつづりを使用してください。つまり、「colour」ではなく「color」と入力してください。
- 名詞を使用してください。「-ing」で終わる言葉や他の動詞は使用しないでください。「testing medians」ではなく「median tests」と入力してください。
- 少ない言葉を使用してください。言葉が追加されるごとに検索を制限します。「distribution」を検索すると1つの大きなリストが表示されます。「normal distribution」と検索すると、distributionのサブリストを表示します。

- 言葉に2つ以上の意味が含まれていることがあります。次の言葉はその内容を狭める際に使用できます。
  a. 「data」はデータ管理上の意味であることを示します。「order」は「データの順番（order of data）」または「オーダー統計(order statistics)」の両方の意味の可能性を含みます。「order data」と検索すると、orderをデータ管理の意味で制限できます。
  b. 「statistics(省略形はstat)」は統計の意味であることを示します。「order statistics」と検索すると、orderを統計の意味で制限できます。
  c. 「graph または graphs」は統計グラフに関する内容であることを示します。「median graphs」と入力すると、中央値のグラフを作成するコマンドの一覧が表示されます。
  d. 「utility(省略形はutil)」はユーティリティコマンドであることを示します。searchコマンド自体はデータ管理、統計、グラフのコマンドではなく、ユーティリティコマンドです。
  e. 「programs または programming(省略形はprog)」はプログラミングの意味を示します。「programming scalar」と入力すると、スカラーに関するプログラミングのサブリストを入手できます。

searchコマンドにはさらに多くの機能があります。詳細は[U] 4.8 searchコマンド: 詳細をご覧ください。

4.7 helpコマンドについて

helpとsearchコマンドは、どちらもユーザが入力の際に引き起こす誤りなどを適切に処理することが重要になります。例えば、頻繁に使われるコマンド名は省略できます。「help regres」または「help regress」と入力すれば、regressコマンドに関するヘルプファイルが開きます。

入力したコマンドをStataの公式ヘルプファイルまたはユーザ作成の追加プログラム中で見つけることができなかった場合、Stataは自動的にsearchコマンドと同等の検索を行います。例えば、「help ranktest」と入力すると「“help for ranktest not found”」という返答の後、「search ranktest」が実行されます。検索結果によると、ranktestコマンドはStata Journal, Volume 7, Number 4に掲載している*Enhanced routines for IV/GMM estimation and testing*の記事に含まれていることが分かります。

Stataではいくつかの省略形について、問題に直面する可能性もあります。例えば、Stataにはコマンド、ksmirnovがありますが、このコマンドのスペルを忘れてしまい、コマンドはksmirだと勘違いした場合、

```
.help ksmir
No entries found for search on "ksmir"
```

そのコマンドをhelpで検索すると、ksmirのヘルプファイルは見つかりません。その後、Stataは自動でその単語を検索しますが、メッセージはksmir の検索は何も結果にヒットしなかったことを示しています。つまり、searchコマンドを使用する際には本当に探したいものを入力するほうがお勧めです。search kolmogorov smirnov.

4.8 searchコマンド：詳細

searchコマンドは、**ヘルプ**メニューからでは使用できない機能をいくつか提供します。
searchコマンドの構文の全体像は次のようになります。

search *word* [*word ...*] [, [all / local / net] author entry exact faq historical or manual sj]

下線部分はStataが理解できる最短の省略形を意味し、[角括弧]内はオプションを示します。

オプションallはデフォルトで設定されており、これはローカルなキーワードデータベースとインターネット上の資料の両方を検索します。

オプションlocalは検索がStataのキーワードデータベースだけに対してのみ行われることを示します。

オプションnetはStataのnetコマンドでアクセス可能なインターネット上の資料を検索します。「search word [*word . . .*], net」は、「net search word [*word . . .*]」とオプションなしで入力する場合と同じです。詳細は[R] netをご覧ください。

オプションauthorはキーワードではなく著者名で検索するように指定します。オプションentryはキーワードではなくエントリIDで検索するように指定します。オプションexactは省略形での検索を防ぎます。

オプションfaq はStataおよび選択したウェブサイト内にあるFAQのエントリを検索します。

オプションhistoricalは検索エントリが歴史上の意味を持つものに制限します。デフォルトでこのようなエントリはリストされません。

オプションorは searchコマンドの後に続く言葉がエントリと関連があるか指定します。デフォルトでは、 searchコマンドの後に入力された言葉の全てが関係あるエントリだけが表示されます。

オプションmanualは検索を*User's Guide* と全ての*Reference*マニュアルのみに制限します。

オプションsjは*Stata Journal* と *Stata Technical Bulletin*.のエントリに制限して検索します。

4.8.1 検索の仕組み

searchコマンドはデータベース(ファイル)を有しており、この中に*User's Guide*、*Reference* マニュアル、文書化されていないヘルプファイル、NetCourses、Stata Press書籍、StataのWebサイトに掲載されているFAQs、StataCorpのYouTubeチャンネルにあるビデオ、StataCorpの公式ブログの記事、選択したユーザ定義のFAQsや例題、*Stata Journal* と*Stata Technical Bulletin*で掲載された記事のタイトルなどが含まれています。このファイルの中にはそれぞれのエントリに関係する言葉、キーワードのリストがあります。

「search *xyz*」と入力するとsearchコマンドはこのファイルを読み取り、キーワードのリストと*xyz*を見比べます。キーワードの中でxyzを見つけた場合、あるいはxyzを省略形として使用できる場合はエントリを表示します。

「search xyz abc」と入力すると、searchコマンドは同じことを実行しますが、両方のキーワードがあるエントリだけを表示します。入力する順番による違いはないので、「search linear regression」と入力しても「search regression linear」と入力しても同じ結果になります。

search コマンドが検索するエントリの合計数については検索データベースがどのように構築されているかによります。特定の検索に関して、リストされるエントリが表示されないことと比べて多くあるほうが良いという仮定の下、過度のキーワードが登録されています。それでも、キーワードに関しては推測の域を出ることがありません。検索する言葉は「normality test」、「normality tests」、「tests of normality」のどれで検索するのが良いのでしょうか。最もよい言葉は「normality test」ですが、この場合、全て同じ結果を表示します。一般的に、単数形の言葉を使い、不必要な言葉は使わないようにしてください。検索するキーワードに関するガイドラインは、先ほどのセクション[U] 4.6 searchコマンドについてで詳しく説明しています。

4.8.2 著者検索

searchコマンドは通常、search に続く単語をキーワード一覧と比較してエントリを表示します。オプションauthorを使用すると、search に続く言葉と著者名を比較します。検索データベースでは、*Stata Journal* と STB 記事、Stata Press書籍、StataCorp の公式ブログ、webページのFAQsの著者名を記載しています。

例えば、[R] kdensityの謝辞にIsa´ıas H. Salgado-Ugarteという人物がいます。この方がStata Journalに他の論文を投稿していないか調べるとします。次のように入力します。

```
. search Salgado-Ugarte, author
(省略)
```

Salgado-Ugarte のような名前は慣れない人は混乱するでしょう。searchコマンドは名前の全てを指定する必要は特にありません。入力したものは名前の全ての“単語”と比較され、どこか1部分でもマッチすればそのエントリはリストされます。ハイフンは特殊な記号なので、無視しても問題ありません。つまり、先ほどと同じエントリのリストを入手するには「Salgado」、「Ugarte」、またハイフンなしの「Salgado Ugarte」でも可能です。

実際のところ、Salgado-Ugarteが書いた全てのエントリを探すには、次のように入力する必要があります。

```
. search Salgado-Ugarte, author historical
(省略)
```

STBの記事(insert)で、データベース内にhistorical付きで保存されたものは、デフォルトでは表示されません。オプション
historicalにより今までの全ての記事が結果として表示されます。

4.8.3 エントリID検索

オプションentryを使用すると、searchコマンドはエントリIDとして入力したものを比較します。エントリIDはタイトルではありません。これは、タイトルの左側に記載されている英数字で、どこを確認するべきか説明しています。
例えば、次の例を見てください。

```
[R] regress . . . . . . . . . . . . . . . . . . . . . . Linear regression
(help regress)
```

この例では“[R] regress”がエントリIDです。次の例では、

```
GS      . . . . . . . . . . . . . . . . . . . .Getting  Started  manual
```

“GS”がエントリIDです。次の例では、

```
SJ-14-4 gr0059 . . . . . Plotting regression coefficients and other estimates
        (help coefplot if installed) . . . . . . . . . . . . . . . . . B. Jann
        Q4/14    SJ 14(4):708--737
        alternative to marginsplot that plots results from any
```

```
            estimation command and combines results from several models
            into one graph
```

“SJ-14-4 gr0059”がエントリIDです。

entryオプションを使用したsearchコマンドは、これらのエントリを検索します。

つまり、*Reference*マニュアル用に目次を作成するには、次のように入力します。

```
. search [R], entry
(省略)
```

16番目に発行されたSTBの目次を作成するには、次のように入力します。

```
. search STB-16, entry historical
(省略)
```

*sbe19*に関連する全ての情報をリストするには、次のように入力します。

```
. search sbe19, entry historical
(省略)
```

繰り返しになりますが、関連する項目のどれかがヒストリカルとして分類されている可能性があるため、ここでもhistoricalオプションを使用します。

4.8.4 FAQ検索

FAQsについて検索するには、faq オプションを使用します。

```
. search logistic regression, faq
(省略)
```

4.8.5 リターンコード

User’s Guide と*Stata Reference*マニュアルのエントリをインデックスで管理する他に、searchコマンドはリターンコードを調べる時にも使用できます。

リターンコード131に関する情報を調べるには、次のように入力しましょう。

```
. search rc 131
        [R] error messages . . . . . . . . . . . . . . . . . . . Return code 131
        not possible with test;
        You requested a test of a hypothesis that is nonlinear in the
        variables. test tests only linear hypotheses. Use testnl.
```

Stataで使用される全てのリターンコードを確認するには、次のように入力します。

```
. search rc
(省略)
```

4.9 net searchコマンド：インターネット上の資料を検索する

メニューから**ヘルプ > 検索...**と操作すると、2種類の項目から選択できます。最初の選択肢、**ドキュメントとFAQの検索**は前のセクションで説明しました。2番目の選択肢は**インターネットリソースの検索**です。Stataのこの機能は、インターネット上の資料を検索する時に使用します。

検索ダイアログボックスでSearch net resourcesを選びキーワードを編集エリアに入力すると、Stataはインターネット上の全てのユーザ定義プログラムを検索します。このユーザ定義プログラムはStata JournalやSTBで出版したものを含みます。ビューワに表示された項目をクリックすると、その項目のヘルプファイルに移動できます。

最終的に、「net search *keywords*」とStataで入力すると、結果を結果ウィンドウに表示することができます。net searchコマンドを使用するときのシンタックスの全容は、[R] **net search**をご覧ください。

5 Stataの種類

目次

5.1 プラットフォーム

Stataは多くのプラットフォームで使用できます。利用できるコンピュータは以下です。

Stata for Windows 64-bit, x86-64

Stata for Mac, 64-bit x86-64

Stata for Linux, 64-bit x86-64

Stataとしての動作はプラットフォームに依存しません。Stataに入力するコマンドは変わりませんし、Stataも乱数に至るまで同じ結果を出力します。それぞれでファイルも共有できます。作成したデータセットは他のコンピュータでも使用できます。同様にグラフ、プログラムや他の出力ファイルも異なるコンピュータで利用できます。プラットフォーム間でファイルをやり取りするには、そのファイルをコピーするだけです。変換などは必要ありません。

動作が速いコンピュータもあれば遅いものもあります。また、メモリ搭載量の多いコンピュータもあるでしょう。Stataを実際に使用する際は、高メモリ高速動作の方が良いでしょう。

Stataを購入すると、上記のプラットフォームのどれにでもインストールできます。StataのライセンスはOSごとに固定している訳ではありません。

5.2 Stata/MP, Stata/SE, Stata/BE

Stataには3つの種類があります。種類と言うよりサイズと言ったほうが適切かもしれません。サイズが大きい方から順に、 Stata/MP, Stata/SE, Stata/BEとなります。 (Stata17以前ではStataの種類をflavorsと呼んでいました。また、Stata/BEはStata/ICと呼ばれていました。)

Stata/MPはStataのマルチプロセッサ対応版です。このStataは2から64までの複数のCPU、あるいはコアで動作します。Stata/MPユーザはいくつのコアを処理に使用するか(1つからライセンスで許可しているコア数まで)決めることができます。Stata/MPはStataの中で最も処理が速い種類です。並列化の詳細は全て内部処理の為、他の種類のStataと同様に使用できます。Stata/MPの動作、またコア数の増加に伴う処理速度の増加に関するレポートはhttps://www.stata.com/statamp/report.pdfで確認してください。

Stata/SEはStata/MPと同じですが、1つのCPUのみを使用します。Stata/SEは複数のCPU、あるいはコアのコンピュータで実行できますが、1つのCPUあるいはコアしか使用しません。SEは「standard edition」の省略形です。

Stata/MPは最速の処理が行えることに加えて最大のデータ量を扱うことができます。Stata/MPは1,099,511,627,775を理論値上限とする観測値の扱いが可能ですが、現実的には上限にたどり前にメモリ容量が不足することと思います。また、120,000個の変数を持つことができます。統計モデルでは65,532個までの変数を組み込むことができます。

Stata/SEは、メモリが十分にあるならば2,147,583,647個の観測値を持つことができます。32,768個までの変数を利用でき、統計モデルでは10,998個までの変数を組み込むことができます。

Stata/BEはStataのBasic版です。メモリが十分にあるならば2,147,583,647個の観測値を持つことができ、2,048個までの変数を利用できます。統計モデルでは800個までの変数を組み込むことができます。

5.2.1 所有しているバージョンを確認する

ライセンス（*License* ）と許可キー（*Authorization Key* ）を確認してください。購入された全てのStataにはインストール中に入力するコードを記載したライセンスと許可キーが同梱されています。これは購入したStataの種類を決定します。

他の種類にアップグレードしたい場合は、Stata社または代理店に問い合わせてください。通常、アップグレードする時に必要なのは正しいコードが記載されているアップグレードライセンスと許可キーです。Stataの種類は全て同じDVDに収録されています。

もし、ある種類のライセンスを購入してそれよりも下位の種類を使用したいという場合は、それらも使用できます。これは、例えば職場では大型コンピュータを使用しているが、家ではパソコンを使用している時などに便利です。しかし、1ライセンス(あるいは購入分だけのライセンス)のみを所有していることを忘れないでください。Stataを自宅と職場、2つのコンピュータにインストールしてそれぞれ異なるタイミングで使用することは法的にも倫理的にも認められています。しかし、2つを同時に使用するのは認められていません。

5.2.2 インストールしているバージョンを確認する

既にStataがインストールしてある場合、普段のようにStataを起動して、「about」と入力します。すると、次のような画面が表示されます。こちらを確認してください。

```
. about
Stata/MP 18.0 for Windows (64-bit x86-64)
Revision date
Copyright 1985-2023 StataCorp LLC
Total usable memory: 8388608 KB
Stata license: 10-user 32-core network perpetual
Serial number: 18
  Licensed to: Stata Developer
               StataCorp LLC
```

5.3 Stata/MP, SE, BEのサイズ制限

Stata/MPはStata/SEやStata/BEに比べて、より多くの変数、大きなモデル、長いコマンド行を使用出来ます。使用可能な変数が大きいので長いコマンド行とマクロは必須になります。大きなモデルとはStata/MPとStata/SEがより多くの独立変数を有した統計モデルで推定できることを意味します。詳細は、

5.4 Stata/MP, SE, BEの処理速度比較

Stata/MPとStata/SEのパフォーマンスを比較したホワイトペーパーを作成しました。詳細はそのホワイトペーパー（https://www.stata.com/statamp/report.pdf）を確認してください。このホワイトペーパーにはコマンドごとのパフォーマンス計測結果が記載されています。

簡単にまとめると、デュアルコアコンピュータは、Stata/SEで実行するコマンドの71%の時間で処理が終わります。もちろん、実際の速度はコマンドに依存するので、SEの半分の時間で終わるコマンドもあれば、処理速度はあまり変わらないものもあるでしょう。統計の推定コマンドは、Stata/MPはStata/SEの59%の時間で処理が済みます。ここで記載している数値は中央値です。パフォーマンスによる処理の高速化の平均値は、中央値より速くなっています。これは全体的に実行までに時間がかかるコマンドの処理の方が高速化されているからです。

4コアで起動するStata/MPはStata/SEの処理速度の50%(すべてのコマンド)と35%(推定コマンド)で実行できます。どちらの数値も中央値です。

Stata/MPは64コアまでサポートしています。

Stata/BEはStata/SEよりも速度は遅くなりますが、これらの違いが顕著になるのはStata/BEの制限の値に近くなっ

た時です。Stata/SEはメモリ使用量がより大きくなり、それ以外にもより大きなデータセットを効率的に処理するために比較確認するためのテーブルメモリを持っています。これらの大きなテーブルの恩恵を感じられるのはStata/BEの上限を超えた後になります。Stata/SEは大きなデータセットを処理するために設計されました。

これらの違いは全てテクニカルな内容で内部処理の項目になります。ユーザの視点では、Stata/MP, Stata/SE, Stata/BE は全て同じように動作します。

5.5 Stata/MP, SE, BEの機能比較

全てのStataの種類と全てのプラットフォームで提供される機能は同じです。違いは処理の速度とこれまでのセクションで説明した制限になります。詳細については、「help stata/mp」「help stata/se」「help stata/be」をご覧ください。

6 メモリの管理

目次

6.1 メモリのサイズについて

Stataはメモリにロードしたデータを元に作業します。正確には、Stataは同時に複数のデータセットをメモリに記憶することができます、詳細は、[D] frames Introをご覧ください。

Stataはセッションが進行すると共に、使用するメモリ量を自動で増減します。StataはOSのメモリも使用できますので、実際のメモリと仮想メモリとの間に差をつけません。仮想メモリは実際の物理メモリがなくなったときにOS側がディクス内部に作成するメモリです。仮想メモリでの動作は遅くなりますが、実際のメモリにロードできない大きいデータセットなどに便利です。Stataが使用できるメモリに上限を設けたいときは「max_memory」を設定してください。詳細は [D] **memory**をご確認ください。Linuxではmax_memoryを設定することを強くお勧めします。詳細は[D] **memory**の中の*Serious bug in Linux OS*をご覧ください。

6.2 データの圧縮

Stataはデータをメモリに保存します。compressコマンドはデータの精度やその他の不利益が起こらない状態で使用するメモリの容量を減らします。詳細は[D] **compress**をご覧ください。適度なタイミングで「compress」を入力する方が良いでしょう。

compressコマンドは保存した形式の確認を行い、データの精度を失わない程度にデータ形式を変更します。例えば、float形式として保存している変数があるとします。実際の値は-127から100の整数のみを有しています。このような場合、compressコマンドはこの変数の保存形式をbyteに変更します。これで各観測値に対して3バイトずつ少なくなります。もし、100個ほどの変数がこのような形式に変更できた場合、観測値ごとに300バイトを節約でき、もし3,000,000個ほど観測値が含まれている場合、合計で900MBほど節約できます。

6.3 maxvarの設定

「“no room to add more variables”, r(900)」というようなエラーメッセージが表示されたとき、いきなりStataの制限に達したと考えないでください。

maxvarコマンドは使用可能な変数の数を指定します。デフォルトの設定はStata/MP, Stata/SE, Stata/BEの種類のうち、どれを使用しているかにより依存します。現在の設定を確認するには「query memory」と入力してください。

Stata/MPでは、最大120,000まで設定できます。Stata/SEでは、最大32,767まで設定できます。maxvarの設定値は、実際に必要な変数の数よりも多めにします。最低でも20は多くしておくと良いでしょう。多すぎないよう加減してください。参考までに、変数10,000個の使用メモリは約0.5MBです。

maxvarを再設定するにはset maxvarコマンドを使用して、以下のように入力します。

```
set maxvar # [, permanently]
```

上記で#には、2,048 ≤ # ≤ 120,000 の範囲を入力できます。maxvarの設定はセッション中に繰り返し行うことができます。permanentlyオプションを使用すると、maxvarの設定をそのセッションだけではなく、その後に行うセッションでも同じ値に設定できます。set maxvarの設定を10,000増加させるごとに、変数名のためのメモリ約1.3MBが変数の値の保存分のほかに必要となります。

6.4 memoryコマンド

memoryコマンドはStataの中でメモリを使用している主要な内容を確認できます。次のコマンドで確認できます。

```
. use https://www.stata-press.com/data/r18/regsmpl
(NLS women 14-26 in 1968)

. memory

Memory usage
                                            Used            Allocated
------------------------------------------------------------------------------
Data                                     856,020           67,108,864
strLs                                          0                    0
------------------------------------------------------------------------------
Data & strLs                             856,020           67,108,864

------------------------------------------------------------------------------
Data & strLs                             856,020           67,108,864
Variable names, %fmts, ...                 4,644               70,527
Overhead                               1,081,344            1,082,136

Stata matrices                                 0                    0
ado-files                                  6,606                6,606
Stored results                                 0                    0

Mata matrices                                  0                    0
Mata functions                                 0                    0

set maxvar usage                       5,281,738            5,281,738

Other                                      4,157                4,157
------------------------------------------------------------------------------
Total                                  7,229,845           73,554,028
```

詳細は[D] memoryをご覧ください。

6.5 保存したデータセットのために一時的なメモリを確保する

Stataの特徴の1つとして、データセットのpreserveとrestoreがあります。これらは、解析の途中で、データを復元することができます。Stata/MPでは、このためのデータセットのコピーをメモリ上に作成して、素早く保存と復元を行います。Stata/SEとStata/BEではディスク上にコピーを作成します。

Stata/MPではこのデータセットのコピーがディスクへのスワップを制御するために、コマンドset max_prereservemで設定を行います。メモリの詳細は[D] preserveをご覧ください。

7 -more-条件

目次

7.1 説明

Stataは、デフォルトの設定において出力結果の表示を途中で止めません。スクリーンに収まらない出力結果は、すでに表示された結果をスクロールバックして確認できます。

Stataでは、出力結果をスクリーン１画面分ごとに出力を一旦停止させることもできます。この機能を有効にする場合、「set more on」コマンドを実行します。詳細は[] moreをご覧ください。

「set more on」を実行すると、Stataはコマンドの出力結果をスクリーン１画面分ごとに一旦停止させるようになります。-more-がスクリーン下部に表示された場合、次の操作のうちどれかを行ってください。

押すボタン...	Stataの挙動...
LキーまたはEnterキー	次の行を表示
Qキー	Breakボタンをクリックと同じ反応
スペースキーまたは他のキー	次の画面を表示

また、*-more-条件をクリア* ボタンをクリックすることも可能です。このボタンは下向き矢印が描かれた緑色の円いボタンです。

-more-はStataの表現方法として、ユーザに他にも表示する内容がある事を示します。しかし、表示するには画面の中にある何か他の項目が移動する事を示します。

7.2 moreを設定する

「set more on」と入力すると、-more-条件が要所で表示されます。

「set more off」と入力すると、-more-条件は表示されず、出力結果は高速スクロールで全て表示されます。

プログラミングを行うユーザの皆様： doファイルの中で「set more」で設定を変更した場合、doファイル完了時に設定はdoファイル実行前の状態に戻ります。

7.3 moreのプログラミングコマンド

adoファイルでプログラムを作成しているとき、画面がいっぱいになったときに
-more-条件の表示に特別な処理は必要ありません。Stataが自動で処理します。

もし、-more-条件を強制的に表示したい場合は、moreコマンドをプログラムに追加できます。moreの構文は以下のとおりです。

```
more
```

moreでは何も引数を使用しません。詳細は[P] **more**をご覧ください。

8 エラーメッセージとリターンコード

目次

8.1 構文の入力を間違えた時

エラーの発生時、Stataはエラーメッセージとリターンコードを返します。例えば次です。

```
. list myvar
no variables defined
r(111);
```

上記では、Stataに myvarという変数の観測値をリストするように指示しました。データをメモリにロードしていないので、Stataは“no variables defined”と“r(111)”を返します。

“no variables defined”がエラーメッセージです。

111はリターンコードと呼ばれています。青いハイパーリンクが付いているリターンコードをクリックすれば、そのエラーについての詳細を確認できます。

8.1.1 エラーを許容する

“no variables defined”というメッセージとr(111)の後は特に何も表示されません。Stataはエラーが発生しなかったかのように動作します。

通常、エラーメッセージを確認すれば修正すべき内容も分かります。もし、メッセージを見てもエラーの理由が分からない場合、先程の数値であるリターンコードをご確認ください。詳細は[P] errorの項目内にまとめられています。

8.1.2 ユーザ定義プログラムやdoファイルはエラーで中断する

ユーザ定義プログラムやdoファイルでエラーが発生すると、そのプログラムあるいはdoファイルはすぐに実行を中止し、エラーメッセージとリターンコードを表示します。

例えば、次のようなdoファイルを見てみましょう。

begin myfile.do

```
use https://www.stata-press.com/data/r18/auto
decribe
list
```

end myfile.do

2行目を見てください。本来入力すべきコマンドは「describe」ですが、誤って「decribe」と入力したようです。「do myfile」と入力してこのdoファイルを実行した時に表示される結果は次のようになります。

```
. do myfile
.use https://www.stata-press.com/data/r18/auto
(1978 automobile data)
```

```
. decribe
command decribe is unrecognized
r(199);
end of do-file
r(199);

. _
```

最初のエラーメッセージとリターンコードは誤った「decribe」を入力したために表示されました。このエラーは、その後のdoファイルの実行を中断したため、有効であるはずのlistコマンドは実行されませんでした。

8.1.3 エラーを容認するための上級プログラミング技術

エラーの原因はスペルミス等に限りません。より本質的な場合もあります。あるデータセットでは有効なコマンドも、他では有効でない場合もあります。上級プログラミングでは、エラーの有無で使用データセットを変えるため、当初からエラーを想定することもあります。

プログラム作成者はリターンコードを確認して、無視できるエラーか判断したり、プログラムが適切に動作するよう変更したりできます。詳細は[P] **capture**に記載されています。

doファイルの最中にエラーで止まるのを防ぐには、doコマンドのnostopオプションを使用してください。

```
. do myfile, nostop
```

8.2 コマンドのタイミングを入手するためのリターンメッセージ

エラーメッセージとリターンコード以外に、リターンメッセージという、通常は表示されないものがあります。通常、「summarize tempjan」は、次のような結果を表示します。

```
. use https://www.stata-press.com/data/r18/citytemp
(City temperature data)
. summarize tempjan
```

Variable	Obs	Mean	Std. dev.	Min	Max
tempjan	954	35.74895	14.18813	2.2	72.6

ここで、次のように入力してみましょう。

```
. set rmsg on
r; t=0.00 10:21:22
```

セッション中のどこかで、Stataはリターンメッセージを表示します。

```
.summarize tempjan
```

Variable	Obs	Mean	Std. dev.	Min	Max
tempjan	954	35.74895	14.18813	2.2	72.6

```
r; t=0.01 10:21:26
```

「r; t=0.01 10:21:26」という行がリターンメッセージです。

「r;」はStataがコマンドを無事に成功したことを示します。

「t=0.01」はコマンド実行の所要時間（Enterキーの入力からこのメッセージの表示まで）を秒単位で示しています。このコマンドは100分の1秒を要したことになります。続きには、現在時刻を24時間表示で表示します。ここでは、コマンドは午前10:21に完了したようです。

Stataでファイル(doファイルと呼ばれます)に保存したコマンドを実行すると、出力のログを取ることができます。中には、doファイルを実行した時に詳細なリターンメッセージが便利であると感じるユーザもいます。このようなユーザは長いプログラムを作成しているので、夜中にプログラムを実行し、その出力結果をログにまとめておきます。翌朝、その出力結果を確認する時に一部分で間違いを見つける事もあるでしょう。その場合はリターンメッセージを確認して、その部分を再実行するのにどれぐらいの時間がかかるのかおおよその見当をつける事ができます。

「set rmsg on」はどこに設定しても構いません。

Stataにリターンメッセージの表示を中止させる場合は、「set rmsg off」と入力してください。

さらに、任意のタイミングでコードのサブセットを入手する方法もあります。詳細は[D] timerをご覧ください。

9 Breakキー

目次

9.1 Stataが実行中の動作を中止する

実行中の動作を中止してStataのドットプロンプトへ戻るには、中断をクリックします。

Windows版Stata：	**中断**ボタンをクリック(赤くて大きなXがあるボタン)、または *Ctrl+Pause/Break*を押す
Mac版Stata：	**中断**ボタンをクリック、または *Command*+.(ピリオド)を押す
Unix(GUI)版Stata：	**中断**ボタンをクリック、または *Ctrl+k*を押す
Unix(console)版Stata：	*Ctrl+c* を押す、または *q*を押す

このマニュアル内の別の箇所では、このアクションは*中断*をクリックする、とだけ記述します。中断(Break)はStataに、今実行している動作を中断して、ユーザがコントロールできる状況に可能な限り早く戻るように伝えます。

入力プロンプトやコマンドを入力中に*中断*をクリックしても、Stataは何も行いません。これは、既にユーザのコントロール下にあるからです。

Stataが何かを行っている最中、例えば新しい変数を作成、データセットをソート、グラフを作成、などに*中断*をクリックすると、Stataは実行中のことを中断し、その実行していたものを元に戻し(Undo)、入力プロンプトを表示します。これで、コマンドを実行する前の状態に戻ります。

例題1

ロジットモデルのフィットを行い、コマンドを入力して、Enterを押した後に、重要な変数を入力し忘れた事に気がついたとします。

```
. logit foreign mpg weight
Iteration 0:   log   likelihood =  -45.03321
Iteration 1:   log   likelihood = -29.898968
-Break-
r(1);

. _
```

*中断*をクリックすると、Stataは—Break—と表示してからr(1);を表示します。*中断*をクリックすると、必ずリターンコード1を返します。これが、エラーコードではなくリターンコードと呼ばれる所以です。1のコードはエラーを意味するものではありません。しかし、コマンドがタスクを完了しなかったことを示します。

9.2 中断をクリックした時の影響

一般的に、中断をクリックしても、影響は何もありません。先程のセクションで説明していますが、Stataはそれまで行っていた計算などを一度元に戻します(Undo)。つまり、Stataの内部の状態は、あたかもコマンドを実行しなかったかのような状態になります。これには2つだけ例外があります。

データをディスクから「import delimited」、「infile」、「infix」のいずれかのコマンドを使用して読み込んでいる場合、既に読み込まれているデータはメモリに残ります。この動作は、読み込みを中断した理由が、現在読み込んでいるデータを一度確認してから続きを読み込みたいから、という考えに基づいています。もし異なっていれば、いつでも「clear」を行ってください。

```
. infile v1-v9 using workdata
(eof not at end of obs)
(4 observations read)
-Break-
r(1);
```

もうひとつの例外はsortです。メモリに大きなデータセットがあり、それにソートをかけた後に考えを改めたとします。

```
. sort price
-Break-
r(1);
```

データセットをあらかじめ変数prodidでソートしていた時に変数priceでソートをしたところを中断すると、ソートは解除されます。つまり、sortコマンド中に*中断*を実行すると、Stataはソートを解除してソートのないデータを表示します。

9.3 プログラミングについて

基本的に、Stataでプログラミングを行う際に中断の扱いで注意するべき事は特にありません。これは、中断によりStata側で自動的に処理するためです。プログラム、またはdoファイルを作成した場合、実行してから*中断*を押すと、Stataは普通の内部コマンドと同じようにそのコマンドの実行を中断します。

上級プログラマーはプログラム実行中にどのようにデータが削除されるか心配するかもしれません。テンポラリ変数を作成して後からドロップする、あるいはテンポラリファイルを作成したので後から削除することもあるでしょう。Stataのユーザが*中断*をクリックした時に、これらのテンポラリ変数が削除されるのはどうすれば分かるのでしょうか。

このようなテンポラリアイテムは名前がtempname, tempvar, tempfileコマンドなどで取得されると自動で削除されます。詳細は[U] 18.7 テンポラリオブジェクトをご覧ください。

しかし、コマンドのグループを継続して実行する必要があるプログラムもあるでしょう。途中で中断すると、ユーザのデータは中間あるいは未定義の状態で保存されます。そのような時のために、Stataは次のコマンドを準備しました。

```
nobreak {
        ...
}
```

詳細は[P] breakをご覧ください。また、Mataの中でBreakキーを使用するには[M-5] setbreakintr()を参照してください。

10 キーボードの使用

目次

10.1 説明

キーボードは、一般的な操作を行えますが、いくつかStata特有の追加機能があります。

- 前に入力したコマンドを再度入手できるキーがあります。また、履歴ウィンドウのコマンドを1回クリックするとリロードでき、2回クリック(ダブルクリック)すると実行できます。この機能は*Getting Started*マニュアルでも紹介しています。
- Unix(console)版Stataを使用するユーザ向けに、コマンド編集用の機能があります。Unix(console)版のインターフェイスではクリックなどの機能を提供しないためです。
- OSやユーザインターフェイスの違いに関係なく、キーボードに*F*(ファンクション)キーがあるなら、これらのキーには特別な意味があり、これらのキーの定義は変更できます。

10.2 F(ファンクション)キー

Windowsユーザ向け：*F3*と*F10*はWindows の内部で予約済みです。このキーはプログラムできません。デフォルトで、Stataでは下記のファンクションキーが定義されています。

*F*キー	定義
F1	help advice;
F2	describe;
F7	save
F8	use

上記のようにエントリの後にセミコロン(;)がある場合、*Enter*キーの入力を示唆します。

Stataはヘルプを検索する方法を複数準備しています。これらの方法を確認するには **ヘルプ** > **アドバイス.** と操作してください。あるいは、シンプルに*F1*を押してください。

describeはStataのコマンドで、メモリにロードしているデータの内容を表示します。詳細は[D] **describe**で紹介しています。通常、「describe」と入力してEnterキーを押します。あるいは*F2*キーを押してください。

saveはメモリにあるデータをファイルに保存するコマンドでuseはデータをロードするコマンドです。詳細は、[D] **use**と[D] **save**をご覧ください。それぞれの構文は同じです。「save」または「use」コマンドにファイル名が続きます。saveとuseコマンドは入力、あるいは*F7*または*F8*を押した後にファイル名を入力する方法、どちらでも使用できます。

これらのファンクションキーの定義は変更できます。例えば、データを一覧表示するコマンドはlistです（このコマンドについては[D] **list**を参照してください）。構文は、listだけを入力するとメモリ内にある全てのデータをリストします。あるいは、listのコマンドの後に変数を入力すると、その変数をリストします(他にも操作の選択肢はあります)。

*F9*にlistを定義したいときは、次のように入力しましょう。

```
. global F9 "list "
```

上の例では、F9は*F*の文字と*9*を入力することをしており、*F9*キーを押すということではありません。コマンドの大文字とスペースの使用について注意してください。

「global」を小文字で入力し、「F9」と入力してから「"list "」と入力します。listの後にあるスペースが重要です。これから先、「list mpg weight」と入力する代わりに*F9*キーを押してから「mpg weight」と入力するとリスト表示できるようになります。*F9*の定義にスペースを入力するのは、*F9*キーを押した後にスペースを押すことなく、変数の名前を入力できるようにするためです。

では、*F5*は全てのデータをリストするように定義しましょう。つまり、listの後にEnterです。その場合、次のように定義します。

```
. global F5 "list;"
```

ここまで設定できると、データを全てリストする方法が2つ設定できたことになります。*F9*を押してから*Enter*キーを押すか、*F5*を押す場合です。*F5*キーを定義した時に入力した、最後のセミコロンは*Enter*を自動的に押すことを意味しています。

*F9*と*F5*の定義を本当に変更するには、そのボタンが意味することをStataが起動するときに毎回定義する必要があります。1つの方法は、Stataが起動するときにglobalコマンドを毎回入力することです。もうひとつの方法はprofile.doというテキストファイルに、このコマンドを入力しておくことです。Stataはprofile.doに入力したコマンドは、profile.doが適切なディレクトリに保存されている限り、Stataが起動するたびに実行します。

Windows: [GSW] B.3 Executing commands every time Stata is started
Mac: [GSM] B.1 Executing commands every time Stata is started
Unix: [GSU] B.1 Executing commands every time Stata is started

*F*キーは自由に設定できます。これらには文字列を記憶させることができるので、
*F*キーを押すとそれらの文字を入力したことと同じになります。

❑ **テクニカルノート**

[*Unix(console)版Stataユーザ*向け]時々、Unixは*F*キーに特別な意味を追加することがあります。その場合、Stataが設定している定義よりも優先されます。Stataでは*F*キーを使用する別の手段を備えています。[*Ctrl+F*]を押し、一度キーを離してから、0から9の数字を入力します。Stataは[*Ctrl+F*]プラス1は*F1*キーと同じであると認識、[*Ctrl+F*]プラス2 は*F2*と認識する、といった要領です。[*Ctrl+F*]プラス 0は*F10*キーを意味します。これらのキーはtermcapまたはterminfoのエントリが正しくマッピングされている時のみ使用できます。

❑

❑ **テクニカルノート**

他の国で使用されているキーボードでは、左向きのシングルクオテーションマーク(‘)はアクセント記号として使用されます。この場合、この記号をファンクションキーにマッピングしておくことをお勧めします。実際には、左向きのシングルクオテーションマーク(‘)と右向きのシングルクオテーションマーク(’)をそれぞれ隣り合ってマッピングを行ったほうが便利でしょう。

Stata内でdoファイルエディタを開き、次の2行をdoファイルエディタに入力します。

```
global F4   ‘
global F5   ’
```

ファイルをprofile.doとしてStataのディレクトリに保存します。すでにprofile.doファイルが設定されている場合は、先ほどの2行を既存のprofile.doに追加してください。

Stataを閉じて、再度起動してください。すると、開始メッセージ画面を表示します。

```
running C:¥Program Files¥Stata18¥profile.do...
```

あるいは、Stataがどこにインストールされているのかによりますが、そのパスが表示されます。*F4*と*F5*を押して、目的としていたシングルクオテーションが表示されることを確認します。

開始メッセージが表示されなかった場合、ホームフォルダ内にprofile.doが保存できていない可能性があります。もちろん、他のファンクションキーにマッピングを行ってもかまいませんが、*F1*, *F2*, *F7*, *F8*は既に使用されています。

❑

10.3 Stataでキーを編集する

利用しているOSによりますが、一般的に提供されている編集キーは使用できます。つまり、Stataでは一般的な操作方法で入力内容を編集できます。Stataのコマンドウィンドウは標準的な編集ウィンドウです。

また、履歴ウィンドウのコマンドをコマンドウィンドウに入力することも可能です。履歴ウィンドウでコマンドをクリックするとコマンドウィンドウにロードするので、内容を編集できます。あるいは、履歴ウィンドウで行をダブルクリックすると、そのコマンドはロードされ、かつ実行されます。

履歴ウィンドウの情報をコマンドウィンドウで使用するには*PgUp*と*PgDn*キーを使用する方法もあります。*PgUp*を押すとコマンドウィンドウに直近で入力したコマンドをロードします。もう一度押すと、さらに上の行のコマンドをロードします。*PgDn*は履歴を反対の方向に移動します。

もうひとつ、ユーザにとって便利な編集キーは *Esc*です。このキーはコマンドウィンドウをクリアします。まとめると、次の通りです。

押すキー	結果
PgUp	履歴ウィンドウのコマンドをひとつずつ前に移動してコマンドウィンドウに表示
PgDn	履歴ウィンドウのコマンドをひとつずつ後ろに移動してコマンドウィンドウに表示
Esc	コマンドウィンドウをクリア

10.4 Unix(console)版Stataでキーを編集する

特定のキーを押すと、現在入力している行を編集できます。Stataはさまざまなコンピュータとキーボードをサポートしているので、全てのStataユーザ間で編集キーの位置と名前は必ずしも同じではありません。

それでも、全てのキーボードで標準的なアルファベットキー(*QWERTY*キーボードなど)と、*Ctrl*キーは存在します。キーボードの中には*PgUp*や*PgDn*のような追加のキーが右や上、左にある場合もあります。

このマニュアル内で、Stataの編集キーはどのユーザのキーボード上にも表示されない名前で表記します。例えば、PrevLineはStataの編集キーのひとつで、直前の行を再取得するものになります。探していただいて結構ですが、キーボード上では見つけられないでしょう。では、PrevLineはどこにあるのでしょうか。可能な限り、ユーザが容易に想像できる場所にキーを配置しました。*PgUp*キーがあるキーボードでは、*PgUp*がPrevLineを表します。もう1つ、[*Ctrl+R*]はどのユーザのキーボード（どのバージョンのUnix、キーボードのブランド）でもPrevLineを意味します。

PrevLineをしてください、と指示をするときは、*PgUp*または*Ctrl+R*を押してください。編集キーは次の通りです。

編集キーの名前	編集キー	機能
Kill	PCでは*Esc*、*Ctrl+U*	行を削除し、最初から再開する
Dbs	PCでは*Backspace*、他のコンピュータではBackspace または*Delete*	1つ戻り、該当の文字を消す
Lft	←、数字のテンキーでは*4*、*Ctrl+H*	カーソルを1文字分左に動かす(ただし、文字は消さない)
Rgt	→、数字のテンキーでは*6*、*Ctrl+L*	カーソルを1文字分右に動かす
Up	↑、数字のテンキーでは*8*、*Ctrl+O*	2行以上の行を使用しているコマンドで、1行分カーソルを上に移動する。あわせてNextLineをご確認ください。
Dn	↓、数字のテンキーでは*2*、*Ctrl+N*	2行以上の行を使用しているコマンドで、1行分カーソルを下に移動する。あわせてNextLineをご確認ください。
PrevLine	*PgUp*、*Ctrl+R*	直前に入力した行を再表示するPrevLineを複数回押すと、押した分だけコマンドを遡ります。
NextLine	*PgDn*、*Ctrl+B*	PrevLineの反対
Seek	PCでは*Ctrl+Home*、*Ctrl+W*	指定した行番号に移動する。Seekを押す前に、行番号を押してください。例えば、*3*と入力してSeekを押した場合、PrevLineを3回押したときと同じになります。
Ins	*Ins*、*Ctrl+E*	挿入モードに切り替える。挿入モードでは、入力した文字はカーソルの位置に挿入されます。
Del	*Del*、*Ctrl+D*	カーソルの位置にある文字を削除する
Home	*Home*、*Ctrl+K*	カーソルを行の開始位置に戻す
End	*End*、*Ctrl+P*	カーソルを行の最終位置に移動する
Hack	PCでは*Ctrl+End*、*Ctrl+X*	カーソル位置で行を切り取る
Tab	PCでは→|、*Tab*、*Ctrl+I*	変数名を拡充する
Btab	PCでは|←、*Ctrl+G*	Tabの反対

例題1

これらの編集キーの使用法を文書で説明するのは困難です。実際に試すのがいいでしょう。
ですが、何も例題が無いのも分かりにくいので、ひとつ例題を準備しました。

```
. summarize price waht
```

「summarize price waht」と入力し、Lftキー（←キーまたは*Ctrl+H*）を3回押してカーソルをwahtのaまで戻します。このまま*Enter*キーを押すと、Stataは「summarize price waht」を認識するので、*Enter*キーを押す時のカーソル位置はどこでも構わないことを物語っています。もし「summarize price」を実行したいなら、もう1文字分左側にカーソルを移動してHackキーを押します。しかし今回の「waht」は「weight」の入力ミスと仮定します。

キーボードで*e*を押せば、aを置き換えてeが表示されます。そしてカーソルはhの元に移動します。これで、「weht」が画面に表示されます。Insを押してStataを挿入モードにします。この状態で*i*と*g*を押しましょう。そうすると、入力行は「summarize price weight」となります。

このコマンドが入力したいコマンドなので、*Enter*キーを押します。挿入したい文字を入力する前に*Ins*キーを毎回押す必要はありません。*Ins*キーは切り替えを行います。もう一度キーを押すと、Stataは挿入モードを終了するので、カーソル位置の文字を置き換えるようになります。*Enter*キーを押すと、Stataは挿入モードをリセットするので、各コマンドを実行するごとに挿入モードであるかどうかを覚えておく必要はありません。

◁

テクニカルノート

Stataは編集を行う際に、ターミナルレコード内の情報を参照します。/etc/termcap(5)の下か、System Vでは、/usr/lib/terminfo(4)にあります。操作中の機能がうまく動作していないと感じたら、ターミナルに関する情報のうち、termcapファイルかterminfディレクトリのものが正しくない可能性があります。システム管理者に確認を取ってください。

❑

10.5 Stataで以前の行を編集する

このセクションで説明する内容に加えて、履歴ウィンドウではレビューバッファーの内容を確認できます。

これらの行を再度取得する1つの方法はPrevLineとNextLineキーです。PrevLineとNextLineは操作時のキーにつけた名前であることを覚えておいてください。キーボード上にこのような名前のキーはありません。直前のセクションを確認して、コンピュータ上のどのキーがPrevLineとNextLineに当てはまるのか、確認してください。今回、時間を短縮するために、ヒントを記載します。PrevLineはおそらく*PgUp*キーに対応し、NextLineはおそらく*PgDn*キーに対応します。

例えば、3行前のコマンドを再度実行するとします。その場合、PrevLineを3回押してから*Enter*を押しましょう。もし、間違えてPrevLineを4回押してしまったら、NextLineを押して、バッファー内の履歴の新しい方に移動します。何行前あるいは後ろに移動するのか、数える必要はありません。これは、PrevLineまたはNextLineを選択すると、その内容がモニターに表示されるからです。探している行が見つかるまで、キーを押してください。

また、Unix(console)ユーザには、#reviewという前の行を参照する方法も便利です。

例題2

「#review」のみを入力すると、直近で実行した5つのコマンドが表示されます。例えば、以下のようになります。

```
. #review
5 list make mpg weight if abs(res)>6
4 list make mpg weight if abs(res)>5
3 tabulate foreign if abs(res)>5
2 regress mpg  weight weight2
1 test weight2=0
.  _
```

表示されたリストから、最後にユーザが入力したコマンドは「test weight2=0」でした。あるいは、履歴ウィンドウを確認して、入力したコマンドの履歴を確認してください。

◁

例題3

目的のコマンドが、直近の5つのコマンドの中に無い事も想定されます。ユーザはStataに必要な行数分だけさかのぼるように指示できます。例えば、「#review 15」と入力すると、直近15行分のコマンドを表示します。

```
. #review 15
15 replace resmpg=mpg-pred
14 summarize resmpg, detail
13 drop predmpg
12 describe
11 sort foreign
10 by foreign: summarize mpg weight
9 * lines that start with a * are comments.
8 * they go into the review buffer too.
7 summarize resmpg, detail
6 list make mpg weight
5 list make mpg weight if abs(res)>6
4 list make mpg weight if abs(res)>5
3 tabulate foreign if abs(res)>5
2 regress mpg  weight weight2
1 test weight2=0
.  _
```

もし、10行前の行を再実行するなら、10と入力してからSeekを押すか、PrevLineを10回押してください。前のコマンドを呼び出すのにどちらの方法を使用しても、編集キーを使用してコマンドの内容を編集できます。

10.6 変数名をTabで自動入力する

素早く変数名を入力するもうひとつの方法は、Stataの変数名完成機能を使用する事です。変数名の初めの数文字をコマンドウィンドウに入力して*Tab*キーを押すだけで利用できます。これで、Stataは変数名の残りの部分を自動的に入力します。もし、2つ以上の変数が入力した文字に当てはまる場合、Stataは可能な限り完成させ、入力した文字列が変数固有の省略形ではないことをビープ音で知らせます。

Stataの変数名完成機能はファイル名にも対応しています。まず″（ダブルクォーテーション）を入力し、その後ファイル名またはディレクトリ名の初めの数文字をコマンドウィンドウに入力して*Tab*キーを押すだけで利用できます。これで、Stataは残りの部分を自動的に入力します。もし、2つ以上の候補が入力した文字に当てはまる場合、Stataは可能な限り完成させ、入力した文字列が変数固有の省略形ではないことをビープ音で知らせます。ファイル名またはディレクトリ名を入力後、″（ダブルクォーテーション）を入力します。

Stataの個別機能

11 言語構文

目次

11.1 概要

いくつかの例外を除き、基本的なStataの言語構文は次の通りです。

[by varlist:] command [varlist] [=exp] [if exp] [in range] [weight] [, options]

必須項目はそのまま記述し、任意で追加する限定子やオプション項目は大括弧[ ]の中に入力します。この構文の中で、varlist は変数の名前、commandはStataのコマンド、expは算術演算、rangeはデータ範囲、weightは加重方法、optionsはオプションのリストを意味します。

11.1.1 varlist

varlistを使用するほとんどのコマンドは、特に変数指定が無くても使用できます。varlistの入力がない場合、コマンドは_all(全てのvarlist)でデータセット内にある全ての変数であると認識します。データを変更したり破壊するコマンドでは、Stataは varlistを明確に書き出すように求めてきます。詳しくは[U] 11.4 varnameとvarlistsをご覧ください。

複数のコマンドではvarlistではなく、varnameを設定します。varnameは変数を1つだけ示す際に使います。例えば、tabulateコマンドはvarnameを設定します。詳細は[R] tabulate onewayをご覧ください。

例題1

summarizeコマンドは指定した変数の平均、標準偏差、範囲などをリスト表示します。[R] summarizeで説明していますが、構文を表す記述は次のようになります。

summarize [varlist] [if] [in] [weight] [, options]

このマニュアルの中には、オプションについてまとめている表もありますが、まずは構文の内容に集中しましょう。summarize以外の言葉は全て大括弧に囲まれているので任意であると分かります。よって、このコマンドの最もシンプルな形は「summarize」になります。引数を使わないで「summarize」のみを入力すると、「summarize _all」と入力する事と同じです。これはデータセット内の変数全てについて要約統計量を表示します。下線部分は使用可能な最短省略形です。つまり、単に「su」と入力することもできます。詳細は[U] 11.2 省略のルールをご確認ください。

optionsを示す表は次の通りです。

options	説明
メイン	
detail	追加の統計量を表示する
meanonly	表示を抑制する、平均のみを計算、プログラマー向けのオプション
format	変数の表示形式の使用
separator(#)	#個分の変数ごとにセパレータを挿入、デフォルトの値はseparator(5)

つまり、「summarize, detail」や「summarize, detail format」と入力することも可能だと分かります。

もうひとつの例題として、dropコマンドがあります。これは変数あるいは観測値をデータセットから削除します。変数を削除する場合は、次のような構文になります。

drop varlist

dropコマンドにはオプションが無いため、オプション表はありません。

実際に、何もオプションとして存在できません。dropだけを入力すると、エラーメッセージとして“varlist or in range required”が表示されます。データセット内の全ての変数を削除するには、「drop _all」と入力する必要があります。

構文形式を見なくても、varlistが必要であることを想像できるでしょう。dropコマンドはデータを変更(破壊)するので、Stataはコマンド全てを入力するように求めてきます。構文の項目によると、varlistは大括弧[ ]で囲まれていないので必須項目であることが分かります。また、dropには下線部分が無いので、省略することはできません。

11.1.2 by varlist:

by varlist:のbyで指定しているvarlist値が等しいサブグループに、同じコマンドをそれぞれ繰り返します。by varlist:のプリフィックスをつけると、varlist値でグループ分けしたそのグループごとにデータセットを作成、保存してから各データセットにコマンドを実行した結果と同じになります。by varlistを使用するには、データをあらかじめソートしておく必要があります。最も、sortオプションがあるのでそれを使うことも可能です。詳細は[U] 11.5 by varlist: 構文をご覧ください。

例題2

「summarize marriage_rate divorce_rate」と入力すると、変数marriage_rateとdivorce_rateの平均、標準偏差、範囲を、データ内全ての観測値を元に計算します。

```
. use https://www.stata-press.com/data/r18/census12
(1980 Census data by state)
. summarize marriage_rate divorce_rate
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
marriage_r~e |         50    .0133221    .0188122   .0074654   .1428282
divorce_rate |         50    .0056641    .0022473   .0029436   .0172918
```

次に、「by region: summarize marriage_rate divorce_rate」と入力してみます。すると、地域ごとに1 つずつ表を作成します。

```
. sort region
. by region: summarize marriage_rate divorce_rate
-------------------------------------------------------------------------------
-> region = N Cntrl
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
marriage_r~e |         12    .0099121    .0011326   .0087363   .0127394
divorce_rate |         12    .0046974    .0011315   .0032817   .0072868
-------------------------------------------------------------------------------
-> region = NE
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
marriage_r~e |          9    .0087811     .001191   .0075757   .0107055
divorce_rate |          9     .004207    .0010264   .0029436   .0057071
-------------------------------------------------------------------------------
-> region = South
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
marriage_r~e |         16    .0114654    .0025721   .0074654   .0172704
divorce_rate |         16     .005633    .0013355   .0038917   .0080078
-------------------------------------------------------------------------------
-> region = West
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
marriage_r~e |         13    .0218987    .0363775   .0087365   .1428282
divorce_rate |         13    .0076037    .0031486   .0046004   .0172918
```

データセットは「by variables:」でソートする必要があります。

```
. use https://www.stata-press.com/data/r18/census12
(1980 Census data by state)
. by region: summarize marriage_rate divorce_rate
not sorted
r(5);
. sort region
. by region: summarize marriage_rate divorce_rate (output appears)
```

あるいは、byプリフィックスとsortオプションを使用する事もできます。

```
. by region, sort:summarize marriage_rate divorce_rate (output appears)
```

by varlist:はほとんどのStataのコマンドで使用できます。どのコマンドで使用できるか確認するには、構文構成を確認してください。例えば、regionごとに、marriage_rateとdivorce_rateの間で相関を見るには、「by region: correlate marriage_rate divorce_rate」と入力しましょう。

◁

❑ テクニカルノート

by varlist:内で入力するvarlistは、Stata/MPでは最大で120,000変数、Stata/SEでは最大で32,767変数、Stata/BEでは最大で2,048変数を設定できます。これはデータセット内で設定できる最大値です。例えば、自動車に関するデータが手元にあり、そのデータから市場の種類（market）と製造元（origin）による平均を算出したいとします。その場合、「by market origin: summarize」と入力します。この例では、varlistは2つの変数、marketと originを指定しています。データがあらかじめmarketとoriginでソートされていない場合、最初に「sort market origin」と入力してソートします。

❑

❑ テクニカルノート

by *varlist*:で設定するvarlistには、文字列変数、数値変数あるいは両方を使うことができます。上記例の一つでは、regionは文字列変数です。形式はstr7です。もしこの例題で変数regionが数値（例えば1, 2, 3, 4や12.2, 16.78, 32.417, 152.13）であっても、コマンドは問題なく実行できます。

❑

11.1.3 if exp（if表現）

if expは設定している条件が真（*true*）である場合のみ、この範囲にコマンドを実行します（つまり、数式がゼロではない、ということになります。詳細は[U] 13 関数と計算式の入力をご覧ください。）

▷ **例題3**

「summarize marriage_rate divorce_rate if region=="West"」と入力すると、西部地域の情報を表示します。

```
. summarize marriage_rate divorce_rate if region == "West"
```

Variable	Obs	Mean	Std. dev.	Min	Max
marriage_r~e	13	.0218987	.0363775	.0087365	.1428282
divorce_rate	13	.0076037	.0031486	.0046004	.0172918

「region=="West"」で入力している2つの等号はエラーではありません。Stataは2つの等号を使用して同等性を検定し、1つの等号では置換を意味します。詳細は[U] 13 関数と計算式の入力をご覧ください。

1つのコマンドにつき、最大で1つのifを使用できます。西部の観測値が0.015よりも多いものを抽出して要約統計量を算出する場合、「summarize marriage_rate divorce_rate if region=="West" if marriage rate>.015」とは入力しないでください。代わりに、次のように入力します。

```
. summarize marriage_rate divorce_rate if region == "West" & marriage_rate > .015 Variable      Obs       Mean
```

	Std. dev.	Min	Max		
marriage_r~e	1	.1428282	.	.1428282	.1428282
divorce_rate	1	.0172918	.	.0172918	.0172918

条件同士をつなげる場合、“&”の変わりに*and*は使用できません。1つの条件あるいはもう1つの条件に当てはまる観測値を抽出する際は“|”記号を使用してください。例えば、「summarize marriage_rate divorce_rate if region==="West" | marriage rate>.015」と入力すると、regionがWestであるか、marriage_rateが0.015よりも大きい観測値が抽出されます。

例題4

if表現はbyプリフィックスと組み合わせて使うことができます。「by region: summarize marriage_rate divorce_rate if marriage_rate>.015」と入力すると、それぞれの地域ごとの表(地域につき1 つ)を作成します。

```
. by region: summarize marriage_rate divorce_rate if marriage_rate > .015
```

-> region = N Cntrl

Variable	Obs	Mean	Std. dev.	Min	Max
marriage_r~e	0				
divorce_rate	0				

-> region = NE

Variable	Obs	Mean	Std. dev.	Min	Max
marriage_r~e	0				
divorce_rate	0				

-> region = South

Variable	Obs	Mean	Std. dev.	Min	Max
marriage_r~e	2	.0163219	.0013414	.0153734	.0172704
divorce_rate	2	.0061813	.0025831	.0043548	.0080078

-> region = West

Variable	Obs	Mean	Std. dev.	Min	Max
marriage_r~e	1	.1428282	.	.1428282	.1428282
divorce_rate	1	.0172918	.	.0172918	.0172918

出力結果を確認すると、北東部と北中央部ではmarriage_rateが0.015を超える州は無かったのに対し、南部と西部ではそれぞれ2つと1つの州で見つかっています。

◁

11.1.4 in range（in範囲）

in rangeは特定の観測値の範囲にのみコマンドを実行します。指定するには、#$_1$[/#$_2$]の形式をとります。ここで#$_1$と#$_2$は正または負の整数を示します。負の整数は「データの最後から数える」という意味になります。つまり、-1は最後の観測値を示します。最初の観測値(#1)として入力する値は最後の観測値(#2)として入力する値と等しいあるいは小さい必要があります。

データセット内の最初と最後の観測値はそれぞれfとl（小文字）でも表現できます。Fはfと同意として認識し、Lはlと同意として認識します。範囲はデータセット内の絶対な観測番号で指定します。結果としてin限定子はby varlist:プリフィックスとの併用はできません。 詳しくは[U] 11.5 by varlist:構文をご覧ください。

▷ **例題5**

「summarize marriage_rate divorce_rate in 5/25」と入力すると、5-25までの観測値のmarriage_rate と divorce_rateの要約統計量を計算します。

```
. summarize marriage_rate divorce_rate in 5/25
```

Variable	Obs	Mean	Std. dev.	Min	Max
marriage_r~e	21	.009814	.0019835	.0074654	.0153734
divorce_rate	21	.0048067	.0013655	.0029436	.0080078

確かに、これは少しおかしいでしょう。しかし、summarizeコマンドではなく、listコマンドを使用すると、この利用方法は分かりやすくなります。例えば、marriage_rateがワースト10に入る州の名前を確認したい場合、「sort marriage_rate」とソートしてから「list marriage_rate in 1/10」を実行すれば、確認できます。

「summarize marriage_rate divorce_rate in f/l」と入力するのは「summarize marriage_rate divorce_rate」と入力するのと同じで、全ての観測値について要約統計量を算出します。

◁

▷ **例題6**

「summarize marriage_rate divorce_rate in 5/25 if region == "South"」と入力すると、観測値が5-25の間にあり、なおかつregionが南部（South）である値を元に、要約統計量を計算します。

```
. summarize marriage_rate divorce_rate in 5/25 if region == "South"
```

Variable	Obs	Mean	Std. dev.	Min	Max
marriage_r~e	4	.0116711	.0033151	.0074654	.0153734
divorce_rate	4	.006049	.0017672	.0038917	.0080078

inとifの順番は特に関係ありません。つまり、コマンドは「summarize marriage_rate divorce_rate if region == "South" in 5/25」でも同じ結果を表示します。

◁

例題7

inで指定する範囲内で負の数を使用する場合、sortと組み合わせると便利です。例えば、自動車に関するデータがあり、最も大きな燃費率を有している最初の5つをリストする際には、次のように入力します。

```
. use https://www.stata-press.com/data/r18/auto
(1978 automobile data)
. sort mpg
. list make mpg in -5/l
```

	make	mpg
70.	Toyota Corolla	31
71.	Plym.Champ	34
72.	Datsun 210	35
73.	Subaru	35
74.	VW Diesel	41

11.1.5 =exp

=expは変数に指定する値を設定するときに使用します。最も多く使用するのは、generateとreplaceコマンドとの組み合わせです。数式に関する詳細は[U] **13 関数と計算式の入力**をご覧ください。また、generateとreplaceコマンドに関しては[D] **generate**をご覧ください。

例題8

表現	意味
generate newvar=oldvar+2	newvarという新しい変数を作成し、その値はoldvar+2の値と等しくなります
replace oldvar=oldvar+2	既存の変数の内容を変更します
egen newvar=rank(oldvar)	変数newvarを作成し、既存の変数oldvarのランクを含みます(詳細は[D] **egen**をご覧ください)

11.1.6 加重

加重（weight）は各観測値に付加される重みを示します。weightの構文は次のとおりです。

[weightword=exp]

ここでは、大括弧[]を実際に入力し、weightword は次の言葉のうちの一つを入力します。

weightword	意味
weight	加重のデフォルト形式 fweightまたはfreque
ncy	度数加重
pweight	サンプリング加重
aweightまたはcellsize	解析的加重
iweight	重点加重

下線部分は利用可能な最短省略形を示します。つまり、weightコマンドはwやweなどと省略できます。

例題9

それぞれの加重に関する違いを説明する前に、人口(母集団)で重みをつけた変数median_ageをアメリカにある全50の州のデータから計算します。データセットには変数popもあり、これはそれぞれの州の総人口を示します。

```
. use https://www.stata-press.com/data/r18/census12
(1980 Census data by state)
. summarize median_age [weight=pop] (analytic weights assumed)
```

Variable	Obs	Weight	Mean	Std.dev.	Min	Max
median age	50	225907472	30.11047	1.66933	24.2	34.7

ユーザに50の観測値があることを示す以外に、Stataは加重合計が225,907,472になることを伝えきます。これは1980年の国勢調査でアメリカに住んでいる総人口と同じ数です。加重の平均は30.11です。Stataは“解析的な”加重が必要であると推測したことが分かります。

weightは各コマンドの中で最も「自然な」加重を使用し、weight,pweight,aweight, iweightのどれかを実行します。漠然と weightを指定すると、コマンドがどの加重の形式を推測したのか、先ほどの例のように表示します。全てのコマンドが全種類の加重法を使用できるとは限りません。各コマンドの構文項目の下にあるメモでは、使用できる加重法の種類を示します。

Stataでは4種類の加重法を使用できます。

1. fweight、または度数加重は複製のある観測値を示します。fweightは常に整数です。例えば、fweightが観測値と関連しており5である場合、まったく同じ観測値が5回見つかったことを示します。
2. pweight、またはサンプリング加重は、あるデータが特定のサンプリング方法によってこのサンプルの中に含まれた確率の逆数を示します。例えば、pweightの100があると、これは元となる母集団の中にある100個分のデータを表していることになります。この加重スケール（尺度）は推定されるパラメータや標準誤差では問題になりません。ただし、合計を推定していたり、有限な母集団をsvyコマンドから計算している時はその限りではありません。詳細は[SVY] Surveyをご覧ください。
3. aweight、または解析的加重はある観測値の分散に反比例します。つまり、j番目の観測値の分散は $\sigma_2=w_j$ であると推定され、w_jは加重を示します。一般的に、観測値は平均を示し、加重はその平均を計算する際に使用した要素の数を示します。ほとんどのStataのコマンドでは、aweightで記録しているスケール（尺度）は無関係です。Stataは、使用する前にN（データの数）の合計として再スケールを実行します。
4. iweight、または重点加重はその観測値の相対的な重要性を示します。統計的な定義は特にありません。これは全てを含むことができるカテゴリです。iweightをサポートするコマンドは、そのコマンドごとに対応方法を指定します。通常、このオプションは特定の計算を行わせたいプログラマーなどに使用してもらうために準備しています。

加重とそれぞれの意味に関する詳細は[U] 20.24 加重推定をご覧ください。

テクニカルノート

加重を指定しない場合、結果は[fweight=1]と指定した時と等しくなります。

11.1.7 オプション

多くのコマンドでは特定のオプションを使用できます。オプションはそれぞれのコマンドと共にリファレンスマニュアルにて説明しています。オプションを使用するには、コマンドの後にカンマを入力してから目的の項目を入力してください。

例題10

「summarize marriage_rate」と入力すると、変数marriage_rateの平均、標準偏差、最小値、最大値を表示します。

```
. summarize marriage_rate
```

Variable	Obs	Mean	Std. dev.	Min	Max
marriage_r~e	50	.0133221	.0188122	.0074654	.1428282

summarizeの構文は次のようになります。

summarize [varlist] [*if*] [in] [weight] [, options]

オプション表は次のようになります。

options	説明
メイン	
detail	追加の統計量を表示する
meanonly	表示を抑制する、平均のみを計算、プログラマー向けのオプション
format	変数の表示形式の使用
separator(*#*)	*#*個分の変数ごとにセパレータを挿入、デフォルトの値はseparator(5)

つまり、summarizeコマンドと共に使用できるオプションは、detail,meanonly,format, separator()となります。これらのオプションで使用可能な最短省略形は、detailではd、meanonlyではmean、formatではf、 separator()ではsep()になります。詳細は[U] 11.2 省略のルールをご覧ください。

「summarize marriage_rate, detail」と入力すると、表は選択した分位(パーセンタイル)、4つの最小と最大値、歪度、尖度を表示します。

```
. summarize marriage_rate, detail
```

marriage_rate

Percentiles		Smallest		
1%	.0074654	.0074654		
5%	.0078956	.0075757		
10%	.0080043	.0078956	Obs	50
25%	.0089399	.0079079	Sum of wgt.	50
50%	.0105669		Mean	.0133221
		Largest	Std. dev.	.0188122
75%	.0122899	.0146266		
90%	.0137832	.0153734	Variance	.0003539
95%	.0153734	.0172704	Skewness	6.718494
99%	.1428282	.1428282	Kurtosis	46.77306

コマンドの中には、必須オプションもあります。例えば、ranksumコマンドにはグループ分けをする変数を指定するため、by(groupvar)オプションが必須です。groupvarはvarnameの特殊な名前です。これは、各観測値がどのグループに属するのか見分ける際に使用します。

❑ テクニカルノート

コマンドvarlistを入力した場合、オプションはコマンドのどこに入力しても大丈夫です。例えば、「summarize marriage_rate divorce_rate if region=="West", detail」と入力しても、「summarize marriage_rate divorce_rate, detail, if region=="West"」と入力しても、Stataは同じ結果を表示します。2つ目のカンマはオプションのリストが一度終わり、コマンド行に戻ることを示しています。detailの後にカンマを入力し忘れると、Stataはエラーを返します。これは「if region=="West"」をコマンドの一部としてではなく、オプションとして読み取ろうとして失敗するためです。

ただし、varlistの途中でオプションを入力してはいけません。ただし、varlistの途中でオプションを入力してはいけません。「summarize marriage_rate, detail, divorce_rate」と入力するとエラーが返されます。

オプションは続けて指定する必要はありません。オプションは続けて指定する必要はありません。つまり、「summarize marriage_rate divorce_rate, detail, if region=="South", noformat」と入力できます。detailとnoformatのどちらもオプションとして正しく認識されます。

❑

❑ テクニカルノート

ほとんどのオプションはオンオフの切り替えを行います。つまり、そのオプションを実行するかしないかのどちらかになります。どちらがデフォルトなのか、覚えておくのは難しいこともあるので、次のルールをよく確認してください。これは全てのオプションに適用されます。オプションが任意のオプションである場合、noオプションも任意で設定するオプションです。つまり、もしsummarizeのデフォルトがdetailかnodetailだったか分からない状態で詳細は必要としない場合、「summarize, nodetail」と入力すると目的の状態で結果を出力できます。この場合、nodetail1オプションを入力する必要は特にありません。しかしStataは特にエラーは表示しません。

いくつかのオプションでは*引数*が使用できます。例えばStataにはkdensityコマンドがあり、n(#)オプションを使用すると評価する密度の推定を算出する際に元にするポイント数を設定します。オプションが引数を設定する場合、その引数は小括弧に囲まれます。

中には、1つ以上の引数を使用するオプションもあります。その場合、それぞれの引数はカンマで区別してください。例えば、次のような構文構造を見かけることがあります。

```
saving(filename[, replace])
```

ここではreplaceが任意の2つ目の引数です。リスト、例えば変数リスト（varlist）や数字リスト（numlist）は1つの引数として認識されます。もし構文が次のようになっている場合、

```
powers(numlist)
```

数字のリストは1つの引数になるので、中に入力する要素はカンマで区切る必要はありません。例えば、powers(1 2 3 4)のように入力します。実際には、このコマンドにカンマを使用したとしても問題なく実行します。つまり、powers(1,2,3,4)と入力しても問題ありません。

いくつかのオプションでは文字列引数を使用できます。regressコマンドにはeform()オプションがあり、これは文字列の引数を使用します。たとえば、eform("Exp Beta")のようになります。安全を期すため、文字列はダブルクオテーションで囲むことをお勧めしますが、必須ではありません。ダブルクオテーションを使用しない場合、2つ以上のスペースは1つのスペースとして認識されます。つまり、eform(Exp beta)はeform(Exp beta)と同じであると認識されます。

❑

11.1.8 numlist(数値リスト)

numlistとは数字のリストです。Stataは範囲を指定するにあたり、特定の記載方法で短く設定できます。

Numlist	意味
2	1つの数字
1 2 3	3つの連続した数字
3 2 1	3つの連続した数字の逆順
5 1 1.5	3つの異なる数字
1 3 -2.17 5.12	4つの順番が入れ替わった数字
1/3	3つの連続した数字：1 2 3
3/1	同じ3つの連続した数字の逆順
5/8	4つの連続した数字：5, 6, 7, 8
-8/-5	4つの連続した数字：−8, −7, −6, −5
-5/-8	4つの連続した数字：−5, −6, −7, −8
-1/2	4つの連続した数字：−1, 0, 1, 2
1 2 to 4	4つの連続した数字：1, 2, 3, 4
4 3 to 1	4つの連続した数字：4, 3, 2, 1
10 15 to 30	5つの等間隔の数字：10, 15, 20, 25, 30
1 2:4	上の1 2 to 4と同じ
4 3:1	上の4 3 to 1と同じ
10 15:30	上の10 15 to 30と同じ
1(1)3	3つの連続した数字：1, 2, 3
1(2)9	5つの等間隔の数字：1, 3, 5, 7, 9
1(2)10	同じく5つの等間隔の数字：1, 3, 5, 7, 9
9(-2)1	5つの等間隔の数字：9, 7, 5, 3, 1
-1(.5)2.5	等間隔の数字：−1, −0.5, 0, 0.5, 1, 1.5, 2, 2.5
1[1]3	1(1)3と同じ
1[2]9	1(2)9と同じ
1[2]10	1(2)10と同じ
9[-2]1	9(2)1と同じ
-1[.5]2.5	−1(.5)2.5と同じ
1 2 3/5 8(2)12	8つの数字：1, 2, 3, 4, 5, 8, 10, 12
1,2,3/5,8(2)12	同じ8つの数字
1 2 3/5 8 10 to 12	同じ8つの数字
1,2,3/5,8,10 to 12	同じ8つの数字
1 2 3/5 8 10:12	同じ8つの数字

poissonコマンドのconstraints()オプションは、constraints(numlist)という構文を持っています。つまり、constraints(2 4 to 8)やconstraints(2(2)8)等と入力します。

11.1.9 datelist(日付リスト)

datelistは日付や時間のリストで、よくグラフを作成する時に日付や時間の設定がある変数を使う場合に使用します。日付と時間がどのようにStataに保存や編集されるか確認するには[U] 25 日付と時間で作業をするを確認してください。カレンダーの日付、つまり%td日付は、Stataの中で1960年1月1日からの日数で記録しています。つまり、0を入力すると1960年1月1日、1を入力すると1960年1月2日、16,541を入力すると2005年4月15日を意味します。同じように、-1は1959年12月31日を、-2は1959年12月30日を、−16,541は1914年9月18日を示します。この場合、datelistは次のような日付のリストを示します。

```
15apr1973 17apr1973 20apr1973 23apr1973
```

あるいは、最初と最後の日付と増分を示します。

```
17apr1973(3)23apr1973
```

または、その両方の組み合わせで設定します。

```
15apr1973 17apr1973(3)23apr1973
```

スペース、スラッシュ、カンマなどを使用して指定する日付は括弧で囲まれている必要があります。

```
(15 apr 1973) (april 17, 1973)(3)(april 23, 1973)
```

等間隔に開いているカレンダーの日付はあまり使い勝手が良くありません。しかし、他の時間の単位では等間隔の期間ごとにデータがあるのはとても有用です。例えば、次のようになります。

```
1999q1(1)2005q1
```

こちらは%tqが使用されています。1999q1(1)2005q1は1999年第一四半期から2005年第一四半期までの各四半期を意味します。1999q1(4)2005q1は各年の第一四半期を示します。

日付のリストを確認する際、Stataはまず関連する変数の形式を確認します。それから、日付/数値の変換関数を使用します。例えば、変数が%td形式を有している場合、td()関数を使用して日付を変換します。%tq形式の変数の場合、tq()関数を使用する、というように進んでいきます。詳細は[D] **Datetime**の中の*Conveniently typing SIF values*をご覧ください。

11.1.10 プリフィックスコマンド

Stataには他のコマンドにプリフィックス(前置き)として使用するコマンドがあります。上記セクション、[U] **11.1.2 by varlist:**で説明したby varlist:は、このプリフィックスコマンドの例になります。このセクション内では、byの使い方を確認しながら、説明を行います。

```
by region: summarize marriage_rate divorce_rate
```

次にこのコマンドを設定します。

```
by region, sort: summarize marriage_rate divorce_rate
```

また、次のように入力することもできます。

```
by region, sort: summarize marriage_rate divorce_rate, detail
```

上記のそれぞれのコマンドは指定したデータの範囲にsummarizeコマンドを実行します。

byそのものはStataの一般的な構文に準拠します。

```
by varlist[, options]: ...
```

「by region, sort: summarize marriage_rate divorce_rate, detail」の中で、regionはbyプリフィックスのvarlistを示し、sortはbyプリフィックスのオプションになります。これは、marriage_rateとdivorce_rateがsummarizeのvarlistになり、detailがsummarizeのオプションになるのと同じです。

byだけがプリフィックスコマンドではありません。一覧は次の通りです。

プリフィックスコマンド	説明
by	データのサブセットにそれぞれコマンドを実行する
collect	コマンドを実行し結果を収集してテーブルに含める
frame	フレームで指定されたデータにコマンドを実行する
statsby	byと同じですが、それぞれのコマンド実行時に統計量を計算する
rolling	移動するサブセットに対してコマンドを実行し、統計量を計算する
bootstrap	ブートストラップサンプルに対してコマンドを実行する
jackknife	データのジャックナイフサブセットに対してコマンドを実行する
permute	ランダムな順列にコマンドを実行する
simulate	人工的に作成されたデータにコマンドを実行する
svy	サーベイサンプリングで集めたデータにコマンドを実行して結果を調整する
mi estimate	多重代入しているデータにコマンドを実行し、多重代入法 (MI)のために結果を調整する
bayes	ベイズ回帰としてモデルの推定をする
fmm	有限混合モデルとしてモデルの推定をする
nestreg	リグレッサブロックでコマンドを実行し、ネストモデルの比較検定結果を表示する
stepwise	ステップワイズ変数の組み込み/除外と共にコマンドを実行する
xi	因子変数との関係を拡張した後にコマンドを実行する。ほとんどのコマンドでは、xiではなく因子変数を使用する方が好ましい(詳細は[U] 11.4.3 因子変数をご覧ください)
fp	1つのリグレッサを使用する、分数多項式でコマンドを実行する
mfp	複数リグレッサを使用する分数多項式でコマンドを実行する
capture	コマンドを実行してリターンコードを取得する
noisily	コマンドを実行して出力を表示する
quietly	コマンドを実行して出力を表示しない
version	特定のバージョン下でコマンドを実行する

最後のグループ（capture, noisily, quietly, version）はStataのプログラミングで使用するプリフィックスコマンドです。過去のコマンドとの関係などの理由から、capture, noisily, quietlyはコロン（:）を使用しなくてもStataは動作します。つまり、プログラマーは次のように入力できます。

```
quietly regress ...
```

もちろん、以下も利用できます。

```
quietly: regress ...
```

他ではすべてコロン（:）が必要です。プリフィックスコマンドの詳しい説明などに関しては対応するリファレンスマニュアルに加えて、Baum (2009)をご覧ください。

11.2 省略のルール

Stataはコマンドなどを省略できます。このマニュアルでは、読みやすさや間違いを避けるため、基本的にコマンド、変数名、オプションなどは全て省略しないようにしています。

```
. summarize myvar, detail
```

経験豊富なStataユーザは、よく使うコマンドを中心に、次のように省略します。

```
. sum myv, d
```

一般的に、コマンド、オプション、変数名は、特定できる最も短い文字列まで省略できます。

このルールは、コマンドまたはオプションが簡単にやり直しできない操作を実行する際は適用されません。つまり、コマンドは全てを入力する必要があります。

また、頻繁に使用されるいくつかのコマンドとオプションに対しては、一般的なルールよりも短く省略できます。

変数名には、この一般ルールが全て適用されます。

11.2.1 コマンドの省略

コマンドやオプションの使用可能な最短省略形は、それぞれのコマンド構文の項目を確認してください。最短省略形は下線部分で表します。

generate
append
rotate
run

下線がない場合、そのコマンドは省略できません。例えば、上記例ではreplaceは省略することができません。これは、replaceコマンドはデータを変更するので、間違いを防ぐために設定されていません。

renameコマンドはren, rena, renamのどれを入力しても同じコマンドとして認識しますまた、さらに短い省略形を使用できる場合もあります。dから始まるコマンドには、decode, describe, destring, dir, discard, display, do, dropなどがあり、これによるとdescribe固有の省略形はdescとなります。しかし、describeコマンドは頻繁に使用されるコマンドのため、1文字、つまりdまで省略可能です。同じように、listコマンドは1文字、lにまで省略できます。

一般的な省略ルールの例外に当てはまるのは、データの変更や破壊をするコマンドです。これはコマンドを全て記述する必要があります。アルファベットのdから始まる2つのコマンド、discardとdropはデータを元に戻すことができないという観点から破壊的である、と認識されます。よって、これら2つのコマンドを実行するには、省略しないで全て入力する必要があります。

この一般的なルールに対して例外なのは、adoファイルとして使用されるコマンドです。これらのコマンドは省略できません。adoファイルコマンドは外部で指定し、それぞれの名前はファイルの名前と対応するためです。

11.2.2 オプションの省略

オプションの省略は、先ほど説明したコマンドの省略と同じ理論で成り立っています。それぞれのオプションで最短利用可能省略形を確認するにはコマンド構文を確認してください。summarizeの構文のオプション部分まで確認してみましょう。

summarize ..., detail format

つまり、detailオプションはd, de, det, ... , detailなどと省略できます。同じように、formatオプションはf, fo, ... , formatのように省略できます。

clearとreplaceオプションは数多くのコマンドと共に使用されます。clearオプションは、ディスクに保存した後にメモリ上のデータに追加した変更点があっても、メモリ内にある全てのデータを消して継続したいときなどに使用します。clearオプションに省略形は存在しません。つまり、例えば「use newdata, clear」のように使用します。

replaceオプションは、既存のデータセットの上に上書きする形でデータを更新します。例えば、mydata.dtaが既にあるときに「save mydata」と入力すると、Stataは “file mydata.dta already exists”というメッセージを返して、上書きはしません。Stataにデータセットの上書きを認めるには、「save mydata, replace」と入力します。なお、replaceオプションは省略できません。

❑ テクニカルノート

replaceオプションはclearオプションよりもより強力な修正を施すため、実際に使用する前に一度確認することをお勧めします。clearオプションを間違えると、数時間の作業結果を無くすことになりますが、replaceオプションを間違えると、数日間の作業結果をなくすことになりかねません。

❑

11.2.3 変数名の省略

- 変数名はメモリにロードしているデータの中で、それぞれの変数を区別できる最短の文字列まで省略できます。

  例えば、データセットに次の4つの変数、state, mrgrate, dvcrate, dthrateがある場合、dvcrateはdvcrat, dvcra, dvcr, dvc, dvのどれでも省略して使用できます。「list dv」と入力すると、dvcrateのデータをリスト表示します。しかし、変数dvcrateはdと省略できません。これは、dだけの省略では、dvcrateとdthrateの変数の見分けがつかないからです。もし「list d」と入力すると、Stataは“ambiguous abbreviation”というメッセージを表示します。（dから始まる“全ての”変数をリストしたい場合は「list d*」と入力します。詳細は[U] 11.4 varname and varlistsをご覧ください。）

- ~記号は「ゼロまたは多くの文字が入る」という意味で使用できます。例えば、変数r~8 は変数rep78, またはrep1978, repair1978, r8を示します。（~記号は[U] 11.4 varname and varlistsで説明している*記号と同様に使用できますが、「この条件に当てはまる変数は1つだけ」という制限がつきます。）

  先ほど、変数は省略可能と説明しました。dvcrate変数を示すのに、dvcrを使用することができますが、他の変数でdvcrを使用している場合はエラーが返されます。dvcrと入力するのとdvcr~と入力するのは同じです。

11.2.4 プログラマ向けの省略

Stataには変数名やコマンド名の省略に関してプログラマーを手助けする便利なコマンドや関数が複数あります。そのコマンドや関数は以下の表にまとめました。

コマンド/関数	説明
unab	一般的な変数リストを拡張し、省略を外す
tsunab	時系列演算子を含む可能性がある変数リストを拡張して省略を外す
fvunab	時系列演算子または因子変数を含む可能性がある変数リストを拡張して省略を外す
unabcmd	コマンド名を省略しない
novarabbrev	変数の省略をオフにする
varabbrev	変数の省略をオンにする
set varabbrev	変数の省略をサポートするか設定する
abbrev(s,n)	文字列関数でsをn表示桁数に省略する
abbrev(s,n)	上記のMataに対応するもので、sとnは行列になる

11.3 名前付けの規則

名前は32文字のアルファベット（A-Zとa-z）、数字（0-9）、アンダーバー（_）を使用する文字の組み合わせです。

プログラミングを行うユーザの皆様：ローカルマクロの名前は最大31文字までしか使用できません。詳細は[U] 18.3.1 ローカルマクロをご覧ください。

Stataは以下の名前を予約しています（ユーザは使用できません）。

```
_all          _n          _r_lb
_b            _N          _r_p
byte          _pi         _r_se
_coef         _pred       _r_ub
_cons         _r_b        _r_z
double        _rc         _se
float         _r_ci       _skip
if            _r_cri      str#
in            _r_crlb     strL
int           _r_crub     using
long          _r_df       with
```

ユーザはこれらの予約している名前を変数名として登録できません。

名前の1文字目はアルファベットまたはアンダーバーです。しかし、アンダーバーで変数名を始める事はお勧めしません。Stata内部のビルトイン変数は全てアンダーバーで始まり、Stata社の開発者たちが新しい変数を追加する権利を有しているからです。

Stataは大文字と小文字を区別します。つまり、myvar, Myvar, MYVARは全て3つの異なる名前です。Stata内にある変数以外の項目を含む大部分のオブジェクトはこの名前付けの規則に従います。

11.4 varname and varlist

varlistとは変数名のリストです。varlistの中にある変数名は新しい（未作成の）変数を示すか、既存の変数のどちらかを示します。newvarlistは常に新しい（未作成の）変数を意味します。同じように、varnameはひとつの変数を指します。これは既存でも未作成でも同じです。newvar は常にひとつの未作成の変数を示します。

コマンドの構文では変数名を、例えば「groupvar」のように記述しますが、これは変数名（varname）を表しています。異なる変数を示す名前は、その変数の用途に対するヒントを提示しています。例えば、groupvarはデータ内のグループを定義する変数にあたります。varnameやvarlistを示すものとして、ほかに共通に使用されているものは次があります。

depvar：推定コマンドでの従属変数を意味します。

indepvars：推定コマンドでの独立変数を含むvarlistを意味します。

xvar：しばしばグラフのx軸上にプロットされる連続実数の変数を意味します。

yvar：しばしばxvarの関数としてy軸上にプロットされる変数を意味します。

clustvar：その観測が属するクラスターまたはグループを識別する数値変数を意味します。

panelvar：横断的時系列データとしても知られるパネルデータのパネルを識別する数値変数を意味します。

timevar：%td、%tc、または%tCという形式の数値変数を意味します。

11.4.1 既存変数のリスト

既存の変数名のリストでは、変数の名前は繰り返し設定できます。

例題11

「list state mrgrate dvcrate state」と入力すると、変数stateは2回（左端の列と右端の列）リストされます。

既存の変数の名前は上記の[U] 11.2 省略のルールにあるように省略できます。また、「*」を使用することもできます。これは「ゼロ個以上の文字がこの位置に当てはまる」を意味します。例えば、変数名の一部分に*を続けて入力する（例えば、sta*）と、これはこの文字で開始する全ての変数を参照していることになります。また、*を変数名の一部分の前に入力する（例えば、*rate）と、これはこの文字で終わる全ての変数を参照することになります。*を文字と文字の間に入力する（例えば、m*rate）と、これはここで指定した文字で始まって終わる変数を参照することになります。変数の省略の中に2つ以上の*を入力することもできます。

例題12

変数poplt5, pop5to17, pop18pがデータの中にある場合、「pop*」と入力すると、これら3つ全ての変数を示します。例えば、「list state pop*」と入力すると、リスト表示されるのは、state, poplt5, pop5to17, pop18pという4つの変数です。

変数inc1990, inc1991, ... , inc1999, incfarm1990, ... , incfarm1999, pop1990, ... , pop1999, s1990, ... , ms1999という変数のセットを有しているデータセットで「*1995」と入力すると、これはinc1995, incfarm1995, pop1995, ms1995を示す、省略形です。つまり、「list *1995」と入力できます。

同じデータセットで「list i*95」と入力すると、inc1995とincfarm1995のリストを表示します。

「list i*f*95」と入力すると、これはincfarm1995をリスト表示する省略形になります。

~は*の代わりに使用できる記号で、同じ事を意味します。ただし、~記号で複数の変数が合致する場合、それら全ての変数を利用せず、メッセージを返します。

例題13

先ほどの例題では、「list i~f~95」と入力してもincfarm1995の変数を表示できます。しかし、例えばこのデータセットにinfant1995という変数がある場合、「list i*f*95」は2つの変数をリストしますが、「i~f~95」と入力すると、”ambiguous abbreviation”というメッセージが返されます。

また、?を使用すると?記号部分に一文字だけ入れるようになります。*はゼロ以上の文字が入ることを意味するので、例えば?*は1つ以上の文字が、??*は2つ以上の文字があてはまります。

例題14

rep1, rep2, ..., rep78という変数を有しているデータセットでは、rep?はrep1, rep2, ..., rep9, を指し、rep?? はrep10, rep11, ..., rep78を示します。

2つの変数名の間にダッシュ(-)を入れると、これらの変数の間に保存している変数を全て指定できます。これには、両サイドの変数も含みます。保存している順番はdescribeコマンドで確認してください。保存順で変数のリストは作成されます。

例題15

データセットの変数state, mrgrate, dvcrate, dthrateがこの順番で保存されている場合、「list state-dvcrate」と入力すると、これは「list state mrgrate dvcrate」と入力する事と同義になります。どちらの場合でも、3つの変数がリストされます。

11.4.2 新しい変数のリスト

未作成の*変数のリスト*では、変数の名前の繰り返しや省略はできません。

変数の前半部分が同じ文字で始まり数字で終わる場合のみ、2つの変数名の間にダッシュ (-) を入力して範囲を指定することができます。このダッシュ表記は、変数の名前の後の数字を昇順で表示します。

例えば、「input v1-v4」と入力するのと「input v1 v2 v3 v4」と入力するのは同じ意味になります。「infile state v1-v3 ssn using rawdata」と入力するのは、「infile state v1 v2 v3 ssn using rawdata」と同じです。

新しく変数名の指定を必要とする多くのコマンドは*stub**という表記での実行が可能です。例えば、predictコマンドで4つの新しい変数を作成する場合は、predict pred*と入力して、変数pred1, pred2, pred3, pred4を新しく作成します。

また、保存形式を変数の名前の前に入力すると、デフォルト以外の保存形式を強制的に設定します。数値の保存形式はbyte, int, long, float(デフォルト), doubleの5つがあります。文字列の保存形式はstr#です。ここで、#は1から2045までの数字が入り、これは文字列あるいはstrLの最大の長さを示します。詳しくは[U] 12 データをご覧ください。

例えば、「var1 str8 var2 var3」と入力されている場合、var1と var3はデフォルトの保存形式で、var2はstr8形式（文字列保存形式で、最大文字数は8）を使用します。

「var1 int var2 var3」と記載すると、var2はint形式で保存されます。変数名をまとめる時は括弧を使うことで、同じ保存形式に設定できます。「var1 int(var2 var3)」と入力すると、var2とvar3の両方をintとして保存します。同じように「var1 str20(var2 var3)」と入力すると、var2と var3の両方をstr20として保存します。異なる保存形式は[U] 12.2.2 数値の保存形式と[U] 12.4 文字列にリストしています。

例題16

「infile str2 state str10 region v1-v5 using mydata」と入力するとstateとregion変数の文字列をmydata.rawファイルから読み込み、それぞれをstr2とstr10の保存形式で保存します。あわせて、変数v1からv5もあり、これらはデフォルトの保存形式（float）で設定されます（set typeコマンドで他の形式をデフォルトにしている場合は異なります）。

「infile str10(state region) v1-v5 using mydata」と入力するとほぼ同じ結果を導き出しますが、今度はstateとregionの保存形式は、どちらもstr10になります。（一度データをインポートしてからcompressコマンドを使用して、文字列を短くすることもできます。詳細は[D] compressをご覧ください。この項目はぜひ一度目を通しておいてください。）

テクニカルノート

数値変数にはコロンを使用することで値ラベル名を設定できます。（値ラベルの説明については、[U] 12.6 データセット、変数、値ラベルを参照してください。例えば、「var1 var2:myfmt」は変数var2がmyfmtという名前で保存してある値ラベルと関連付けをします。これは、var1 var2を入力した後に「label values var2 myfmt」というコマンドを実行したときと同じ効果になります。

そのコマンドの中で値ラベルを関連付けるにはコロンを使用すると便利です。詳細は[D] inputと[D] infile (free format)をご覧ください。

例題17

「infile int(state:stfmt region:regfmt) v1-v5 using mydata, automatic」と入力すると、mydata.rawから州と地域の情報を読みこみ、int形式で保存します。あわせて、変数v1からv5も読み込み、こちらはデフォルトの保存形式で保存します。

以前の例題では、州と地域はどちらも文字列でした。文字列は数値として保存できるのでしょうか。（質問の答えについては、[U] 12.6 データセット、変数、値ラベル を参照してください。コロンは使用する値ラベルを指定し、automaticオプションは文字列に数字のコード分けを行うように指示します。州ごとの数値はStataがその場で設定することになりますが、state変数に保存されます。数字のコードと値ラベルの関連付けに関してはstfmtという値ラベルに保存されます。同じように、地域にも数字のコードが割り振られ、その値はregionに保存され、関連付けに関してはregfmtに保存されます。

ここでデータをlistすると、 変数stateとregionは文字列と同じように見えます。例えばstateは、AL, CA, WAのように表示されますが、実際に格納しているものは1, 2, 3, 4というような数字だけです。

11.4.3 因子変数

因子変数は既存の変数に拡張を加える変数リストです。因子変数と共に使用できるコマンドの場合、変数名をデータから入力するほかに、因子変数を入力してください。ちなみに、次のように表示されます。

```
i.varname
i.varname#i.varname
i.varname#i.varname#i.varname
i.varname##i.varname
i.varname##i.varname##i.varname
```

因子変数はカテゴリカル変数から指標変数を作成し、これらはほとんどの推定コマンドや推定後コマンド、そしてわずかではありますが、それ以外のコマンドで使用できます。

例えば、変数groupは1，2，3の値を使用します。Stataのlistコマンドは因子変数を使えるので、この因子変数がどのように拡張されているのかを確認してみます。次のように入力してみましょう。

```
. list group i.group in 1/5
```

	group	1.group	2.group	3.group
1.	1	0	0	0
2.	1	0	0	0
3.	2	0	1	0
4.	2	0	1	0
5.	3	0	0	1

このデータの中に1.group，2.group，3.groupという変数は存在しません。単にgroupという変数があるだけです。しかし、「i.group」と入力すると、Stataは変数1.group，2.group，3.groupが存在するように挙動します。1.group，2.group，3.groupのことを仮想変数と呼びます。1.groupはgroupが1のとき、1を取り、それ以外では0となります。2.groupはgroupが2のとき、1を取り、それ以外では0となります。3.groupはgroupが3のとき、1を取り、それ以外では0となります。

因子変数演算子が反映するカテゴリカル変数は、正の整数のみを使用する必要があります。

❑ テクニカルノート

先ほど、group = 3の時には1を表示し、それ以外の時は0を表示する、と記載しました。groupの値が欠損値の場合、3.groupは欠損値になります。正確に説明すると、group = 3の時には1を表示し、group ≥ .となるときにはシステム欠損値（.）を表示し、それ以外の場合は0になります。

❑

❑ テクニカルノート

先ほど、「i.group」と入力すると、Stataは変数1.group, 2.group, 3.groupが存在するように計算するといいましたが、これはi.groupと入力すると仮想変数を作成していると考えることもできます。しかし、それは誤りです。仮想変数は常に存在します。

実際には、i.groupは1.group, 2.group, 3.groupを示す、省略形の表示です。因子変数を使用できるコマンドでは、仮想変数を指定できます。つまり、上記のリストは次のようにコマンドを入力しても作成できます。

```
. list group 1.group 2.group 3.group in 1/5
```

#.varnameはvarname = # が成り立つ時に1を表示し、varname ≥ .となるときにはシステム欠損値(.)が成り立ち、

それ以外の場合は0になるように定義します。つまり、値として1, 2, 3しかなくても、4.groupは定義されています。全ての省略形の中で、4.groupは0 になります。4.groupを参照しても、“virtual variable not found”というようなエラーは表示しません。

❑

回帰分析で、因子変数を利用すると、1カテゴリがベースカテゴリとして決定されます。例えば、次のようなコマンドでは、

```
. regress y i.group
```

このコマンドは次のように入力しても同様の結果が得られます。

```
. regress y 1b.group 2.group 3.group
```

1b.groupは他の仮想変数と異なります。このbはベースとなる値であることを示しています。1b.groupは0に等しい仮想変数です。これは次のように入力することで確認できます。

```
. list group i.group in 1/5
```

	group	1.group	2.group	3.group
1.	1	0	0	0
2.	1	0	0	0
3.	2	0	1	0
4.	2	0	1	0
5.	3	0	0	1

i.groupの集合が線形回帰に組み込まれた場合、1b.groupは全て同じ値なので推定から除外され、2.groupと3.groupの係数がgroup = 1からの変化分を反映します。

11.4.3.1 因子変数演算子

i.groupは因子変数と呼ばれていますが、正確には、groupは因子変数の演算子が適用されるカテゴリカル変数であるといえます。因子変数の演算子は、5つあります。

演算子	説明
i.	指標子を指定する演算子
c.	連続数として扱う演算子
o.	指標子や変数を読み飛ばす演算子
#	交差項を作るバイナリ演算子
##	全要因交差項を作るバイナリ演算子

i.groupと入力すると、group内にある特殊な値に対して指標子を作成します。マニュアルの中では、少し簡単にi.groupはgroup内のレベルごとに指標子を作成する、といいます。または、もう少し内容を短くしてi.groupはgroupの指標子を作成する、とも表現します。

c.演算子は連続することを示します。解説は後ほど行います。

o.演算子は連続変数あるいはカテゴリカル変数のレベルの中で無視すべき変数を指定します。例えば、o.ageは連続変数ageを無視するように指示します。また、o2.groupはgroup = 2の指標子は無視する意味になります。

#と##は、それぞれ交差項と全要因交差項といいます。ペアになっている変数に使用する指標子です。i.group#i.sexは変数groupとsexの組み合わせで、それぞれ指標子を作成します。group#sexは同じ事を意味します。つまり#を使用するとi.プリフィックスを使用することを暗に示しています。

group#c.age(またはi.group#c.age)は変数groupのレベルと連続変数ageの交差項を意味します。ここでは、i.gro

upをまず作成し、各レベルの変数ageをかけています。i.groupは仮想変数1b.group, 2.group, 3.groupに拡張することが分かっているので、group#c.ageは1b.group*age, 2.group*age, 3.group*ageをまとめたものと同じです。1.group*ageはゼロになります。これは、1.groupがゼロだからです。2.group*ageは、group = 2である場合はageの値になりますが、それ以外は0です。3.group*ageはgroup = 3である場合はageの値になりますが、それ以外は0です。yの値によるageのgroup#c.ageでは、1.group*ageが無視され、2.group*ageはベースグループを元にしたgroup = 2の場合の年齢係数を計測し、3.group*ageは同じようにベースグループを元にしたgroup = 3の場合の年齢係数を計測します。

もう少し演算子の使用に関する例を紹介します。

因子指定	結果
i.group	groupのレベルの指標子を指定
i.group#i.sex	それぞれの変数groupとsexの組み合わせの指標子を二元の交差項で作成
group#sex	i.group#i.sexと同じ
group#sex#arm	それぞれの変数group, sex, armの組み合わせの指標子を三元の交差項で作成
group##sex	i.group i.sex group#sexと同じ
group##sex##arm	i.group i.sex i.arm group#sex group#arm sex#arm group#sex#armと同じ
sex#c.age	2つの変数、男性のageとそれ以外では0、女性のageとそれ以外では0、が成り立つ。ageがモデル内にある場合、2つの仮想変数のうち1つがベースとして設定される
sex##c.age	i.sex age sex#c.ageと同じ
c.age	ageと同じ
c.age#c.age	ageの二乗
c.age#c.age#c.age	ageの三乗

いくつかの因子変数は同じ変数リスト内で指定されています。例えば次のようになります。

```
. regress y  i.sex i.group sex#group age sex#c.age
```

あるいは、以下でも同じです。

```
. regress y  sex##group sex##c.age
```

11.4.3.2 ベースレベル

回帰コマンドでi.groupと入力すると、group = 1がベースレベルになります。特に指定しない場合、最も小さいレベルがベースレベルとなります。

因子変数のベースレベルを指定する場合、ib.演算子を使用します。構文は次の通りです。

基準演算子[a]	説明
ib#.	#をベースとして使用、# =変数の値
ib(##).	順番に並べた上の#番目の値をベースにする[b]
ib(first).	最も小さな値をベースとして使用（デフォルト）
ib(last).	最も大きな値をベースとして使用
ib(freq).	最も頻度が多い値をベースとして使用
ibn.	ベースレベルなし

注[a] i は無視できます。例えば、ib2.groupと入力する代わりにb2.groupと入力することもできます。
注[b] 例えば、ib(#2).は2番目の値を基準値として使用します。

つまり、group = 3をベースとして使用する場合は、「ib3.group」と入力します。これを行うには、次のように入力します。

```
. regress y  i.sex ib3.group sex#ib3.group age sex#c.age
```

または次のように入力します。

```
. regress y  i.sex ib3.group sex#group age sex#c.age
```

つまり、ベースは一度だけ設定する必要があります。1度以上ベースを設定する場合は、同じベースレベルである必

要があります。コマンドの中でベースレベルを変更しようとすると、エラーが返されます。

ib3.groupと入力した場合、仮想変数は1.group, 2.group, b3.groupとなります。

ib(freq).groupと入力すると、仮想変数は最も頻度が多く出ている変数により、b1.group, 2.group, 3.group、または1.group, b2.group, 3.group、あるいは1.group, 2.group, b3.groupのいずれかになります。

11.4.3.3 ベースレベルを永久的に設定する

fvsetコマンドを使用すると、ベースレベルを永続設定して、固定できます。詳細は[R] fvsetをご覧ください。たとえば、次のように実行します。

```
. fvset base 3  group
```

これはベースをgroupの3に設定しています。この設定はデータに記録されるので、このデータを再保存すると、以降のセッションでは、ベースレベルはいつもgroup3になります。

ベースレベルをデフォルトに戻したい場合は、次のように入力してください。

```
. fvset base default  group
```

ベースレベルをグループの最大値に設定したい場合は、次のように入力します。

```
. fvset base last  group sex arm
```

詳細は[R] fvsetをご覧ください。

明確に設定したかによらず、ib.を使用するとベースレベルを一時的に変更できます。

11.4.3.4 レベルを設定する

i.groupと入力すると、仮想変数はb1.group, 2.group, 3.groupとなります。i.groupと入力したかによらず、これらへはアクセスができます。例えば、if構文での使用もできます。

```
.list if 3.group
                  (省略)
.generate  over_age  =  cond(3.group,  age-21,  0)
```

このセクションを通して#.group（例えば、3.group）がi.groupとは大きく異なると見えるように説明してきましたが、正式に記載するとi3.groupとなります。iは省略することができます。ポイントとしては、i3.groupはi.groupの特殊事例である、ということです。i3.groupはgroupの3番目のレベルの指標子を指定し、i.groupはgroupにおける全てのレベルの指標子を指定します。つまり、上記のコマンドは次のように入力できます。

```
.list if i3.group
                  (省略)
.generate  over_age  =  cond(i3.group,  age-21,  0)
```

同じように、仮想変数であるb1.group, 2.group, 3.groupは正式に記述するとib1.group, i2.group, 3.groupとなります。先頭に付くiは、その後に続く文字が数字かbでベースを指定している場合は無視できます。

また、ユーザはレベルの範囲、仮想変数の範囲を指定できます。それには、i(numlist).varnameを使用します。これは推定コマンドと共にフィットで使用するモデルを指定するときに便利です。数値リスト（numlist）を設定するときには、iを省略することはできません。

例	説明
i2.cat	cat = 2のときの指標子
2.cat	i2.catと同じ
i(2 3 4).cat	3つの指標子, cat = 2, cat = 3, cat = 4を示す。 i2.cat i3.cat i4.catと同じ
i(2/4).cat	i(2 3 4).catと同じ
2.cat#1.sex	cat = 2とsex = 1が成り立つときには1になる指標子で、それ以外の場合は0になる
i2.cat#i1.sex	2.cat#1.sexと同じ

使用するレベルを選択するほかに、入れるべきではない（無視すべき）レベルをo.演算子を使用して指定できます。コマンドの中でio(numlist).varnameを使用する場合、numlistで指定された以外のvarnameにあるレベルの指標子も含まれます。無視するレベルがo.演算子で指定されると、i.演算子はそれ以外ということで設定され、varnameの残りの指標子は設定できます。

例	説明
io2.cat	cat = 2を取り除いたcatのレベルの指標子
o2.cat	io2.catと同じ
io(2 3 4).cat	cat = 2, cat = 3, cat = 4の3つのレベルを取り除いたcatのレベルの指標子
o(2 3 4).cat	io(2 3 4).catと同じ
o(2/4).cat	io(2 3 4).catと同じ
o2.cat#o1.sex	それぞれの変数catとsexの組み合わせの指標子を cat = 2とsex = 1のレベルを取り除いて使用

11.4.3.5 変数の集団に演算子を適用する

因子変数の演算子は括弧とあわせて使用することで変数のグループに適用できます。例えば、次のように入力します。

```
i.(group sex arm)
```

これは、i.group i.sex i.armと同じです。

次の例題では、変数 group, sex, arm, cat はカテゴリカル変数で、age, wt, bpは連続変数です。

例	拡張
i.(group sex arm)	i.group i.sex i.arm
group#(sex arm cat)	group#sex group#arm group#cat
group##(sex arm cat)	i.group i.sex i.arm i.cat group#sex group#arm group#cat
group#(c.age c.wt c.bp)	group#c.age group#c.wt group#c.bp
group#c.(age wt bp)	group#(c.age c.wt c.bp)と同じ

括弧は、実際に入力する分量を減らせると共に、さらに読みやすくできます。例えば、以下のようになります。

```
. regress y  i.sex i.group sex#group age sex#c.age c.age#c.age sex#c.age#c.age
```

これよりも、次のように書いてある方が分かりやすいです。

```
. regress y  sex##(group c.age c.age#c.age)
```

11.4.3.6 時系列演算子で因子変数を使用する

因子変数はL.やF.のような時系列演算子と共に組み合わせて使用できます。その中では、時系列計算などで使用する因子変数のラグ項やリード項を指定でき、iL.groupやLi.groupと入力します。この中で、Lとiの順番は関係ありません。L.group#L.armやL.group#c.ageと入力できます。

例題としては、次を参照してください。

```
. regress y  b1.sex##(i(2/4).group cL.age cL.age#cL.age)
. regress y  2.arm#(sex#i(2/4)b3.group cL.age)
. regress y  2.arm##cat##(sex##i(2/4)b3.group cL.age#c.age) c.bp
>            c.bp#c.bp c.bp#c.bp#c.bp sex##c.bp#c.age
```

11.4.3.7 ビデオ例題

Introduction to factor variables in Stata, part1: The basics

Introduction to factor variables in Stata, part 2: Interactions

Introduction to factor variables in Stata, part 3: More interactions

11.4.4 時系列の変数リスト

時系列の変数リスト（varlists）は既存の変数における変数リストの変化形です。コマンドが時系列の変数リストを使用できる時は、時系列演算子を追加できます。例えば、L.gnpは変数gnpのラグ値になります。時系列演算子は次の通りです。

演算子	意味
L.	ラグ x_{t-1}
L2.	2期間に渡るラグ x_{t-2}
...	
F.	リード x_{t+1}
F2.	2期間に渡るリード x_{t+2}
...	
D.	差 $x_t - x_{t-1}$
D2.	差の差 $x_t - x_{t-1} - (x_{t-1} - x_{t-2}) = x_t - 2x_{t-1} + x_{t-2}$
...	
S.	“季節的”な差 $x_t - x_{t-1}$
S2.	ラグの2(季節的な)差 $x_t - x_{t-2}$
...	

時系列演算子は繰り返したり組み合わせたりできます。L3.gnp は変数gnpの3番目のラグ項を意味します。LLL.gnp, LL2.gnp, L2L.gnpも同じ意味です。LF.gnpはgnpと同じです。DS12.gnpは12期間の差の中の1期間の差になります。LDS12.gnpは同じコンセプトで、1回ラグが取られています。

D1.= S1.は成り立ちますが、D2. ≠ S2., D3. ≠ S3.などとなっていきます。D2.は期間の差に対する差を示します。S2.は2期間の差を示します。変数gnpの12 期間の差の差を計算したい場合は、D2S12.gnpと入力します。

演算子は大文字で入力しても小文字で入力しても、違いはありません。ほとんどのユーザはD2S12.gnpではなくd2s12.gnpと入力します。

演算子は好きなように入力できます。Stataは内部でそれぞれの演算子を標準形式に直して使用します。ld2ls12d.gnpと入力すると、実行された変数はL2D3S12.gnpと表示します。

operator#を使用することに追加して、Stataはoperator(numlist)の形式で入力するとそれが演算された変数であると理解します。例えば、L(1/3).gnpと変数リスト部分に入力すると、これはL.gnp L2.gnp L3.gnpと同じになります。演算子は括弧で囲んだ変数のリストに対しても適用できます。例えば、次のようになります。

```
.use http://www.stata-press.com/data/r18/gxmpl1
.list year L(1/3).(gnp cpi)
```

	year	L. gnp	L2. gnp	L3. gnp	L. cpi	L2. cpi	L3. cpi
1.	1989	.	.	.	.	.	.
2.	1990	5837.9	.	.	124	.	.
3.	1991	6026.3	5837.9	.	130.7	124	.
4.	1992	6367.4	6026.3	5837.9	136.2	130.7	124
5.	1993	6689.3	6367.4	6026.3	140.3	136.2	130.7
6.	1994	7098.4	6689.3	6367.4	144.5	140.3	136.2
7.	1995	7433.4	7098.4	6689.3	148.2	144.5	140.3
8.	1996	7851.9	7433.4	7098.4	152.4	148.2	144.5

括弧を使用する表記については、どの演算子でも使用できます。D(1/3).gnpと入力すると、1番目から3番目の差を表示します。

括弧を使う表記は、複数の演算子を使用するような表記でも使う事ができます。例えば、

L(0/3)D2S12.gnpのようになります。

演算子リストには1つの括弧の組み合わせを使用でき、それにはnumlistを囲むことができます。詳細は[U] 11.1.8 numlistをご覧ください。

時系列演算子L.とF.は因子変数と組み合わせることができます。因子変数regionのレベルに対して指定変数のラグをとる場合、iL.regionと入力します。また、レベルの指標変数を設定して、変数のラグを計算している、ともいえますこれらは同じ内容になります。

両方の因子変数と時系列演算子はnumlistと括弧の表記は組み合わせることができます。例えば、「iL(1/3).region」は変数regionの最初の3つのレベル指標子のラグを指定しています。regionに4 つのレベルがある場合、これは「i1L1.region i2L1.region i3L1.region i4L1.region i1L2.region i2L2.region i3L2.region i4L2.region i1L3.region i2L3.region i3L3.region i4L3.region」と入力するのと同じです。さらに表記について記載します。「i(1/2)L(1/3).(region education)」と入力すると、変数regionとeducationに対して、レベル1と2の指標変数の最初の3つのラグを指定します。

❑ テクニカルノート

D.とS.の時系列演算子は因子変数と組み合わせて使うことはできません。これはこのような組み合わせは2つの意味に受け取られるからです。iD.aとあると、レベル指標子で変数aの前の期間からの差を求めているのか、2つの期間の差を意味するレベル指標子なのか、区別がつかないためです。これらは通常、同じ値ではありません。また指標子の数自体も異なります。さらに、この計算が必要になるのは稀です。

❑

変数リストで時系列演算子を使う前に、tssetコマンドを使用して時間変数を設定する必要があります。

```
. list l.gnp
time variable not set
r(111);
. tsset time
(省略)
. list l.gnp
(省略)
```

詳細は[TS] tssetをご覧ください。時間変数は整数を入力する必要があります。また、データは時間変数でソートしてください。tssetを行うと自動的にソートされますが、今後以下のようなエラーが表示されることもあるかもしれません。

```
.list l.mpg
not sorted
r(5);
```

それから「sort time」と入力するか「tsset」を行って順番を整列させます。

時系列演算子は時間変数が示す内容に注意します。L2.gnpはgnp_{t-2}を意味し、データセットの中に欠損値があっても、それは変わりません。次のデータセットでは、1992年分の観測値が欠損となっています。

```
. use http://www.stata-press.com/data/r18/gxmpl2
. list year gnp l2.gnp, separator(0)
```

	year	gnp	L2. gnp
1.	1989	5837.9	.
2.	1990	6026.3	.
3.	1991	6367.4	5837.9
4.	1993	7098.4	6367.4
5.	1994	7433.4	.
6.	1995	7851.9	7098.4

← 正しく入力されています

演算子はそれぞれの表現で使用することができます。

```
. generate gnplag2 = l2.gnp
(3 missing values generated)
```

Stataは断面的な時系列データも理解します。時系列データの断面がある場合、データをtssetするときにこれを記述します。

```
.tsset country year
```

詳細は[TS] **tsset**をご覧ください。最も、それを入力するか、次の情報を入力します。

```
.xtset country year
```

xtsetはパネルデータを設定するコマンドです。ちょうどtsset で時系列データを設定する様に使用します。この状態ではこの2つのコマンドは同じ事を行っています。パネルデータセットは必ずしも断面的な時系列データセットとは限りませんが、その場合、2つ目の変数は時間ではありません。つまり、xtsetは次のように定義できます。

```
.xtset country
```

詳細は[XT] **xtset**をご覧ください。

11.4.4.1 ビデオ例題 .

Time series, part 3: Time-series operators

11.5 by varlist: 構文

by varlist: command

byプリフィックスを使用すると、固有の値または組み合わせのvarlistにある変数セットに対してコマンドを実行します。varlistは数値変数、文字列変数、あるいは数値変数と文字列変数の混合でも使用できます。（varlistは時系列演算子を使用することはできません。）

byは任意に使用するプリフィックスコマンドで、varlistにある変数を分類項目として分類したそれぞれのグループに、同じコマンドを実行します。

それぞれの項目を反復している間、byグループの最初の変数との関係から、_nと_Nのシステム変数が設定されます。詳細は**[U] 13.7 Explicit subscripting**をご覧ください。inをby varlist:と共に使用することはできません。これは、inを設定する際、観測の絶対値を設定するのではなく、相対的な値を設定するためです。

テクニカルノート

inとbyを組み合わせて使用できない事は、あまり大きな問題ではありません。これはif条件でinの機能以上の事ができるからです。例えば、全てのbyグループの中の最初の3つの観測値にコマンドを実行したい場合、次のように入力します。

```
. by varlist: command if _n<=3
```

コマンドで実行した結果は、それぞれの観測値グループで別々のデータセットを作成してから保存（save）し、それを別々に使用（use）してコマンド実行した場合と同じになります。

例題18

byを使用した例題は上記の[U] 11.1.2 by varlist:で紹介しています。_n, _N,行番号の指定方法に対するbyの影響を[U] 13.7 Explicit subscriptingで説明しています。

byではまず、データがソートしてあることが前提になります。例えば、北東部と西部の420の町の1月から7月までの気温(華氏)の平均値のデータで、地域（by region）ごとの平均を入手するには、次のように入力します。

```
. use http://www.stata-press.com/data/r18/citytemp3, clear
(City temperature data)
. by region: summarize tempjan tempjuly
not sorted
r(5);
```

Stataは入力したコマンドを実行しませんでした。これは、データがregionでソートされていないからです。この場合、ソート（sort）をregionで行ってからコマンドを実行するか、byのsortオプション（同じ効果になります）を使用する必要があります。

```
. by region, sort: summarize tempjan tempjuly
```

-> region = NE

Variable	Obs	Mean	Std. dev.	Min	Max
tempjan	164	27.88537	3.543096	16.6	31.8
tempjuly	164	73.35	2.361203	66.5	76.8

-> region = N Cntrl

Variable	Obs	Mean	Std. dev.	Min	Max
tempjan	284	21.69437	5.725392	2.2	32.6
tempjuly	284	73.46725	3.103187	64.5	81.4

-> region = South

Variable	Obs	Mean	Std. dev.	Min	Max
tempjan	250	46.1456	10.38646	28.9	68
tempjuly	250	80.9896	2.97537	71	87.4

-> region = West

Variable	Obs	Mean	Std. dev.	Min	Max
tempjan	256	46.22539	11.25412	13	72.6
tempjuly	256	72.10859	6.483131	58.1	93.6

例題19

上記例と同じ例題を元に、by regionを使用して1月の平均気温と7月の平均気温の回帰を推定しましょう。どちらの気温も華氏で表示されます。

```
. by region: regress tempjan tempjuly
```

-> region = NE

Source	SS	df	MS		
Model	1529.74026	1	1529.74026	Number of obs	= 164
Residual	516.484453	162	3.18817564	F(1, 162)	= 479.82
				Prob > F	= 0.0000
Total	2046.22471	163	12.5535258	R-squared	= 0.7476
				Adj R-squared	= 0.7460
				Root MSE	= 1.7855

tempjan	Coefficient	Std. err.	t	P>|t|	[95% conf. interval]	
tempjuly	1.297424	.0592303	21.90	0.000	1.180461	1.414387
_cons	-67.28066	4.346781	-15.48	0.000	-75.86431	-58.697

-> region = N Cntrl

Source	SS	df	MS		
Model	2701.97917	1	2701.97917	Number of obs	= 284
Residual	6574.79175	282	23.3148644	F(1, 282)	= 115.89
				Prob > F	= 0.0000
Total	9276.77092	283	32.7801093	R-squared	= 0.2913
				Adj R-squared	= 0.2887
				Root MSE	= 4.8285

tempjan	Coefficient	Std. err.	t	P>|t|	[95% conf. interval]	
tempjuly	.9957259	.0924944	10.77	0.000	.8136589	1.177793
_cons	-51.45888	6.801344	-7.57	0.000	-64.84673	-38.07103

-> region = South

Source	SS	df	MS		
Model	7449.51623	1	7449.51623	Number of obs	= 250
Residual	19412.2231	248	78.2750933	F(1, 248)	= 95.17
				Prob > F	= 0.0000
Total	26861.7394	249	107.878471	R-squared	= 0.2773
				Adj R-squared	= 0.2744
				Root MSE	= 8.8473

tempjan	Coefficient	Std. err.	t	P>|t|	[95% conf. interval]	
tempjuly	1.83833	.1884392	9.76	0.000	1.467185	2.209475
_cons	-102.74	15.27187	-6.73	0.000	-132.8191	-72.66089

```
-> region = West
      Source |       SS       df       MS              Number of obs =     256
-------------+------------------------------           F(1, 254)     =    2.84
       Model |  357.161728     1  357.161728            Prob > F      =  0.0932
    Residual |  31939.9031   254  125.74765            R-squared     =  0.0111
-------------+------------------------------           Adj R-squared =  0.0072
       Total |  32297.0648   255  126.655156           Root MSE      =  11.214

------------------------------------------------------------------------------
     tempjan |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
    tempjuly |   .1825482   .1083166     1.69   0.093    -.0307648    .3958613
       _cons |    33.0621    7.84194     4.22   0.000     17.61859    48.5056
------------------------------------------------------------------------------
```

この回帰によると、北東部で7月の平均気温が1度上昇すると、1月の平均気温は1.3度上昇することになります。しかし、西部では、0.18度上昇するにとどまります。これはかろうじて有意である、といえます。

◁

❑ テクニカルノート

byには2つ目の構文があり、これはデータなどの安全面に基準を置く場合は特に有効です。

by varlist$_1$ (varlist$_2$): command

これはStataにデータがvarlist$_1$とvarlist$_2$でソートされている事を確認してから、それが正しいことを前提にvarlist$_1$でコマンドを実行します。例えば、以下のようになります。

```
. by subject (time): generate finalval = val[_N]
```

このように入力すると、各観測値の最終観測値であるvalをそれぞれのデータの内容として、新しい変数（finalval）が作られます。最終値は、その項目内で時間を元にデータがソートしてある時の最後の値になります。上記コマンドはデータが変数subjectとtimeでソートされている事を確認しています。ソートができている時は次のコマンドを実行します。

```
. by subject: generate finalval = val[_N]
```

データが正しくソートできていない場合、エラーメッセージが表示されます。もちろん、直接次のように入力することもできます。

```
. by subject: generate finalval = val[_N]
```

これはユーザが、しっかりとデータがソート済みであることを確認した場合を前提としています。

byの2つ目の構文はbyのsortオプションからも使用できます。

```
. by subject (time), sort: generate finalval = val[_N]
```

これは、次の構文と同じです。

```
. sort subject time
. by subject: generate finalval = val[_N]
```

❑

by:構文を使用したさまざまな例題に関しては、Mitchell (2010, chap. 7)をご覧ください。あわせて、Cox (2002)もご確認ください。

11.6 ファイル名の規則

いくつかのコマンドでは*ファイル名*を指定する必要があります。ファイル名は使用しているOSごとに一般的な方法を元に設定されます。

Windows	Unix	Mac
mydata	mydata	mydata
mydata.dta	mydata.dta	mydata.dta
c:mydata.dta	~friend/mydata.dta	~friend/mydata.dta
"my data"	"my data"	"my data"
"my data.dta"	"my data.dta"	"my data.dta"
myproj¥mydata	myproj/mydata	myproj/mydata
"my project¥my data"	"my project/my data"	"my project/my data"
C:¥analysis¥data¥mydata	~/analysis/data/mydata "~/my project/my data"	~/analysis/data/mydata "C:¥my project¥my data" "~/my project/my data"
..¥data¥mydata	../data/mydata	../data/mydata
"..¥my project¥my data"	"../my project/my data"	"../my project/my data"

標準ASCII文字以外に使用されるUnicodeのUTF形式がOSにより異なるため、ファイル名へのUnicodeの使用は推奨されません。例えば、ファイル名の保存形式はLinuxでUTF-8であるのに対して、WindowsでUTF-16となり、標準ASCII文字以外をファイル名に用いたファイルをOS間でまたいだ移動を行うと、使用が出来なくなる可能性があります。

ほとんどの場合、ファイル名は実際にロードするファイルです。また、ファイル名にはURLの指定もできます。例えばuse https://www.stata-press.com/data/r18/nlswork です。

Stataをインストールできる全てのOSにおいてファイル名にスペースを含むことができます。これはStataも同じです。しかし、その場合、ファイル名をダブルクオテーションで囲む必要があります。次のように入力します。

```
. save "my data"
```

これは「my data.dta」を作成します。また、次のように入力します。

```
. save my data
```

これは、エラーを返します。

一般的に（例外となるコマンドはcopy, dir, ls, erase, rm, type）、あらかじめ記述していない場合、Stataはファイルの拡張子を自動的に設定します。例えば、「use mydata」と入力すると、Stataは「use mydata.dta」であると推測します。これは.dtaはStataがデータファイルを保存する一般的な拡張子だからです。

下記はStataがさまざまなコマンドで使用する、デフォルトファイル拡張子です。

.ado	自動的にロードするdoファイル
.dct	テキストデータ辞書
.do	doファイル
.dta	Stata形式のデータセット
.dtas	Stata形式のフレームセット
.dtasig	データ署名ファイル
.gph	グラフ
.grec	グラフエディタのレコーディング（テキスト形式）
.irf	インパルス応答関数データセット
.log	テキスト形式のログファイル
.mata	Mataのソースコード
.mlib	Mataライブラリ
.mmat	Mata行列
.mo	Mataのオブジェクトファイル
.raw	テキスト形式のデータ
.smcl	SMCL形式のログファイル
.stbcal	ビジネスカレンダー
.ster	保存した推定量
.sthlp	ヘルプファイル
.stjson	Stataのコレクション結果、ラベル、スタイル
.stpr	プロジェクトファイル
.stptrace	パラメータトレースファイル。詳細は[MI] **mi ptrace**参照。
.stsem	SEMビルダファイル
.stswm	空間重み行列
.stswp	Doファイルエディタのバックアップファイル（swapファイル）
.stxer	lassoコマンドを含む.sterの補助ファイル
.sum	ネットワーク越しの変換を確認するchecksumファイル

ファイルの名前をつけるときに.dta拡張子を使用する必要はありません。もし特定の拡張子を入力した場合、デフォルトではなく、そちらを優先します。例えば、データセットが myfile.datとして保存されている場合、「use m

yfile.dat」と入力すれば使用できます

もし、データセットが特に設定も無くmyfileとだけ保存されている時は、ファイル名の後にピリオドを打ち、拡張子が無いことを示すことができます。このファイルを使用する場合は、「use myfile.」と入力してください。

❑ テクニカルノート

Stataは先ほどのファイル拡張子の他に、12個のファイル拡張子を使用します。これらのファイルは上級プログラマーやStataの内部処理に使用するものです。具体的には以下です。

.class	オブジェクト指向プログラミング向けのクラスファイル。詳細は[P] class
.dlg	ダイアログのリソースファイル
.idlg	ダイアログのリソースのインクルードファイル
.ihlp	ヘルプのインクルードファイル
.key	searchのキーワードデータベースファイル
.maint	メンテナンスファイル(Stataの内部使用のみ)
.mnu	メニューファイル(Stataの内部使用のみ)
.pkg	ユーザサイトのパッケージファイル
.plugin	コンパイルした追加(DLL)
.scheme	グラフスキーマを制御するファイル
.style	グラフのスタイルファイル
.toc	ユーザサイトの説明ファイル

❑

11.6.1 Macユーザへの注意点

「myfolder/myfile」という表記を見たことはありますか。この表記はパスと呼ばれています。これはファイルやフォルダ（ディレクトリとも呼ばれる）の場所を指示するものです。

この表記が好きではないなら、無理して使用する必要はありません。他の方法として、現在使用しているフォルダ内にあるファイルでのみ操作するように制限を設けることもできます。その制限が厳しすぎる場合でもMac版Stataには多くのメニューやボタンがあるので、特に大きな問題は無く使用を回避できるでしょう。逆に、この表記がとても便利だと感じる可能性もあります。この表記を便利だと感じたなら、その定義をこちらで確認してください。

/記号はパス区切りと呼ばれ、パスの中に含まれるフォルダ名やファイル名を区切ります。区切り文字がない状態で始まる場合、そのパスは相対的に現在のフォルダ内を意味します。

例えば、myfolder/myfileというパスは、myfolderフォルダ内にあるmyfileファイルを参照しています。このmyfolderは現在開いているフォルダの中にあります。

..記号は現在のフォルダが入っているフォルダ（1つ外にあるフォルダ）を指します。つまり、../myfileは現在のフォルダが入っている(上の階層の)フォルダ内にあるmyfileで、../nextdoor/myfileはそのフォルダ内にあるnextdoorフォルダ内にあるmyfileです。

パスがパス区切りから始まる場合、そのパスは絶対パスと呼ばれ、現在のフォルダに関係なく、固定されたファイルまたはフォルダへのパスを示します。固定パスの先頭にある/ はルートディレクトリを意味します。つまり、OSがブートしているメインのハードドライブを指します。例えば、/myfolder/myfileというパスは、myfolderフォルダ内にあるmyfileファイルを参照しています。myfolderはメインのハードドライブ内にあります。

11.6.2 ホームディレクトリへのショートカット

Stataは~記号をユーザのホームディレクトリと認識します。したがって、ホームディレクトリにあるmydirフォルダの中のmydata.dtaファイルは次で参照できます。

```
~¥mydir¥mydata.dta
```

上記はWindows版Stataの場合です。

```
~¥mydir¥mydata.dta
```

上記はMac版StataおよびUnix版Stataの場合です。

11.7 参考文献

Baum, C. F. 2016. *An Introduction to Stata Programming.* 2nd ed. College Station, TX: Stata Press.

Buis, M. L. 2020. Stata tip 135: Leaps and bounds. *Stata Journal* 20: 244-249.

Cox, N. J. 2002. Speaking Stata: How to move step by: step. *Stata Journal* 2: 86-102.

---. 2009. Stata tip 79: Optional arguments to options. *Stata Journal* 9: 504.

---. 2023. Stata tip 151: Puzzling out some logical operators. *Stata Journal* 23: 293-297.

Cox, N. J., and C. B. Schechter. 2019. Speaking Stata: How best to generate indicator or dummy variables. *Stata Journal* 19: 246-259.

---. 2023. Stata tip 152: if and if: When to use the if qualifier and when to use the if command. *Stata Journal* 23: 589-594.

Daniels, L., and N. Minot. 2020. An Introduction to Statistics and Data Analysis Using Stata. *Thousand Oaks*, CA:Sage.

Kolev, G. I. 2006. Stata tip 31: Scalar or variable? The problem of ambiguous names. *Stata Journal* 6: 279-280.

Mitchell, M. N. 2020. *Data Management Using Stata: A Practical Handbook.* 2nd ed. College Station, TX: Stata Press.

Ryan, P. 2005. Stata tip 22: Variable name abbreviation. *Stata Journal* 5: 465-466.

12 データ

目次

12.1 データとデータセット

*データ*は長方形の表に数値や文字列を格納します。各行は全ての変数の観測値で、各列はひとつの変数(項目)に関する全ての観測値を入力します。変数は変数名（*variable name*）で割り当てられます。観測値は順番に1から_Nまで番号が割り振られます。以下のデータの例は最初の5つの奇数(odd)と偶数(even)をそれぞれ表記し、文字列変数も併記しています。

```
     odd   even    name
1.     1      2    Bill
2.     3      4    Mary
3.     5      6     Pat
4.     7      8   Roger
5.     9     10    Sean
```

この例では、観測値は1から5まで順に番号を振られ、変数はodd, even, nameと名前が付いています。観測値は順番に振られた数字、変数は名前で参照されます。

*データセット*はデータ、ラベル、形式、メモ、構造情報をあわせたものです。

*データ*と*データセット*に関する、全ての要素がここで定義されます。Long (2009)は長年のStataユーザによる苦労や経験を元に、正確かつ繰り返し行うためのデータ管理について、アドバイスを提供しています。Mitchell (2010)はStataによる数多くの例題を提供しています。

12.2 数字

数字は符号、整数部分、小数点、小数桁部分、eまたはE、符号がついた整数の指数部分から成り立ちます。数字はカンマを含みません。つまり、1,024という数字は1024（あるいは1024.や1024.0）として入力する必要があります。有効な数字の例は次の通りです。

```
5
-5
5.2
.5 5.2e+2
5.2e-2
```

❑ テクニカルノート

Stataでは16進数/2進数で数字を表記することもできます。

[+|-]0.0[〈ゼロ〉]{X|x}-3ff

または

[+|-]1.〈16進数〉[〈16進数〉]{X|x}[+|-]〈16進数〉[〈16進数〉]

先頭にある桁は常に0か1です。表現する数字がゼロのときのみ、0で表記します。16進の小数点の右側には、最大で13桁まで使用できます。指数は-3ffから+3ffまでサポートします。数字は16進数(基数16)の桁で表現されます。aX+bの数字は、a × 2^bをあらわします。例えば、1.0X+3は23または8です。1.8X+3は、1.816を小数で表記すると1 + 8/16 = 1.5となり、1.5 × 23 = 1.5 × 8 = 12が成り立つので12になります。

Stataはこの形式で数字を表示できます。詳細はこの後の[U] **12.5.1 数値形式**をご覧ください。たとえば、次のように実行します。

```
. display 1.81x+2
6.015625
. display %21x 6.015625
+1.8100000000000X+002
```

この16進数表記は数値解析の際に特に注目されます。

❑

12.2.1　欠損値

数値は、特殊な数値である欠損値（ピリオド(.)で表現）をとる場合もあります。欠損値は数字を指定できる箇所であるなら、どこでも指定できます。欠損値は一般的な数値とは1つ異なり、どのような算術演算を行ったとしても、結果として出力されるのは欠損値になります。

実際には、Stataでは27の欠損値を設定できます。今説明した「.」はその27個のうちの1つです。他に.a, .b, ..., .zまであり、これらは拡張欠損値です。「.」はデフォルトまたはシステム欠損値と呼ばれる欠損値です。ユーザの中には、この拡張欠損値を使用して、特定の値が不明である理由（例えば、質問を尋ねなかった、回答を得られなかった等）を示すこともあります。他のユーザは特に拡張欠損値を利用する必要が無いため、システム欠損値「.」のみを使います。

Stataのデフォルトあるいはシステム欠損値は、欠損値に算術演算を行った時や算術演算が定義されていない時(例えば、0で割るときや負の数を含む対数計算など)に出力されます。

```
. display 2/0
.
. list
```

	a
1.	.b
2.	.
3.	.a
4.	3
5.	6

```
. generate x = a + 1
(3 missing values generated)
. list
```

	a	x
1.	.b	.
2.	.	.
3.	.a	.
4.	3	4
5.	6	7

数値の欠損値は“大きな整数値”として表現されます。大きさの順番は次の通りです。

$$\text{全ての数字} < . < .a < .b < \cdots < .z$$

つまり、次のような表現の時は注意が必要です。

$$age > 60$$

これは、変数ageが60よりも大きい実測された数字か、全ての数字よりも大きい欠損値を表示します。同じように、

$$gender \neq 0$$

これは変数genderがゼロではない、あるいは欠損値の場合に真（true）となります。

欠損値を除外するには、「.」よりも小さい値を求めるようにしてください。また、同じように欠損値のみを探す方法は、「.」以上の値を求めることで表示できます。例えば、以下のようになります。

```
. list if age>60 & age<.
. generate agegt60 = 0 if age<=60
. replace agegt60 = 1 if age>60 & age<.
. generate agegt60 = (age>60) if age<.
```

❑ テクニカルノート

Stata 8より前のバージョンでは、欠損値として使用できるのはピリオド(.)のみでした。

古いプログラムやdoファイルを問題なく動作させるため、versionが8以前の場合は全ての欠損値を同じように扱うようになります。つまり、「. == .a == .b == .z」となることにより、‘exp==.’と‘exp!=.’は以前と同じように使用できます。

❑

▷ **例題1**

夫婦の収入に関するデータについて、夫の収入を変数hincomeにまとめ、妻の収入を変数wincomeにまとめました。listコマンドを入力して、データが含む情報を確認します。

```
. use https://www.stata-press.com/data/r18/gxmpl3
. list
```

	hincome	wincome
1.	32000	0
2.	35000	34000
3.	47000	.b
4.	.z	50000
5.	.a	.

変数wincomeの3番目と5番目の観測値は欠損値となっており、その値はゼロではない事が1番目の観測値から分かります。

hincomeとwincomeの合計を、generateコマンドを使用して新しい変数incomeを作成すると、3つの欠損値が作成されます。

```
. generate income = hincome + wincome
(3 missing values generated)
. list
```

	hincome	wincome	income
1.	32000	0	32000
2.	35000	34000	69000
3.	47000	.b	.
4.	.z	50000	.
5.	.a	.	.

generateコマンドは警告として3つの欠損値が作成されたことを示します。データをリストすると、47,000＋欠損値は欠損値を出力することが分かります。

◁

テクニカルノート

Stataは数値の欠損値を特定の保存形式内で最も大きい27個の数値として設定します。詳細は[U] 12.2.2 数値の保存形式をご覧ください。また、これには、2つの重要な意味が隠されています。1つ目は欠損値を有する変数をソートする場合、欠損値は最後に配置されます。なお、欠損値間のソート順は前に説明した欠損値の特性によります。

```
. sort wincome
. list wincome
```

	wincome
1.	0
2.	34000
3.	50000
4.	.
5.	.b

2つ目は関係演算子と欠損値に関する内容です。欠損値はどの数値よりも大きくなることを忘れないでください。

```
. list if wincome > 40000
```

	hincome	wincome	income
3.	.z	50000	.
4.	.a	.	.
5.	47000	.b	.

観測値4と5がリスト内にあるのは、‘.’ と ‘.b’はどちらも欠損値で、これは40,000よりも大きい値になるからです。関係演算子の詳細は[U] 13.2.3 関係演算子をご覧ください。

例題2

統計的な出力を行ううえで、Stataは欠損値を含む観測値を無視します。先ほどの例題1からの続きとなりますが、変数hincomeとwincomeの要約統計量をsummarizeコマンドで求めると、次のような結果を得ます。

```
. summarize hincome wincome
```

Variable	Obs	Mean	Std. dev.	Min	Max
hincome	3	38000	7937.254	32000	47000
wincome	3	28000	25534.29	0	50000

いくつかのコマンドでは、変数の中にあるデータがひとつでも欠損値の場合、全ての観測値(行)を使用しません（casewise deletionといいます）。correlateコマンドを変数hincomeとwincomeで使用すると、次のような結果になります。

```
. correlate hincome wincome (obs=2)
```

	hincome	wincome
hincome	1.0000	
wincome	1.0000	1.0000

相関係数は2つの観測値のみを元に計算されたことがわかります。

12.2.2 数値の保存形式

数値は、5つの保存形式、byte, int, long, float(デフォルト), doubleの中のひとつで保存されます。byteは1バイトで保存されます。intは2バイト、longとfloatは4バイト、doubleは8バイトで保存します。次の表はそれぞれの保存形式で保存可能な最大と最小の値を記します。

保存形式	最小	最大	限りなく0に近づく	バイト
byte	−127	100	±1	1
int	−32,767	32,740	±1	2
long	−2,147,483,647	2,147,483,62	±1	4
float	$-1.70141173319 \times 10^{38}$	$1.70141173319 \times 10^{38}$	$\pm 10^{-38}$	4
double	$-8.9884656743 \times 10^{307}$	$+8.9884656743 \times 10^{307}$	$\pm 10^{-323}$	8

integer（整数）という数の特徴を保存形式であるintと混同しないでください。例えば、数字の5はどの保存形式で保存されていても、整数であることは変わりません。つまり、ある数式が整数になると記載されていても、保存形式はintである必要はありません。

12.3 日付と時間

Stataには9つの日付、時間、日付と時間のエンコードがあり、これらを共に%t変数または値といいます。これらは、次の通りです。

```
%tC     カレンダーの日付と時間、閏秒で修正付き
%tc     カレンダーの日付と時間、閏秒を無視
%td     日
%tw     週
%tm     月
%tq     四半期
%th     半年
%ty     年
%tb     ビジネスカレンダー
```

%tyと%tb以外の日付と時間は0＝1960年の1月の開始の時、を元にしています。%tcと%tCは1960年1月はじめからミリ秒単位で値を記録しています。%tdは日数を記録します。他の変数は週、月、四半期、半年をそれぞれ記録します。%tyはシンプルに年を記録し、%tbはユーザ定義のビジネスカレンダー形式で記録します。

日付と時間を使用する際の操作に関する全容は[U] **25 日付と時間を使用する**をご覧ください。

12.4　文字列

このセクションではStataによる文字列の取り扱いについて説明します。このセクションは次のようなサブセクションに分割されています。

12.4.1　概要

文字列は連続した記号(文字)です。

```
Samuel Smith
California
U.K.
```

一般的に（ただし、必ずしもそうであるとは限りませんが）文字列はダブルクォーテーション（“”）に囲まれています。

```
"Samuel Smith"
"California"
"U.K."
```

クォーテーションに挟まれた文字列は「*string literals*（文字列リテラル）」と呼ばれます。

文字列はStataのデータセットに文字列変数として保存されます。

```
. use https://www.stata-press.com/data/r18/auto, clear
(1978 automobile data)
. describe make
```

Variable name	Storage type	Display format	Value label	Variable label
make	str18	%-18s		Make and model

文字列変数の保存形式はstr1, str2, ..., str2045とstrLがあります。例えば、変数makeはstr18型の変数です。18文字までの文字列を入力できます。文字列は全て18文字長でなければならないというわけではありません。

```
. list make in 1/2
```

	make
1.	AMC Concord
2.	AMC Pacer

str18型の変数には18バイトよりも長い文字列を格納できませんが、万が一それより長い文字列を入力したとしても、Stataが自動で必要な長さのstr#型へ変更するので、特に憂慮する必要はありません。

```
. replace make = "Mercedes Benz Gullwing" in 1
variable make was str18 now str22
(1 real change made)
```

このほかStataにおいて、データセットの情報を参照できるラベルやメモにも文字列を格納できます。詳細は[U] 12.6 データセット、変数、値ラベル、[U] 12.7 データに附属したメモをご覧ください。またStataプログラムでは、文字列スカラー、マクロ、構造情報、保存された分析結果の中にも文字列を格納できます。

Stataは文字列用の関数としてstrlen()やsubstr()を提供します。

```
. generate len = strlen(make)
. generate str first5 = substr(make, 1,5)
. list make len first5 in 1/2
```

	make	len	first5
1.	Mercedes Benz Gullwing	22	Merce
2.	AMC Pacer	9	AMC P

多くのStataコマンドでは文字列変数を使用できます。

```
. generate str brand = word(make, 1)
. tabulate brand
```

brand	Freq.	Percent	Cum.
AMC	2	2.70	2.70
Audi	2	2.70	5.41
BMW	1	1.35	6.76
Buick	7	9.46	16.22
Cad.	3	4.05	20.27
Chev.	6	8.11	28.38
Datsun	4	5.41	33.78
Dodge	4	5.41	39.19
Fiat	1	1.35	40.54
Ford	2	2.70	43.24
Honda	2	2.70	45.95
Linc.	3	4.05	50.00
Mazda	1	1.35	51.35
Merc.	6	8.11	59.46
Mercedes	1	1.35	60.81
Olds	7	9.46	70.27
Peugeot	1	1.35	71.62
Plym.	5	6.76	78.38
Pont.	6	8.11	86.49
Renault	1	1.35	87.84
Subaru	1	1.35	89.19
Toyota	3	4.05	93.24
VW	4	5.41	98.65
Volvo	1	1.35	100.00
Total	74	100.00	

Stata 14からは、文字列にUnicode文字を含めることができるようになりました。それらはUTF-8でエンコードされます。つまり、上記の文字のような標準ASCII文字(「下位ASCII」としても知られ、0から127までのコード値で保存される文字)はそのまま使い続けることができます。また、既存のファイル内に残るラテン文字、あるいは中国語、キリル文字、日本語におけるアルファベットも使い続けることができます。ただし、

データセット、doファイル、adoファイルにASCII文字以外が含まれる場合、特別な措置が必要になります。詳細は[U] 12.4.2 Unicode文字列の取り扱いをご覧ください。

12.4.2 Unicode文字列の取り扱い

もし標準ASCII文字以外のUnicode文字を使用していないのであれば、特に措置を講じる必要はありません。たとえ標準ASCII文字以外を使用している場合であっても、多くのケースでは、やはり特別な措置は必要ありません。常に成立する普遍的な規則は存在しませんが、一般的にそれらを切り分けるためのガイドラインとされるものはいくつか存在します。

その前に、基本として文字とバイトの違いを押さえる必要があります。文字とは、普段目にしているものです。例えば、“a”、“Z”、“@”などはすべて文字です。一方、バイトとは文字をエンコードしたもので、コンピュータ上にデータとして保存されるものです。

標準ASCII文字では、1バイトに収められた値と文字のコードとの間に1対1の対応がつきます。これに対し、UTF-8でエンコードされたUnicode文字は、1文字当たり2から4バイト分の情報を使用します。このため、Unicode文字を含む文字列には全てのUnicode文字が認識できる機能が必要となります。詳細は[U] 12.4.2.1 Unicode文字列関数をご覧ください。拡張ASCII文字として知られる文字が、14以前のStataで作成したファイルに含まれている場合、14では正しく表示されなかったり、期待通りの動作をしなかったりします。これらを回避するには、古いデータセットや、doファイルなどのテキストファイルに拡張ASCIIが含まれる場合、それらを適切に変換する必要があります。詳細は[U] 12.4.2.6 Stata 13以前のStataを使用していたユーザへのアドバイスをご覧ください。

データセットで標準ASCII文字以外を使用している場合、またはほかの人のためにコマンドを作成する場合、以下のセクションのご一読を推奨します。

12.4.2.1 Unicode文字列関数

Stataにおける文字列関数のいくつかはUnicodeを認識でき、文字列を1バイトずつデータの連続でなくUnicode文字の連続と認識することができます。時として、このUnicode対応の関数で正確な戻り値を得る必要があります。たとえば、make(製造元)のデータにClénet Coachworksで製造された車の情報が含まれた場合です。

文字列の長さを正確に知りたい場合、strlen()でなくustrlen()を使用することになります。前者の関数では18が得られるのに対し、後者では17が得られます。

Unicode対応の関数はほかにもあります。Unicode文字列を大文字化、または小文字化する関数（それぞれustrupper()とustrlower()）、あるいはタイトルに適するように変換する関数（ustrtitle()）などです。これから使おうとする関数に、Unicode対応した別名の関数があるかどうかを確認するには、[FN] 文字列関数を参照してください。

ASCII文字の範囲を超えるUTF-8文字が変数に含まれる場合には必ずしもUnicode対応関数が必要になる、という訳ではないのでご注意下さい。文字列の長さはともかく、“Mercedes”を“Merc.”に置き換えを行う場合には、usubinstr()ではなくsubinstr()を使用します。“Mercedes”と“Merc.”のどちらにもUTF-8文字は含まれないからです。

その他、Unicode対応の関数には表示桁数に関するものがあります。これは主にプログラミングでの使用を想定した関数です。詳細は[U] 12.4.2.2 Unicode文字の表示をご覧ください。

少しでも必要性を感じたり、ほかの人から普通に使用されるプログラムを作成したりする場合、Unicode対応関数が用意された関数に対してはUnicode対応関数を使用するようにしてください。Unicode対応関数は、元の関数名の最初にuを付けた関数名となっています。詳細は[FN] 文字列関数をご覧ください。

12.4.2.2 Unicode文字の表示

Stataでは、Stataの結果およびビューワウィンドウに表示される等幅フォントに関して、縦方向の整列を確実に行うため、*表示桁(display column)*という新たな概念を導入しました。文字がStataに映る際、1文字につき1または2の表示桁数が自動で割り振られます。

ほとんどの文字が1表示桁なため、ASCII文字以外のUTF-8文字をデータに含めている場合でも、文字数と表示桁に差が見られることはまずないことでしょう。ただ一部では、中国語、韓国語、日本語といった(CJK)文字のように1文字で2表示桁分を使用する文字もあります。

表示桁数は取得が必要な場面が時に存在します。Unicode文字を認識できる関数と同様、表示桁数を認識できる関数が用意されています。それらは“ud”という接頭語付きの関数名になっています。例えば、udstrlen(string)という関数を用いて表示桁数を取得できます。ある文字列の先頭から10表示桁分を抜き出すには、udsubstr(string,1,10)を使用できます。詳しくは[FN] **文字列関数**をご覧ください。

12.4.2.3 エンコード

エンコード（エンコーディング）は、コンピュータが文字列を保存するときに行う方法です。Stataがテキストを保存する際に用いるASCII やUTF-8は、エンコードの中の一形式です。標準ASCII文字は1文字を0から127までの値に直し、それぞれ1バイトで保存します。“a”、“Z”、“@”は、標準ASCII文字で、バイト値はそれぞれ97、90、64です。

一方、“a´”も一つの文字です。UTF-8エンコードでは、この文字が2バイトで保存されます（195と161)。Unicodeでは標準ASCII外の文字は全て2バイト以上で保存され、各バイト値は128から255までの値をとります。一部の文字は3または4バイトを使います。

あらゆるバイト値の組み合わせ全てに対し、対応するUnicode文字がある訳ではありません。Unicode文字は2バイト以上で保存されるので、たとえば128以上255以下の値を持つ1バイトのみに対応する有効なUnicode文字はありません。14以前のStataで作成したファイルに拡張ASCIIが含まれる場合、それらのいくつかはUnicode文字として無効なバイトデータの塊りであることが見込まれます。詳細は[U] **12.4.2.6 Stata 13以前のStataを使用していたユーザへのアドバイス**をご覧ください。

ほかのエンコードで保存されたテキストがStataファイルに含まれる場合、正しく表示したり、一部の関数を正しく動かしたりするためにはそれらのファイルをUTF-8に変換する必要があります。ファイルのUTF-8への変換には、元のエンコード形式を知る必要があります。最も広く用いられていると考えられるのはWindows-1252です。よく用いられているエンコード、あるいはStataが変換可能なエンコードの一覧については、[D] **unicode encoding**のunicode encoding listまたはunicode encoding aliasセクションを参照してください。、

The unicode analyze、unicode translateというコマンドはテキストファイルやデータセットの変換に役立ちます。詳細は[D] **unicode translate**をご覧ください。合わせて、[U] **12.4.2.6 Stata 13以前のStataを使用していたユーザへのアドバイス**もご覧ください。

12.4.2.4 Unicodeのロケール

ロケールは、言語圏内を慣習ごとに分けた共同体の識別に用います。“en”で英語を示すときのように一般的な一言語を示したり、“en_US”や“en_HK”で英語（米国）や英語（香港）を示すときのように言語圏内の特定の国・地域を示したりします。

ロケールでは、*タグ(tag)*を使って特殊言語としての情報を定義します。この*タグ*にはlanguage、script、country、variant、keywordsの各タグが含まれます。典型的には、languageが必須で、ほかのタグは任意です。多くの場合、Stataで使用するタグはlanguageとcountryです。たとえば、“en_US”はlanguageを英語に、countryを米国に指定します。

言語に依存する特定の演算を正しく行うためには、ロケールが適切に設定される必要があります。例えば、英語では“i”の大文字は“I”です。トルコ語では“i”の大文字は“İ”（すなわち“I”の上にドットがある文字、Unicode文字は¥u0130）です。大文字への変換法を指定するには、たとえばustrupper("i", "en_US")のように、関数ustrupper()の中でロケールを指定する必要があります。

ロケールに依存する関数は、ustrupper()、ustrlower()、ustrtitle()、ustrword()、ustrwordcount()、ustrcompare()、ustrcompareex()、ustrsortkey()、およびustrsortkeyex()です。

上記の関数の中で、特に明示的にロケールを指定しなかった場合、Stata内のlocale functionsの現在値が用いられます。この現在値は、以下のコマンドで確認できます。

```
. display c(locale_functions)
```

および

```
. unicode locale list
```

2つ目のコマンドでは、サポートされるロケールの一覧を表示します。しかし、set locale_functions設定を変更する必要がある場合は稀です。

デフォルト値の決まり方を含むロケール設定の詳細は[P] set locale_functionsをご覧ください。

12.4.2.5 Unicode文字を含む文字列のソート

このセクションは、Unicode文字を含む文字列のソートである*照合(collation)*と、その基になる規則について説明します。多くの方にとって、このセクションをスキップしても問題ないセクションです。

ASCII以外のUnicode文字を使用しない方は、このセクションをスキップしても問題ありません。また、ほかのコマンドやプリフィックスを使用するためだけにsortを用いる方も、このセクションをスキップしても問題ありません。この場合にはたとえば、Unicode文字を含む変数idについて、以下のような使い方をする場合が当てはまります。

```
. statsby id: regress y x1 x2
```

このように、変数idを係数を求めるためのグループ分けにのみ利用し、グループ順を問題にしない場合、このセクションはスキップして構いません。通常のsortで十分対応できます。

データをグループ分けするだけであれば、このセクションの手順は適用する必要がありません。例えば、2つのデータセットに対し、mergeにて変数idを基準とした1対1対応で統合を行う場合です。これを行うには、次のように入力します。

```
. merge 1:1 id using ...
```

最後に、データ内のUnicode文字に言語に特有な規則を適用しない方も、このセクションをスキップしても問題ありません。例えば、“café”と“cafe”の区別は必要であっても、どちらが先に並ぶかは問題としない場合です。

私たちが普段行う方法で文字列をソートするには、次の4つの規則を踏まる必要があります。

1. ソートはロケールに依存する。
2. ソートキーを生成する必要がある。変数自体によるソートはできない。
3. Unicode文字列を操作する設定は複数ある。
4. varlistによるソートでは、文字列の連結が必要になる。

文字列の比較には、1番目と3番目の規則が必要になります。各規則について、以下で詳述します。まず、ソートがどのように行われるかを把握しておきます。

Stataのコマンドsortと論理演算子>、<は文字のバイト値を基準に並び順を決定します。例えば、“a”のバイト値は97であり、“A”のバイト値は65であるので、“a” > “A”となります。同様に、バイト値が90の“Z”は、“a” > “Z”となります。つまり、“Z”から始まる文字列は、“a”から始まる文字列よりも前に並ぶという、英語の辞書とは異なる順番になります。

別の例として、以下のデータがある場合を考えます。

```
. list mystr
```

	mystr
1.	Quick
2.	quick
3.	brown
4.	Fox
5.	jump

上のデータをsortし、その後listすると、次のような結果を得ます。

```
. sort mystr
. list
```

	mystr
1.	Fox
2.	Quick
3.	brown
4.	jump
5.	quick

上記は通常みられない並びです。

私たちが普段用いる順番でmystrをソートするには、Unicode照会アルゴリズム(Unicode collation algorithm(UCA))と呼ばれるUnicodeツールを使用して、言語の特性を踏まえた方法を用います。ロケールで、大文字小文字の別や発音区別符を考慮に入れるかどうかを指定することで、UCAにより普段の方法に近い順番で並べ替えができます。

StataとMataでは、ustrcompare(), ustrcompareex(), us trsortkey(), ustrsortkeyex() の各関数でUCAの設定をします。Stataでは、さらにcollatorlocale()、collatorversion()でも設定を行えます。

公式なUCA仕様についてはhttp://www.unicode.org/reports/tr10/をご覧ください。

規則1：ソートはロケールに依存する。

Unicode文字の並び順は言語、およびその言語特有の任意のタグ、keywordsに依存して変化します。

関数ustrcompare()、ustrsortkey()では、何も指定しなければ現在の言語のデフォルトの規則が適用になります。locale_functionsの現在値を利用するか、または関数内で指定することにより個別にロケールを指定するかは自由に選択できます。ロケールについての詳細は[U] 12.4.2.4 Unicodeのロケール、ロケールに依存した照合については、[D] unicode collator をご覧ください。

並び順に関する上級設定は、関数ustrcompareex()、ustrsortkeyex()で行います。これらの関数は、大文字小文字や発音区別符による並び順の違いを細かく調整する照合キーワードを指定できます。“pinyin”と“stroke”というキーワードを例にとります。

中国語において、これらのキーワードを設定したときの並び順は異なります。有効なキーワードとその説明については、http://unicode.org/repos/cldr/trunk/common/bcp47/collation.xmlを参照ください。

規則2：ソートキーを生成する必要がある。

ロケールにより適用される規則を全て活用して適切なデータのソートを行うには、*ソートキー*を作成する必要があります。ソートキーは、UCAにより作成され、Unicode文字列のソートに使用される文字列です。Unicode文字列のソートは、それ自体でなくソートキーに対して行います。ソートキーは可読性がなく、データ管理以外の目的で使用されません。

ソートキーの作成はustrsortkey()、またはustrsortkeyex()のどちらかで行います。作成後、ソートキーを収める変数を基にデータをsortします。sortによる結果と、上記の関数を用いたUnicode照合による結果の違いを以下に例示します。

```
. generate sortkey = ustrsortkey(mystr, "en")

. sort sortkey

. list mystr
```

	mystr
1.	brown
2.	Fox
3.	jump
4.	quick
5.	Quick

上記では、mystrが適切にソートされていますが、この並べ替えはmystr自身でなく、sortkeyに対するソートで行われています。Stataが認識しているのは、sortkeyによるソートのみです。すなわち、byのようにソート順に依存する関数を用いる場合、mystrでなく、次のようにsortkeyを用います。

```
. by sortkey:...
```

ロケールの設定あるいはustrsortkeyex()による上級設定により作成されるソートキーは、ロケール間や上級設定間で互換性がなく、再利用できません。例えば、"en"というロケールの下で作成したソートキーと、"fr"の下で作成したものとを比較することに意味はありません。

❑ テクニカルノート

実際に用いられたロケールが、想定とは異なる場合があります。ロケールを指定しない場合、異なるコンピュータ、ユーザーアカウントからは異なるソートキーが生成する場合があります。次に実際に用いられるロケールは関数collatorlocale()で取得できます。

❑

❑ テクニカルノート

Unicode標準は常時新たな文字、言語規則を追加および変更を行っており、現在のバージョンのUCAによるソートキーが、ほかのバージョンのUCAによるソートキーと異なる場合があります。

今後の作業のために、関数collatorversion()から、照合処理の現在のバージョンを取得し、作成したソートキーと共に（例えば変数のメモに）保存しておくとよいでしょう。

保存したソートキーを作成したバージョンが、現在のものと異なる場合、最新のバージョンでの結果に刷新または、新旧の違いを比較するには、現在のバージョンで新たなソートキーを作成してください。

❑

規則3：Unicode文字列を操作する設定は複数ある。

UCAは一見直感的ですが、細かく見てみると驚くべき性質が隠れています。以下のような、文字列の比較するケースで考えます。

```
. display ustrcompare("café","cafe","fr")
1
```

上記では、仏語ロケール("fr")を用いて文字列“cafe´”と“cafe”を比較します。Stataが返す値は、“cafe´”が“cafe”より大きいことを意味する1です。データのソートでは、“cafe´”が“cafe”の後ろに並ぶことになります。

次に、以下を考えます。

```
. display ustrcompare("café du monde","cafe new york","fr")
-1
```

上記で、“café du monde”が“cafe new york”より小さいことを意味する-1が返されたことに驚かれるのではないでしょうか。

この結果の原因は、2単語目の頭文字“d”と“n”の違いがUCAにおいて*第一段階の違い*とされるのに対し、1単語目の最後の文字“é”と“e”の違いが*第二段階の違い*とされるからです。第一段階の違いのほうが後にあるにも関わらず、第二段階の違いよりも優先されます。

ほとんどの場合、関数ustrcompare()、ustrsortkey()のデフォルトの設定を変更する必要はありません。第一段階から第四段階までの違いを基準に並べ方を変えるかなど、Unicodeの並べ方に関する上級設定は、ustrcompareex()、ustrsortkeyex()で行います。詳細は[FN] String functionsをご覧ください。

規則4：varlistによるソートでは、文字列の連結が必要になる。

Unicode文字列のソートキーを作成する際、規則3を適用することにより重要な影響が表れます。通常の場合、2つの変数を基準としたソートを行うには、単純に次のようにします。

```
. sort string1 string2
```

しかし、UCAの機能を活かして2つ以上の変数のソートを行うには、まず変数を結合し、その後ソートを実施します。

```
. generate  string3  =  string1  +  string2
. generate sortkey = ustrsortkey(sting3, "fr")
. sort sortkey
```

上記を行わずにソートを実施すると、2つ目の変数string2にある第一段階の違いが、1つ目の変数string1にある第二段階の違いに対し、優先されません。

12.4.2.6　Stata 13以前のStataを使用していたユーザへのアドバイス

このセクションでは、Stata 13などの以前のStataによる古いファイルを最新式のStataで開くことや、以前のStataとのファイルの共有について説明します。

13以前のStataでは、Unicodeはサポートしていませんでした。データセット、doファイル、adoファイルで標準ASCII文字のみを使用する場合、最新式のStataで使い続けるためにファイルに対して特別な措置を講じる必要はありません。また、saveoldで以前のフォーマットでのファイル保存も可能です。doファイル、adoファイルはそのまま共有できます。

13以前のStataで用いたファイルに拡張ASCII文字が含まれる場合、最新式のStataで使い続けるにはUTF-8でそれらの文字列を変換する必要があります。まず、unicode analyzeコマンドでファイルに変換が必要か分析し、もし必要ならばunicode translateでUTF-8エンコードへ変換します。詳細は[D] **unicode translate**をご覧ください。文字列変数を変換するには、ustrfrom()を用います。

Unicode文字を含むファイルを13以前のStataのユーザと共有したい場合、saveoldによる旧フォーマットでの保存はできますが、Unicodeを認識できない以前のStataでは正しく文字が表示できない場合があることにご注意ください。saveoldを実行する前に、ustrto()で文字列変数の拡張ASCIIエンコードからUTF-8エンコードに変換してください。ustrfrom()やustrto()の実行にあたっては、generateで新たな変数を作成し、既存変数のreplace前後で適切な変換ができたかどうか比較して確認することを推奨します。ustrfrom()とustrto()はMataの文字列行列にも使用できます。

12.4.3　識別データを有している文字列

文字列変数は多くの場合、患者名、町名、州名などの識別情報を含んでいます。これらの文字列は一般的にリストされていますが、直接統計的な解析で使われることはほとんどありません。しかし、データは文字列でソートしたり、文字列の情報を元にデータを融合することもできます。

12.4.4　カテゴリデータによる文字列

いくつかの文字列は直接分析で使用する情報、例えば患者の性別（“male”と“female”）などの情報を含むこともあります。Stataはこのような情報を数値にエンコードして、数値として保存する事を推奨しています。Stataの統計ルーチンは文字列を欠損値のように扱います。Stataは文字列を数値のコードとして変換したり、コードを文字列に変換し直すコマンドとしてencodeとdecodeの2つを提供しています。 詳細は[U] **24.2 カテゴリカルな文字列変数**と[U] **11.4.3 因子変数**をご覧ください。

12.4.5　数値データを有している文字列

もし、文字列が数字を表す記号や文字を有している場合、例えば変数myvarが“1”, “1.2”, “-5.2”を含んでいるとき、これらの値は real()関数かdestringコマンドを使用すると数値に変換できます。たとえば、次のように入力します。

```
. generate newvar = real(myvar)
```

数値の表記を文字列に変換したい場合は、string()関数かtostringコマンドを使用してください。例えば、次のように入力します。

```
. generate as_str = string(numvar)
```

詳細は[FN] **String functions** 、[D] **destring**をご覧ください。

12.4.6 文字列リテラル

文字列リテラルは印刷可能な文字のまとまりがクォーテーションで挟まれている状態です。クォーテーションは文字列の一部分であると認識されません。クォーテーションは文字列の最初と最後を示す、区切りの役割を担っているだけです。文字列リテラルの例は次の通りです。

```
"Hello, world" "String" "string"
" string" "string " ""
"x/y+3" "1.2"
```

上記の文字列の例は全て異なります。大文字・小文字の区別も重要です。文字列の最初と最後のスペースも重要になります。また、“1.2”はクォーテーションで囲まれているので、数字ではなく文字列になります。

Stataを使用していて、クォーテーションで文字列を区切ることができない、という事はありません。ですが、データを入力する時のように、クォーテーションなしで入力ができる事もあります。クォーテーションによる区切りのない文字列は、単語の前後についているスペースで区切られます。クォーテーションで区切られている文字列はそのまま使用されます。

上記のリストは、次のように書き換えることもできます。

```
‘"Hello, world"’
‘"String"’
‘"string"’
‘" string"’
‘"string "’
‘""’
‘"x/y+3"’
‘"1.2"’
```

‘"と"’は合体(compound)ダブルクォーテーションと呼ばれます。

合体ダブルクォーテーションは、クォーテーションそのものを含む文字列を入力する方法です。

```
‘"Bob said, "Wow!" and promptly fainted."’
```

合体ダブルクォーテーションには、さらに合体ダブルクォーテーションを含ませることもできます。

```
‘"The compound quotes characters are  ‘" and "’ "’
```

12.4.7 str1-str2045とstr

strはgenerateコマンドと共に使用します。これに関する解説は後ほど行います。

str1からstr2045はStataの長さ固定の文字列保存形式です。

長さ固定の由来は、データセット内で変数がstr#として保存される時に、各観測数で#バイトの保存領域が必要だからです。#の値を必要以上に大きくするのは得策ではありません。Stataのcompressコマンドは必要以上に長いstr#文字列を短くします。

```
. use https://www.stata-press.com/data/r18/auto, clear
(1978 automobile data)
. compress
  variable mpg was int now byte
  variable rep78 was int now byte
  variable trunk was int now byte
  variable turn was int now byte
  variable make was str18 now str17
  (370 bytes saved)
```

[U] 12.4.1 概要では、generateコマンドとstrを同時に使用した例を確認できます。

```
. generate str brand = word(make, 1)
```

strはgenerateコマンドと共に使用し、generateコマンドに必要最低限の長さを持ったstr#を作成するように指示します。上記例の出力からは確認できませんが、generateコマンドは変数brandをstr7保存形式の変数として作成しました。

Stataは必要に応じて自動でstr#保存形式を調整します。

```
. replace make = "Mercedes Benz Gullwing" in 1
variable make was str17 now str22
(1 real change made)
```

実際には、保存すべき文字列が2,045バイトを超える場合、generateとreplaceコマンドはさらに多くの文字を格納できる、strL形式を使用します。strLについては次のセクションを確認してください。

12.4.8 strL

strL変数は0から20億バイトまでの文字列を格納できます。

「L」はlongのLで、「スタール」と発音します。

strL変数は2,045バイトを超える長さである必要はありません。

str#変数は2,045バイトまでの文字列を格納するので、strLとstr#は重なります。この重なりは数値の保存形式でのintとfloatの重なりと同じです。intとして保存できる数値はfloatとして保存できるのと同じように、str#として保存できる文字列はstrLとして保存できます。逆の場合は成立しません。さらに、strL変数はバイナリ文字列を格納できますが、str#変数はテキスト文字列のみ格納します。つまり、str#/strLとint/floatの保存形式同士の関係は同一であるといえます。ところにより、str#変数ではなく、strL変数を使用するほうが良い場合もあります。これは、int変数よりもfloat変数を使用する場合と同じような状況です。

strL変数はstr#変数と同じように動作します。次の例は[U] 12.4.1 概要で行ったことの繰り返しですが、strL変数を使ってみます。

```
. use https://www.stata-press.com/data/r18/auto, clear
(1978 automobile data)
. generate strL mymake = make
. describe mymake
                        Storage   Display    Value
Variable name   type    format    label      Variable label
------------------------------------------------------------------------------
mymake          strL    %9s
. list mymake in 1/2

     +-----------------+
     |         mymake  |
     |-----------------|
  1. | AMC Concord     |
  2. |   AMC Pacer     |
     +-----------------+
```

str#の値を置き換えたように、strLの値も置き換えることができます。

```
. replace mymake = "Mercedes Benz Gullwing" in 1
(1 real change made)
```

str#変数で行ったように、strL変数を使用して関数を実行できます。

```
. generate len = strlen(mymake)
. generate strL first5 = substr(mymake, 1, 5)
. list mymake len first5 in 1/2
```

	mymake	len	first5
1.	Mercedes Benz Gullwing	22	Merce
2.	AMC Pacer	9	AMC P

テーブルを作成することも可能です。

```
. generate strL brand = word(mymake, 1)
. tabulate brand
```

brand	Freq.	Percent	Cum.
AMC	2	2.70	2.70
Audi	2	2.70	5.41
BMW	1	1.35	6.76
(省略)			
Volvo	1	1.35	100.00
Total	74	100.00	

次の項目のみ、Stataには制限があるのでご注意ください。

1. 2つのデータセットを統合するときのマッチングキーとしてstrL変数の値を使用することはできません。
2. strL変数はfillinコマンドとあわせては使用できません。

strL変数はstr#変数とは異なる方法で保存されます。str#変数は観測値ごとに#バイトの保存領域が必要になります。strL変数は各観測値の文字列ごとに実際に使用するバイト数が必要になります。つまり、文字列のバイト数が少ない場合はstr#の保存形式よりもstrLの形式のほうが実際に使用するメモリが少なくなります。しかし、ほとんどのstrLは80バイトのオーバーヘッドが必要になります。ただし、空欄の文字列の場合は8バイトになります。

strLと　str#のどちらでも、実際にどちらのほうが保存する際にメモリの使用量が少なくなるかは、各文字列に依存します。compressコマンドを使うと、それを確認できます。

```
. compress
  variable mpg was int now byte
  variable rep78 was int now byte
  variable trunk was int now byte
  variable turn was int now byte
  variable len was float now byte
  variable make was str18 now str17
  variable mymake was strL now str22
  variable first5 was strL now str5
  variable brand was strL now str8
  (12,420 bytes saved)
```

compressコマンドはメモリの使用量を抑えるために全てのstrL変数をstr#の値に変換しました。

しかし、compressコマンドはメモリの使用量を抑えるとしても、str#変数をstrL変数には変換しません。これは、strLの保存形式ではできないことがstr# では行えるためです。

str#をstrLに変換するには、recastコマンドを使用してください。

```
. * variable make is currently str17
. recast strL make
. describe make
              Storage   Display    Value
Variable name   type    format     label      Variable label
-------------------------------------------------------------------------------
make            strL    %-9s                  Make and model
. compress make
  variable make was strL now str17
  (5,607 bytes saved)
```

12.4.9　strL変数と複製値

実際にユーザが気にする必要はありませんが、strL形式で保存している同一の値が複数の観測値にある場合、Stataは1つの値のみを保存して他の観測値にはその保存データをコピーします。これは「コレーシング」と呼ばれ、メモリの使用を抑えることができます。

Stataは変数作成時に自動的にほとんどのstrL変数のコレーシングを行いますが、時として、同一の値があることを見逃す場合があります。compressコマンドを実行すると、Stataはコレーシングを行える箇所を探します。次のように確認できます。

```
. compress x
x is strL now coalesced
(11,301,687 bytes saved)
```

上記の内容もあり、strL変数がある時はcompressコマンドを適宜使用することをお勧めします。

12.4.10　strL変数とバイナリ文字列

strLはバイナリ文字列を格納できます。バイナリ文字列とは、技術的な話をすると、バイナリ0を有している文字列を指します。少々おかしいシチュエーションになりますが、状況を説明するために例題を紹介しましょう。

```
. use https://www.stata-press.com/data/r18/auto, clear
(1978 automobile data)
. replace make = "a" + char(0) + "b" in 1
  variable make was strL now str17
(1 real change made)
. list make in 1
```

	make
1.	a¥0b

listコマンドはバイナリゼロを¥0として表示します。

同じ実験を、変換を阻害するためにnopromoteオプション付きでstr#変数で行います。すると、先ほどとは異なる結果になります。

```
. use https://www.stata-press.com/data/r18/auto, clear
(1978 automobile data)
. replace make = "a" + char(0) + "b" in 1, nopromote (1 real change made)
. list make in 1
```

make
a 1.

str#文字列にとって、バイナリ0は文字列の終点を意味します。つまり、最初の観測値には「a」が格納されています。

str#変数はバイナリ0を格納することはできませんが、strLなら可能です。

compressコマンドはこの条件より、バイナリ0がある時はstrL形式を使用します。最初の例でcompressを入力した場合、compressコマンドは変数makeをstr#形式に変更しません。これは、1つの値がstr#として保存できないためです。このcompressコマンドの挙動は1つの値が1.5であることを理由にfloat形式をint形式に変換しないのと同じ理由です。

12.4.11 strL変数とファイル

strLで行うことができる便利な機能のひとつに、ファイルの内容物を格納できることがあります。例えば、10人の患者のデータがあります。いくつかのデータは担当医のメモのコードと紐付けられています。これらのメモは、notes_2217.xyz, notes_2221.xyz, notes_2222.xyzのように保存されています。この場合、次のように入力できます。

```
. generate strL notes = fileread("notes_2217.xyz") in 1
. replace notes = fileread("notes_2221.xyz") in 2
. replace notes = fileread("notes_2222.xyz") in 3
. ...
```

もっと簡単にデータを入力するには、次のように入力します。

```
. generate str fname = "notes_" + string(patid) + ".xyz"
. generate strL notes = fileread(fname)
```

元のファイルはStataに保存されている情報から再作成する事もできます。全てのファイルを再作成するには、次のように入力しましょう。

```
. generate len = filewrite(fname, notes)
```

「Diabetes Mellitus Type 1」という言葉がメモ内にあるか、また担当医が病名としてT1DMを記録しているか、確認するには次のように入力します。

```
. generate t2dm = (strpos("notes", "T1DM")) != 0
```

もちろん、この場合notes_*.xyzファイルがテキスト形式、あるいはテキストのような形式であり、T1DMが“T1DM”と表示されるかどうかにもよります。

なお、strpos()をはじめとしたStataの文字列の関数はバイナリ文字列もサポートしています。

12.4.12　文字列表示形式

文字列を表す形式は%[-]#s、例えば%18sや%-18sとなります。#は2,045までの数字を入れることができます。#は入力範囲の幅を指定します。%#sは右寄せで範囲内に表示します。また、%-#sは文字列を左寄せで表示します。

Stataはstr#変数を使用する際に適切な形式を指定します。デフォルト形式は%#sなので、変数がstr18の場合、デフォルトの形式は%18sとなります。

StataはstrL変数を使用する際には常に固定の形式、%9sを使用します。strL変数は必要なだけの長さになるため、どの形式が良いのかはそれぞれ異なります。つまり、質問はどれほどの長さの文字列を表示したいか、に変わります。

形式が文字列の長さに対して短すぎた場合、文字列がstr#でもstrLの場合でも、Stataは通常# − 2の文字数を表示し、最後に2つのドットを表示します。「通常」という言葉を使用するのは、いくつかのコマンドはこの方法と異なる事があるためです。

12.4.13　strLまたはstr#変数の内容を全て見るには

デフォルトで、listコマンドは長い文字列の最初の部分のみを表示して、その後に2つのドットを記します。listコマンドで表示される文字数は結果ウィンドウの幅によります。

listコマンドは、notrimオプションを使用すると、長い文字列（strLやstr#）の最初の2,045バイトを表示します。

```
. list, notrim
                (省略)
. list mystr, notrim
                (省略)
. list mystr in 5, notrim
                (省略)
```

長い文字列を表示するもう1つの方法はdisplayコマンドです。displayコマンドでは、全内容を確認できます。mystr変数で5番目の観測値を表示するには、次のようにします。

```
. display _asis mystr[5]
                (省略)
```

このコマンド1つで、とても長い文字列の場合、数百ページにも及ぶページが出力される事もあります。なお、このリストを中断するには、*Break*を行います。

最初の5,000文字を表示したい場合、次のように入力しましょう。

```
. display _asis usubstr(mystr[5], 1, 5000)
```

標準ASCII以外のUnicode文字の表示は、[U] 12.4.2.2 Unicode文字の表示をご覧ください。

レアなケースですが、文字列はSMCL出力結果を有していることもあります。SMCLはStataのテキストマークアップ言語です。変数がSMCLを有している可能性があるのは、fileread()を使用してStataのログファイルを読み込んだときのことです。その場合、テキストが正しく形式化されているか確認するには次のように入力します。

```
. display as txt mystr[1]
                (省略)
```

displayコマンドのほかの機能については、[R] displayをご覧ください。

12.4.14 プログラマ向けの注意点

マクロの最大長よりもstrLのほうが長くなっています。つまり、次のことが言えます。

1. マクロで使用した文字列表現は途中で切断される恐れを考慮する必要はありません。
2. 拡張マクロはクォーテーション−‘″‘macname’″’−で囲み、文字列リテラルを作成することで、途中で分断される危険を回避します。
3. マクロはバイナリ文字列を含むことはできません。バイナリ文字列を使用する場合は、strLの一機能として準備されている文字列スカラーを使用してください。詳細は[P] scalarをご覧ください。
4. 文字列表現の結果が必ずしもマクロに入るというわけではありません。確実に入力される自信がある場合は、そのまま結果をマクロとして保存してください。自信が無い場合は文字列スカラーを使用して、strLを組み込むようにしましょう。
5. strL変数の内容が必ずしもマクロに入るというわけではありません。代わりに、文字列スカラーを使用してください。
6. プログラミングでは、文字列スカラーは数値のスカラーと同じように使用します。

```
program ...
        version 18.0
        ...
        tempname mystr
        ...
        scalar ‘mystr’ = ...
        ...
        generate ... = ...‘mystr’...
        ...
end
```

上記コードの「mystr」はテンポラリネームを有しているマクロです。‘mystr’は表現ではなく、文字列スカラーの参照先となります。

12.5 形式：データ表示をコントロールする

形式は数値あるいは文字がどのように表示されるか指示します。例えば、325.24という数字があります。これをどのように表示すればいいでしょうか。325.2、325.24、325.240、3.2524e+02、3.25e+02、あるいはこれ以外の表示方法でしょうか。*display format*(表示形式)はStataにこのようなデータをどのように表示するか指示します。常にどのような形式で表示するか指定する必要はありません。Stataはある程度推測を行い、一般的に使われている表示形式を選択します。しかし、ユーザは常に形式を指定することができます。

12.5.1 数値形式

Stataの数値形式は次のように作成されています。

1番目に入力	%	形式使用を開始
任意入力	−	左側に揃える
任意入力	0	小数点前のゼロを表示(1)
2番目に入力	数字w	値の幅を指定
3番目に入力	.	
4番目に入力	数字d	数字で表示する桁数を指示して小数点を続ける
5番目に次のいずれかを入力		
	e	科学的表記、例えば1.00e+03を示す
	f	固定表記、例えば1000.0を示す
	g	一般表記、Stataが表示する桁数を自動的に決定
任意入力	c	カンマ形式を使用(eとは併用不可)

(1) 0を指定して「小数点前のゼロを表示」出来るのはf形式のみです。

例えば、次のように表現されます。

```
%9.0g     一般形式、9列の幅
                                  sqrt(2) =     1.414214
                                    1,000 =         1000
                      10,000,000 =    1.00e+07
%9.0gc    一般形式、9列の幅、カンマ付き
                                  sqrt(2) =     1.414214
                                    1,000 =       1,000
                      10,000,000 =    1.00e+07
%9.2f     固定形式、9列の幅、小数点以下2桁
                                  sqrt(2) =          1.41
                                    1,000 =       1000.00
                     10,000,000 = 10000000.00
%9.2fc    固定形式、9列の幅、小数点以下2桁
                                  sqrt(2) =          1.41
                                    1,000 =     1,000.00
                     10,000,000 = 10,000,000.00
%9.2e     指数形式、9列の幅
                                  sqrt(2) =     1.41e+00
                                    1,000 =     1.00e+07
                      10,000,000 =    1.00e+07
```

Stataは、e, f, gの3つの数値形式を有しています。それぞれの形式は直前にあるパーセント記号(%)で表記を開始します。その後にw.*d*の値が入力されます。wとdはそれぞれ整数を表します。最初の整数、wは形式の幅を指定します。2つ目の整数*d*は小数点以下何桁まで表示するか指定します。dはwよりも小さい数字である事が必須です。最後に、形式のタイプを指定する文字（e, f, g）を入力します。またオプションとしてcを追加で入力すると、結果の数値にはカンマが含まれる形になります（cはeと共に使用することはできません）。

デフォルトでは、全ての数値は%w.0g形式が設定されていて、wはそのときの変数の最大の値を表示できる形式になっています。%w.0g形式はフォーマットに関するルールの中で、値の精度を損なうことなく、可能な限り読みやすい形式にしたものです。g形式は数値の小数点の表示位置を変更して、現在の値の読みやすさを改善します。

各数値の保存形式におけるデフォルトの形式は次のようになります。

```
                    byte      %8.0g
                    int       %8.0g
                    long      %12.0g
          float     %9.0g           double  %10.0g
```

値の形式を変更するにはformat varname %fmtコマンドを使用してください。

%w.0gに追加して使用できるものは、%w.0gc形式で、これを使用するとカンマ付き形式で表示されます。千は、%9.0g形式では「1000」となり、%9.0gc形式では「1,000」となります。

%w.0gや%w.0gcを使用する代わりに%w.dgと%w.dgc（この場合d > 0が成り立つ必要があります）を使用することができます。例えば、%9.4gと%9.4gcがあります。この数字の4はおおよそ有効桁数が4であることを示しています。例えば、3.14159265を%9.4g形式で表示すると3.142、31.4159265は31.42、314.159265は314.2、3141.59265は3142となります。この形式は厳密に言うと有効桁数ではないのは、31415.9265は31416となり、3.142e+04と表示されないためです。

f形式を選択すると、値は全て同じだけの小数点以下の値で表示されます。これは、小数点の精度が欠けることになっても、桁数は変わりません。f 形式はC f形式と同じような形式です。Stataのf形式はFortran F形式と類似しています。しかし、Fortran F形式と異なるのは数値が大きすぎて指定されたf形式では表示できないときにg形式に変更できる点です。

%w.dfに追加して、%w.dfcの値はカンマ付きで数値を表示します。

e形式はC eあるいはFortran Eと類似した形式です。各値が先行桁と共に表示され(必要に応じて、マイナス記号が付きます)、その次に小数点を表示してから指定した桁数が続きます。数字の後に文字eに続けてプラスまたはマイナスの記号と共に10の指数を表示(前に附属する符号で変更)されるので、その値を表示する値に掛けます。e形式が指定される場合、幅は小数点の後に続く桁数を最低でも7桁多く指定する必要があります。これは先行する符号や桁、小数点、e、10の指数を表示するためです。

例題3

以下に、5つの観測値を有している3つの変数、e_fmt、f_fmt、g_fmtがあります。3つの変数には全て同じ値が入力されています。違いは表示形式のみです。describeコマンドは数値の表示形式を変数の保存形式の右側に示します。

```
. use https://www.stata-press.com/data/r18/format, clear
. describe
Contains data from https://www.stata-press.com/data/r18/format.dta
Observations:             5
   Variables:             3                          12 Mar 2020 15:18
```

Variable name	Storage type	Display format	Value label	Variable label
e_fmt	float	%9.2e		
f_fmt	float	%10.2f		
g_fmt	float	%9.0g		

```
Sorted by:
```

これらの変数の形式は、次のように入力して設定します。

```
. format e_fmt %9.2e
. format f_fmt %10.2f
```

変数g_fmtで形式を設定する必要が無かったのは、Stataが自動的に%9.0g形式に設定するからです。最も、「format g_fmt %9.0g」と入力しても同じ結果を得ることができます。データの結果をリストすると、次のようになります。

```
. list
```

	e_fmt	f_fmt	g_fmt
1.	2.80e+00	2.80	2.801785
2.	3.96e+06	3962322.50	3962323
3.	4.85e+00	4.85	4.852834
4.	-5.60e-06	-0.00	-5.60e-06
5.	6.26e+00	6.26	6.264982

テクニカルノート

上記の説明は、これで全て、という訳ではありません。もうひとつ、数値を使用して分析するユーザ向けに重要な形式があります。%21x形式は10進数（基数が10）を16進数（基数が16）に変更して表示します。数字は16進数(基数16)の桁で表現されます。aX+bの数字は、$a \times 2^b$をあらわします。例えば、次のように表現されます。

```
. display %21x 1234.75
+1.34b0000000000X+00a
```

つまり、10進数の数字である1,234.75は、16進数では1.34bX+0aと表示されます。これは、次のように表されます。

$$\left(1 + 3\cdot 6^{-1} + 4\cdot 16^{-2} + 11\cdot 16^{-3}\right) \times 2^{10}$$

16進数の表記は10進数と次のように対応します。

16進数	10進数
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
a	10
b	11
c	12
d	13
e	14
f	15

❑

詳しくは[U] 12.2 数字をご覧ください。

12.5.2　ヨーロッパ式数値形式

e, f, gの数値形式は、w.dではなくw,dと幅と小数点以下の桁数を指定すると、カンマ（,）が小数点位置を表す記号として使われます。例えば、%9,0g形式は、普通に操作すると1.5と表示するところを1,5と表示します。

fcまたはgcでヨーロッパの値を指定すると、ピリオドの変わりにカンマが使用されます。
例えば、%9,0gc形式は、普通に操作すると1,000.5と表示するところを1.000,5と表示します。

もし、この表示方法をデフォルトとして使用したい場合、Stataの「set dp comma」コマンドを使用する方法があります。「set dp comma」コマンドは、Stataにほぼ全ての%w.d{g|f|e}形式を%w,d{g|f|e}として認識するように指示します。Stata内のほとんどの箇所では小数点を表すのにピリオドを使用しています。つまり、%w,d{g|f|e}をデータに対して正しく設定していたとしても、表示するデータのみに影響を与える事になります。例えば、summarizeコマンドを入力して要約統計量を算出したり、regressコマンドを使用して回帰結果を計算すると、小数点はピリオドで表示されます。

「set dp comma」はこの設定を変更してStata全てに影響を与えます。「set dp comma」では、データが%w.d{g|f|e}の形式または%w,d{g|f|e}の形式、どちらで設定されていてもかまいません。全ての結果はカンマを小数点として使用しています。

```
. use https://www.stata-press.com/data/r18/auto
(1978 automobile data)
. set dp comma
. summarize mpg weight foreign
```

Variable	Obs	Mean	Std. dev.	Min	Max
mpg	74	21,2973	5,785503	12	41
weight	74	3019,459	777,1936	1760	4840
foreign	74	,2972973	,4601885	0	1

```
. regress mpg weight foreign
```

Source	SS	df	MS		
Model	1619,2877	2	809,643849	Number of obs	= 74
				F(2, 71)	= 69,75
Residual	824,171761	71	11,608053	Prob > F	= 0,0000
				R-squared	= 0,6627
Total	2443,45946	73	33,4720474	Adj R-squared	= 0,6532
				Root MSE	= 3,4071

mpg	Coefficient	Std. err.	t	P>|t|	[95% conf.	interval]
weight	-,0065879	,0006371	-10,34	0,000	-,0078583	-,0053175
foreign	-1,650029	1,075994	-1,53	0,130	-3,7955	,4954422
_cons	41,6797	2,165547	19,25	0,000	37,36172	45,99768

小数点をピリオドに戻すには、「set dp period」と入力してください。

❑ テクニカルノート

「set dp comma」はStataの内部に多くの変更を行います。しかし、古いユーザ作成プログラムの中には、これらの変更に適用できないものも出てくる事があります。古いユーザ定義プログラムを使用している場合、「set dp comma」を入力してからプログラムを実行すると、プログラムは動作せずに構文のエラーを返してくることがあります。

プログラムを使用していて本来表示されないはずのエラーが表示された場合、dpをperiodに設定し直してみてください。詳細は[D] formatをご覧ください。

また、set dp commaはStataの数値の出力に影響を与えますが、入力には影響を与えません。小数点を使用するには、ピリオドで入力する必要があります。「set dp comma」を行った場合でも、入力は次のように行ってください。

```
. replace x=1.5 if x==2
```

❑

12.5.3 日付と時間の形式

日付と時間の形式は実際には数値形式です。これは、Stataが時間をミリ秒の数字で、日、週、月、四半期、半年を1960年1月1日を起点とした日数で保存しています。詳細は[U] 25 日付と時間の作業をご覧ください。

%t形式の構文は次の通りです。

1番目に入力	%	形式が開始する事を指示
任意入力	-	結果を左揃えできる
2番目に入力	t	
3番目に入力	*文字*	単位を指定
任意入力	*他の文字*	日付や時間の表示形式を指定

単位を指定する時に入力する文字は次の一覧から選択してください。

```
C   1960年1月1日からのミリ秒、うるう秒の修正付き
c   1960年1月1日からのミリ秒、うるう秒を無視
d   1960年1月1日からの日数
w   1960年第1週からの週の数
m   1960年1月からのカレンダー月数
q   1960年第一四半期からの四半期数
h   1960年半年からの半年の数
```

どのように日付/時間を表示するか指定するコードは複数ありますが、通常、入力する必要はありません。ほとんどのユーザはデフォルトの日付/時間の際には%tcを、日付に関しては%tdを使用しています。 詳しくは[D] Datetime display formatsをご覧ください。

12.5.4　文字列形式

文字列形式に関する構文は次の通りです。

1番目に入力	%	形式が開始する事を指示
任意入力	-	結果を左揃えできる
2番目に入力	*数字*	結果の幅を指定
3番目に入力	s	

例えば、%10sは文字幅10の文字列形式です。[U] 12.4.2.2 Unicode文字の表示.

strwの形式は、デフォルトで%ws、あるいは%9sの長い方になります。例えば、str10変数は%10s形式になります。文字列はマイナス記号が表記されていない限り、フィールド上では右揃えで表示されます。つまり、%-10sと入力すると左揃えで表示できます。

例題4

Stataの自動車データはmakeという文字列変数を含んでいます。

```
. use https://www.stata-press.com/data/r18/auto
(1978 automobile data)
. describe make
              Storage   Display    Value
Variable name   type    format     label      Variable label
-------------------------------------------------------------
make            str18   %-18s                 Make and model
. list make in 63/67
```

	make
63.	Mazda GLC
64.	Peugeot 604
65.	Renault Le Car
66.	Subaru
67.	Toyota Celica

これらの値は%-18sと表示形式で指定されているため、左揃えです。これらの値を右揃えにしたい場合は、次のように表示形式を変更してください。

```
. format %18s make
. list make in 63/67
```

	make
63.	Mazda GLC
64.	Peugeot 604
65.	Renault Le Car
66.	Subaru Toyota C
67.	elica

12.6 データセット、変数、値ラベル

Stataの中の要素にはラベル付けを行えます。このラベルはデータセット、変数、値に付ける事ができます。

12.6.1 データセットラベル

それぞれのデータセットには80文字まで入力できる*データセットラベル*があります。このラベルは最初、空欄になっています。「label data ″テキスト″」と入力するとデータセットラベルを定義できます。

例題5

1980年の結婚率、離婚率、結婚年齢の中央値の州ごとのデータを入力しました。describeコマンドはメモリにあるデータを表示します。

```
. describe

Contains data
Observations:        50
   Variables:         4                          6 Apr 2022 15:43
```

Variable name	Storage type	Display format	Value label	Variable label
state	str8	%14g		
median_age	float	%9.2f		
marriage_rate	float	%9.0g		
divorce_rate	float	%9.0g		

```
Sorted by:
```

describeによると、4つの変数、state, median_age, marriage_rate, divorce_rateがあり、それぞれ50個の観測値があります。変数stateの保存形式は str8です。変数median_ageはfloatの保存形式、変数marriage_rateと divorce_rateはどちらもlongで保存されています。各変数の表示形式（詳細は[U] **12.5 形式：データ表示をコントロールする** をご覧ください）も表示します。 最後に、データはソートされている訳ではなく、データセットは前回保存されてから変更されていません。

このデータで「label data ″1980 state data″」と入力して、データセットにラベルを付けましょう。このコマンドを入力してから、もう一度describeコマンドを実行すると、次のようになります。

```
. label data ″1980 state data″
. describe Contains data
  Observation      50                            1980 state data
 Variables:        4
-------------------------------------------------------------------------------
Variable name   Storage   Display    Value
                type      format     label      Variable label
-------------------------------------------------------------------------------
state median_   str14     %14s
age marriage_   float     %9.0f
rate divorce_   float     %9.0g
rate            float     %9.0g
-------------------------------------------------------------------------------
Sorted by:
```

データセットラベルはdescribeとuseコマンドを実行すると表示されます。

12.6.2　変数ラベル

名前の他に、それぞれの変数には80文字分の*変数ラベル*の入力ができます。変数ラベルは、最初は空白です。「label variable *変数名* ″*テキスト*″」コマンドを使用して、新しい変数ラベルを設定します。

例題6

上記で示したデータに続けて、値とラベルを組み合わせるには、次のように入力しましょう。

```
. label variable median_age ″Median age″
. label variable marriage_rate ″Marriages per 100,000″
. label variable divorce_rate ″Divorces per 100,000″
```

これ以降、describeコマンドの結果は次のようになります。

```
. describe
Contains data
Observations:            50                          1980 state data
  Variables:             4                          6 Apr 2022 15:43
-------------------------------------------------------------------------------
              Storage   Display    Value
Variable name   type    format     label      Variable label
-------------------------------------------------------------------------------
state           str14   %14s
median_age      float   %9.2f                 Median age
marriage_rate   float   %9.0g                 Marriages per 100,000
divorce_rate    float   %9.0g                 Divorces per 100,000
-------------------------------------------------------------------------------
Sorted by:
```

Stataが結果を出力する時、表示範囲に問題が無いならば、変数名の代わりに変数ラベルを使用して表示します。

12.6.3 値ラベル

*値ラベル*は対応する値と言葉のマッピングを定義します。この言葉は数値が何を意味するのか説明するためのものです。マッピングは「label define *ラベル名 #"文字列" #"文字列"*」コマンドで名前をつけると共に定義します。．．*ラベル名*の最大文字数は32文字です。*#*は整数、あるいは拡張欠損値（.a, .b, . . . , .z）である必要があります。*文字列*の最大文字数は32,000文字です。名前をつけるマッピングは「label values *変数名 ラベル名*」コマンドで関連付けています。

例題7

定義というと、実際の値ラベルの機能以上の難しさと複雑さがあるように感じてしまいます。例題として、1つ紹介します。あるデータセットでそれぞれの個人の性別をコード化（0を男性、1を女性）しています。このデータセットに従業員番号と給与がある場合、次のようになります。

```
. use https://www.stata-press.com/data/r18/gxmpl4
(2007 Employee data)
. describe
Contains data from https://www.stata-press.com/data/r18/gxmpl4.dta
  Observation      7                          2007 Employee data
 Variables:        3                          11 Feb 2016 15:31
-------------------------------------------------------------------------------
              Storage   Display    Value
Variable name   type    format     label      Variable label
-------------------------------------------------------------------------------
empno           float   %9.0g                 Employee number
sex sa          float   %9.0g                 Sex
lary            float   %8.0fc                Annual salary, exclusive of bonus
-------------------------------------------------------------------------------
Sorted by:
```

```
. list
```

	empno	sex	salary
1.	57213	0	34,000
2.	47229	1	37,000
3.	57323	0	34,000
4.	57401	0	34,500
5.	57802	1	37,000
6.	57805	1	34,000
7.	57824	0	32,500

では、マッピングとしてsexlabelを作成しましょう。0を“Male”(男性)、1を“Female”(女性)として定義してこのラベルを変数sexと関連付けます。次のコマンドを実行してください。

```
. label define sexlabel 0 "Male" 1 "Female"
. label values sex sexlabel
```

実行後、データは次のようになります。

```
. describe
Contains  data  from  https://www.stata-press.com/data/r18/gxmpl4.dta
Observations:         7                           2007 Employee data
  Variables:          3                           11 Feb 2020 15:31
```

Variable name	Storage type	Display format	Value label	Variable label
empno	float	%9.0g		Employee number
sex sa	float	%9.0g	sexlabel	Sex
lary	float	%8.0fc		Annual salary, exclusive of bonus

```
Sorted by:
     Note: Dataset has changd since last saved
. list
```

	empno	sex	salary
1.	57213	Male	34,000
2.	47229	Female	37,000
3.	57323	Male	34,000
4.	57401	Male	34,500
5.	57802	Female	37,000
6.	57805	Female	34,000
7.	57824	Male	32,500

値ラベルはデータをlistしたときのみ使用するわけではなく、変数sexと関連付けたれたsexlabelがdescribeコマンドを実行すると表示されます。

◁

テクニカルノート

値ラベルと変数は同じ名前でもかまいません。例えば、上記例の値ラベルをsexlabelという名前にする代わりにsexと名前をつけることもできます。この場合、変数sexと値ラベルsexの関連付けを行うには「label values sex sex」と入力します。

❑

例題8

Stataのencodeとdecodeコマンドは簡単に文字列変数と数値変数の切り替えを行うことができます。先ほどの例題を元に説明を続けます。0を男性、1を女性とコードを割り振る代わりに、文字列変数を作成して"male"または"female"と直接入力していたとします。

```
. use https://www.stata-press.com/data/r18/gxmpl5, clear
(2007 Employee data)
. describe
Contains data from https://www.stata-press.com/data/r18/gxmpl5.dta
Observations:             7                  2007 Employee data
  Variables:              3                  11 Feb 2022 15:37
```

Variable name	Storage type	Display format	Value label	Variable label
empno	float	%9.0g		Employee number
sex sa	str6	%9s		Sex
lary	float	%8.0fc		Annual salary, exclusive of bonus

```
Sorted by:
. list
```

	empno	sex	salary
1.	57213	male	34,000
2.	47229	female	37,000
3.	57323	male	34,000
4.	57401	male	34,500
5.	57802	female	37,000
6.	57805	female	34,000
7.	57824	male	32,500

このデータを元に数値化した変数genderを文字列変数から作成します。この変換を行う目的としては、例えば「anova salary sex」と入力して給与と性別で一元分散分析（one-way ANOVA）を行うと観測値がないというメッセージが表示されるのを回避するためです。Stataの統計的なコマンドは文字列変数を欠損値のように扱うため、上記のようにエラーになります。統計的なコマンドは数値データでのみ機能します。

```
. encode sex, generate(gender)
. describe
Contains data from https://www.stata-press.com/data/r18/gxmpl5.dta
Observations:         7                          2007 Employee data
   Variables:         4                          11 Feb 2022 15:37
```

Variable name	Storage type	Display format	Value label	Variable label
empno	float	%9.0g		Employee number
sex sa	str6	%9s		Sex
lary g	float	%8.0fc		Annual salary, exclusive of bonus
ender	long	%8.0g	gender	Sex

```
Sorted by:
     Note:Dataset has changed since last saved.
```

encodeコマンドは新しいlong保存形式の変数genderを作成し、新しい値ラベルgenderを作成します。値ラベル genderは文字列maleに1を、文字列femaleに2を関連付けます。この状態でlistコマンドを使用してデータをリストすると、変数genderとsexの間に違いは見受けられなくなります。しかし、この2つは異なります。Stataの統計コマンドは変数genderを扱うことはできますが、変数sexを扱って計算することはできません。詳細は[U] 24.2 カテゴリカルな文字列変数をご覧ください。

❑

❑ テクニカルノート

従業員のデータではなく、例えば、性転換の処置を受けている患者のデータがあるとします。これは、つまり、性転換の処置前と後の2つの性別情報のための変数が作成されます。これらの変数はpresexとpostsexという名前がついているとします。これらの変数に*同じ*値ラベルを設定するには、次のように入力してください。

```
. label define sexlabel 0 "Male" 1 "Female"
. label values presex sexlabel
. label values postsex sexlabel
```

❑

❑ テクニカルノート

Stataの入力コマンド（inputとinfileコマンド）は、値ラベルの言葉を数値のコードに変換できます。encodeとdecodeは文字列と数値のマッピングを交互に変換できるので、入力の際、あるいはもっと後に文字列を数値に変換することもできます。

例えば、次のように表現されます。

```
. label define sexlabel 0 "Male" 1 "Female"
. input empno sex:sexlabel salary, label empnoeempno        sex
                      salary
  1.57213 Male 34000
  2.47229 Female 37000
  3.57323 0 34000
  4.57401 Male 34500
  5.57802 Female 37000
  6.57805 Female 34000
  7.57824 Male 32500
  8.end
```

label defineコマンドは値ラベルsexlabelを定義します。「input empno sex:sexlabel salary, label」の表記はStataに3つの変数(empno, sex, salary)をキーボードから入力し、値ラベルsexlabelを変数sexに関連付けします。そして、言葉の中で数値に変換するべきものを変換します。正しく設定できているか確認するために、入力したデータをlistコマンドで表示してみます。

```
. list
```

	empno	sex	salary
1.	57213	Male	34000
2.	47229	Female	37000
3.	57323	Male	34000
4.	57401	Male	34500
5.	57802	Female	37000
6.	57805	Female	34000
7.	57824	Male	32500

入力した観測値3の値とStataがリストした値を比べてみましょう。入力した値は「57323 0 34000」です。つまり、観測値3に入力した値は0ですが、Stataが観測値をリストしたとき、Stataは値をMaleと表示しました。これはlabel defineコマンドでゼロはMaleと同じであると定義しているためです。

このデータにもう1つ観測値を追加しましょう。

```
. input, label
        empno         sex      salary
  8. 67223 FEmale 33000
'FEmale' cannot be read as a number
  8. 67223 Female 33000
  9. end
```

最初は「67223 FEmale 33000」と入力しましたが、それに対してStataは“’FEmale’ cannot be read as a number”というメッセージを返します。Stataは大文字と小文字を区別するため、FEmaleとFemaleは別のものであると区別されます。Stataはもう一度入力するように指示してきたので、今度は正確に入力しました。

❑

❑ テクニカルノート

automaticオプションと組み合わせると、Stataは言葉を数字に変更できるだけではなく、マッピングも自動的に作成します。再び、データを入力しましょう。今回はキーボードで入力するのではなく、ファイルから読み込みます。employee.rawという名前のテキスト文書があり、その内容としては次のようになっているとします。

```
57213 Male 34000
47229 Female 37000
57323 Male 34000
57401 Male 34500
57802 Female 37000
57805 Female 34000
57824 Male 32500
```

infileコマンドはこれらのデータを読み込み、自動的にマッピングを行います。

```
. label list sexlabel
value label sexlabel not found
r(111);
. infile empno sex:sexlabel salary using employee, automatic
(7 observations read)
```

最初のコマンド、「label list sexlabel」は今までに値ラベルとしてsexlabelを定義していないことを確認するだけのものです。Stataは特に問題なくinfileコマンドを実行しました。これで、次のようなデータを準備できました。

```
. list
```

	empno	sex	salary
1.	57213	Male	34000
2.	47229	Female	37000
3.	57323	Male	34000
4.	57401	Male	34500
5.	57802	Female	37000
6.	57805	Female	34000
7.	57824	Male	32500

もちろん、sexはもうひとつの数値変数です。実際にはMaleとFemaleのような文字列を有しているわけではありません。これは数値が自動的にマッピングされたラベルを表示していることになります。マッピングの項目が何かを確認するには、label listコマンドを使用します。

```
. label list sexlabel
sexlabel:
           1 Male
           2 Female
```

この情報から、Stataが1をMale、2をFemaleに設定したことがわかります。実際のデータを確認したい場合(値ラベルではなく、データを直に確認したい場合)、 nolabelオプションを使用すると確認できます。

```
. list, nolabel
```

	empno	sex	salary
1.	57213	1	34000
2.	47229	2	37000
3.	57323	1	34000
4.	57401	1	34500
5.	57802	2	37000
6.	57805	2	34000
7.	57824	1	32500

❑

12.6.4 他言語でのラベル

データセットはデータ、変数、値を含めたラベルで最大100言語まで格納できます。メモリにあるデータセットで使用できる言語を確認するには、label languageを入力してください。すると、次のように表示されます。

```
. label language
Language for variable and value labels
    In this dataset, value and variable labels have been defined in only one language:   default
    To create new language:          . label language <name>, new
    To rename current language:      . label language <name>, rename
```

あるいは、次のようになります。

```
. label language
Language for variable and value labels
    Available  languages:
            default
            de
            en
            sp
    Currently set is:                . label language sp
    To select different language:    . label language <name>
    To create new language:          . label language <name>, new
    To rename current language:      . label language <name>, rename
```

現在、サンプルデータセットはsp(スペイン語)ラベルが設定されています。

```
. describe Contains data
Observations:        74                          Automóviles.  1978
  Variables:         12                          3 Oct 2020 13:53
```

Variable name	Storage type	Display format	Value label	Variable label
make	str18	%-18s		Marca y modelo
price	int	%8.0gc		Precio
mpg	int	%8.0g		Consumo de combustible
rep78	int	%8.0g		Historia de reparaciones
headroom	float	%6.1f		Cabeza adelante
trunk	int	%8.0g		Volumen del maletero
weight	int	%8.0gc		Peso
length	int	%8.0g		Longitud
turn	int	%8.0g		Radio de giro
displacement	int	%8.0g		Cilindrada
gear_ratio	float	%6.2f		Relacio´n de cambio
foreign	byte	%8.0g		Extranjero

```
Sorted by: foreign
```

1つ以上の言語でラベルを作成するには、新しい言語を設定して、通常の手順でラベルを設定してください。詳細は[D] label languageをご覧ください。

12.7　データに附属したメモ

データセットにはメモを含むことがあります。メモはnotesコマンドで定義して確認できるテキストの文書です。noteの後にコロン（:）を入力してメモとして登録するテキストを入力します。

```
. note:  Send copy to Bob once verified.
```

定義したメモは、次のように入力するといつでも表示できます。

```
. notes
_dta:
  1.  Send copy to Bob once verified.
```

メモはデータと共に保存されます。つまり、データセットを保存すればこのメモを再度表示させることができます。

また、メモを追加することもできます。

```
. note: Mary also wants a copy.
. notes
_dta:
  1.  Send copy to Bob once verified.
  2.  Mary also wants a copy.
```

今まで追加したメモは、データ全体に添付されています。これが_dtaのプリフィックスを使用してリストする理由です。また、変数にメモをつけることもできます。

```
. note state: Verify values for Nevada.
. note state: What about the two missing values?
. notes
_dta:
  1.  Send copy to Bob once verified.
  2.  Mary also wants a copy.       state:
  1.  Verify values for Nevada.
  2.  What about the two missing values?
```

データをdescribeコマンドで表示すると、データセットにメモが附属しているか、あるいは変数にメモがあるか等を確認できます。

```
. describe
Contains data from states
  Observations:    50                           1980 state data
  Variables:        4
                                                (_dta has notes)
-------------------------------------------------------------------------------
              Storage   Display    Value
Variable name   type    format     label      Variable label
-------------------------------------------------------------------------------
state           str8    %9s                 * 
median_age      float   %9.0g                 Median Age
marriage_rate   long    %12.0g                Marriages per 100,000
divorce_rate    long    %12.0g                Divorces per 100,000
                                             * indicated variables have notes
-------------------------------------------------------------------------------
Sorted by:
     Note:Dataset has changed since last saved.
```

この機能に関する説明は[D] notesをご覧ください。

12.8 構造情報

構造情報はStataにとって秘匿部分ですが、これはStataのプログラマーにとってはとても便利な機能です。実際に前述のnotesコマンドはこの構造情報を使用して挿入しています。

データセットそのものと、中に含まれているそれぞれの変数には構造情報のセットがあります。構造情報は、varname[charname]のように、名前をつけて参照できます。ここでvarnameは変数あるいは_dtaを指す名前です。構造情報はテキストを含み、Stataのデータと共にStata形式である.dtaデータと共に保存されます。つまり、データをロードする度に呼び出されます。

構造情報はどのように使われるのでしょうか。[XT] **xt**コマンドはパネル変数の名前が必要ですが、これらのコマンドの中には時間変数を必要とするものもあります。xtsetはパネル変数を指定する時とオプションとして時間変数を指定する時に使用します。ユーザがデータに対してxtsetを行う必要があるのは、1回のみです。Stataはその後、他のStataセッションでもこの情報を覚えています。Stataはこれを次の構造情報で格納します。_dta[iis]はパネル変数の名前を有し、_dta[tis]は時間変数の名前を格納します。xtコマンドが実行されると、コマンドはこれらの構造情報を確認し、パネルと時間の変数名を取得します。この情報が見つからない場合、データはxtsetを行われていないことを示し、エラーメッセージが表示されます。構造情報がどのようにセットされているか、はユーザから隠されており、どのようにパネル変数や時間変数がStataに記憶されているのか、記載されていません。

Stataユーザとしては、いくつかのコマンドで構造情報をどのように設定して、削除するのか説明されているもののみを確認すれば十分です。*変数名*のついた変数のcharname（*構造情報名*）でxを指定するには、次のように入力します。

```
. char varname[charname] x
```

データの構造情報名charnameを xに 指定するには、次のように入力します。

```
. char _dta[charname] x
```

構造情報を削除するには、次のように入力します。

```
. char varname[charname]
```

varnameは変数名かデータ名です。一度も設定していなくても、構造情報は削除できます。

構造情報の最も重要な機能は、Stataがセッションをまたいで記録をしていること、つまり、データと共に保存していることです。

❑ テクニカルノート

プログラマーは詳細を確認したいでしょう。技術的な説明は[P] **char**で確認できます。概要として、varnameとcharnameを示すためには、シングルクォーテーションを使用して、‘varname[charname]’と入力すると参照できます。詳細は[U] 18.3.13 構造情報の参照をご覧ください。

varnameと関係する全ての構造情報の名前を集めるには、次のように入力してください。

```
. local macname : char varname[]
```

構造情報の最大収容サイズは67,784バイト分入力できます（Stata/BE, Stata/SE, Stata/MP共通）。名前と構造情報の関連付けは慣習に則っています。プログラマーとして新しい構造情報を作成してadoファイルに組み込む場合、最低1つの大文字を入力することを忘れないでください。現在の慣習では、全てが小文字の構造情報は公式Stataのために予約されています。

❑

12.9　データエディタと変数マネージャ

今まで、この章ではStataのコマンド入力でデータ管理を行う場合の説明をしてきました。しかし、Stataは画面上でデータを確認および管理できる強力な機能としてデータエディタと変数マネージャを準備しています。

データエディタはスプレッドシート形式でデータ編集を行うことができます。データの入力、既存のデータの編集、読み取り専用モードでのデータの確認、ほぼ全ての希望するデータ管理機能を再現可能な状態のGUIで実行できます。データエディタを開くには、**データ** > **データエディタ** > **データエディタ(編集)** または **データ** > **データエディタ** > **データエディタ(ブラウズ)**と操作してください。[GS] **6 Using the Data Editor** (GSM, GSU, GSW)では、データエディタに関するチュートリアルの内容を確認できます。詳細は[D] **edit**をご覧ください。

変数マネージャはデータ内にある全ての変数をリストし、プロパティを設定できるツールです。変数のプロパティは名前、ラベル、保存形式、表示形式、値ラベル、メモを指します。変数マネージャは変数をソートしてフィルタをかけることができます。これは多くの変数を有しているデータで作業を行っているときにとても便利です。変数マネージャはコマンドウィンドウのための変数リスト (varlists) を作成する際にも使用できます。変数マネージャを開くには、**データ** > **変数マネージャ**と操作してください。[GS] **7 Using the Variables Manager** (GSM, GSU, GSW)では、変数マネージャに関する説明を確認できます。

データエディタと変数マネージャは、どちらもStataにコマンドを送信することで、希望の操作を実行します。インターフェイスの操作からもどのような変更を行ったのかログを残すことができ、同時にインタラクティブに操作することもできます。最終的には実行した操作のログができるので、後ほど分析を再実行したい際などに便利です。

12.10　データフレーム

ここまでは、メモリ上の単一のデータセットを利用したStataの使用例を紹介してきました。しかし、Stataは同時に複数のデータセットをメモリに記憶して、フレーム、またはデータフレームとして保存することができます。使用するフレームを切り替えたり、フレーム間でデータをコピーまたは変数の別名を作成したり、他のフレームのデータを利用して分析を行ったり、フレーム間でキーとなる変数を設定して、リンクさせることができます。詳細は[D] **frames Intro**をご覧ください。

12.11　参考文献

Cox, N. J. 2006. Stata tip 33: Sweet sixteen:Hexadecimal formats and precision problems. *Stata Journal* 6: 282-283.

——. 2010a. Stata tip 84: Summing missings. *Stata Journal* 10: 157-159.

——. 2010b. Stata tip 85: Looping over nonintegers. *Stata Journal* 10: 160-163.

Cox, N. J., and C. B. Schechter. 2018. Speaking Stata: Seven steps for vexatious string variables.*Stata Journal* 18: 981-994.

Daniels, I., and N. Minot. 2020. *An Intorduction to Statistics and Data Analysis Uising Stata*. Thousand Oaks, CA: SAGE.

Long, J. S. 2009. *The Workflow of Data Analysis Using Stata*. College Station, TX: Stata Press.

Longest, K. C. 2020. *Using Stata for Quantitative Analysis*. 3nd ed. Thousand Oaks, CA: Sage.

Mitchell, M. N. 2010. *Data Management Using Stata:A Practical Handbook*. College Station, TX: Stata Press.

Rising, W. R. 2010. Stata tip 86: The missing() function. *Stata Journal* 10: 303-304.

13 関数と表現

目次

[U] **11 言語構文**をまだ読んでいない場合は、この章を読む前にご一読ください。

13.1 概要

計算式の入力例は、次のようになります。

```
        2+2
miles/gallons
myv+2/oth
(myv+2)/oth
ln(income)
age<25 & income>50000
age<25 | income>50000
age==25
name=="M Brown"
fname + " " + lname
substr(name,1,10)
val[_n-1]
L.gnp
```

上記のような表現は、コマンドの構文の中にexpがあるところで使用できます。例題の1つは[D] **generate**にあります。

```
generate newvar = exp [if] [in]
```

expは新しい変数の内容を指定します。任意で2つ目の表現を追加でき、その数式では定義するサンプルの範囲を指定します。他の例としては、[R] **summarize**があります。

```
summarize [varlist] [if] [in]
```

任意で追加できる式はサンプルを制限し、その要約統計量を算出する範囲を指示します。

算術表現と文字列表現は一般的な方法で指定でき、通常の適用順で行われます。
計算の順番を変更するには、括弧を自由に使ってください。

例題1

myv+2/othはmyv+(2/oth)として認識されます。計算する順番を変更したい場合は、(myv+2)/othと入力してください。

13.2 演算子

Stataには4つの演算子の区分、つまり算術、文字列、関係、論理があります。それぞれの区分に関しては、以下の説明を確認してください。

13.2.1 算術演算子

Stataで使用している算術演算子は、 + (加算), - (減算), * (乗算), / (除算), ^ (べき乗), 数字の前で使用する - (負の記号)です。欠損値に対する算術演算や、不可能な算術演算(例えば、0で割る)は欠損値を出力します。

例題2

-(x+y^(x-y))/(x*y)と記載されている場合、以下の式を表します。

$$-\frac{x + y^{x-y}}{x \cdot y}$$

xまたはyが欠損値か0の場合、出力結果は欠損値になります。

13.2.2 文字列演算子

文字列演算の中では、+と*記号は使用できます。

+は2つの文字列を連結する際に使用します。Stataはその表現の内容を見て、+記号が加算と連結、どちらを意味するのか判断します。+が数値の間にあれば、Stataは加算します。+ が文字列の間にあれば、Stataはその文字列を連結します。

例題3

"this"+"that"という表現は、"thisthat"を出力します。 2+3 という表現は数字の5を出力します。Stataは+の両脇にある引数が同じ種類で無い時は、“type mismatch”というエラーを返します。つまり、2+"this"や 2+"3"はエラーになります。

+記号の両側にある引数はどれだけ複雑でもかまいません。

```
substr(string(20+2),1,1) + strupper(substr("rf",1+1,1))
```

上記の表記の文字列表現の結果は"2F"になります。詳細については[FN] String functionsをご覧ください。特に、substr(), string(), strupper()関数を説明しています。

*はその文字列を0回以上の回数、複製するときに使用します。Stataはその前後関係を見て、*記号が乗算と文字列複製、どちらを意味しているのか判断します。*が数値の間にあれば、Stataは乗算します。*が文字列と数値の間にある場合、Stataはその数値が示す回数分だけ文字列を複製します。

例題4

"this"*3という表現は、"thisthisthis"を出力します。2*3という表現は数字の6を出力します。Stataは*の両脇にある引数がどちらも文字列の場合、“type mismatch”というエラーを返します。つまり、"this"*"that"という表現はエラーになります。

例題3の文字列連結と同じように、引数は任意で複雑にできます。

13.2.3　関係演算子

関係演算子は、>（より大きい），<（より小さい），>=（以上），<=（以下），==（等しい），!=（等しくない）があります。等しさを示す関係演算子が等号2つになることに注意してください。この記述方法は関係による等号と=expで定義する内容を区別するために行っています。

テクニカルノート

~は!の代わりに使用できます。つまり、論理的な“ではない”（not）を表現するときに使用できます。等しくない事を示す演算子は~=と入力します。

関係演算子ではその結果は真か偽で表します。関係演算子は数値表現でも文字列を表すサブ表現でも使用できます。3>2は真で、 "zebra">"cat"も真になります。後者の場合、"zebra"は辞書の並び順（アルファベット順）で"cat"の後に表示される関係を示しています。Stataの中では、大文字の言葉は小文字よりも先になるので、"cat">"Zebra"は真となり、成り立ちます。

関係演算子の中で欠損値を使用することもできます。xが数値の場合、x>=.の表現は、xの値が欠損値である場合は*真*に、それ以外の場合は*偽*になります。欠損値は欠損値ではないどの値よりも大きくなります。詳細は[U] 12.2.1 欠損値をご確認ください。

例題5

ageとincomeについてのデータを持っており、データのサブセットを25歳より下に設定してリストを行います。次のように入力しましょう。

```
. list if age<=25
```

ちょうど25歳の人のデータをサブセットとして表示するには、次のように入力します。

```
. list if age==25
```

等号を2つ使用している点に注意してください。「list if age=25」と入力するとエラーになります。

関係演算子が関係を評価して真偽を確かめていると考えるのは便利ですが、実際には数字で算出してそれを評価しています。真という結果は1で定義され、偽という結果は0で定義されます。

例題6

真偽の定義はダミー変数を作成するのに便利です。例えば、以下のようになります。

```
generate incgt10k=income>10000
```

変数incomeの値が$10,000以下の場合は値を0とし、変数incomeが$10,000よりも大きい場合は1を入力します。欠損値は全ての記入されている値よりも大きく定義されているため、新しい変数incgt10kでは、欠損値がある場合も1の値を入力します。次のように入力すれば、その危険は回避できるでしょう。

```
generate incgt10k=income>10000 if income<.
```

変数incomeで欠損値が存在する場合は、新しい変数incgt10kでも欠損値を有することになります。詳しくは[U] 26 カテゴリデータと因子変数を使用するの例題をご確認ください。

❑ テクニカルノート

算術演算子と関係演算子はどちらも数値に対して作用するので、これら2つの演算子の両方を一度に使用してはならない、という事にはなりません。例えば、(2==2)+1は2になります。これは、2==2は1(真)となり、1 + 1は2になるからです。

関係演算子は算術演算子が全て計算された後に判断されます。つまり、(3>2)+1は2となりますが、3>2+1は0になります。関係演算子の場合、（数値ではなく）論理的な解釈でも確認できます。3>2+1が偽であることは理にかなっているといえるでしょう。

❑

13.2.4 論理演算子

論理演算子には& (and), | (or), ! (not)があります。論理演算子はゼロではない値（*欠損値*を含む）を*真*と判断し、ゼロを*偽*と判断します。

例題7

変数ageとincomeを有するデータセットがあり、その中で$50,000より多く稼いでいる人のとなりに、25よりも下の年齢で$30,000よりも多く稼いでいる人をリスト表示する場合、次のように入力します。

```
list if income>50000 | income>30000 & age<25
```

&は|よりも有意になります。不安があるなら、次のように入力することもできます。

```
list if income>50000 | (income>30000 & age<25)
```

どちらの場合でも、この構文では変数incomeの値が欠損値でもlist表示されます。
これは、*欠損値*は50,000よりも大きいからです。

◁

❑ テクニカルノート

関係演算子のように、論理演算子は*真*には1を、*偽*には0を返します。例えば、表現 「5 & .」は1になります。論理演算子では、 !を除き、全ての算術や関係演算子が計算された後に実行されます。つまり、3>2 & 5>4 は(3>2) & (5>4)と同じになり、結果は1になります。

❑

13.2.5 全ての演算子を含んだ時の評価順

全ての演算子で評価される順番は 、!（または ~), ^, -（負の符号), /, *, -（減算), +, !=（または ~=), >, <, <=, >=, ==, &, | となります。

13.3 関数

Stataは算術的関数、確率と密度の関数、行列関数、文字列関数、日付や時間に対応する関数、プログラマ向けの特殊な関数、を提供しています。これら全ての関数の詳細は*Stata Functions Reference Manual*に記載しています。Stataの行列プログラミング言語、Mataはさらに多くの関数を有しており、その関数については*Mata Reference Manual*に記載されています。あるいはヘルプ文書を確認（「help mata functions」と入力）してください。

関数は単純にルールのセットです。関数に引数を受け渡し、関数が定義されているルールに従ってその引数を評価します。関数は最終的には引数を評価するサブルーチンで、それだけでは何も実行しません。関数はStataのコマンドと共に使用する必要があります。関数は関数名、開小括弧、表現あるいはカンマによって分けられた複数の表現、閉小括弧、の形で定義されます。

たとえば、次のように表現されます。

```
. display sqrt(4) 2
```

または

```
. display sqrt(2+2) 2
```

これは、最も簡単な関数の使い方です。この例では、算術関数sqrt()を使用しました。これは1つの数字（あるいは表現）をその引数として受け取り、その値の平方根を返します。この例では、Stataのdisplayコマンドと組み合わせて関数が使われています。単純に関数だけを入力した場合、次のようになります。

```
. sqrt(4)
```

Stataは次のようなエラーメッセージを返します。

```
command sqrt is unrecognized
r(199);
```

関数は変数に対しても作用します。例えば、対数正規分布から値を集めたランダム変数を生成する場合、次のように入力します。

```
. set obs 5
Number of observations (_N) was 0, now 5
. generate y = runiform()
. replace y = invnormal(y) (5 real changes made)
. replace y = exp(y) (5 real changes made)
. list
```

	y
1.	.686471
2.	2.380994
3.	.2814537
4.	1.215575
5.	.2920268

入力を減らすには、次のように入力すれば同じ結果を得られます。

```
. generate y = exp(rnormal())
```

関数は引数として関数を使用できます。

全ての関数は特定の定義域で定義され、特定の範囲内で値を返します。その関数の定義域外の引数が与えられた場合、関数は欠損値かエラーメッセージのうち、適切な方を返します。例えば、log()関数に引数としてゼロを入力すると、log(0)は欠損値になります。これはゼロが自然対数関数の定義域外にあるからです。log()関数に文字列引数を入力すると、Stataは“type mismatch”エラーを返します。これは、log()関数は数値関数で文字列に関数を適用できないからです。

関数の範囲外の結果を算出する引数を受け渡すと、関数は欠損値を返します。文字列を引数として受け付ける場合、文字列はダブルクォーテーションで囲む必要があります。例外としては文字列保存形式を指定している変数を使用する場合です。

13.4 システム変数（_variables）

計算式には_variablesを含めることができます。これはビルトインのシステム変数で、Stataによって作成され更新されます。システム変数と呼ばれるのは、該当する名前がアンダーバー「_」で始まるからです。

_variablesには、次の項目があります。

[eqno]_b[varname]（同義語：[eqno]_coef[varname]）は最近にフィットしたモデル（ANOVA，回帰，Cox，ロジット，プロビット，多項ロジットなど）で使用しているvarnameにかかる係数（マシンによる精度までの）値を設定します。詳細な説明は以下にある[U] 13.5 係数と標準誤差にアクセスするのセクションをご確認ください。

_consは直接使用して参照すると常に数字の1と等しくなります。また、間接的（例えば、_b[_cons]）に使用すると切片の項を示します。

_nは現在の観測値の数を格納します。

_Nはデータセット内に保有している全ての観測値、あるいは現在のby()グループに属している観測値の数を示します。

_piはπの値を格納します。

[eqno]_se[varname]は最近にフィットしたモデル（ANOVA，回帰，Cox，ロジット，プロビット，多項ロジットなど）で使用しているvarnameにかかる標準誤差の係数の（マシンによる精度までの）値を設定します。詳細な説明は以下にある[U] 13.5 係数と標準誤差にアクセスするのセクションをご確認ください。

13.5 係数と標準誤差にアクセスする

モデルでフィットした後、係数や標準誤差を取得してその後の数式に使用することができます。簡単に推定値、残差などの値を入手するには、[R] predict（と[U] 20 推定と推定後コマンド）をあわせてご覧ください。

13.5.1 単一推定式モデル

まず、係数と変数が一対一の関係がある、1つの推定式を算出する推定方法（例えば、logit，ologit，oprobit，probit，regress，tobit）について考えてみましょう。_b[varname]（同義語：_coef[varname]）はvarnameに関する係数を格納し、_se[varname]は標準誤差を格納します。どちらもコンピュータの精度で記録されます。つまり、_b[age]は「regress response age sex」を実行した後の変数ageの計算した係数を、_se[age]は係数の標準誤差を参照します。_b[_cons]は定数を、_se[_cons]は標準誤差を示します。つまり、次のように入力できます。

```
. regress response age sex
. generate asif = _b[_cons] + _b[age]*age
```

13.5.2　複数推定式モデル

複数の推定式を作成するモデルで係数や標準誤差を参照する構文は、1つの推定式を出力する場合の構文と基本的に同じです。しかし、_b[]と_se[]は大括弧[]を使用して推定式の番号を入力してください。最も、同じ出力結果を算出するには他の入力方法もあり、それは次のように記載します。

```
[eqno]_b[varname]
[eqno]_se[varname]
```

しかし、_b[]は_coef[]と入れ替えても使用出来ます。なお、_b[]は完全に無視できるので、多くのStataユーザは無視します。

```
[eqno][varname]
```

また、2つ目の大括弧[]は完全に無視できます。

```
[eqno]varname
```

また、_b[] や_se[]は繋げる事が出来、eqnoとvarnameの間にコロン（:）を入力できます。

```
_b[eqno:varname]
```

推定式番号、eqnoを指定する方法は絶対推定式番号か間接推定式番号の2つになります。絶対推定式番号を使用する場合、数字は「#」記号の後に続きます。つまり、[#1]displはdisplの第一推定式にある係数を意味します（そして、[#1]_se[displ]は第一推定式の標準誤差を表します）。この方法は、簡単なモデル、例えばregressコマンドでも使用できます。regressコマンドは1つの推定式を推定します。つまり[#1]displは_b[displ]と同じようにdisplの係数を推定します。同じように [#1]_se[displ]と_se[displ]は同じです。この理論は双方向に働きます。複数の推定式がある場合、_b[displ]はdisplの最初の推定式の係数を算出し、_se[displ]は標準誤差を算出します。_b[varname](_se[varname])は[#1]varname([#1]_se[varname])のもう一つの表記方法となります。

推定式も間接的に表現できます。[res]displはresという名前の推定式の中の変数displに対応する係数を示します。フィットするモデルに従属変数という概念があるなら、推定式内では頻繁にその従属変数名で参照されます。つまり、[res]displは変数resを求める推定式におけるdisplの係数を示す可能性もあります。

多項ロジット（mlogit）、多項プロビット（mprobit）や他の類似コマンドでは、推定式は1つの従属カテゴリカル変数を元に名づけられています。このようなモデルでは、1つの従属変数があり、その変数に記録される出力（取り込む）値に対応する推定式があります。ただし、ベースとなる出力は除きます。[res]displは結果であるresを出力する推定式に対応する変数はdisplであると認識します。もし出力結果であるresが元となる出力の場合、Stataは[res]displをゼロとして扱います（これは、[res]_se[displ]でも同じです。）

多項式の出力結果について継続して説明します。出力変数は数値でなければなりません。構文[res]displは数値を出力する変数があり、その変数に値ラベルとしてresが関連付けられている場合のみ使用できます。ラベル付けが行われていないデータを使用する場合、一般的な複数推定式構文([##]varnameと[##]_se[varname])を使用して、#番目の推定式の中の変数varnameの係数と標準誤差を算出します。

mlogitでは、データにラベルが付いていない場合、[#]varnameと[#]_se[varname]（つまり、「#」が無い状態）の構文を使用して係数や標準誤差をvarnameに対して等式の出力#と表記する事ができます。

13.5.3 因子変数と時系列演算子

時系列に対応した変数は、一般的な変数と同じように参照するので、変数の名前を入力します。時系列に関連する変数には、時系列を示す演算子も付随します。詳細は[U] 11.4.4 時系列変数でご覧ください。次のように入力できます。

```
. regress open L.close LD.volume
. display _b[L.close]
. display _b[LD.volume]
```

この表現の中で、因子変数にi.groupのように参照することはできません。例えば、i.groupには3つの階層があります。1b.group, 2.group, 3.groupのような表現は3つの仮想変数の数に由来します。表現の中にある指標子を示すには、例えば、_b[i2.group]や _b[2.group]を使用します。つまり、指標変数の入力する指標子と因子変数のレベルを指定する事になります。因子変数を使用して回帰を行う場合を考えて見ましょう。

```
. use https://www.stata-press.com/data/r18/fvex, clear
(Artificial factor variables' data)
. regress y i.sex i.group sex#group age sex#c.age
```

Source	SS	df	MS		
Model	221310.507	7	31615.7868	Prob > F	= 0.0000
Residual	1170122.5	2,992	391.083723	R-squared	= 0.1591
				Adj R-squared	= 0.1571
Total	1391433.01	2,999	463.965657	Root MSE	= 19.776

y	Coeffi cient	Std. err.	t	P>|t|	[95% conf.	interval]
sex						
female	32.29378	3.782064	8.54	0.000	24.87807	39.70949
group						
2	9.477077	1.624075	5.84	0.000	6.292659	12.66149
3	18.31292	1.776337	10.31	0.000	14.82995	21.79588
sex#group						
female#2	-6.621804	2.021384	-3.28	0.001	-10.58525	-2.658361
female#3	-10.48293	3.209	-3.27	0.001	-16.775	-4.190858
age	-.212332	.0538345	-3.94	0.000	-.3178884	-.1067756
sex#c.age						
female	-.226838	.0745707	-3.04	0.002	-.3730531	-.0806229
cons	60.48167	2.842955	21.27	0.000	54.90732	66.05601

例えば、因子変数レベル2のグループの係数を使用する場合は、_b[2.group]と入力しました。また、レベル3を使いたい時は_b[3.group]と入力します。2つのレベルにまたがる2つの因子変数における係数のやり取りについては、その関係演算子を指定してそれぞれの変数のレベルも指定します。例えば、sex = 1 (female)とgroup = 2の係数を使用するには、_b[1.sex#2.group]と入力します。（1が女性（female）であると、label listと入力して確認しました。）変数のうちの1つの交差項が連続である場合、それを明確に指示する（_b[1.sex#c.age]）か、c.を使用しないで_b[1.sex#age]と入力します。

普通の変数を参照するよりも、交差項を参照するほうが難しいです。さらに高度なのは、複数の数式を使用する推定子の係数を参照することです。どの形式の係数なのか分からない場合、実際に行った推定を、coeflegendオプションと共に実行し直してみてください。

```
. regress, coeflegend
```

Source	SS	df	MS		
Model	221310.507	7	31615.7868	Number of obs	= 3,000
Residual	1170122.5	2,992	391.083723	F(7, 2992)	= 80.84
				Prob > F	= 0.0000
				R-squared	= 0.1591
				Adj R-squared	= 0.1571
Total	1391433.01	2,999	463.965657	Root MSE	= 19.776

y	Coefficient	Legend
sex		
female	32.29378	_b[1.sex]
group		
2	9.477077	_b[2.group]
3	18.31292	_b[3.group]
sex#group		
female#2	-6.621804	_b[1.sex#2.group]
female#3	-10.48293	_b[1.sex#3.group]
age	-.212332	_b[age]
sex#c.age		
female	-.226838	_b[1.sex#c.age]
_cons	60.48167	_b[_cons]

Legend列は推定の中にある係数を参照するにはどのように入力するか、記述します。

もし、推定結果が式と因子変数の両方を有している場合、上記[U] 13.5.2 複数式モデルと何も変わりません。*varname* に入力する内容が、少し複雑になるだけです。

13.6 Stataのコマンドから結果にアクセスする

推定コマンドだけではなく、ほとんどのStataコマンドは後から表現で参照できる結果を格納します。e(name), r(name), s(name), c(name)と実行することで参照できます。

```
. summarize age
. generate agedev = age-r(mean)
. regress mpg weight
. display "The number of observations used is " e(N)
```

ほとんどのコマンドはrクラスとして分類されています。これは、結果をr()に格納します。返される結果、例えばr(mean)、はコマンド実行直後から取得可能です。それを参照するには可能な限り早い段階で参照してください。というのも、これは次のコマンドがr()の値を置き換えるためです。

eクラスコマンドは、Stataの推定コマンド、つまりモデルをフィットするコマンドです。e()に格納されたコマンドは、次の推定モデルがフィットされるまで使用可能です。

sクラスコマンドは構文解析コマンドです。つまり、プログラマーが使用するコマンドで、入力したコマンドを解析する際に使用します。いくつかのコマンドは何でもs()に保存します。

cクラスコマンドというものはありません。c()は常に入手可能な値を格納しています。例えば、c(current_date)（今日の日付）、c(pwd)（現在の作業フォルダ）、c(N)（観測値の数）、などとなります。c()値には多くの種類があり、それらは[P] creturnに記載されています。

Stataの全てのコマンドは、rクラス、eクラス、sクラス、そしてもしコマンドが何も格納しない場合はnクラスのどれかに分類されます。rは「リターン」を意味し、これは結果を返す(リターンする)ためです。eは「エスティメーション」を意味し、推定(エスティメーション)結果からきています。sは「ストリング」(文字列)を意味しています。最後に、分かりにくいですが、nはヌル（ではない）にあたります。

何が保存されているのか確認するには、リファレンスマニュアルのそれぞれのコマンドの中にある*Stored results*セクションをご確認ください。コマンドのクラスが分かっている場合（それほど難しくないので、推測できます）、何を格納しているのか確認するにはreturn list, ereturn list, sreturn listの該当するものを入力します。

詳細は[R] Stored resultsと[U] 18.8 他のプログラムの計算結果にアクセスするをご確認ください。

13.7 行番号の指定方法

個々の変数の観測値は変数に行番号をつける事で参照できます。行番号の指定方法は変数名の後に大括弧[]内に表現を入れることで指定されます。行番号で表現する数値は整数になるように四捨五入します。そして、確定した観測値が返されます。もし行番号の値が1よりも小さい、あるいは_Nよりも大きい場合、欠損値が返されます。

13.7.1 ラグとリードを生成する

次のようなコマンドを入力すると、

```
. generate y = x
```

Stataはこの次のコマンドを入力したかのように認識します。

```
. generate y = x[_n]
```

つまり、yの最初の値に対応するのがxの最初の値で、yの2番目の観測値はxの2つ目の観測値に対応する、となります。例えば、次のように入力すると

```
. generate y = x[1]
```

それぞれのyの観測値をxの最初の値に紐付けします。例えば、次のように入力します。

```
. generate y = x[2]
```

それぞれのyの観測値をxの2番目の値に紐付けします。例えば、次のように入力します。

```
. generate y = x[0]
```

Stataはyの値全てに欠損値を入力します。これは0番目の観測値は存在しないからです。同じことが次のように入力すると起こります。

```
. generate y = x[100]
```

この状態で、100個よりも少ない観測値があるときは、欠損値になります。

大括弧[]を入力すると、行番号の指定方法を設定します。行番号の指定方法の記述と_variable _nを使用するとラグ値を作成できます。変数xのラグ値は次のようなコマンドで入手できます。

```
. generate xlag = x[_n-1]
```

ラグ項やリード項について詳しく知りたい場合、おそらく、時系列データを使用していると想定できるので、時系列演算子、例えばL.xを使う方がいいかもしれません。時系列演算子は変数リストと共に使用でき、データ内のギャップについて考慮されるので、こちらの方がより安全で便利です。詳細は[U] 11.4.4 時系列変数リストと[U] 13.10 時系列演算子をご覧ください。時系列演算子を使用したい場合でも、上記の仕組みがどのようになっているのか、理解することも大切です。

ビルトインのアンダーバーがついた変数_nはStataの中で現在の観測値の番号を意味します。上記理由から、

```
. generate y = x[_n]
```

変数xの観測値1の中にあるxがyの中にある観測値1の方にコピーされ、他の観測値に関しても同じように操作することになっています。次のようなコマンドがあります。

```
. generate xlag = x[_n-1]
```

_n-1は1つ前の観測値を評価します。最初の観測値については、_n-1 = 0となるため、xlag[1]は欠損値となります。2つ目の観測値については_n-1 = 1となり、xlag[2]は x[1]の値を設定する、となります。

同じように、xのリードは次のように作成できます。

```
. generate xlead = x[_n+1]
```

この場合、新しい変数xleadの最後の値は欠損値となります。これは、_n+1は_N（データセット内にあるデータセット数）よりも大きくなるためです。

13.7.2　グループ内で行番号を追加する

コマンドの先頭にby varlist:プリフィックスがついている場合、行番号とアンダーバーをつけた_nと_Nは現在処理されているデータのサブセットに対応する形で評価されます。例えば、次の（何の変哲もない）データを見てみましょう。

```
. use https://www.stata-press.com/data/r18/gxmpl6
. list
```

	bvar	oldvar
1.	1	1.1
2.	1	2.1
3.	1	3.1
4.	2	4.1
5.	2	5.1

_n, _Nと行番号の指定方法がどのような仕組みになっているのか確認するために、新しい3つの変数を作成し、それぞれを実行して値をリストしましょう。

```
. generate small_n = _n
. generate big_n = _N
. generate newvar = oldvar[1]
. list
```

	bvar	oldvar	small_n	big_n	newvar
1.	1	1.1	1	5	1.1
2.	1	2.1	2	5	1.1
3.	1	3.1	3	5	1.1
4.	2	4.1	4	5	1.1
5.	2	5.1	5	5	1.1

small_n（これは_nと同じです）は1から5までの値を取ります。そしてbig_n（これは_Nと同じです）は5です。これは、それほど驚くべきことではありません。データ内には5つの観測値があります。_nは観測値の数を数えるのに使用し、_Nは合計の観測値を示します。変数newvarは oldvar[1]として定義され、その値は1.1です。確かに、変数oldvarの最初の値は1.1です。

では、これらと同じ3つのステップを繰り返しましょう。今回はby bvar:プレフィックスを使用します。まず、small_n, big_n, newvarの値をドロップして、まっさらな状態ではじめましょう。

```
. drop small_n big_n newvar
. by bvar, sort: generate small_n=_n
. by bvar: generate big_n =_N
. by bvar: generate newvar=oldvar[1]
. list
```

	bvar	oldvar	small_n	big_n	newvar
1.	1	1.1	1	3	1.1
2.	1	2.1	2	3	1.1
3.	1	3.1	3	3	1.1
4.	2	4.1	1	2	4.1
5.	2	5.1	2	2	4.1

結果は、先ほどのものとは異なります。先ほど、_nと_Nはbyグループを元に分けられた、サブセットのデータに対して適用されると説明しました。つまり、 small_n (_n)はbvar = 1の場合、1から3までの値をとり、bvar = 2の場合は1から2までの値をとります。big_n (_N)は最初のグループでは3、2番目のグループでは2になります。最後に、newvar (oldvar[1]) はそれぞれ1.1と4.1になります。

例題8

これまでの説明で、いくつか驚くような事ができます。

例えば、それぞれの州ごとのデータを有しており、そのデータの中に州を全部で4つの地域に分けるregionという変数が組み込まれているとします。またデータ内に変数xがあり、新しい変数avgxを作成して回帰に使用したい、とすると、この新しい変数は州がある地域ごとにxの平均を計算します。つまり、カリフォルニアでは、xの観測値とその地域内にある州の平均avgxがあります。次のように入力しましょう。

```
. by region, sort: generate avgx=sum(x)/_n
. by region: replace avgx=avgx[_N]
```

まず、by regionプリフィックスを使用して、「generate avgx」 では変数xの累積値をそのときまでの観測数で割ります。「, sort」オプションは変数regionの順番に応じてソートします。結果としてこのコマンドで、変数 regionグループ内の変数xにおける個別平均を作成します。この中で、この個別平均を元にした、regionグループ内の全体の平均になる、最後の値が目的の値です。つまり、「by region」ではavgxにある観測値を、その地域内の最後の値、avgx[_N]で置き換え (replace) ます。

これらのコマンドを入力すると確認できる情報は次の通りです。

```
. use https://www.stata-press.com/data/r18/gxmpl7, clear
. by region, sort: generate avgx=sum(x)/_n
. by region: replace avgx=avgx[_N] (46 real changes made)
```

この例題の中では、xの観測値の中で欠損となっているものはありません。もし欠損値があったとすると、誤った結果を入手していたことになります。個別平均を作成した際、次のように入力しました。

```
. by region, sort: generate avgx=sum(x)/_n
```

ここでの問題は、sum()関数に関連しているわけではありません。sum()が欠損値に遭遇すると、合計にゼロを入力します。ここでの問題は_nにあります。例えば、最初の回帰の2番目の観測値は、xに関して欠損値を算出したとしましょう。Stataがその地域の3番目の観測値を処理する際、2つの要素の合計を計算し、(そのうちの1つは欠損値です)合計を2ではなく、3で割ります。この問題に対しては、次のように対処します。

```
. by region: generate avgx=sum(x)/sum(x<.)
```

_nで値を割る代わりに、全ての欠損していない観測値数で割ります。つまり、sum(x<.)になります。

もし、目的が平均を求めるだけの場合、「egen avgx=mean(x), by(region)」と入力すればより簡単に必要な値を入手できます。詳細は[D] egenをご覧ください。egenはStataの中で書かれていますが、上記がegenのmean()関数が機能する仕組みです。一般的な箇所は理解しておいたほうがいいでしょう。

◁

例題9

ある実験の最中の患者のバイタルサインを記録したデータがあります。変数の一覧の中には変数patientがあり、これは患者の名前かIDを示します。変数timeはバイタルサインを取得した日付/時間/出来事を記録します。変数vitalはバイタルサインを入力しています。1つ以上のバイタルサインが記録してあることが想定されますが、今回は考え方を説明するため、変数は1つだけにします。それぞれの観測値は、患者と時間の組み合わせを表します。

では、この形式のデータが1,000人分あり、同じ患者のデータに対して患者の初期状態のバイタルサインを記録する新しい変数origを作成します。

```
. use https://www.stata-press.com/data/r18/gxmpl8, clear
. sort patient time
. by patient: generate orig=vital[1]
```

ここで、vital[1]は最初の患者の最初の値ではなく、現在の患者の最初の値を参照しています。これは、generateコマンドがby patientプリフィックスと共に使用しているためです。

◁

例題10

これらの患者のデータを使って、もうひとつ例題を確認しましょう。この患者のデータから新しいデータセットを作成して患者の個人情報、最初のバイタルサインを記録した時間とその値に追加して、最後のバイタルサインの時間と値も記録します。その中で、一人の患者につき1つの観察値として記録する場合、次のように入力しましょう。

```
. sort patient time
. by patient: generate lasttime=time[_N]
. by patient: generate lastvital=vital[_N]
. by patient: drop if _n!=1
```

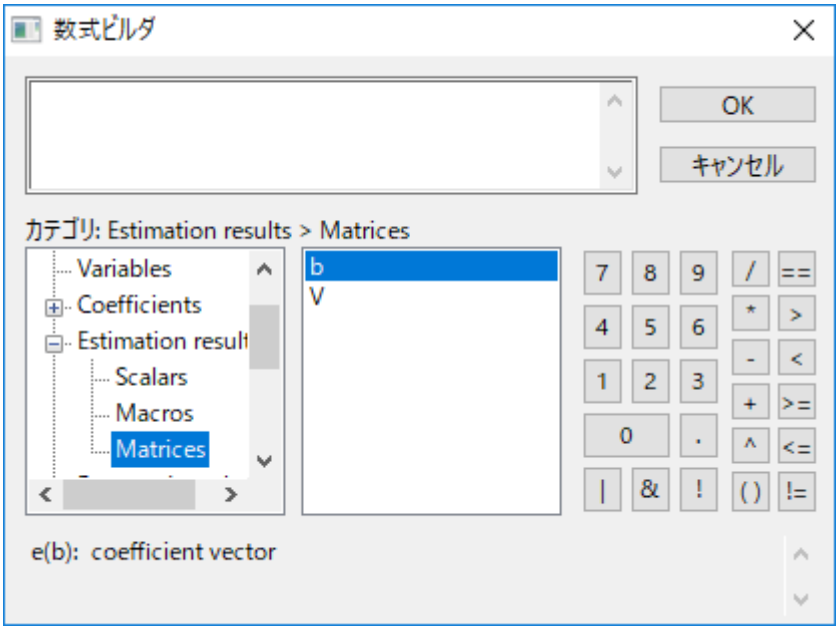
◁

行番号を指定したりグループ内で行番号を使用するさまざまな例題に関しては、Mitchell (2010, chap. 7)をご覧ください。

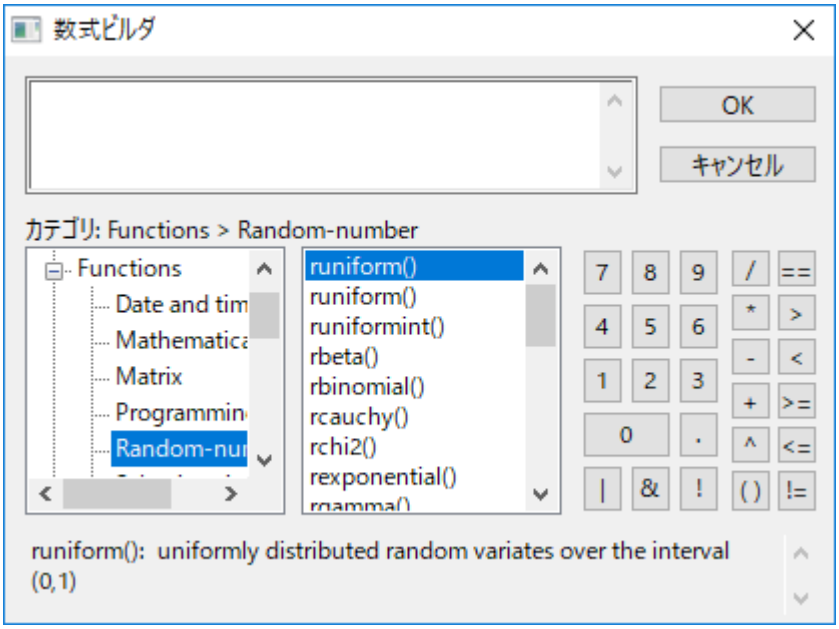
13.8 数式ビルダを使用する

Stataに搭載の数式ビルダは、前述の各関数を使用して計算式を作成できる便利なツールです。数式ビルダを開くには、コマンドのダイアログボックスでexpを入力できるボックスの右にある**作成...**ボタンをクリックします。

数式ビルダでは、数学定数、変数、システム限界、ローカルまたはグローバルマクロ、データセット、変数のメモなど、計算式に含めることのできるほぼ全ての数式をインタラクティブに閲覧し、追加することができます。特に、正確に名前を把握していない推定結果やシステム値へアクセスする場合に有用です。



数式ビルダ内で関数を選択すると、その説明と引数の順番が下部に表示になるので便利です。



数式ビルダの使用法についてはビデオ例題 をご覧ください。

13.9 因子変数のレベルによる指標値のレベル

Stataの因子変数はカテゴリカル変数の仮想指標変数にアクセスでき、その関係性を確認するのに使用しました。詳細は、[U] 11.4.3 因子変数と[U] 26 カテゴリデータと因子変数 をご覧ください。これらの仮想指標変数を表現に使用して、その仮想変数があたかもデータ内に存在しているかのように設定できます。もし、[U] 11.4.3 因子変数 内の因子変数の変数リストについて読んでいない場合、このセクションを読む前に読んでください。

もし、groupというカテゴリカル変数がありその中に値として1、2、3がある場合、次のような表現が利用できます。

```
. generate group1 = 1.group
```

この例では、 group = 1の場合は仮想因子変数1を入力し、group ≠ 1の場合は0を入力して、実際の変数group1にしました。これはgroupに欠損値がない場合のみ当てはまります。グループが欠損値を含む場合、1.groupが欠損している場合を追加する必要があります。

これらの仮想変数はそれぞれの関係にも拡張していきます。もうひとつの変数、sexがあり、そのなかで男性は0を女性は1を示すとき、次が成り立ちます。

```
. generate sex0grp2 = 0.sex#2.group
```

これは変数sex0grp2を作成します。この場合、sex = 0とgroup = 2が成り立つ場合に1が入ります。もしsexまたはgroupが . (欠損値)の場合は欠損値を返し、それ以外の場合は0を返します。

仮想指標変数はどんな表現（if表現を含む）でも利用できます。

❑ テクニカルノート

仮想指標変数を表すのに、省略形であるiのプリフィックスを外して今まで記述してきました。具体的に言うと、i2.groupではなく、2.groupと記述してきました。この、iプレフィックスを外すことができない場合が3つあります。それは変数名がe, d, xのどれかに当てはまる場合です。これらの文字は数字を表す際（例えば1e-3）にも使用するため、もうひとつの記述方法として1.e-3とも書き表せます。変数eがある場合、1.e-3の記述は数字（0.001）として認識すればいいのか、それとも仮想変数1.e から数字の3を引いたものか判断がつきません。長年の慣例により、この場合は数字0.001として認識されます。1.eを仮想指標変数として認識させるには、iプリフィックスを追加して、i1.eとする必要があります。

❑

13.10 時系列演算子

時系列演算子では、gnpのラグ項をL.gnpと入力すると入手できます。また、2番目のラグはL2.gnpと入力、と続きます。また、リード(またはフォワード)項（F）、差（D）、季節性の差（S）の演算子も存在します。

時系列演算子は変数リストや表現と共に使用できます。まだ読んでいない場合、[U] 11.4.4 時系列変数リストをあわせてご覧ください。このセクションは時系列演算子をgenerateコマンドのような表現の中で使う方法を示します。しかし、新しい変数を作成する必要は必ずしもありません。時系列演算子を直接使用することができます。

13.10.1 ラグ、リード、差の生成

時系列分析の内容では、gnp[_n-2]と参照するよりもL2.gnpを使用するほうが好ましいです。これは、欠損値がある可能性があるためです。では、観測番号4はt = 25のデータを、観測番号5はt = 27のデータを有しているとします。L2.gnpは正しい値を示します。L2.gnpの観測番号5は観測番号4の値になります。これは時系列演算子がtを探して表示するからです。機械的なgnp[_n-2]は観測を2つ分戻りますが、これは目的の結果ではありません。

これと同じ考え方が差を求めるときも使われます。この例では、D.gnpは観測番号5(t = *27*)が欠損しています。これは、t = *26*の値が無いため、 t = *27*の1つ分の差は計算することができないためです。

時系列演算子は変数リストや数式と共に使用できるので、次のように入力します。

```
. regress val L.gnp r
```

または

```
. generate gnplagged = L.gnp
. regress val gnplagged
```

どちらかを入力する前に、tssetコマンドを使用してStataに時系列変数であることを指示する必要があります。詳細は[TS] tssetをご覧ください。データをtssetした後に構文構造の中でexpとある場合、時系列演算子を使用した変数を入力できます。

```
. summarize r if F.gnp < gnp
```

または

```
. generate grew = 1 if gnp > L.gnp & L.gnp < .
. replace grew = 0 if grew >= .& L.gnp < .
```

または

```
. generate grew = (gnp > L.gnp) if L.gnp < .
```

13.10.2 時系列演算子と因子変数

変数リストでは、因子変数はL(ラグ)とF(リード)のような時系列演算子と組み合わせて使用できます。変数を生成して指標子groupの2レベル分(group = 2)のラグを生成する場合は、次のように入力します。

```
. generate lag2group = 2L.group
```

演算子は表現を入力できる場所であれば、どこにでも使用できます。2レベル目のラグがグループ1に属する物を選ぶには「if i2L.group」と入力しましょう。

これらは関係を表すものと組み合わせることができます。sex = 1とgroup = 3の関係に関するラグを作成するには、次のように入力します。

```
. generate lag1sexX3grp = 1L.sex#2L.group
```

因子変数と時系列演算子についての詳細は[U] 11.4.3 因子変数と[U] 11.4.4 時系列変数リストをご覧ください。

13.10.3 グループの演算子

Stataはパネル、あるいは断面的な時系列データも理解します。例えば、次のように入力します。

```
. tsset country time
```

これで、時系列データを使用することを宣言しています。時間変数はtimeで、異なる国ごとに時系列データをまとめています。

断面的と時間の識別子に対してtssetすると、簡単な時系列データを有しているように処理できます。

```
. generate grew = (gnp > L.gnp) if L.gnp < .
```

これで、正しい結果を入手できます。L. 演算子はパネルの最後にある観測値と次のパネルの最初にある観測値を混同することはありません。

13.10.4 ビデオ例題

Time series, part 3: Time-series operators

13.11 値ラベル

[U] 12.6 データセット，変数，値ラベルのセクションをまだ読んでいない場合、まず読んできてください。ラベルは数値の代わりに表現の中で使用したり、関連している値を表現する際に使用します。この使用法を使う場合、ラベルをダブルクオテーションの中に入力してからコロンで続け、値ラベルを続けます。

例題11

値ラベルyesnoは、1のときにyesのラベルを、0のときにnoを表示します。そうすると、"yes":yesno (yesnoの中でyesの値) は1と評価されます。ダブルクオテーションでラベルが定義されていない場合、または値ラベルそのものが見つからなかった場合、欠損値が返されます。つまり、"maybe":yesnoは欠損として評価されます。

```
. use https://www.stata-press.com/data/r18/gxmpl9, clear
. list
```

	name	answer
1.	Mikulin	no
2.	Gaines	no
3.	Hilbe	yes
4.	DeLeon	no
5.	Cain	no
6.	Wann	yes
7.	Schroeder	no
8.	Cox	no
9.	Bishop	no
10.	Hardin	yes
11.	Lancaster	yes
12.	Poole	no

```
. list if answer=="yes":yesno
```

	name	answer
3.	Hilbe	yes
6.	Wann	yes
10.	Hardin	yes
11.	Lancaster	yes

上記の例題の場合、変数answerは文字列変数ではありません。これは数値変数で、その数値が値ラベルyesnoと関連付けられているだけです。yesnoでは1のときにyesを、0のときにnoを表示しているので、「list if answer==1」と入力しても、同じ結果を得ることができます。しかし、「list if answer=="yes"」とは入力できません。これは変数answerは文字列変数ではないからです。入力すると、“type mismatch”というエラーメッセージが返されます。

◁

13.12 精度とそれに関する問題

以下の、Stataの簡単なセッションを確認してください。

```
. drop _all
. input x y
            x           y
  1. 1 1.1
  2. 2 1.2
  3. 3 1.3
  4. end
. count if x==1
      1
. count if y==1.1
      0
```

```
. list
```

	x	y
1.	1	1.1
2.	2	1.2
3.	3	1.3

2つの変数、xとyを有するデータセットを作成しました。1番目の観測値として、xには1が、yには1.1が入力されています。何回変数xが数値1を有していたか数えるコマンド(countコマンド)を使用すると、1回入力されていたことを示します。同じように、何回変数yが数値1.1を有していたか数えるコマンド(countコマンド)を使用すると、1回も無かったことを示します。何がおかしいのでしょうか。データをlistコマンドでリスト表示すると、変数yの最初の観測値は1.1であることが分かります。

表示される内容は異なっているように見えますが、Stataは間違えていません。Stataは数字を最初は2進数で保存します。1.1という数字は2進数で正確に表現することができません。つまり、有限な2進数で1.1と等しい値は存在しないのです。

❑ テクニカルノート

1.1という数字は、2進数では1.0001100110011...と表されます。このピリオドは2 進数の桁数を表します。2進数のコンピュータが遭遇する1/10のような数字を格納する問題は、人間のような10進数で数字を表す際の1/11、つまり0.0909090909...の扱いと似ています。

2進数の精度に関する詳細とStataがどのように2進数のフローティング桁数を保存するのか確認するにはGould (2011a)をご覧ください。

❑

リスト表示の中で1.1と表示しているのは、実際には1.1000000238419となります。これは10^{8}桁目で2つほど外れています。Stataに対して明確に指示しない限り、全ての数字はfloatという保存形式で保存されます。この形式は単一精度または4バイト実数とも呼ばれます。反対に、Stataは内部の計算を全てdoubleという保存形式で行っています。これは、倍精度あるいは8バイト実数とも呼ばれます。この部分が難しく、複雑な状況にしています。

上記の例ではfloatの保存形式で1.1という数字と、doubleの保存形式で1.1という数字を比較しました。倍精度の1.1の表し方は単一精度の表記よりもより近いですが、それでも異なります。つまり、これら2つの数字は異なるものとして認識されます。

この問題を回避する方法はいくつかあります。1.1を1.1と同じであると扱っていないという問題は、保存形式の精度と内部で計算する精度が同じ場合は問題になりません。つまり、全てのデータをdoubleとして保存すれば、問題はありません。ただし、この方法ではコンピュータのメモリをより多く使用します。しかし、データがその精度を維持する必要があるとは考えにくく、計算に多大な影響を与えるとは考えにくくなります。

❑ テクニカルノート

このような保存形式による数値の不一致が計算結果に影響を当たる可能性は、低いです。先ほどもありましたが、Stataは内部の計算を全て倍精度で行っているため、計算結果の不一致は起こりにくくなっています。初めから単一精度のみを扱っていれば、また話は異なってきますが、現時点では注意する必要は無いでしょう。

❑

もうひとつの解決方法として、float()関数を使う方法があります。float(x)はxの値をfloatまで精度を下げて計算します。 先ほどの例で、「count if y==float(1.1)」と入力していた場合、その値は1つある、という結果を得ていたことになります。

13.13 参考文献

Cox, N. J. 2006. Stata tip 33: Sweet sixteen: Hexadecimal formats and precision problems. *Stata Journal* 6: 282-283.

———. 2011a. Speaking Stata: Compared with ..... *Stata Journal* 11: 305-314.

———. 2011b. Speaking Stata: Fun and fluency with functions. *Stata Journal* 11: 460-471.

———. 2011c. Stata tip 96: Cube roots. *Stata Journal* 11: 149-154.

Cox, N. J. and C. B. Schechter 2019. Speaking Stata: How best to generate indicator or dummy variables. *Stata Journal* 19: 246-259.

Crow, K. 2012. Building complicated expressions the easy way. The Stata Blog: Not Elsewhere Classified. http://blog.stata.com/2012/02/07/building-complicated-expressions-the-easy-way/.

Gould, W. W. 2006. Mata Matters: Precision. *Stata Journal* 6: 550-560.

———. 2011a. How to read the %21x format, part 2. The Stata Blog: Not Elsewhere Classified. http://blog.stata.com/2011/02/10/how-to-read-the-percent-21x-format-part-2/.

———. 2011b. Precision (yet again), Part I. The Stata Blog: Not Elsewhere Classified. http://blog.stata.com/2011/06/17/precision-yet-again-part-i/.

———. 2011c. Precision (yet again), Part II. The Stata Blog: Not Elsewhere Classified. http://blog.stata.com/2011/06/23/precision-yet-again-part-ii/.

———. 2012. The penultimate guide to precision. The Stata Blog: Not Elsewhere Classified. http://blog.stata.com/2012/04/02/the-penultimate-guide-to-precision/.

Linhart, J. M. 2008. Mata Matters:Overflow, underflow and the IEEE floating-point format. *Stata Journal* 8:255-268. Mitchell,

M. N. 2010. *Data Management Using Stata: A Practical Handbook*. College Station, TX: Stata Press.

Weiss, M. 2009. Stata tip 80: Constructing a group variable with specified group sizes. *Stata Journal* 9: 640-642.

14 行列の表現

目次

14.1 概要

Stataには2種類の行列プログラム言語があります。1つはStataの古い行列言語と呼ばれ、もうひとつはMataです。StataのMataは新しい言語で、これら2つの言語の間には、一言では説明できない関係があります。

この章ではStataの古いほうの行列言語について説明します。そして、新しい方の言語は他のマニュアル*Mata Reference Manual* を確認するか、「help mata」とコマンドを入力して確認してください。

新しいMataの方がほぼ全ての領域において良いといえるでしょう。しかし、今でも古い行列言語が使われているのはStataがより良く、深く理解しているためです。MataがStataの情報などを受け渡す場合、行列を指定しても、Mataは古い方の行列言語を使用しなければいけません。つまり、ユーザは古い言語と新しい言語、2つを知っている必要が出てきます。

実際はこの説明から想像されるほど難しいわけでも混乱を招く訳でもありません。Stataの古い行列言語は簡単に使用でき、それほど難しいわけではありません。ただ、非常に多くのプログラミングを行う場合、Mataの言語も理解しておいたほうがいいでしょう。

14.1.1 行列の定義

Stataの行列の定義は、行列の数学部分を越える細かい内容も含みます。Stataにとって、行列はr × cが成り立つ名前のついた存在で、倍精度の数字(欠損値を含む)が四角形のように並べられ、それらは行と列の名前により区別されています。行列の次元については[R] **Limits**をご覧ください。

```
. matrix list A A[3,2]
    c1  c2
r1   1   2
r2   3   4
r3   5   6
```

ここに、3 × 2の行列Aがあり、要素としては1, 2, 3, 4, 5, 6が含まれています。1行2列目（数学では$A_{1,2}$と記述し、StataではA[1,2]と記述）は2を格納しています。列はc1と c2、行は r1, r2, r3という名前がついています。これらはStata機械的につけるデフォルトの名前です。数学では、行や列の名前には特に役割はありませんが、出力をラベル付けする際には、とても大きな助けになります。

名前は数字のように操作可能です。例えば、以下のようになります。

```
. matrix B=A'*A
. matrix list B symmetric B[2,2]
            c1   c2
c1  35
c2  44   56
```

B = A'Aであると定義しました。Bの行名と列名は同じです。積算はどのa × b と b × cが成り立つ行列でも定義でき、結果はa × cになります。よって、結果の行列の中で、行名は最初の行列の行の名前で、列名は2番目の行列の列の名前になります。最初の行列、Aを置き換える形でA'Aを作成します(そのときに名前も変わりました)。そして、表示されている名前を取得します。

14.2 行と列の名前

行列の行と列には常に名前があります。Stataは行列が作成されたときに名前を設定しますが、行列の計算の中で演算子がこれらの名前を変更していきます。よって、一般的に、行列の計算の結果が出るときに正しい名前が設定されます。

例えば、行列の計算で、b = $(X'X)^{-1}X'y$を実際にデータに行ってみます。

```
. use https://www.stata-press.com/data/r18/auto
(1978 automobile data)
. matrix accum XprimeX = weight foreign (obs=74)
. matrix vecaccum yprimeX = mpg weight foreign
```

```
. matrix b = syminv(XprimeX)*yprimeX'
. matrix list b b[3,1]
                mpg
 weight  -.00658789
foreign  -1.6500291
  _cons   41.679702
```

結果に表示される行や列の名前はユーザがStataに何か特殊なコマンドを使用して指示した、というわけではありません。XprimeX行列を作成したとき、結果は次のようになります。

```
. matrix list XprimeX symmetric XprimeX[3,3]
             weight     foreign        cons
 weight  7.188e+08
foreign       50950          22
  _cons      223440          22          74
```

matrix accumコマンドでX'X行列を作成して、そのときに使用した変数名から行と列の名前を設定します。(1,1)の要素はweight観測値の二乗の合計で、(2,1)要素はweightとforeignを掛け合わせたものの合計なので、名前は正しいといえます。

同じように、matrix vecaccumはy'X行列を作成し、使用した変数名を行と列の名前に使用します。つまり、「matrix vecaccum yprimeX = mpg weight foreign」は次のような結果をもたらします。

```
. matrix list yprimeX yprimeX[1,3]
      weight   foreign    _cons
mpg  4493720       545      1576
```

最後のステップ、「matrix b = invsym(XprimeX)*yprimeX'」では名前を変更しています。ゆっくりと気をつけて考えれば、そのルールはご自身で導き出すことができます。invsym()（逆数）は変換と似ています。そのため、行と列の名前は交換されます。しかし、ここでは、行列は対称なので、名前はそのままにされます。行列の乗算は、1つ目の行列の列の名前と2つ目の行列の行の名前を取ります。最終的な結果は次のようになります。

```
. matrix list b
b[3,1]
                mpg
 weight  -.00658789
foreign  -1.6500291
  _cons   41.679702
```

すると、この行列の解釈はmpg = −0.00659 weight − 1.65foreign + 41.68 + eとなります。

昔、研究者たちは行列表記を使用すると、複雑な計算の説明を簡単にできることに気がつきました。もし、行列を利用することに気がつかなかったとしたら、どうなっていたでしょうか。行列要素を直接指定する長く、複雑な計算式ごとに、さまざまな名前をつけて計算に利用していたと想像できます。

14.2.1 行と列の名前の意義

ほとんどの場合、Stataの行列はプログラムの推定子で使用し、Stataは行と列の名前を使用して見やすい結果を出力しています。例えば、あるコードを、インタラクティブあるいはプログラムで記述します。そのコードは、次のようなベクターbと共分散行列Vを作成します。

```
. matrix list b
b[1,3]

           weight   displacement          cons
y1     -.00656711      .00528078      40.084522
. matrix list V
symmetric V[3,3]

                    weight   displacement          cons
      weight      1.360e-06
displacement      -.0000103      .00009741
       _cons     -.00207455      .01188356      4.0808455
```

これで、あと2行コードを追加すると、一般的な推定出力を算出できます。

```
. ereturn post b V
. ereturn display
```

	Coefficient	Std. err.	z	P>|z|	[95% conf.	interval]
weight	-.0065671	.0011662	-5.63	0.000	-.0088529	-.0042813
displacement	.0052808	.0098696	0.54	0.593	-.0140632	.0246248
_cons	40.08452	2.02011	19.84	0.000	36.12518	44.04387

Stataのereturnコマンドがこの出力形式で結果を表示するのは、係数ベクトルと分散行列の行と列の名前が原因です。また、通常、bとVを生成するために、特殊なことを行う必要はありません。これは、行列の名前がどのように働くのかを踏まえ、自動的に作成されるためです。

行と列の名前はプログラムを作成する上でエラーを見つける手助けにもなります。行列bとVを作成するためにコードを入力しましたが、そこにエラーがある場合、誤った行や列の名前を出力します。先ほど作成したベクトルbではなく、次のような結果を出力する可能性があります。

```
. matrix list b
b[1,3]

           weight           c2          cons
y1     -.00656711        42.23      40.084522
```

この状態で推定結果をpostする場合、Stataはこのコマンドを拒絶します。Stataは行列の名前から、誤りがあることが想定したためです。

```
. ereturn post b V
name conflict
r(507);
```

なお、Stataは行列の演算に対する、一般的なルールに従います。名前はその中の一環です。行列は位置により合計されます。つまり、C = A + Bを作成するという指示は、どのような名前になるかにかかわらず、$C_{11} = A_{11} + B_{11}$を作成します。また、異なる行列名を足し合わせることもエラーにはなりません。

```
. matrix list a
symmetric a[3,3]
               c1          c2          c3
    mpg       14419
 weight     1221120   1.219e+08
  _cons         545       50950          22
. matrix list b
symmetric b[3,3]

                    c1          c2          c3
displacement   3211055
         mpg    227102       22249
       _cons     12153        1041          52
. matrix c =  a + b
. matrix list c
symmetric c[3,3]

                     c1          c2          c3
displacement     3225474
         mpg     1448222   1.219e+08
       _cons       12698       51991          74
```

行列の行と列の名前は、出力結果を表示するのに使用します。

14.2.2 2部分の名前

行と列の名前はコロン(:)で分けられた、*数式名*:opvarnameの2つの部分から構成されています。

今までの例題を元に見てみると、数式名（equation_name）は空欄で、opvarnameは因子変数や時系列演算子を使用しない、とてもシンプルな変数名を表しています。数式名が空欄になるのは良くあることです。1つの数式のモデル（例えば、regress, probit, logistic）を実行します。そして、結果の行列を表示すると、opvarnameだけを使用している行と列の名前になります。

時系列データを使用する人は、行や列の名前がopvarname形式で保存されていることに気をつけてください。時系列変数にとっては、opvarnameは時系列演算子であるL., D., L2D.をプレフィックスとして使用します。詳細は[U] 11.4.4 時系列変数リストをご覧ください。例えば、次のように表現されます。

```
. matrix list example1
symmetric example1[3,3]
                                      L.
                rate        rate      _cons
  rate     3.0952534
L.rate      .0096504    .00007742
 _cons    -2.8413483   -.01821928    4.8578916
```

この行列は変数rateとL.rateに線形回帰を行った後、共分散行列を取得したものになります。 L.rateという名前の行や列は、普通に変数rateと同じように考えていただくか、以前の例題のr1, r2, c1, c2, weight, foreignのようにお考えください。

行や列の名前もopvarname形式で保存されるので、因子変数を使用する時は注意してください。因子変数では、opvarnameは1つの仮想指標変数を参照して因子変数を作成するものを全て指します。例えば、3.groupはgroup = 3の時に1を入力して他の場合は0を入力する値を示す仮想変数を表します。1.sex#3.groupはsex = 1とgroup = 3が成り立つ時であれば、1でそれ以外の場合は0を示す仮想変数で、1.sex#c.ageはsex = 1で他の値が0の時にageの値を当てはめる仮想変数を示します。例えば、次のように表現されます。

```
. matrix list example2
symmetric example2[5,5]
                      0b.           1.       0b.sex#       1.sex#
                     sex           sex         c.age         c.age        _cons
  0b.sex               0
  1.sex                0    7.7785864
0b.sex#c.age           0    .08350827     .00231307
1.sex#c.age            0   -.09705697    5.606e-17      .00223195
  _cons                0   -3.2868185   -.08350827    -2.131e-15    3.2868185
```

1.sex#c.ageは先ほどのrateや L.rateと同じような行名と列名です。因子変数と有効な因子変数の構文に関しては、**［U］ 11.4.3因子変数, ［U］ 26 カテゴリカルデータと因子変数の操作, ［U］ 13.9 因子変数のレベルによる指標値のレベル of, ［U］ 20.12 係数の操作**をご覧ください。

因子変数演算子は時系列演算子であるL.や F.と組み合わせて使用することができます。つまり、opvarnameは1L.sex（sexの第1レベルの指標のラグ）や3L2.group（groupの第3レベルの2番目のラグ）のようになります。

数式名は区切られた行列をラベル付けする際に使用し、推定では複数の式を使用する場合で利用します。これがequation_nameのついた行列と、シンプルな（計算されていない）opvarnameの例です。

```
. matrix list example3
symmetric example2[5,5]
                        mpg:          mpg:          mpg:          mpg:          mpg:
                    foreign         displ         _cons       foreign         _cons
  mpg:foreign     1.6483972
    mpg:displ      .004747     .00003876
    mpg:_cons    -1.4266352  -.00905773      2.4341021
weight:foreign   -51.208454  -4.665e-19      15.224135     24997.727
  weight:_cons    15.224135   2.077e-17    -15.224135    -7431.7565     7431.7565
```

これが、equation_nameと演算する変数名を含んだ例になります。

```
. matrix list example4
symmetric example3[5,5]
                        val:          val:          val:       weight:       weight:
                                        L.
                       rate          rate         _cons       foreign         _cons
     val:rate     2.2947268
   val:L.rate      .00385216      .0000309
    val:_cons    -1.4533912     -.0072726     2.2583357
weight:foreign   -163.86684     7.796e-17     49.384526     25351.696
 weight:_cons     49.384526   -1.566e-16    -49.384526      -7640.237      7640.237
```

val:L.rateは列名です。これは、先ほどのセクションであったように、c2と foreignが列名であったことと変わりません。

例えば、この最後の行列が作成したプログラムで生成した分散行列で、そのプログラムは係数ベクトルbも算出した、とします。

```
. matrix list b
b[1,5]
            val:          val:          val:       weight:       weight:
                            L.
           rate          rate         cons       foreign         cons
y1     4.5366753   -.00316923      20.68421    -1008.7968     3324.7059
```

これが、結果をまとめて表示した表です。

```
. ereturn post b example4
. ereturn display
```

	Coefficient	Std.err.	z	P>|z|	[95% conf.interval]	
val						
rate						
L1	4.536675	1.514836	2.995	0.003	1.567652	7.505698
	-.0031692	.0055591	-0.570	0.569	-.0140648	.0077264
_cons	20.68421	1.502776	13.764	0.000	17.73882	23.6296
weight						
foreign	-1008.797	159.2222	-6.336	0.000	-1320.866	-696.7271
_cons	3324.706	87.40845	38.036	0.000	3153.388	3496.023

matrix listを使用して行と列の名前を表示してきましたが、これはmatrix list は全ての行列で使用可能だからです。推定結果で操作を行っている時は、より便利に行列の行と列の名前を参照する方法があります。これは、推定結果は分散行列の行や列の名前と同じ名前があり、それとベクトル係数の列の名前でもあります。より良い方法は、ほぼ全ての推定コマンドで使用可能な、coeflegend表示オプションを使用することです。例えば、次のように表現されます。

```
. use https://www.stata-press.com/data/r18/fvex
(Artificial factor variables' data)
. generate t = _n
. tsset t
  (省略)
. sureg (y = sex##group) (distance = d.age il2.sex)
  (省略)
```

```
. sureg, coeflegend
Seemingly unrelated regression
```

Equation	Obs	Params	RMSE	"R-squared"	chi2	P>chi2
y	2,998	5	20.03657	0.1343	464.08	0.0000
distance	2,998	2	181.3797	0.0005	0.92	0.6314

	Coefficient	Legend
y		
sex		
female	21.59726	_b[y:1.sex]
group		
2	11.42832	_b[y:2.group]
3	21.6461	_b[y:3.group]
sex#group		
female#2	-4.892653	_b[y:1.sex#2.group]
female#3	-6.220653	_b[y:1.sex#3.group]
_cons	50.5957	_b[y:_cons]
distance		
age		
D1.	.2230927	_b[distance:D.age]
L2.sex		
female	1.300898	_b[distance:1L2.sex]
_cons	57.96172	_b[distance:_cons]

「matrix list e(V)」や「matrix list e(b)」を実行すれば行や列の名前は確認できますが、matrix listでは記載できる空間に限りがあるため、読みにくい表示になるでしょう。coeflegendでは、名前は結果の表の中のLegend列で簡単に確認できます。matrix listコマンドとcoeflegend オプションの違いは、coeflegend は名前を_b[]で囲むことです。これは、coeflegend の本来の目的は数式や推定後コマンドなどで係数をどのように入力するか示すオプションだからです。詳細は[U] 13.5 係数と標準誤差にアクセスすると[U] 20.12 係数の操作をご覧ください。そのなかで、_b[]は必須になります。

14.2.3 行と列の名前を設定する

行や列の名前を再設定するにはmatrix rownamesとmatrix colnamesコマンドを使用します。

名前を再設定する前に、matrix listを実行して名前が誤っていることを確認します。多くの場合、これらは自動的に修正されます。行列を手で入力する場合、行の名前は機械的に r1, r2,...,となり、列の名前はc1, c2,...となります。

```
. matrix a = (1,2,3¥4,5,6)
. matrix list a a[2,3]
    c1  c2  c3
r1   1   2   3
r2   4   5   6
```

現在の行や列の名前が何かにかかわらず、matrix rownamesと matrix colnamesは名前を再設定できます。

```
. matrix colnames a = foreign alpha _cons
. matrix rownames a = one two
. matrix list a a[2,3]
     foreign     alpha     _cons
one        1         2         3
two        4         5         6
```

*演算子*をopvarnameの一部として設定できます。

```
. matrix colnames a = foreign l.rate _cons
. matrix list a a[2,3]
                    L.
     foreign      rate     _cons
one        1         2         3
two        4         5         6
```

ここで設定する値は仮想因子変数の指標となりえます。これらの名前は ベース(b.)とオミット (o.)の指標です。

```
. matrix colnames b = 0b.sex 2o.arm 1.sex#c.age 1.sex#3.group#2.arm
. matrix list b b[2,4]
                                        1.sex#
            0b.         2o.      1.sex#   3.group#
           sex         arm       c.age      2.arm
one          1           2           3          3
two          5           6           7          8
```

因子変数演算子に関する詳細は[U] 11.4.3 因子変数をご覧ください。また、数式名を設定できます。

```
. matrix colnames a = this:foreign this:l.rate that:_cons
. matrix list a a[2,3]
        this:     this:     that:
                    L.
     foreign      rate     _cons
one        1         2         3
two        4         5         6
```

詳細は[P] matrix rownamesをご覧ください。

14.2.4　行と列の名前を取得する

matrix listは行列の行や列の名前を表示します。プログラミングでは、マクロのlocalを使用すると行や列の名前を確認できます。

```
local ...: rowfullnames matname
local ...: colfullnames matname
local ...: rownames matname
local ...: colnames matname
local ...: roweq matname
local ...: coleq matname
```

rowfullnamesとcolfullnamesはフルの名前（*数式名*:opvarnames）をリストして表示します。

rownamesとcolnamesは数式を飛ばしてopvarnamesのみを返します。これも、一覧をリストで表示します。

roweqとcoleqは順にリストされた数式の名前を返します。詳細は[P] **macro**と[P] **matrix define**をご覧ください。

14.3 ベクトルとスカラー

Stataはベクトルを有していません。ベクトルは行列の特殊なケースとして位置づけられ、matrixコマンドで制御されます。

Stataにスカラーはありますが、ベクトルと同じように特殊なケースとして扱うことができるので、必要性は低いでしょう。スカラーの詳細については[P] **scalar**をご覧ください。

14.4 行列を手で入力する

行列を入力するときには、次のコマンドを使用します。

```
matrix input matname = (...)
```

または

```
matrix matname = (...)
```

どちらの場合でも、行列は行ごとに入力します。ひとつずつの要素はカンマ（,）で区切り、改行はバックスラッシュ(¥)で表記します。inputという言葉を入力しない場合、数式パーサーを使用して行列を入力することになります。

```
. matrix a = (1,2¥3,4)
. matrix list a a[2,2]
    c1  c2
r1   1   2
r2   3   4
```

この利点は、どの要素に対しても数式を使用できることです。

```
. matrix b = (1, 2+3/2 ¥ cos(_pi), _pi)
. matrix list b b[2,2]
           c1          c2
r1          1         3.5
r2         -1   3.1415927
```

一方、この記述方法は行列の大きさが小さい(50要素程度)必要があります。

matrix inputはそのような制限はありません。しかし、各要素に対して数式は使用できません。

```
. matrix input c  = (1,2¥3,4)
. matrix input d = (1, 2+3/2 ¥ cos(_pi), _pi)
invalid syntax
r(198);
```

行列を入力し終わった後に行の名前や列の名前を設定する場合は、その方法を前のセクション、[U] 14.2.3 行と列の名前を設定するで紹介しているので確認してください。

小さな行列については、ダイアログボックスで入力することもできます。メニューで **データ** > **行列(ado言語)** > **行列を手入力**と操作するか、コマンドウィンドウに「db matrix input」と入力しましょう。ダイアログボックスは、小さくて対称的な行列では特に便利です。

14.5 Stataのコマンドで作成した行列にアクセスする

Stataの全ての推定コマンドを含むいくつかのコマンドでは、その後の操作で使用できる行列を準備します。推定コマンドを実行した後、「ereturn list」と入力すると何を使用できるか確認できます。

```
. use https://www.stata-press.com/data/r18/auto
(1978 automobile data)
. probit foreign mpg weight
  (省略)
. ereturn list
scalars:
                e(rank) =  3
                   e(N) =  74
                  e(ic) =  5
                   e(k) =  3
                e(k_eq) =  1
                e(k_dv) =  1
          e(converged) =  1
                  e(rc) =  0
                  e(ll) =  -26.84418900579868
          e(k_eq_model) =  1
                e(ll_0) =  -45.03320955699139
                e(df_m) =  2
                e(chi2) =  36.37804110238542
                   e(p) =  1.26069126402e-08
                e(N_cdf) =  0
                e(N_cds) =  0
                e(r2_p) =  .4039023807124773

macros:
             e(cmdline) : "probit foreign mpg weight" e(cmd) : "probit"
            e(estat_cmd) : "probit_estat"
              e(predict) : "probit_p"
           e(marginstok) : "default Pr"
         e(marginsnotok) : "stdp DEviance SCore"
                e(title) : "Probit regression"
            e(chi2type) : "LR"
                 e(opt) : "moptimize" e(vce) : "oim"
                e(user) : "mopt   probit_d2()"
            e(ml_method) : "d2"
            e(technique) : "nr"
               e(which) : "max"
              e(depvar) : "foreign"
          e(properties) : "b V"
matrices:
                   e(b) :  1 x 3
                   e(V) :  3 x 3
                e(mns) :  1 x 3
              e(rules) :  1 x 4
               e(ilog) :  1 x 20
           e(gradient) :  1 x 3
functions:
              e(sample)
```

ほとんどの推定コマンドはe(b)（係数ベクトル）とe(V)（推定子の分散-共分散行列）は残します。

```
. matrix list e(b)
e(b)[1,3]
       foreign:     foreign:     foreign:
           mpg       weight        _cons
y1  -.10395033   -.00233554     8.275464
```

どの行列表現でも e(b) と e(V) は参照できます。

```
matrix ‘b’ = e(b)
. matrix list myb
myb[1,3]
       foreign:     foreign:     foreign:
           mpg       weight        _cons
y1   -.10395033   -.00233554    8.275464
. matrix c = e(b)*invsym(e(V))*e(b)’
. matrix list c symmetric
c[1,1]
             y1
y1  22.440542
```

14.6 データを累積して行列を作成する

推定子のプログラミングを行っているとき、XとZをデータ行列としたX’X, X’Z, X’WX, X’WZ形式の行列はしばしば作成されます。matrix accum, matrix glsaccum, matrix vecaccum, matrix opaccum はこれらのような行列を作成します。詳細は[P] **matrix accum**をご覧ください。

データを直接行列にロードして数式パーサーを使用して行列を作成する方法はお勧めしません。しかし、これを行いたい場合は[P] **matrix mkmat**をご覧ください。また、[P] **matrix mkmat**の最後にあるテクニカルノートは必ず読むようにしてください。matrix accumコマンドの使用法については、さまざまなお勧めがあります。

14.7 行列演算子

新しい行列を作成したり、既存の行列を置き換えるには、次のように入力します。

```
matrix matname = matrix_expression
```

例えば、以下のようになります。

```
. matrix A = invsym(R*V*R’)
. matrix IAR = I(rowsof(A)) - A*R
. matrix beta = b*IAR’ + r*A’
. matrix C = -C’
. matrix D = (A, B ¥ B’, A)
. matrix E = (A+B)*C’
. matrix S = (S+S’)/2
```

以下の演算子が提供されています。

	演算子	記号
	単項演算子	
	マイナス	-
	転置	'
2項演算子		
	(優先順位が低い)	
	行をつなげる	¥
	列をつなげる	,
	加算	+
	減算	-
	乗算	*
	スカラーによる除算	/
	クロネッカー積	#
	(優先順位が高い)	

括弧は計算の順番を変更するのに使用できます。

特に、,と¥は演算子です。(1,2)は1 × 2行列(ベクトル)を作成し、(A,B)はrowsof(A) = rowsof(B)となるrowsof(A) × colsof(A)+colsof(B)行列を作成します。 (1¥2)は2 × 1 行列 (ベクトル) を作成し、(A¥B)はcolsof(A) = colsof(B)となるrowsof(A)+rowsof(B) × colsof(A)行列を作成します。つまり、数式は次のようになります。

```
matrix R = (A,B)*Vinv*(A,B)'
```

これも、Stataで使用できます。

14.8 行列関数

これから紹介する関数に追加して、それぞれの用途に合わせて次の参照先もご確認ください。1つの値の分解に関しては[P] **matrix svd**を、対称行列における固有値と固有ベクトルに関しては[P] **matrix symeigen**を、そして非対称な行列における固有値に関しては[P] **matrix eigenvalues**をご覧ください。行列関数の詳細な説明は[FN] **Matrix functions**をご覧ください。

行列を返す行列関数：

cholesky(M)	I(n)	nullmat(matname)
corr(M)	inv(M)	sweep(M,i)
diag(v)	invsym(M)	vec(M)
get(systemname)	J(r,c,z)	vecdiag(M)
hadamard(M,N)	matuniform(r,c)	

スカラーを返す行列関数：

colnumb(M,s)	el(M,i,j)	rownumb(M,s)
colnfreeparms(M)	el(M,i,j)	rownfreeparms(M)
colnumb(M,s)	issymmetric(M)	rownumb(M,s)
colsof(M)	matmissing(M)	rowsof(M)
det(M)	mreldif(X,Y)	trace(M)

14.9 行番号

1. 行列とスカラー数式では、matname[r,c]（ここで、rとcはスカラー表現を示します）を使用してmatnameの1つの要素をスカラー値として使用できます。

   例題：
   ```
   matrix A = A / A[1,1]
   generate newvar = oldvar / A[2,2]
   ```

2. 行列表現では、matname[s_r,s_c]（s_rとs_cは文字列を示します）を使用すると、1つの要素のサブ行列を求めます。返される要素は行や列の名前を元に検索します。

   例題：
   ```
   matrix B = V["price","price"]
   generate sdif = dif / sqrt(V["price","price"])
   ```

3. 行列表現では2つの構文を組み合わせてmatname[r,s_c]や
   matname[s_r,c]を参照できます。

   例題：
   ```
   matrix b = b * R[1,"price"]
   ```

4. 行列表現では、matname[r_1..r_2,c_1..c_2]を使用してサブ行列を参照できます。r_1, r_2, c_1, c_2はスカラー数式でも使用可能です。r_2が欠損値の場合、それはmatnameの最後の行を参照しているとみなされます。c_2が欠損値の場合、matnameの最後の列を参照しているとみなされます。つまり、matname[r_1...,c_1...]は認められています。

   例題：
   ```
   matrix S = Z[1..4, 1..4]
   matrix R = Z[5..., 5...]
   ```

5. 行列表現では、matname[s_{r1}..s_{r2},s_{c1}..s_{c2}]を使用して、サブ行列を参照できます。s_{r1}, s_{r2}, s_{c1}, s_{c2}は文字列表現となります。返される行列は行や列の名前を確認しています。

   文字列表現が数式の名前のみの場合、その数式の行や列が返されます。

   例題：
   ```
   matrix S = Z["price".."weight", "price".."weight"]
   matrix L = D["mpg:price".."mpg:weight", "mpg:price".."mpg:weight"] matrix T1 = C["mpg:", "mpg:
   "]
   matrix T2 = C["mpg:", "price:"]
   ```

6. 行列表現では、上記のどの構文でも組み合わせて使用できます。例題：
   ```
   matrix T1 = C["mpg:", "price:weight".."price:displ"] matrix T2 = C["mpg:", "
   price:weight"...]
   matrix T3 = C["mpg:price", 2..5] matrix T4 = C["mpg:price",
   2]
   ```

行列の要素を定義する場合、次の表現を使います。

```
matrix matname[i,j] = expression
```

ここで、iとjはスカラー表現です。この場合、行列matnameは既に作成されている必要があります。
例題：

```
matrix A = J(2,2,0) matrix A[1,2] = sqrt(2)
```

7. 行列内のサブ行列を入れ替える場合、同じ構文を使用します。数式の右辺がスカラーとなるか1 × 1の行列の場合、要素が置き換えられます。数式の結果が行列となる場合、サブ行列の左上の要素(i,j)が置き換えられます。この場合、行列matnameは既に作成されている必要があります。

   例題：

```
matrix A = J(4,4,0) matrix A[2,2] = C'*C
```

14.10 スカラー数式で行列を使う

スカラー数式は、Stataのマニュアル内でexpと記載されます。

```
generate newvar = exp if exp ...
replace newvar = exp if exp ...
regress ... if exp ...
          if exp {... }
while exp {... }
```

最も重要な内容としては、スカラー数式はgenerateや replaceコマンド、多くのコマンドと共に使用できるif表現、プログラムコントロールで使用するifとwhileコマンドで使用します。

これらのコマンドの中で行列を参照するのは、ifとwhileコマンド以外はないでしょう。

どの場合でも、行列を参照することはできます。しかし数式の中で行列を評価し、その結果として行列を返すことはできません。つまり、trace(A)を参照することはできますが、trace(A+B)を参照することはできません。

数式の評価をする際に行列の評価が必要になるのか、それを見極めるのはとても難しいことです。上級ユーザも時として驚かされます。もし、“matrix operators that return matrices not allowed in this context”, r(509)というエラーメッセージを受け取ったら、このような状況になります。

これを解決するには、数式を2つに分けましょう。例えば、次のようになります。

```
if trace(A+B)==0 {
...
}
```

から

```
matrix AplusB = A+B
if trace(AplusB)==0 {
...
}
```

または、

```
matrix Trace = trace(A+B)
if Trace[1,1]==0 {
...

}
```

14.11 参考文献

Miura, H. 2012. Stata graph library for network analysis. *Stata Journal* 12: 94-129.

15 出力を保存し印刷する—ログファイル

目次

15.1 概要

Stataは実行しているセッションをログファイルと呼ばれるファイルに保存できます。しかし、自動的にログを取り始める訳では無いので、Stataにセッションの記録を開始するよう、指示する必要があります。デフォルトで、出力するログファイルはStata Markup and Control Language(SMCL)（詳細は[P] **smcl**をご覧ください）という形式で保存され、このファイルにはユーザが入力したコマンドとStataが出力した結果を記録します。ファイルは印刷したり、一般的なテキストファイルに変換できるので、ワープロソフトで作成した文書に組み込むことができます。

ログを開始するには：	. log using *ファイル名*
実行しているStataのセッションは*ファイル名*.smclのファイルに保存されます。	
一時的にログの記録を止めるには：	
一度ログを中断する：	. log off
再開：	. log on
ログを取るのを終了してファイルを閉じるには：	. log close
ファイル名.smclを印刷、または：	. translate*ファイル名*.smcl*ファイル名*.log
ファイル名.logに変換したので、ワープロファイルでロードできます。	
WindowsかMacでは、*ファイル名*.smclから	
PDFの作成ができる：	. translate*ファイル名*.smcl*ファイル名*.pdf

ログを開始する別の方法：	
既存のログに追加：	. log using *ファイル名*, append
既存のログファイルを置き換える:	. log using *ファイル名*, replace

GUIを使う：	
ログを開始するには：	**ログ**ボタンをクリック
ログの記録を中断するには：	**ログ**ボタンをクリックして、**中断**を選択
再開するには：	**ログ**ボタンをクリックして、**再開**を選択
ログを終了してファイルを閉じるには：	**ログ**ボタンをクリックして、**閉じる**を選択
前の、または現在のログを印刷するには：	**ファイル** > **表示...**と操作してファイルを選択してからビューワを右クリックして、**印刷**を選択

また、cmdlogはユーザが入力したコマンドだけを記録するログを作成します。結果を含んでいないログでも、セッションを再現するには十分です。

コマンドのみのログを開始するには：	. cmdlog using *ファイル名*
ログを取るのを終了してファイルを閉じるには：	. cmdlog close
セッションを再構築するには：	. do *ファイル名*.txt

15.1.1 ログの開始と終了

Stataを起動すると共に、log using sessionと入力（あるいは**ログ**ボタンをクリック）して今後の操作の記録を撮るログを開始します。

```
.log using  session
-------------------------------------------------------------------------------
      name:  <unnamed>
       log:  C:¥example¥session.smcl log type:  smcl
 opened on:  17 Mar 2023, 12:35:08

.use https://www.stata-press.com/data/r18/census5

(1980 Census data by state)

. tabulate region [freq=pop]

    Census |
    region |      Freq.     Percent        Cum.
-----------+-----------------------------------
        NE |  49,135,283       21.75       21.75
   N Cntrl |  58,865,670       26.06       47.81
     South |  74,734,029       33.08       80.89
      West |  43,172,490       19.11      100.00
-----------+-----------------------------------
     Total | 225,907,472      100.00

. summarize median_age
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
  median_age |         50       29.54    1.693445        24.2       34.7

. log close
      name:  <unnamed>
       log:  C:¥example¥session.smcl log type:  smcl
 closed on:  17 Mar 2023, 12:35:38
-------------------------------------------------------------------------------
```

これで、ディスクにsession.smclというファイルが保存されます。これをテキストエディタやワープロソフトで開くと、次のような画面になります。

```
{smcl}
{com}{sf}{ul off}{txt}{.-} name:  {res}<unnamed
      >
   {txt}log:  {res}C:¥example¥session.smcl
  {txt}log type:  {res}smcl
 {txt}opened on:  {res}17 Mar 2023, 12:35:08

{com}.use https://www.stata-press.com/data/r18/census5
{txt}(1980 Census data by state)

{com}.tabulate region [freq=pop]

    {txt}Census {c |}
     region {c  |}      Freq.     Percent        Cum.
{hline 12}{c +}{hline 35}
         NE {c |}{res}  49,135,283       21.75       21.75
{txt}     N Cntrl {c |}{res}  58,865,670       26.06       47.81
（省略）
```

この画面で確認できるものがSMCLと呼ばれる形式で、Stataが理解できる形式です。Stataのtypeコマンドを使用してコマンドを入力したときに表示する画面は次の通りです。

```
. type session.smcl
-------------------------------------------------------------------------------
      name:  <unnamed>
       log:  C:¥example¥session.smcl log type:  smcl
 opened on:  17 Mar 2023, 12:35:08

.use https://www.stata-press.com/data/r18/census5 (1980 Census data b
y state)

.tabulate region [freq=pop]
    Census
    region |      Freq.     Percent        Cum.
-----------+-----------------------------------
        NE |  49,135,283       21.75       21.75
   N Cntrl |  58,865,670       26.06       47.81
     South |  74,734,029       33.08       80.89
      West |  43,172,490       19.11      100.00
-----------+-----------------------------------
     Total | 225,907,472      100.00

. summarize median_age
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
  median_age |         50       29.54    1.693445        24.2       34.7

. log close
      name:  <unnamed>
       log:  C:¥example¥session.smcl log type:  smcl
 closed on:  17 Mar 2021, 12:35:38
-------------------------------------------------------------------------------

. _
```

今表示している画面は、先ほど表示した画面の完全な複製です。Stataを使用してファイルを印刷すると、このまま印刷されるので、完全なコピーになります。

SMCLファイルは単なるテキストファイルに変換できます。この方がワープロソフトなどに組み込む際には便利です。「translate *ファイル名*.smcl *ファイル名*.log」と入力すると、*ファイル名*.smclをテキストに変換して*ファイル名*.logが保存されます。

```
. translate session.smcl session.log
```

出力ファイルであるsession.logは次のようになります。

```
-------------------------------------------------------------------------------
      name:  <unnamed>
       log:  C:¥example¥session.smcl log type:  smcl
 opened on:  17 Mar 2023, 12:35:08

.use https://www.stata-press.com/data/r18/census5 (1980 Census data by
state)

.tabulate region [freq=pop]
    Census |
    region |      Freq.     Percent        Cum.
-----------+-----------------------------------
        NE |  49,135,283       21.75       21.75
   N Cntrl |  58,865,670       26.06       47.81
     South |  74,734,029       33.08       80.89
(省略)
```

translateコマンドを使用して*ファイル名*.smclを*ファイル名*.logに変換する際、*ファイル名*.logが存在しない事が条件になります。

```
. translate session.smcl session.log file session.log alr
eady exists
r(602);
```

既にファイルを作成済で、そのファイルを上書きする場合replaceオプションを使用します。

```
. translate session.smcl session.log, replace
```

詳細は[R] translateをご覧ください。

WindowsとMacではSMCLファイルをPDFに変換できます。これで簡単に共有可能です。

```
. translate session.smcl session.pdf
```

詳細は[R] translateをご覧ください。

もしSMCLが必要ないのであれば、直接、テキスト形式のログファイルを出力できます。操作としては下記のようにテキストオプションを指定する方法と、

```
. log using session, text
```

そしてファイル名を入力する際に.logファイルを指定する方法があります。

```
. log using session.log
```

logコマンドでログを開始または終了するときにヘッダーまたはフッターを表示しないようにするには、log usingまたはlog closeコマンドで、nomsgオプションを指定します。詳細は[R] logをご覧ください。

15.1.2 既存のログに追加する

Stataは既存のログファイルを意図せず上書きする操作は行いません既に同名のログファイルがあり、そのまま継続してログの記録を残したい場合、3つの選択肢があります。

- 新しいログファイルを作成する
- 新しいログを既存ログファイルに追加するコマンド「log using *ログ名*, append」を使う
- 既存のログファイルを置き換えるコマンド「log using *ログ名*, replace」を使う

例えば、既存のログファイルsession.smclが手元にあるときに次を入力します。

```
. log using session, append
```

すると、既存のログファイル、session.smclの最後に新しいログが追加されます。

15.1.3 ログを終了して再度開始する

セッションの記録を開始した後には、ロギングをオン/オフできます。ロギングをオフにすると、Stataはログの記録を取る事を中断しますが、ログのファイルは開いたままになります。この状態からロギングをオンに戻すと、Stataは継続してセッションの記録を続けます。つまり、ログをオフにする直前の行から続きを記録します。

例えば、何か興味深い結果が算出された場合、「log using results」と入力して（あるいは、**ログ**ボタンをクリックしてresults.smclを開いて）ください。それから、その結果を出力したコマンドを再入力します（あるいは、履歴ウィンドウにあるコマンドをダブルクリック、または*Pgup*キーを押してコマンドを再表示してからでも操作可能です。詳細は[U] 10 キーボードの使用をご覧ください）。これで目的の算出結果がログファイルに追加されます。

この後、ある程度解析を進めないとログに記録すべき解析結果は出力されない事が予想できます。「log close」と入力する代わりに「log off」を入力するか、**ログ**をクリックしてから**中断**を選択します。これ以降、何もログファイルには追加されません。次にログに記録すべき解析結果が出力された時は「log on」を入力（あるいは**ログ**をクリックして**再開**を選択）して、そのコマンドを再実行すれば、ログに記録されますその後、再び「log off」を入力してください。このように、ログのオン/オフを切り替えながら作業を行うことができます。

15.2 ログ内にコメントを挿入する

Stataは、「*」から始まる行をコメントとして認識して無視します。つまり、インタラクティブに操作を行っていてコメントを残したい場合、「*」を入力すれば、コメントを入力できます。

```
. * check that all the spells are completed
. _
```

Stataはコメントを実行せずに無視しますが、記録中はこのコメントをログ記録します。

❑ テクニカルノート

logコマンドは#review(詳細は[U] 10 キーボードの使用をご覧ください)と組み合わせて使用できます。これは今までのセッション内の操作を確認する際に便利です。例えば、コンピュータの前でStataを操作しており、目的としていた解析ができた状態にあるとしましょう。しかし、解析はできたのですが、ずばり何を行ったのか思い出せない状態です。何か間違いは無かっただろうか。同じ結果を導き出せるだろうか。確認しようと思ったのですが、出力結果のログは記録していませんでした。この場合、#reviewを入力すると実行したコマンドを確認でき、logと組み合わせると記録を取る事ができます。また、履歴ウィンドウでは実行したコマンドを確認できます。これらのコマンドは保存できます。保存するには、目的の部分を選択し、履歴ウィンドウで右クリックをして**選択範囲を保存...**を選びます。

あるいは「log using *ファイル名*」と入力し、続けて「#review 100」と入力します。Stataは最大過去100回分のコマンドをリストします。logファイルの記録中なのでこのリストはファイルに保存されます。最後に「log close」と入力してください。

❑

15.3 入力したコマンドだけをログで取る

ログファイルはセッション中に起こる事を全て記録します。これは、ユーザが入力した情報とその結果としてStataが出力した情報の両方を示します。

Stataはコマンドログファイルを出力できます。これはユーザが実行したコマンドのみをまとめたファイルです。このファイルは後からdoファイルを作成するときに非常に便利です。

cmdlogはコマンドログを作成します。基本的な構文は次の通りです。

```
cmdlog using ファイル名 [ , append replace ]      ファイル名.txtを作成する
cmdlog off                                        コマンドログの作成を一度中断する
cmdlog on                                         コマンドログの作成を再開する
cmdlog close                                      コマンドログファイルを閉じる
```

詳細は[R] logをご覧ください。

コマンドログは一般的なテキストファイルです。例えば、次のようにStataに入力します。

```
. cmdlog using session
(cmdlog C:¥example¥session.txt opened)
. use https://www.stata-press.com/data/r18/census5
(Census Data)
. tabulate region [freq=pop]
  (省略)
. summarize median_age
  (省略)
. cmdlog close
(cmdlog C:¥example¥session.txt closed)
```

すると、作成したmycmds.txtファイルは次の内容を含みます。

```
use https://www.stata-press.com/data/r18/census5
tabulate region [freq=pop]
summarize median_age
```

両方のログの形式(セッション全てのログとコマンドログ)は同時に作成する事も可能です。コマンドログファイルは、後からdoファイルとして使用できます。詳細は[R] doをご覧ください。

15.4 ログボタンの代替案

logコマンドと同じ事を行う（ただし、cmdlogコマンドでは無いので注意）にはStataのGUIインターフェイスを使用できます。操作は**ログ**ボタンをクリックするか、**ファイル**メニューから**ログ**を選択してください。

また、ビューワを使用するとログを表示できます。これは、作成途中のログでも同じです。**ファイル > 表示...**と選択してください。ログを作成中ならば、表示するファイル名は現在記録中のログファイル名を表示します。そのまま**OK**をクリックしてください。定期的に**再読み込み**ボタンをクリックすると、ビューワの内容が更新されます。

前に作成したログの内容は、同じようにビューワで見る事ができます。

ビューワを開くには、**ファイル > 表示...**と操作するか、viewコマンドを使用してください。

```
. view myoldlog.smcl
```

15.5 ログを印刷する

ビューワからログファイルを印刷できます。**ファイル > ビューワ...**と選択するか、コマンドとして「view *ログファイル名*」を入力してログをビューワにロードします。そしてビューワ内で右クリックして**印刷**を選択してください。

また、他の方法でログを印刷する事もできます。詳細は[R] translateをご覧ください。

15.6 同時使用のために複数のログファイルを作成する

プログラマーや上級ユーザは一度に2つ以上のログファイルを作成したい時もあるでしょう。例えば、1つのログファイルはセッション中の情報を情報を全て記録し、もう1つはセッションの一部分のみのを記録したログファイルが欲しい時などが当てはまります。

logコマンドのname()オプションを使用すると、複数のログを作成できます。詳細は[R] logをご覧ください。

16 doファイル

目次

16.1 説明

キーボードでコマンドを直接入力する代わりに、コマンドを含んだテキストファイルを作成してStataにそのファイル内のコマンドを実行するように指示できます。このようなファイルは、doコマンドで実行するので*doファイル*と呼びます。

doファイルは標準的なテキストファイルで、「do *ファイル名*」と入力して実行します。doファイルは、どのテキストエディタでも作成できます。Stata内に組み込まれているdoファイルエディタも使用できます。詳細は[GSW] **13 Using the Do-file Editor—automating Stata**をご覧ください。コマンドの入力やダイアログボックスの使用でなくdoファイルを使用した場合、いくつかの利点があります。データを管理したり分析する手順をdoファイルとしてまとめておくと、後からその作業を簡単に繰り返すことができます。また、doファイルを作成すると、デバッグ作業を行う手順を簡略化できます。分析の一箇所を変更する場合、変更箇所のコマンドをdoファイル内で修正するほうが、手順の1番目から作業をし直すより時間と労力を短縮できます。この章では、doファイルの技法について説明します。Long (2009)は全ての調査・研究プロジェクトで使用するべきと述べると共に、長年Stataを使用する上で培ってきた経験を元にしたデータ管理と統計分析についてのアドバイスを提供しています。

例題1

doファイルを使用するとバッチ処理のような環境を作成することができます。これは、実行したいコマンドを全てdoファイルに入力すると、そのファイルでコマンドを一括実行できるためです。では、テキストエディタあるいはワープロソフトを使用してmyjob.doという、以下の3行を含んだファイルを作成したとします。

```
------------------------------------------------------------------------------ begin myjob.do
use https://www.stata-press.com/data/r18/census5
tabulate region
summarize marriage_rate divorce_rate median_age if state!="Nevada"
------------------------------------------------------------------------------ end myjob.do
```

そしてStataを起動し、Stataにこのファイルを実行(do)するように指示します。

```
. do myjob

. use https://www.stata-press.com/data/r18/census5 (1980 Census data b
y state)

. tabulate region
```

Census region	Freq.	Percent	Cum.
NE	9	18.00	18.00
N Cntrl	12	24.00	42.00
South	16	32.00	74.00
West	13	26.00	100.00
Total	50	100.00	

```
. summarize marriage_rate divorce_rate median_age if state !="Nevada"
```

Variable	Obs	Mean	Std. dev.	Min	Max
marriage_r~e	49	.0106791	.0021746	.0074654	.0172704
divorce_rate	49	.0054268	.0015104	.0029436	.008752
median_age	49	29.52653	1.708286	24.2	34.7

この結果を出力するのに、「do myjob」しか入力していません。ファイルの拡張子を指定しなかったので、Stataは「do myjob.do」であると推測して実行しました。詳細は[U] 11.6 ファイル名の修正をご覧ください。

16.1.1 バージョン

doファイルを作成する際、最初の行にdoファイルを作成したときのStataのバージョンを記載するようにお勧めしています。つまり、先ほどのmyjob.doを次のように修正します。

```
begin myjob.do
version 17.0
use https://www.stata-press.com/data/r18/census5
tabulate region
summarize marriage_rate divorce_rate median_age if state!="Nevada"
end myjob.do
```

必ずしもこのバージョン情報が記載されているとは限りません。このマニュアル内に紹介している多数の例題の多くには「version 18.0」行は入力していません。

しかし、今後doファイルを将来も使うならば、この行を含めてください。これは新バージョンのStataでもdoファイルを継続使用可能にするためです。Stataの開発は継続して続けられています。様々なものがそれまで予想もしなかった方向に変更されることもあります。

例えば、Stata 3.0では、重み付けを指定する新しい構文が始めて使われるようになりました。もし、Stata 2.1で作成した古いdoファイルの中に重み付けをしたデータを分析する物がある場合、そのdoファイルの最初に「version 2.1」の記述が無いと、新しいStataではいくつかのコマンドでは、構文エラーを返してきます。先頭の行に「version 2.1」があれば、ファイルは問題なく実行できます。

Stata 10のバージョンでは、xtsetを導入しました。その中で、xtコマンドを使用するには、まずデータをxtsetする必要があるという変更が追加されました。それまでのバージョンでは、各xtコマンドの最後にグループと時間変数を指定するオプションを設定していました。この変更の後でも、version 9または前のバージョンをdoファイルの中で指定すれば、xtコマンドは以前と同じように使用できます。

バージョン間の違いに関しての概要リストや自動的に表示されないバージョン情報については、「help version」を実行してください。

version記述を含む古いバージョンのdoファイルを実行する時に、バージョンを元に戻すという心配は必要ありません。doファイルが完了したら、Stataが自動的に元のversionの情報に戻します。

13やそれ以前のStataユーザとのファイル共有については[U] 12.4.2.6 Stata 13以前のStataを使用していたユーザへのアドバイスをご覧ください。

16.1.2 doファイル内のコメントと空行

doファイルには自由に空行を組み込むことができます。先ほどの例題では、doファイルは次のように表現することもできます。

```
begin myjob.do
version 18.0
use https://www.stata-press.com/data/r18/census5
tabulate region
summarize marriage_rate divorce_rate median_age if state!="Nevada"
end myjob.do
```

doファイルにコメントを追加するには4つの方法があります。

1. 入力行を「*」から開始します。Stataは「*」がある行を無視します。*はMata内では使用できません。
2. コメントを/* */の区切り文字内に記入します。
3. コメントを2本のスラッシュ、つまり//の後に入力します。//の後に続く行の内容はコメントとして認識されます（ただし、//が「http://... 」の一部である場合はその限りではありません）。
4. コメントを3本のスラッシュ、つまり///の後に入力します。///の後に続く行の内容はコメントとして認識されます。しかし、///行の次の行は///行と同じ行として扱われます。///は長いコマンド行をdoファイル内で複数の行に分割できる記述法です。

4つのコメント メソッドのいずれかを周囲のテキストから空白文字で区切る必要があることに注意してください。

❑ テクニカルノート

/* */, //, ///はdoファイルとadoファイルでのみ使用できるコメントを指示する識別子です。これらはインタラクティブな操作、つまりコマンドウィンドウなどでは使用できません。しかし、「*」はコメントの識別子として、インタラクティブに使用できます。

❑

myjob.doにコメントを加えると、次のようになります。

```
begin myjob.do
* a sample analysis job
version 18.0
use https://www.stata-press.com/data/r18/census5
/* obtain the summary statistics:*/
tabulate region
summarize marriage_rate divorce_rate median_age if state!="Nevada"
end myjob.do
```

あるいは、以下でも同様です。

```
begin myjob.do
// a sample analysis job
version 18.0
use https://www.stata-press.com/data/r18/census5
// obtain the summary statistics:
tabulate region
summarize marriage_rate divorce_rate median_age if state!="Nevada"
end myjob.do
```

どのコメント識別子を使用するかはユーザに任されています。/* */を使う方法の利点の1つは、コマンドと同じ行の中で、コマンドの後ろに付け加える事ができる点です。

```
begin myjob.do
* a sample analysis job
version 18.0
use https://www.stata-press.com/data/r18/census5
tabulate region                    /* obtain summary statistics */ summarize marriage_rate di
vorce_rate median_age if state!="Nevada"
end myjob.do
```

実際、/* */は行の中でも、好きなところに入力することができます。

```
begin myjob.do
* a sample analysis job
version 18.0
use /* confirm this is latest */ https://www.stata-press.com/data/r18/census5
tabulate region                    /* obtain summary statistics */
summarize marriage_rate divorce_rate median_age if state!="Nevada"
end myjob.do
```

//または///でも、同じように使用できます。

```
begin myjob.do
// a sample analysis job
version 18.0
use https://www.stata-press.com/data/r18/census5
tabulate region                    // obtain summary statistics
summarize marriage_rate divorce_rate median_age if state!="Nevada"
end myjob.do
```

または

```
begin myjob.do
// a sample analysis job
version 18.0
use /// confirm this is latest
https://www.stata-press.com/data/r18/census5
tabulate region                    // obtain summary statistics
summarize marriage_rate divorce_rate median_age if state!="Nevada"
end myjob.do
```

16.1.3 doファイル内の長い行

Stataをインタラクティブに使用している場合、*Enter*キーを押すとStataはその行のコマンドを実行します。スクリーンよりも長い行を入力するには、そのまま入力を続けると、自動的に折り返して表示するかスクロールできるようになります。

doファイルでも、テキストエディタやワープロソフトでそのように設定しているなら、同じように動作します。もうひとつ設定する方法があり、それは区切り文字（デリミタ）としてセミコロン「;」行を設定する方法です。行の終わりを示す区切り文字として「;」を設定するには#delimitを使用します。コメントの間に改行を入力したい場合は/* */のコメント区切りを使うか、改行してもコメントを継続する///をご利用ください。

例題2

次のdoファイルは一時的に行の終わりのデリミタを変更しました。

```
                                                                  fragment of example.do
use mydata
#delimit ;
summarize weight price displ headroom rep78 length turn gear_ratio if substr(company,
          1,4)=="Ford" |
             substr(company,1,2)=="GM", detail ; gen byte ford = substr(compa
ny,1,4)=="Ford" ;
#delimit cr
gen byte gm = substr(company,1,2)=="GM"
                                                                  fragment of example.do
```

一度行の区切り文字をセミコロン（;）に変更すると、短い行でもその行の終わりにはセミコロンを入力する必要があります。キャリッジリターン(改行)はスペースと同じように扱われます。区切りをキャリッジリターンに戻すには「#delimit cr」と入力してください。

#delimitコマンドはdoファイルでのみ使用できます。インタラクティブなStataの操作では使用できません。doファイルの最後で改行をキャリッジリターンに直すのはStataが自動的に行うので、特に操作する必要はありません。

例題3

長いコメント行のもうひとつの対処法は、/* */のコメント括弧を使用するか///の行統合の識別子を使用して、改行部分をコメントとすることです。つまり、コードは次のようになります。

```
                                                                  fragment of example.do
use mydata
summarize weight price displ headroom rep78 length turn gear_ratio /*
      */ if substr(company,1,4)=="Ford" |        /*
      */      substr(company,1,2)=="GM",  detail gen byte ford = substr(c
ompany,1,4)=="Ford" gen byte gm = substr(company,1,2)=="GM"
                                                                  fragment of example.do
```

または

```
                                                                  fragment of example.do
use mydata
summarize weight price displ headroom rep78 length turn gear_ratio /// if substr(company,1,
        4)=="Ford" |                            ///
           substr(company,1,2)=="GM",  detail
gen byte ford = substr(company,1,4)=="Ford" gen byte gm = subst
r(company,1,2)=="GM"
                                                                  fragment of example.do
```

16.1.4 doファイルでのエラーの扱い方

doファイルは実行後、ファイルの最後まで到達する、exitコマンドが実行される、エラーが発生する(*リターンコードが0以外*)、のいずれかの場合にそのファイルを止めます。エラーが発生した場合、doファイルの残りのコマンドは実行されません。

doファイルを実行中に*Break*を実行すると、Stataはあたかもエラーが発生したかのように挙動してdoファイルを途中で止めます。これは、リターンコードがゼロではないのでエラーが発生したときと同じ挙動です。リターンコードに関する詳細は[U] 8 エラーメッセージとリターンコードをご覧ください。

例題4

doファイルを実行した後に*Break*を押すと、次のようになります。

```
. do myjob2
. version 18.0
. use census (Census data)
. tabulate region Census
    region |      Freq.     Percent        Cum.
— Break —  |
r(1);
end of do-file Break
r(1);   —
. _
```

*Break*を実行すると、Stataは—Break—と表示してリターンコード1を返します。Stataは“end of do-file”と表示した後に—Break—とリターンコード1をもう一度表示するので、動作が繰り返したように見えます。この繰り返しのメッセージに関しては特に心配する必要はありません。最初のメッセージは、*Break*が実行されたのでStataが実行していたtabulateコマンドを中断していたことを示します。そして、2つ目のメッセージは*Break*を実行したのでStata自体を中断した事を示します。

例題5

もう一度、例題を試して見ましょう。しかし、今回は実際にエラーを組み込んでみます。myjob2.doを次のように変更します。

```
begin myjob2.do
version 18.0
use censas
tabulate region
summarize marriage_rate divorce_rate median_age if state!="Nevada"
end myjob2.do
```

致命的な入力ミスのケースを演出するために、use census5の代わりに use censasを入力しました。censas.dtaというファイルが存在しない事を想定しているので、エラーが検出されるはずです。Stataにファイルを実行(do)するように指示を出すと、次のようになります。

```
. do myjob2
. version 18.0
. use censas
file censas.dta not found r(601);
end of do-file r(601);
. _
```

「use censas」を実行すると、Stataは“file censas.dta not found”というメッセージと共にリターンコード601を返します。それからStataは“end of do-file”と表示した後にもう一度リターンコード601を返しました。メッセージが2回表示されたのは、先程の例題で*Break* を実行した時の理由と同じです。useコマンドはリターンコード601を返し、doファイル自体も同じリターンコードを返したため、結果としてコードが繰り返されました。重要なことは、Stataはエラーが存在したためdoファイルの実行を止めた、という事です。

◁

❑ テクニカルノート

「do *ファイル名*, nostop」と入力すると、doファイル実行中にエラーがあってもそのまま実行するように指示できます。結果は次の通りです。

```
. do myjob2, nostop
. version 18.0
. use censas
file censas.dta not found r(601);
. tabulate region
no variables defined r(111);
summarize marriage_rate divorce_rate median_age if state!="Nevada" no variables define
d
r(111);
end of do-file
. _
```

doファイル内の最初のコマンドが失敗したため、他のコマンドは全てうまくいきませんでした。この事態を避けるため、Stataは通常、doファイルにエラーがあると実行を止めます。しかし、もしこの例題の中で実行可能なコマンドがあれば、その箇所までは実行できたでしょう。一般的に、この方法でdoファイルのコードを作成する事はお勧めしません。これは、エラーで実行が中止しないと意図しない弊害が発生する可能性があるためです。

❑

16.1.5 doファイルの出力のログをとる

doファイルのログの出力はインタラクティブなセッションでログを出力するのと同じように実行できます。詳細は[U] 15 ログファイルの保存と印刷をご覧ください。

多くのユーザは、doファイルの中にコマンドとしてログを開始するコマンドと終了するコマンドを組み込んでいます。

```
begin myjob3.do
version 18.0
log using myjob3, replace
* a sample analysis job
use census
tabulate region                    // obtain summary statistics
summarize marriage_rate divorce_rate median_age if state!="Nevada" log close
end myjob3.do
```

doファイルを「log using myjob3, replace」で開始します。ここで重要になるのは「replace」オプションです。このオプションが無ければ、doファイルを簡単に再実行できなくなります。myjob3.smclが既に作成されており、logコマンドにファイルを置き換える指示がされていない（つまり、replaceオプションを使用していない）場合、doファイルは実行を中止し、“file myjob3.smcl already exists”とメッセージを返します。もちろん、メッセージを表示させないには、doファイルを実行する前にログファイルを削除すれば問題なくdoファイルを実行できます。

16.1.6 -more-を阻止する

Stataはデフォルトで—more—設定をoffにしています。詳細は[U] 7 -more- 条件をご覧ください。

「set more on」により—more—設定をonにしていると、画面が埋まるとその都度出力が止まるため、doファイルをログ出力つきで実行している場合はわずらわしく感じるでしょう。

この問題を解決するには、doファイルに一行、「set more off」を追加してください。—more—設定はdoファイルの実行が終わると、doファイル実行前の元の状態に戻ります。

16.2 他のdoファイルを呼び出す

doファイルは他のdoファイルを呼び出すことができます。例えば、データをinfileコマンドでインポートし、いくつか変数を作成してからstep1.dtaとして保存するmakedata.doを作成します。そして、step1.dtaを解析するコマンドをまとめてanlstep1.doを作成しました。ここで、3つ目のdoファイルを次のように作成します。

```
begin master.do
version 18.0
do makedata
do anlstep1
end master.do
```

これで2つのdoファイルをまとめることができます。

doファイルが他のdoファイルを呼び出し、そのdoファイルがさらに他のdoファイルを呼び出し、と階層を深く進んでいく事ができます。Stataのdoファイルは64階層まで下げることができます。

ここで気をつけてほしいのは、上記のmaster.doは同じように追加した1,000個のdoファイルを順番に呼び出したとしても、その階層は2のままです。

16.3 doファイルの作成と実行

16.3.1 Windowsのためのdoファイルの作成と実行

1. doファイルを実行するには上記のようにdoコマンドの後にファイル名を入力します。
2. **ファイル** > **doファイルの実行...**と選択するとdoファイルを実行できます。
3. doファイルの作成、保存、実行にはdoファイルエディタを使用できます。詳細は[GSW] 13 Using the Do-file Editor-automating Stataをご覧ください。doファイルエディタを使用するには、**doファイルエディタ**ボタンをクリックするかコマンドウィンドウにdoeditと入力します。Stataには複数のdoファイルなどを管理するためのプロジェクトマネージャがあります。詳細は[P] Project Managerをご参照ください。
4. doファイルのアイコンをダブルクリックすると、doファイルエディタが開きます。
5. doファイルはバッチモードで実行できます。詳細は[GSW] B.5 Stata batch modeで確認してください。簡単に言えば、Windowsのコマンドウィンドウで次を実行します。

   ```
   C:¥data> "C:¥Program Files¥Stata18¥Stata" /s do myjob
   ```

   または

   ```
   C:¥data> "C:¥Program Files¥Stata18¥Stata" /b do myjob
   ```

   C:¥Program Files¥Stata18がStataの保存場所であることを前提に、上記でバッチモードを実行できます。/bと/sは作成するログの種類の指定ですが、ここでは詳述しません。上記を実行すると、Stataはdoファイルをバックグラウンド実行します。doファイルが終了すると、タスクバーのStataアイコンが点滅します。アイコンをクリックしてStataを閉じてください。doファイルの完了前に動作を中断するには、タスクバーのアイコンをクリックし、実行中の作業のキャンセルを指示します。doファイルの終了時、アイコンを点滅させずStataを閉じるには、コマンド入力時に/eも入力します。

ログとして出力を記録するには、doファイルを実行する前にログを開始するか、log usingコマンドとlog closeコマンドをあらかじめdoファイルに入力してください。

Stataをこの方法で実行すると、以下のように動作します。

a. Stataが自動的にログを開きます。/sを指定すると、StataはSMCLログファイルを開きます。/bを指定すると、Stataはテキストのログを開きます。doファイルがxyz.doという名前になっている場合、ログファイルは同じディレクトリ内にxyz.smcl(/sの場合)あるいはxyz.log(/bの場合)として保存されます。

b. もしdoファイル内で他のログファイルを指定していてそれも開く場合、Stataは2つの出力結果を保存します。

c. Stataは—more—条件を無視するとともに、インタラクティブ操作中では一度中断する事態が発生しても、doファイルを止めないために無視します。

16.3.2 Macのためのdoファイルの作成と実行

1. doファイルを実行するには上記のようにdoコマンドの後にファイル名を入力します。
2. **ファイル** > **doファイルの実行...**と選択するとdoファイルを実行できます。
3. doファイルの作成、保存、実行にはdoファイルエディタが使用できます。詳細は[GSM] 13 Using the Do-file Editor-automating Stataをご覧ください。**doファイルエディタ**ボタンをクリックするかコマンドウィンドウにdoeditと入力します。Stataには複数のdoファイルなどを管理するプロジェクトマネージャがあります。詳細は[P] Project Managerをご参照ください。
4. doファイルのアイコンをダブルクリックすると、doファイルエディタが開きます。

5. Stata.doというdoファイルをダブルクリックすると、Stataが既に起動していない場合はStataが起動し、現在の作業フォルダをdoファイルのあるパスに設定します。
6. doファイルはバッチモードで実行できます。詳細は［GSM］B.3 Stata batch modeで確認してください。簡単に言えば、ターミナルウィンドウを開いて次の内容を入力します。

```
% /Applications/Stata/Stata.app/Contents/MacOS/Stata -s do myjob
```

または

```
% /Applications/Stata/Stata.app/Contents/MacOS/Stata -b do myjob
```

これは、StataBEが/Applications/Stataのパスに保存してあることを前提としています。-bと-sはどのようなログを作成するか指示するものですが、ここでは詳述しません。この方法でStataを実行すると、Stataはバックグラウンドでdoファイルを実行します。doファイルが完了すると、ドックにあるStataのアイコンがStataを前面に移動するまではねます。その後、Stataを閉じてください。doファイルが完了する前に止めたい場合、ドックにあるStataのアイコンを右クリックして、**終了**を選んでください。

ログとして出力を記録するには、doファイルを実行する前にログを開始するか、log usingコマンドとlog closeコマンドをあらかじめdoファイルに入力してください。

Stataをこの方法で実行すると、以下のように動作します。

a. Stataが自動的にログを開きます。-sを指定すると、StataはSMCLログファイルを開きます。-bを指定すると、Stataはテキストのログを開きます。doファイルがxyz.doという名前になっている場合、ログファイルは同じディレクトリ内にxyz.smcl(-sの場合)あるいは xyz.log(-bの場合)として保存されます。
b. もしdoファイル内で他のログファイルを指定していてそれも開く場合、Stataは2つの出力結果を保存します。
c. Stataは—more—条件を無視するとともに、インタラクティブ操作中では一度中断する事態が発生しても、doファイルを止めないために無視します。

16.3.3 Unixのためのdoファイルの作成と実行

1. doファイルを実行するには、上記のようにdoコマンドの後にファイル名を入力すれば実行できます。
2. **ファイル** > **doファイルの実行...**と選択するとdoファイルを実行できます。
3. doファイルを作成、保存、実行するのにdoファイルエディタを使用できます。詳細は［GSU］13 Using the Do-file Editor-automating Stataをご覧ください。**doファイルエディタ**ボタンをクリックするかコマンドウィンドウにdoeditと入力します。Stataには複数のdoファイルや他のファイルを管理するためのプロジェクトマネージャがあります。詳細は［P］Project Managerをご参照ください。
4. Unixのプロンプトで、次のように入力します。

```
$ xstata do filename
```

または

```
$ stata do filename
```

これで、Stataを起動してdoファイルを実行します。doファイルが完了すると、Stataは普通の方法で次のコマンドを尋ねてきます。代わりにStataを終了したい時はdoファイルの最後に「exit, STATA clear」という行を追加してください。

ログとして出力を記録するには、doファイルを実行する前にログを開始するか、log usingコマンドとlog closeコマンドをあらかじめdoファイルに入力してください。

5. Unixのプロンプトで、次のように入力します。

```
$ stata -s do filename &
```

または

```
$ stata -b do filename &
```

これでdoファイルをバックグラウンド実行します。上記の2つの例は、xstataでなくstataを使用しています。普段GUIバージョンであるxstata使用するユーザも、stataと入力してください。例では1箇所だけ違いがあります。1つは-s、もう1つは-bを指定します。これらは出力ログの種類を決定します。それぞれ次のように動作します。

a. Stataが自動的にログを開きます。Stataは、-sを指定するとSMCLログファイルを、-bを指定するとテキストのログを開きます。doファイルがxyz.doという名前の場合、ログファイルは現在のディレクトリ(stataコマンドを実行したディレクトリ)内にxyz.smcl(-sの場合)あるいはxyz.log(-bの場合)として保存されます。

b. もしdoファイル内で他のログファイルを指定していてそれも開く場合、Stataは2つの出力結果を保存します。

c. Stataは—more—条件を無視するとともに、インタラクティブ操作中では一度中断する事態が発生しても、doファイルを止めないために無視します。

言い換えると、バックグラウンドでdoファイルを実行してテキストログを入手するには、次のように入力してください。

```
$ stata -b do myfile &
```

もうひとつの方法は、一般的なリダイレクトを使います。

```
$ stata < myfile.do > myfile.log &
```

最初の方法の方がわずかですが効率的です。どちらの場合でも、Stataはバックグラウンドで実行することが決まっている場合、—more—条件をはじめとした、Stataのdoファイルを中止する挙動は無視します。ただし、もし作成したdoファイルに#delimitコマンドまたはコメント記号（/* */の間に挟まれた文字はコメントとして内容はチェックされない）が組み込まれている場合、2番目のリダイレクトは正常に動作しません。よって、最初の方法をお勧めします。stata -b do myfile &とstata -s do myfile & のどちらを使用するかは好みによります。SMCLログ(-s)のほうが印刷した際の見栄えがいいので、そちらを設定するユーザも多いでしょう。また、この形式はいつでもtranslateコマンドを使用してテキスト形式に変換できます。詳細は[R] translateをご覧ください。

16.4 doファイルでプログラミングする

こちらは上級者向けトピックです。これから、まだ説明していないコンセプトについて説明をします。詳細は[U] 18 Stataのプログラミングをご覧ください。

16.4.1 引数を受け渡す

Stataのプログラムが引数を使用できるように、doファイルも引数を使えます。詳しくは[U] 18 Stataのプログラミングと[U] 18.4 プログラムの引数をご覧ください。実際には、Stataがdoファイルを実行する際に従う手順はプログラムを実行する際の手順と同じです。ローカルマクロはこちらを保存していて、新しいものは定義されます。引数は‘1’, ‘2’のようなローカルマクロに保存されます。doファイルの作成が終わると、他のプログラムのように、以前の設定保存が戻ります。

つまり、doファイルで

1. 選択するデータセットを使う
2. 変数regionをテーブルにする
3. 変数marriage_rateとdivorce_rateの要約統計量を出す

場合、doファイルに次のように入力しましょう。

```
──────────────────────────────────────────────────────────────────begin myxmpl.do──
    use ‘1’
    tabulate region
    summarize marriage_rate divorce_rate
──────────────────────────────────────────────────────────────────end myxmpl.do────
```

このdoファイルを実行するには次のように入力すると実行できます。

```
. do myxmpl census
（省略）
```

最初のコマンド、「use ‘1’」は use census5として認識されます。これはcensus5がdo myxmplの後に入力した最初の引数だからです。

より良いdoファイルは次のようになります。

```
──────────────────────────────────────────────────────────────────begin myxmpl.do────
    args dsname
    use ‘dsname’
    tabulate region
    summarize marriage_rate divorce_rate
──────────────────────────────────────────────────────────────────end myxmpl.do────
```

argsコマンドは引き渡された引数により良い名前を提供しているだけです。args dsnameはdo myxmplがファイル名であることを確認するわけではありません。（もし確認したい場合はsyntaxコマンドを使用すれば確認できます。）しかし、‘dsname’と‘1’を入れ替えると、見ただけでコードの意味が通じるようになります。

もしプログラムが2つの引数を受け取る場合、それらを‘1’と‘2’として設定するか、doファイルの冒頭部分に‘args dsname other’を追加してください。‘dsname’と‘other’で別々に参照するようにはしないでください。

引数の受け渡しについて詳しく知るには[U] 18.4 プログラムの引数をご確認ください。Baum (2009)はdoファイルに関する多くのヒントや例題を掲載しています。

16.4.2 出力を抑える

do *ファイル名*と入力する以外にrunファイル名と実行することもできます。runコマンドはdoコマンドと同じように実行できます。違いはdoファイル内のコマンドに関連する情報や出力はスクリーンに表示しないことです。

例えば、先ほどのmyxmpl.doでrun myxmpl census5と入力すると次のようになります。

```
. run myxmpl census
. _
```

doファイル内の全てのコマンドは実行されましたが、結果は表示されませんでした。

これはあまり便利ではないかもしれません。ただし、doファイルがStataのプログラム定義で作成されており、（詳細は[U] 18 Stataのプログラミングをご覧ください）そのプログラムのコードを確認する必要がなく、ただロードだけしたい場合に runコマンドは便利です。

16.5 参考文献

Baum, C. F. 2009. *An Introduction to Stata Programming*. College Station, TX: Stata Press.

Long, J. S. 2009. *The Workflow of Data Analysis Using Stata*. College Station, TX: Stata Press.

Wiggins, V. L. 2018. How to automate common tasks. *The Stata Blog: Not Elsewhere Classified*. https://blog.stata.com/2018/10/09/how-to-automate-common-tasks/.

17 adoファイル

目次

17.1 説明

Stataにはプログラミング機能が用意されています。Stataプログラムを自作する計画をお持ちでない方にとっても、これは重要な機能です。Stataの機能の多くはStata用プログラムで書かれており、毎日のようにStata社と先進的なユーザによって新しい機能が追加されています。

1. 新しい機能は*Stata Journal*を介して入手できます。新しい機能は出版物であるジャーナルにおいて発表されますが、ソフトウェアはインターネット経由で、無償で入手できます。
2. 追加分のファイルはStata社のフォーラムであるStatalistからも入手できます。このリストサーバでは熱心なユーザ同士で情報やプログラムの交換が頻繁に行われています。フォーラムへの参加方法はhttps://www.statalist.orgを参照してください。
3. それとは別に、Stataプログラムの巨大なアーカイブであるBoston College Statistical Software Components (SSC)を大変役に立ちます。https://www.stata.comからアーカイブを検索し、プログラムを入手することができます。ただし、Stataの中から直接、必要な情報を検索する機能がありますのでご紹介します。Stataのsearchとnetコマンドを利用すると、とても効率的に検索できます。詳細は[R] **search**を参照してください。searchまたはnetコマンドで検索したファイルは、検索結果のウィンドウから、ハイパーリンク機能を利用して即座にインストールできます。SSCアーカイブのファイルを直接インストールするための特別なコマンドsscには複数のオプションが用意されています。詳細は[R] **ssc**をご参照ください。
4. もちろん、自分用のコマンドファイルを作成することも可能です。

この章ではadoファイルを利用する上で必要な情報を解説します。一般ユーザ向けに丁寧にご説明します。オリジナルのadoファイルを作成する場合は[U] **18.11 adoファイル**をご参照ください。

17.2 adoファイルの仕組み

adoファイルは一つのStataコマンドを定義するものです。しかし、Stataコマンドがすべてadoファイルかというと、そうではありません。例えば、summarizeコマンドを実行すると要約統計量を表示しますが、このコマンドはStataの内蔵コマンドです。

信頼区間を求める場合に利用するciはadoファイルです。普通にコマンドを実行する分には、それが内蔵コマンドなのか、adoファイルなのか、識別することはできません。

繰り返しますが、adoファイルはテキストファイルで構成され、その中にはStataのプログラムコードが記述されています。Stataでコマンドを実行すると、最初に内蔵コマンドを検索し、次にディスク上でadoファイルを検索します。目的のadoファイルが検索できたら、Stataは即座にそれを実行しますので、ユーザはあたかも内蔵コマンドが実行されたかのように感じます。

ciコマンドは繰り返しますが、adoファイルです。つまり、ディスク上のどこかにci.adoという名前のテキストファイルが存在します。

adoファイルには基本的にそれに対応するヘルプファイルが存在します。help ciとコマンドウィンドウに入力するか、または **ヘルプ ＞ Stataのコマンド...** と操作してciと入力します。Stataはci.adoに対応したci.sthlpファイルを検索します。ヘルプファイルも同じくテキストファイルであり、Stataのヘルプシステムはその内容を表示します。

17.3 内蔵コマンドとadoファイルの見分け方

コマンドが内蔵コマンドなのか、それともadoファイルのコマンドなのかを見分ける場合はwhichコマンドを利用します。例えば、adoファイルであるlogisiticについてコマンドを実行すると次のようになります。

```
. which logistic
C:¥Program Files¥Stata18¥ado¥base¥l¥logistic.ado
*! version 3.5.4  28feb2017
```

内蔵コマンドsummarizeの場合は次のようになります。

```
. which summarize
built-in command:  summarize
```

17.4 adoファイルの所在

whichコマンドをadoファイルに対して実行すると、Stata画面にファイルの所在を表示します。

```
. which logistic
C:¥Program Files¥Stata18¥ado¥base¥l¥logistic.ado
*! version 3.5.4  28feb2017
```

adoファイルは単純なテキストファイルにStataプログラム記述したものです。その内容を確認する場合はStataのビューワを利用します。もちろん、一般のテキストエディタやワープロソフトでも同じ操作が可能です。

```
. type "C:¥Program Files¥Stata18¥ado¥base¥l¥logistic.ado"
*! version 3.5.4  28feb2017
program define logistic, eclass prop(or svyb svyj svyr swml mi bayes) ///
                                byable(onecall)
        version 6.0, missing
  (省略)
end
```

または

```
. viewsource logistic.ado
  (省略)
```

ファイルの内容を画面で確認する場合はtypeコマンドを利用します。viewssourceコマンドはアドディレクトリを検索し、中身をビューワに保存する際に利用するコマンドです。また、adoファイルに対応するヘルプファイルを同様に閲覧することも可能です。adoファイルに対応したヘルプファイルは、そのadoファイルと同じフォルダに入っています。

```
. type "C:¥Program Files¥Stata18¥ado¥base¥l¥logistic.sthlp", asis
{smcl}
{* *! version 1.4.4  01sep2020}{...}
{viewerdialog logistic "dialog logistic"}{...}
  (省略)
```

または

```
. viewsource logistic.sthlp
  (省略)
```

17.5 adoファイルの保存場所

Stataはadoファイルを7箇所に分類して保存します。大きなカテゴリとしては3つです。

I. 公式adoファイルディレクトリ
   1. (BASE)Stata社が公認しているadoファイルで、Stataと一緒にインストールされます。インストール時に更新されているものは、最新のバージョンが入ります。

II. パーソナルadoファイル
   2. (SITE),御客様の会社(大学)全体で利用するadoファイル用ディレクトリ
   3. (PLUS),御客様が個人的に利用するadoファイル用ディレクトリ
   4. (PERSONAL),個人的に作成したadoファイル用ディレクトリ
   5. (OLDPLACE)個人的に作成し、現在は不使用のadoファイル用ディレクトリ

III. 現在の作業ディレクトリ
   6. (.),仮作成、または、将来利用予定のないadoファイル用ディレクトリ

もちろん、以上のディレクトリはユーザの環境によって異なることが考えられます。とりあえず、これらのディレクトリの情報を確認するにはsysdirコマンドを利用します。

```
. sysdir
   STATA:  C:¥Program Files¥Stata18¥
    BASE:  C:¥Program Files¥Stata18¥ado¥base¥
    SITE:  C:¥Program Files¥Stata18¥ado¥site¥
    PLUS:  C:¥ado¥plus¥
PERSONAL:  C:¥ado¥personal¥
OLDPLACE:  C:¥ado¥
```

17.5.1 公式なadoファイルディレクトリ

公式なadoファイルディレクトリはsysdirで表示されるBASEフォルダです。

```
. sysdir
   STATA:  C:¥Program Files¥Stata18¥
    BASE:  C:¥Program Files¥Stata18¥ado¥base¥
    SITE:  C:¥Program Files¥Stata18¥ado¥site¥
    PLUS:  C:¥ado¥plus¥
PERSONAL:  C:¥ado¥personal¥
OLDPLACE:  C:¥ado¥
```

1. BASEフォルダにはDVDからStataと一緒に、公式adoファイルをインストールします。それらの更新版もこのフォルダに入ります。更新版のインストールはupdateコマンドか、または**ヘルプ** > **アップデートのチェック**と操作します。詳細は[U] 17.8 公式更新版のインストールをご参照ください。

17.5.2 個人的に利用するadoファイルディレクトリ

sysdirを実行すると次のようにPERSONAL,PLUS,SITE,OLDPLACEディレクトリを表示します。

```
. sysdir
   STATA:  C:¥Program Files¥Stata18¥
    BASE:  C:¥Program Files¥Stata18¥ado¥base¥
    SITE:  C:¥Program Files¥Stata18¥ado¥site¥
    PLUS:  C:¥ado¥plus¥
PERSONAL:  C:¥ado¥personal¥
OLDPLACE:  C:¥ado¥
```

1. PERSONALは個人的に作成したadoファイル用ディレクトリです。個人的に利用するファイルはこのフォルダに保存します。詳細は[U] 17.7 doファイルの追加を参照ください。
2. PLUSは個別にインストールしたadoファイル用ディレクトリです。ここへは通常SJまたはSSCアーカイブから入手したadoファイルが入ります。ほかの場所に入ることも時折あります。これらのサイトから直接adoファイルをインストールするにはnetコマンドを利用するか、または、**ヘルプ > Stataジャーナル/コミュニティ投稿コマンド**と操作し、必要な情報をご覧ください。詳細は[U] 17.6 doファイルの追加を参照してください。
3. SITEは個人用のディレクトリとは正反対の位置づけで、いわばPLUSのパブリック版です。ネットワーク環境でStataを利用している場合、管理者がadoファイルをここにインストールすると、ネットワーク下にあるすべてのユーザはあたかも自分のローカルなPLUSディレクトリに目的のadoファイルがインストールされている状態になります。ネットワークサイトの管理者はnetコマンドでサーバにインストール際に、SITEまたはPLUSのどちらにインストールするか選択できます。詳細は[R] netを参照してください。
4. OLDPLACEにはStataの古いバージョン用のadoファイルを保存します。Stata 6よりも古いバージョンで個人的に作成したadoファイルはすべてOLDPLACEに保存されます。Stataの古いバージョンを利用するユーザは何も気にせず新しいバージョンに古いadoファイルをインストールできます。Stataは常にOLDPLACEフォルダ内も検索します。

17.6 追加ファイルのインストール方法

追加ファイルには4つの種類があります。

1. SJなどに掲載されているユーザ定義の追加ファイル
2. ユーザ定義ファイルの更新版
   詳細は[U] 17.9 ユーザ定義ファイルのインストールをご参照してください。
3. 自作のadoファイル
   詳細は[U] 17.7 自作のadoファイルの追加をご参照ください。Stataのフォーラムや友人から入手したadoファイルの場合がこのケースに相当します。
4. Stata社が提供する公式更新版
   詳細は[U] 17.8 公式更新版のインストールをご参照してください。

Stata Journal (SJ)に掲載されているユーザ定義などはインターネット経由で取得します。
インターネットからファイルを取得する場合は次のようにします。

1. **ヘルプ > Stataジャーナル/コミュニティ投稿コマンド**として目的リンクをクリックします。

または

2. net from https://www.stata.comと入力します。

このようにして情報が表示されたら、画面の中から目的のリンクを選択します。その時の操作方法は[GS] 19 Updating and extending Stata-Internet functionality(GSM, GSUを参照するか、またはGSWと操作します)。同様に[U] 29 Using the Internet to keep up to date、[R] net そして、[R] ado updateもご参照ください。

17.7　自作のadoファイルの追加

Stataの追加プログラム([U] 18 Stataのプログラミング))を自作することも可能です。拡張子は.adoとし、ヘルプファイルも同じく作成します。自作したファイルはsysdirコマンンドのPERSONALにコピーします。

```
. sysdir
   STATA:  C:¥Program Files¥Stata18¥
    BASE:  C:¥Program Files¥Stata18¥ado¥base¥
    SITE:  C:¥Program Files¥Stata18¥ado¥site¥
    PLUS:  C:¥ado¥plus¥
PERSONAL:  C:¥ado¥personal¥
OLDPLACE:  C:¥ado¥
```

ファイルをC:¥ado¥personalフォルダにコピーします。

自作のadoファイルを作成している場合、作業中のフォルダに自作ファイルを保存すると便利です。つまり、プログラムのデバッグは自作用adoファイルフォルダで行います。

17.8　公式ファイルの更新版インストール方法

更新版はインターネット上で取得できます。

1. **ヘルプ** > **アップデートのチェック**と操作し、**https://www.stata.com**をクリックします。

または

2. update queryと入力します。

このようにして情報が表示されたら、画面の中から目的のリンクを選択します。その時の操作方法は[GS] 19 Updating and extending Stata-Internet functionality (GSM, GSUを参照するか、または、GSWと操作します)。同様に[U] 29 Using the Internet to keep up to date、そして[R] netもご参照ください。

公式更新版はデバックや新機能の追加がある場合に提供されますが、コマンドの文法や動作が変わることはありません。

更新版をインストール後で、help whatsnew(または**ヘルプ** > **最新情報**)と操作すれば、更新内容を確認できます。

17.9　ユーザが作成した追加ファイルの更新方法

ユーザ定義のファイルについての更新状況を確認する場合は、ado updateコマンドを利用します。更新版がある場合、ado update, updateコマンドでそれらをインストールできます。詳細は[R] ado updateを参照してください。

17.10　参考文献

Cox, N. J. 2006. Stata tip 30: May the source be with you. *Stata Journal* 6: 149-150.

Wiggins, V. L. 2018. How to automate common tasks. *The Stata Blog: Not Elsewhere Classified*. https://blog.stata.com/2018/10/09/how-to-automate-common-tasks/.

18 Stataのプログラミング

目次

Stataのプログラミングは上級者向けの項目です。Stataのプログラミング方法を学ばなくても、十分にStataで生産的な研究活動を行えます。実際、データのインポートや変数の作成、モデルフィットなどはプログラミングの機能が無くても操作できます。しかし、上級者向けと書きましたが、Stataのプログラミングは決して難しいものではありません。プログラミングの記述のやさしさも、Stataの大きな特徴の一つです。実際にStataのプログラミングを試してみて、初めてStataの価値を実感できます。

Stataには2つのプログラミング言語が用意されています。一つはadoファイルと呼ばれるもので、この章で詳しく説明します。adoファイルはStataのコマンドを組み合わせたものであり、分析の自動化や、新しい分析機能を追加する場合に利用します。

もう一つの言語はMataと呼ばれるコンパイル言語で、C/C++のような文法で記述します。行列演算機能に大きな特徴があります。これら2つの言語を組み合わせて利用することも可能です。つまり、adoファイルからMata関数を、逆に、Mata関数内でadoファイルを呼び出すことも可能です。Mataに関する詳細は*Mata Reference Manual*をご参照ください。

Stataスクリプトやプログラムなどを管理する場合はプロジェクトマネージャの機能を利用します。詳細は[P] **Project Manager**をご参照ください。

とりあえず、一度はこの章を読んでみてください。そして難しいと感じたら、そこで中断してもかまいません。Stataのプログラムを書くという必要性が無い方でも、この章を読むことでStataの便利な機能についての情報を身に付けることができます。

プログラミングについてより詳しく学習したい方は、インターネットによるネットコースをご検討ください。[U] **3.6.2 ネットコース**をご参照ください。Baum (2009)にはStataプログラミングに関する実用的な情報が豊富に用意されています。

18.1 説明

Stataでコマンドを実行すると、最初に内蔵コマンドを検索し、次にディスク上でadoファイルを検索します。プログラムが検出できた場合は、そのプログラムをすぐに実行します。

helloというStataコマンドはありませんので、

```
. hello
command hello is unrecognized
r(199);
```

しかし、プログラムhelloを定義してしまえば、次のような応答を作りだすことができます。

```
. hello
hi there
. _
```

実際のプログラムの中身を次に示します。

```
. program hello
  1. display "hi there"
  2. end

. _
```

Stataにおけるプログラムの動作を解説します。プログラムは次のように動作します。

```
program progname
        Stata commands
end
```

Stataのコマンドウィンドウでprognameと入力して実行します。

18.2　プログラムとdoファイルの関係

Stataにおいてプログラムファイルとdoファイルの扱いは基本的に同じです。このセクションでは引数の引き渡し、Stataコマンドの画面出力などの動作について説明しますが、これらはプログラムファイルとdoファイルでも、どちらでも同じように機能します。

プログラム(以下プログラムファイルをこう呼ぶ)とdoファイルの相違点は次の通りです。

1. doファイルを実行する場合、do filenameと入力します。プログラムの場合は単純にプログラム名だけを入力します。
2. プログラムは利用前に定義(メモリーにロード)します。一方、doファイルの場合は、doファイル自体を実行します。プログラムを自動的にロードする方法もいくつかありますが、それは後述します。
3. do filenameと操作すると、Stataはコマンドを実行し、そのコマンドと結果を表示します。プログラム名を入力すると、Stataは分析結果だけを表示し、中身のコマンドは表示しません。見た目としてはこの点が大きく異なります。つまり、doファイルでは処理内容を結果と同様に重視します。一方のプログラムではコマンドは表示しません。プログラムはStataの新しいコマンドと同等であると考えることができます。

次は類似点について整理します。

1. 引数はプログラムとdoファイルのどちらにも同じ形式で引き渡されます。
2. プログラムとdoファイルの中身はどちらもStataコマンドです。doファイルで利用可能なコマンドはプログラムでも利用できます。
3. プログラムから他のプログラムを呼び出すことができます。同様に、doファイルから他のdoファイルを呼び出すこともできます。プログラムからきわめて稀なことかと思いますが、doファイルを呼び出すこともでき、逆にdoファイルからプログラムを呼び出す事もできます。Stataでは64階層までプログラムとdoファイルをネストできます。

一般的にプログラムはdoやadoファイルで定義します。adoファイルの詳細は後述します。

プログラムは操作画面上でインタラクティブに定義できますが、実用的なことを考えるとテキストエディタに入力し、doファイルとして保存すると良いでしょう。

最初に定義したプログラムを確認します。

```
program hello
                    display "hi there"
end
```

これらのコマンドはインタラクティブに入力しますが、複雑な処理の場合は面倒です。そのようなケースではdoファイルにコマンドを入力します。

```
program hello
        display "hi there"
end
```

begin hello.do

end hello.do

Stataの操作画面に戻って次のように入力します。

```
. do hello

. program hello
  1.         display "hi there"
  2. end

.
end of do-file
```

do helloと入力しただけでは何も画面上に表示しません。do helloと入力しても、それはdoファイル中のコマンドでプログラムを定義してに過ぎません。doファイルの内容はhelloというプログラムを定義しているだけで、実行している訳ではありません。プログラムをロードしたら、それを次のようにして実行します。

```
. hello hi there
```

ここまでの操作ではdoファイルとプログラムを一緒に利用しました。インタラクティブな動作を目的に新しいコマンドを定義する場合は

1. doファイル内でprogram ... end 形式で新しいコマンドを記述します。
2. 新しいコマンドを使用する前にdoファイルを実行します。
3. 当該セッションにおいて新しいコマンドを利用します。

doファイルを自動的にロードして上記の操作を行う便利な方法もありますが、それは後述します。この章では上記の方法を利用します。

doファイルとプログラムを一緒に利用するもう一つ方法として、プログラムの定義と実行コマンドを一つのdoファイルの中に記述するという方法があります。

begin hello.do

```
program hello
        display "hi there"
end
hello
```

end hello.do

このdoファイルを実行した時の画面を次に示します。

```
. do hello

. program hello
  1.         display "hi there"
  2. end

. hello
hi there

.
end of do-file
```

ここではdoファイルとプログラムを組み合わせて利用しています。なぜでしょうか。helloの内容が複雑な場合、それをその都度入力するのは面倒です。そのような場合はプログラムをdoファイルの中に書きます。このようにすれば分析をさらに進める過程で、doファイルの中の他の箇所で、このコマンドを利用することもできます。つまり、操作画面からインタラクティブに新しいコマンドとして利用するのではなく、必要に応じてdoファイルの中で他の目的のために利用できます。

プログラムとdoファイルには実に様々な使い方がありますが、インタラクティブにprogramコマンドを利用するケースはあまり見かけません。一般的にはテキストエディタにプログラムを書き下します。そしてdoファイルの中でプログラムを定義し、そのdoファイルを実行します。

もう一つ、覚えておいてください。一度定義したプログラムを再度、同じ名前で定義することができません。

```
. program hello
program hello already defined
r(110);
```

doファイルとして定義したプログラムをhelloと入力して、実行(定義)することができません。

```
. do hello

. program hello
program hello already defined
r(110);

end of do-file
r(110);
```

再度、定義する場合はprogram drop helloとします。このように操作した後で、doファイルの内容を編集します。

begin hello.do

```
program drop hello
program hello
        display "hi there"
end
hello
```

end hello.do

しかし、このように操作しても実際にはうまくいきません。試しにdoファイルを再度実行すると次のようになります。

```
. do hello

. program drop hello
hello not found
r(111);

end of do-file
r(111);
```

この問題を解決するためには次のように操作します。

begin hello.do

```
capture program drop hello
program hello
        display "hi there"
end
hello
```

end hello.do

captureをコマンドの前に付けると、コマンドの内容を画面に表示します。詳細は[P] **capture**をご参照ください。プログラムを記述しているdoファイルでは、プログラムの定義の前にcapture program dropと記述するのが一般的です。

programコマンド自体の詳細は[P] **program**を参照してください。このコマンドはプログラムファイル自体を操作するものです。つまり、プログラムの定義、削除、ディレクトリの表示などの操作を行います。

プログラムではStataのコマンドが利用できますが、コマンドの中に特別な機能を持つものもあります。詳細は*Stata Index*の目次にある*Programming*で始まる項目をご参照ください。

18.3 マクロ

プログラミングについて説明する前にプログラムで利用するマクロについて解説します。

*マクロ*は文字列のことであり、マクロ名で指定します。中身の文字列を意味する場合はマクロの中身という具合に区別して呼ぶことにします。

マクロにはローカルとグローバルの2種類があります。ここで利用頻度の高いローカルマクロからご説明しますが、両者はほぼ同じものと考えてください。

18.3.1 ローカルマクロ

ローカルマクロに利用できる文字数は31です。

localコマンドでローカルマクロの内容をセットします。この操作はインタラクティブに行えます。まずはマクロがどのようなものか、実際に体験して学ぶことにしましょう。次のように入力します。

```
. local shortcut "myvar thisvar thatvar"
```

'shortcut'の中身は"myvar thisvar thatvar"です。シングルクォーテーションでショートカットを囲みます。ここで細かく説明します。

> 'shortcut'と入力した場合、
> 文字列の左右にシングルクォーテーションを付けます。
> Stataはショートカットの中身をmyvar thisvar thatvarと理解します。

マクロの中身にアクセスする場合は、上記のようにマクロ名の前に左シングルクォーテーション(シフトキーを押して@記号)、そして後ろに右シングルクォーテーション(シフトキーを押して7)を付けます。

このようにショートカットを囲むシングルクォーテーションのことをマクロ置換記号と呼びます。shortcutとは文字列の短縮形のことです。上記の例では'shortcut'がmyvar thisvar thatvarを指します。

したがって、次のように入力しますと、

```
. list `shortcut'
```

次のコマンドと同じ動作を実現します。

```
. list myvar thisvar thatvar
```

マクロはStataのどこからでもアクセスできます。例えば、次のように定義します。

```
. local cmd "list"
```

次のように実行します。

```
. `cmd' `shortcut'
```

これはlist myvar thisvar thatvarとまったく同じになります。

次のような例について考えてみましょう。

```
. local prefix "my"
. local suffix "var"
```

その場合、

```
. `cmd' `prefix' `suffix'
```

これはlist myvarと同じことになります。

ここでStataにおけるシングルクォーテーションの用法について確認しておきます。特にマクロと利用する場合は注意してください。(参照[U] 18.3 マクロ). ここでStataにおけるシングルクォーテーションの用法について確認しておきます。このマニュアルで説明する内容は日本語キーボードにおけるシングルクォーテーションの用法です。

英語キーボートを利用している場合は、両キーボードの違いに注意してください。左シングルクォーテーションは日本語キーボードの、シフトキーを押して@記号で入力します。そして、右シングルクォーテーションはシフトキーを押して7で入力します。日本語キーボードの場合、左右のシングルクォーテーションが英語のようにペアになっていません。英語PDFマニュアルとは表示される記号が異なりますので注意しましょう

日本語キーボードの場合、左右のシングルクォーテーションが英語のようにペアになっていません。ちなみに、ダブルクォーテーションの場合は通常通り、shiftキー+2で入力します。

18.3.2 グローバルマクロ

Stataにはローカルとグローバル、2つのマクロがありますが、ここでは、その違いではなく、グローバルマクロの動作について説明します。

グローバルマクロに利用できる文字数は32です。グローバルマクロの定義はlocalコマンドでなく、globalコマンドを利用します。

```
. global shortcut "alpha beta"
```

グローバルマクロの中身を取得する場合は、マクロ名の先頭に$記号を付けて、$shortcutとします。ですから$shortcutは"alpha beta"と同じです。

前のセクションではローカルマクロ名をshortcutとしました。上記の例では'shortcut'がmyvar thisvar thatvarを指します。

ローカルマクロとグローバルマクロで同じ名前を利用できますが、2つの間に関連性は生じませんし、識別できます。

グローバルマクロは、その中身はglobalで操作でき、その際にマクロ名の先頭に$記号を付けることを除けば、ローカルマクロとほぼ同じものです。

18.3.3 ローカルとグローバルの違い

ローカルとグローバルの違いはプライベートに利用するか、またはパブリックに利用するか、という所にあります。

次のようなプログラムがあるとします。

```
program myprog
code using local macro alpha
end
```

myprogにおけるローカルマクロalphaを他のプログラムで編集したり、その内容を参照することはできません。より明確にその違いを示すために次のようなプログラムを想定します。

```
program myprog
        code using local macro alpha
        mysub
        more code using local macro alpha
end
program mysub
        code using local macro alpha
            end
```

myprogは途中でmysubをコールしますが、myprogとmysubは両方ともローカルマクロalphaを利用します。しかし、両プログラムにおけるローカルマクロは中身が異なるかもしれません。mysubのalphaマクロはmyprogのalphaマクロと独立であり、何の関連性もありません。mysubを実行したとしても、そのalphaマクロはmyprogのそれとは無関係です。mysubのalphaマクロはmyprogのalphaマクロと独立であり、何の関連性もありません。

繰り返しますが、myprogのalphaとmysubのalphaはまったくの別物ですローカルマクロを利用する場合、同名のローカルマクロの存在を気にする必要はありません。

ローカルマクロは当該プログラムだけに対応します。一方のグローバルマクロは、すべてのプログラムから利用できます。myprogとmysubがクローバルマクロbetaを利用している場合、それは同一のマクロであると言えます。mysubを実行した時の$betaの中身が何であれ 、mysubの実行結果に対応する値となります。そして、mysubを実行した場合の$betaの内容に関係なく、myporgにコントロールが戻った時の内容が入ります

18.3.4 マクロと数式

ここではローカルマクロとグローバルマクロの適した利用方法を解説します。

次の用法をご覧ください

```
. local one 2+2
          . local two = 2+2
```

(上記の用例をglobalで置き換えて考えることも可能です。)どちらの1行目のコマンドには等号がありませんが、2行目のマクロには等号があります。より正確に書くならば1行目は次のようになります。

```
          . local one "2+2"
```

しかし、Stataではマクロの宣言においてダブルクォーテーションを省略できます。

local one 2+2(ダブルクォーテーションは略)は文字列2+2をマクロoneの中身として保存します。
local two=2+2 は2+2を計算して、その答えの4をマクロtwoの中身として保存します。

すなわち、

```
local macname contents
```

計算結果をマクロに割り当てる場合は、

```
local macname = expression
```

expressionの部分には式を書き、その結果をmacnameに保存します。

2番目の書き方でexpressionの部分には数式、または文字列を利用します。2+2は数式表現です。文字列による表現の利用例を次に示します。

```
. local res = substr("this",1,2) + "at"
```

右辺の結果をresに保存します。

右辺には数式、または、文字のどちらも利用できます。次にその用例を示します。

```
. local a "example"
. local b = "example"
```

どちらもマクロとしてexampleという文字列を保存します。最初のコマンドは文字列をコピーし、2番目のコマンドは"example"という文字列を評価して代入します。2番目のコマンドの右辺にはさらに複雑な式を記述することができます。

マクロを利用する上での注意点や、他のプログラム言語に慣れ親しんだユーザにとっては留意すべき事柄があります。例えば、マクロ`i'に5が入っているものとします。iに1を足して5+1=6にするにはどのようにしたら良いでしょうか。答えは、

```
local i = `i' + 1
```

右辺にはシングルクォーテーションを付け、左辺に付けない理由が分かりますか。重要なポイントとして`i'はローカルマクロiの中身(5)を参照します。つまり、次のような処理を行うことになります。

```
local i = 5 + 1
```

これは目的通りの結果をもたらします。

次の方法はC言語のプログラム経験のあるユーザにとっては慣れ親しんだ方法かもしれません。

```
local ++i
```

Cプログラミングの経験者からすると、local ++iはlocal i=i+1よりも効率的(処理が速い)に思えるかもしれませんが、実際には同じです。ローカルマクロで減算する場合は次のようにします。

```
local --i
```

local --iはi=i-1と同じですが、local --iの処理速度の方が優れています。最後に、

```
local i++
```

としても、iを増やすことはできません。これはローカルマクロiが++を含むように再定義することになります。しかし、i++(およびi--)で期待通りの処理を行う方法もあります。詳細は[U] 18.3.7 マクロの増加/減少関数を参照してください。

18.3.5 ダブルクォーテーション

ローカルマクロ`answ'にはyesまたはnoが入るものと考えてください。answの内容によってプログラムは異なる処理を実行します。

```
if "`answ'" == "yes" {
        ...
}
else {
        ...
}
```

"`answ'"という記述方法が気になると思いますで、中身を実際に置き換えて考えてみます。answの中身に具体的な文字を代入すると

```
if "yes" == "yes" {
```

または

```
if "no" == "yes" {
```

このような形になります。左辺のダブルクォーテーションを省略すると、

```
if no == "yes" {
```

(ここではnoが入っているものとしますが)、これは意図した処理とは異なります。これを文字通り解釈すると、noは文字列でなく、データにおける変数名と解釈されます。

この種の問題は、すべて置換した後の形で考えます。

ダブルクォーテーションは文字列を囲む目的で利用します。"yes", "no", "my dir¥my file", "`answ'"(これらはローカルマクロansの中身を意味します。)ダブルクォーテーションはマクロで利用する場合は、

```
local a "example"
if "`answ'" == "yes" {
                    ...
}
```

また、Stataコマンドでもダブルクォーテーションを利用するケースがあります。

```
. regress lnwage age ed if sex=="female"
. generate outa = outcome if drug=="A"
. use "person file"
```

引用符付きのマクロを使う場合は、ダブルクォーテーションを省略できません。

```
. regress lnwage age ed if sex=="`x'"
. generate outa = outcome if drug=="`firstdrug'"
. use "`filename'"
```

Stataには2組のダブルクォーテーションがあります。一つは""です。もう一つは`""'です。両方とも動作は同じです。

```
. regress lnwage age ed if sex==`"female"'
. generate outa = outcome if drug==`"A"'
. use `"person file"'
```

ですが、 普通の""の代わりに、単純に`""'を利用することはできません。上級者ならば、次のようにプログラムするかもしれません。

```
local a `"example"'
        if `"`answ'"' == `"yes"' {
                    ...
}
```

`"example"'が"example"よりも良く、`"`answ'"'が"`answ'"よりも良く、同様に,`"yes"'が"yes"よりも良いと考えられるでしょうか?確かに`"`answ'"'は"`answ'"よりも好ましいですが、`"example"'と`"yes"'は、"example"と"yes"に比べ、良くも悪くもありません。

`"`answ'"'が"`answ'"よりも良い理由はマクロansが、その中にダブルクォーテーションを含むからです。コンパウンド型のダブルクォーテションはネストさせて利用することができます。例えば、`answ'の中身が“I "think" so”であるとします。その場合、

```
Stataは、if "`answ'"=="yes"というコマンドを見つけると、
"I "think" so"=="yes"として解釈できなくなります。
また、Stataは、if `"`answ'"' == `"yes"'とあったとしても
`"I "think" so"' == `"yes"'と翻訳して解釈できなくなります。
```

左のダブルクォーテーション記号と右のダブルクォーテーション記号は見た目は同じです。これをコンパウンド形で入力すると、左は‘″、右は″’となり判別できるようになります。よって、Stataではコンパウンド形で入力すれば、対応関係は明確になります。‘″I ″think″ so″’はStataが簡単に解釈可能ですが、″I ″think″ so″とすると、判然としません。(試しに″A″B″C″だとした場合の意味について考えると良いかもしれません。はたして、A″B″Cなのか、または引用符付きのAの後ろにB、そして最後に引用符付きのCとしたいのか、判別できません。)

Stataはオープン、およびクローズの引用符は正しく理解できます。ダブルクォーテーションの場合もそれは同じです。‘″I ‘″think″’ so″’(″A″B″C″の意味はどう解釈されるでしょうか。‘″A ‘″B″’ C″’または ″A″’ B ‘″C″’と解釈できます。

コンパウンド形のダブルクォーテーションは、複雑になりがちです。特に、マクロ置換の記号‘’と組む合わせた場合は戸惑うかもしれません。したがって、プログラムの解説の中でもこれを使うことはあまりありません。プログラムを記述する場合、引用符の特定の利用法に固執する必要はありません。例えば、次のような使用方法でも許されます。

```
local a ″example″
if ‘″‘answ’″’ == ″yes″ {
                    ...
}
```

必要に応じてコンパウンド形のダブルクォーテーションを利用(‘″‘answ’″’)し、必要性が無い場合(例えば ″yes″)の場合は普通にダブルクォーテーションだけを利用します。マクロの中にダブルクォーテーションが含まれていないことが分かっている場合や動作に影響を及ぼすことがない場合は、わざわざ、″‘answ’″とする必要はありません。

しかし、上級プログラマの場合、コンパウンド形のダブルクォーテーションを利用する場面があるかもしれません。後々、Stataコマンドの文法を学び、その知識を元に次のようなプログラムを書いたと想定してください。

```
. myprog mpg weight if strpos(make,″VW″)!=0
```

ここでif文以下をローカルマクロで記述することもできます。つまり、‘if’として“if strpos(make,″VW″)!=0”のマクロを記述します。ifの中身をプログラミングする方法を考えてみましょう。例えば、次のように書くこともできます。

```
if ‘″‘if’″’ != ″″ {
      // the if exp was specified
      ...
}
else {
      // it was not
      ...
}
```

ここではマクロ‘if’の部分でコンパウンド形のダブルクォーテーションを利用します。ローカルマクロ‘if’はダブルクォーテーションを含みますので、ここではコンパウンド形のダブルクォーテーションを利用します。

18.3.6　マクロ関数

マクロの右辺に=expと書けますが、それ以外にもローカル及びグローバルマクロの機能を利用可能です。マクロ機能する際はコロン(:)をマクロ名の後ろに付けます。例えば、

```
local       lbl : variable label myvar local filenames : dir ″.″
files ″*.dta″ local       xi : word ‘i’ of ‘list’
```

マクロの機能の一つとして情報取得機能があります。最初の例では変数myvarに変数ラベルをマクロlblに保存します。さらに、情報を収集のための操作に関するものがあります。2番目の例ではマクロfilenamesに、カレントディレクトリに存在する.dtaファイル名を保存します。また、引数を使ってなんらかの操作を実行し、結果を得るという機能もあります。3番目の例ではxiに‘list’の‘i’番目の要素を格納します。マクロ機能に関する詳細は[P] macro をご参照ください。

情報を取得する手段として、c()を利用する方法があります。詳細は[P] creturnをご参照ください。

```
local  today ″‘c(current_date)’″ local curdir ″‘c(pwd)’
″
local    newn = c(N)+1
```

c()は特定の値を取得する機能を持っており、式の中で利用する事によって目的の値を取得することができます。c(current date)は”*dd MON yyyy*”の形式で日付を取得します。最初の例ではコマンド実行日をマクロtodayに格納します。c(pwd)は、例えば C:¥data¥projのような形でカレントディレクトリを取得します。2番目の例ではカレントディレクトリをマクロcurdirに格納します。c(N)はメモリーに展開したデータの個数を取得します。3番目の例題ではマクロnewnにはデータ個数に1を足した数を取得します。

ここでc()の意味を確認します。最初の2つの例は次のように書くこともできます。

```
local   today  = c(current_date) local  curdir  = c(pwd)
```

c()はStataの関数sqrt()と同じ意味で利用します。つまり、式の中でc()を自由に利用できます。c()はマクロ拡張機能の特別なものですが、このc()はマクロ拡張符の中でも利用できます。この点だけはsqrt()とは異なります。

データセットや操作環境に関する情報を取得する場合は[P] macroおよび[P] creturnを参照してください。ある特定の場所に関連する情報を取得するようケースを考えます。カレントディレクトリについては次のようにします。

```
local curdir = c(pwd)
```

または

```
local curdir : pwd
```

どちらのコマンドを利用しても、この場合、同じ情報を取得できます。

18.3.7 マクロの増加/減少関数

マクロの増加関数については[U] 18.3.4 マクロと数式で解説しました。次の構文は

```
command that makes reference to ‘i’ local ++i
```

このようなコマンドの用法はStataの中では一般的です。処理速度が速く、プログラムを参照する度にiを増加させることができます。Stataでは次のように利用します。

```
while ( ‘++i’ < 1000) {
        ...
}
while ( ‘i++’ < 1000) {
        ...
}
while ( ‘--i’ > 0) {
                    ...
}
while ( ‘i--’ > 0) {
                    ...
}
```

ここではStataのwhileコマンドの中での用法を示しましたが、++や--はプログラム中のどこでも利用できます。基本的にマクロ置換の文脈の中で利用します。

マクロ名の前に++または--を利用する場合、マクロは最初に増加/減少を行った上で、置換を実行します。

マクロ名の後ろで++または--を利用する場合、マクロは最初に置換し、それから増加/減少を行います。

❑ テクニカルノート

コマンド行の一部を実行しない場合、インラインで++や--の演算子は利用しないでください。次のような例を考えます。

```
if ( ‘i’ ==0) local j = ‘k++’
```

一方、

```
if ( ‘i’ ==0) {
       local j = ‘k++’
}
```

最初のプログラムはコマンドを解析する前にマクロを拡張してしまうので、目的の処理とは異なる結果を生むと考えられます。最初の例は常にk分増加し、2番目の例は‘i’==0の時だけkだけ増加します。

❑

18.3.8　マクロの記法

次のようなマクロの記法と

```
command that makes reference to ‘=exp’
```

次の2行に分けるコマンドは同じものです。

```
local macroname  = exp
command that makes reference to  ‘macroname’
```

最初の記法は処理速度が速く、しかも簡単に入力できます。長いコマンド文の中で‘=exp’という表現を利用すると、expの部分はStataが内部的に解釈し、その結果をこの部分に代入します。そして、コマンド全体を実行します。例えば、

```
summarize u4 summarize u ‘=2+2’
summarize u ‘=4*(cos(0)==1)’
```

これらのコマンドはすべて同じ結果を返します。expの部分にはStataの一般的なコマンド文、変数、行列、スカラー、他のマクロへの参照などのコマンドを利用できます。必ず引用符でサブマクロを囲むようにします。

```
replace "var" = "group"[ "=" j" +1" ]
```

また、次のようなコマンドは

```
command that makes reference to ":macro function"
```

次の2行に分けるコマンドは同じものです。

```
local macroname : macro function
command that makes reference to  "macroname"
```

したがって次のようなプログラミングも可能です。

```
format y ":format x"
```

変数yに変数xと同じフォーマットを適用します。

❑ テクニカルノート

マクロを拡張する演算子として.(ドット)というものがあります。これはStataのクラスシステムを接続する場合に利用します。詳細は[P] classを参照してください。

マクロ拡張関数macval()は"macval(name)"という具合に利用するコマンドで、nameを最初のレベルだけに制限します。したがって、nameの内部に埋め込まれている、より深いレベルのリファレンスは実行しません。このコマンドを利用するユーザは極めて少ないものと思われますが、これを利用する場合は[P] macroおよび、[P] fileに例題がありますので、参照してください。

❑

18.3.9 ローカルマクロの操作(上級者向け)

ここではマクロの置換機能についても理解を助けるための解説を行います。実際にプログラムの最中に起りそうな例を用いて説明します。

1. 今、マクロとしてx1, x2, x3があるものとします。"x1"とすれば、x1を、"x2"とすれば、x2を参照します。そこで、"x"i""とすると何を参照するでしょうか。"i"には6が入っているものとします。

   最初に内部の参照から開始しますので、

   "x"i""は"x6"となります。

   そして"x6"はローカルマクロx6の内容を参照します。

   マクロがベクトルのようにセットになっています。

2. 次にマクロを隣り合わせた場合の例を説明します。例えば、"alpha"はmyを、そして"beta"がvarを含んでいるものとします。この状態で"alpha""gamma""beta"とすると、どうなるでしょうか。gammaは未定義のままとします。

   Stataはマクロの存在自体は考慮しません。つまり、マクロが定義されていない場合、その内容は空であると解釈します。ローカルマクロgammaが未定義の場合、

   "gamma"は空のまま展開されることになります。

   エラーにはなりません。つまり"alpha""gamma""beta"を評価するとmyvarになります。

3. 逆にローカルマクロをクリアする場合は、次のように空にします。

```
local macname
or   local macname ""
or   local macname = ""
```

18.3.10　グローバルマクロの操作(上級者向け)

グローバルマクロの利用頻度は決して高いものではありません。一般的にはプログラム間でのコミュニーケションのために利用します。ローカルマクロで事足りるような状況で、敢えてグローバルマクロを使う必要はありません。

1. 例えば、xiというグローバルマクロを定義したとします。$xがthisを、 $i には6が入っているとすると、 xiはthis6になります。仮に$xが未定義の場合、$x$iは、未定義のローカルマクロの場合と同じ理由で6だけを返します。
2. マクロをネストする場合は括弧を利用します。つまり、$iに6が入っている場合、${x$i}は${x6}になり、最終的に$x6の内容を出力します。もし、$x6が未定義の場合は何も出力しません。
3. グローバルマクロとローカルマクロを組み合わせて利用することもできます。例えば、ローカルマクロjに7が入っているものとします。そこで、${x‘j’}とします。
   結果として $x7の中身を出力します。
4. 同様にグローバルマクロの内容に対してカッコを利用して情報を転送することもできます。例えば、マクロdriveの中身に“d:”が入っているものとします。driveがローカルマクロの場合、次にようにすると、

   ```
   ‘drive’myfile.dta
   ```

   d:myfile.dtaを出力します。driveがグローバルマクロの場合は次のように入力します。

   ```
   ${drive}myfile.dta
   ```

   次のように入力するのは誤りです。

   ```
   $drive myfile.dta
   ```

   これはd: myfile.dtaを返却します。次のように入力するのは誤りです。

   ```
   $drivemyfile.dta
   ```

   これは.dtaを返却します。
5. Stataは$記号があると、それをグローバルマクロとして解釈します。ですから記号としての$を利用する場合は注意が必要です。例えば、$22.15をdisplayコマンドで表示する場合、display ″¥$22.15″と入力します。しかし、この場合、″$22.15″でもそのまま出力できます。しかし、$thisと画面出力する場合、″$this″ではうまく行きません。″¥$this″と入力する必要があります。また、別の方法として、SMCLコードを利用してドル記号を出力することもできます。すなわち、display ″{c S|}this″とします。詳しくは[P] smclをご参照ください。
6. ドル記号をマクロの中に記述し、置換機能を抑制する方法があります。最初に置換を抑制しない場合の出力について確認します。次のような定義について考えてみます。

   ```
   global baseset ″myvar thatvar″ global bigset ″$baseset thisvar
   ″
   ```

   $bigsetは“myvar thatvar thisvar”と同じです。ここでマクロbasesetを再定義します。

   ```
   global baseset ″myvar thatvar othvar″
   ```

   bigsetの定義は変わりません。その中身は“myvar thatvar thisvar”です。basesetの定義で利用されているbigsetは定義した時のまま変わっていません。bigsetはbasesetとの関係性に関する情報を持っていません。

   代わりに、bigsetを次のように定義したと仮定しましょう。

   ```
   global bigset ″¥$baseset thisvar″
   ```

このように入力します。$bigsetは “$baseset thisvar”に等しく、結局、“myvar thatvar othvar thisvar”となります。bigsetは明らかにbasesetに依存していますが、いつでもbasesetの定義を変更できます。bigsetの定義を自動的に変える方法はを参照してください。

18.3.11 マクロを利用してウィンドウズのファイル名を作成する

Stataは¥(バックスラッシュ)をマクロの展開を抑制するコマンドとして利用します。

ウィンドウズは¥をディレクトリとして解釈します。

一般的にはファイル名の中で¥を利用しても問題は生じません。しかし、プログラム中でWindowsのパスを作成するコードを書く場合は両者を混在させないでください。例えば、ファイル名をfnameとします。

‘path’¥‘fname’

この場合¥があると、‘fname’の中身を解釈しません。この場合は次のようにします。

‘path’/‘fname’

Stataは/をすべてのプラットフォームのディレクトリとして解釈します。

18.3.12 システム値へのアクセス

Stataから円周率のパイ、日付や時刻、カンレントワーキングディレクトリなどのシステムパラメータや設定情報にアクセスする場合があります。

システム値へのアクセスにはStataのcクラス値を利用します。文法はローカルマクロの場合と同じです。円周率のパイを利用する場合は‘c(pi)’とします。このコマンドを利用すると、3.141592653589793という文字列を取得できます。

```
. display sqrt(2*‘c(pi)’) 2.5066283
```

現在時刻を参照する場合は、

```
. display ″‘c(current_time)’″ 11:34:57
```

cクラス値はシステムパラメータや設定を取得する際に利用します。システムパラメータとしてはシステムディレクトリ、システムの仕様、文字列、メモリー設定、メモリー中のデータの属性、出力設定、ネットワーク設定、Stataのシステム設定、デバック設定に関する情報が用意されています。

これらの情報に関する詳細は[P] creturnをご参照ください。次のように入力します。

```
. creturn list
```

現在の設定状態を確認できます。

18.3.13　構造情報の参照

構造情報とは変数に付属する文字情報のことでマクロと似た役割を果たす機能です。詳細は[U] 12.8 構造情報をご参照ください。構造情報とは例えば、mpg[comment]のようにvarname[charname]の形式で書き、中に含まれる文字列を参照する場合はマクロと同じように操作します。

varname[charname]の中身を参照するには‘varname[charname]’と入力します。
例えば、‘mpg[comment]’

中身の情報を設定する場合はcharコマンドを利用します。

```
char varname[charname] [["]text["]]
```

=expという用法を除けば、これはlocalやglobalコマンドと用法は同じです。中身を空にする方法はマクロの場合と同じく、中身に空白を設定します。

次のようにします。　　char varname[charname]
または、　　char varname[charname] ""

構造情報はデータに付属してファイルに保存されます。したがって、データファイルを閉じても情報はファイルに残します。ですから、変数を削除すると、その情報を一緒に失われます。_dta[charname]はデータファイル自体に結合しますので、個別の変数とのつながりはありません。

マクロの場合と同じように、構造情報は引用符を使って参照することもでき、その内容は引用符で囲んだ情報で置換できます。マクロがそうであったように、中身を空にする場合は、空白で置換します。

18.4　プログラムの引数

プログラムやdoファイルを実行する際、プログラム名やdoファイル名の後ろに入力する情報は引数として認識されます。例えば、xyzという名前のプログラムがあるとして次のように入力します。

```
. xyz mpg weight
```

この時、mpgとweightはプログラムの引数であり、mpgが1番目の引数、weightは2番目の引数となります。

プログラムの引数はローカルマクロを介してプログラムに引き渡されます。

マクロ	内容
‘0’	ユーザが入力したそのままの情報が入ります。スペースやダブルクォーテーションもこの中に入ります。
‘1’	1番目の引数(‘0’の最初の単語)
‘2’	2番目の引数(‘0’の二番目の単語)
...	...
‘*’	すべての引数‘1’, ‘2’, ‘3’, ..., の順次全ての単語を空白区切りで。‘0’と良く似ていますが、少し異なります。こちらにはスペースやダブルクォーテーションは含みません。

すなわち、ユーザが入力した情報は3種類の方法で転送することができます。

1. ‘0’を利用すると、ユーザの入力した情報をそのまま、すべての転送できます。
2. スペース区切りで‘1’, ‘2’,...という順番で..引数を区切った形で、個別に転送します。
3. 個別の情報を‘1’ ‘2’ ‘3’と並べて、‘0’と同じような形で転送します。

これらの方法をひとつのプログラムの中で併用することはあまり考えられません。

参考までに言えば、‘*’は古いStataプログラムとの互換性のために用意してあるオプションです。

‘0’のオプションはStataの標準的な構文に則って操作すれば、まったく問題ありませんが、プログラミングの正確な知識を必要とします。詳細は[U] 18.4.4 標準的なStataの構文を理解するを参照してください。

引数の操作のうち、最も簡単なものは、位置のマクロ‘1’, ‘2’, ...を利用する方法です。

このセクションの最初にxyz mpg weightと入力してプログラムを実行するものとしました。‘1’がmpg, ‘2’が weightとなります。そして‘3’には何も入りません。

この二つの変数の相関を求めるプログラムを作成することにします。ご存知のようにStataには相関を求めるコマンド-correlate-が既に用意されいますので、これから作成するプログラムではこのコマンドを利用します。プログラム自体は意味の無いのないものですが、ここでは引数の取り扱いに注目してください。

プログラムは次のようになります。

```
program xyz
        correlate ‘1’ ‘2’
end
```

プログラムを定義したら、次のようにして実行してみましょう。

```
. use https://www.stata-press.com/data/r18/auto
(1978 automobile data)
. xyz mpg weight (obs=74)
```

	mpg	weight
mpg	1.0000	
weight	-0.8072	1.0000

動作を確認できたでしょうか?ここではxyz mpg weightと入力しました。そして引数1としてmpg,2としてweightを設定しました。プログラムはcorrelate ‘1’ ‘2’を実行します。その内容はcorrelate mpg weightとなります。

ここに示した例は、順番でしてする引数の例です。あまり実用的な例ではありませんが、利用方法はご理解いただけると思います。ただ、Stataの構文を意識した場合、少し難易度は上がりますが、xyzというコマンドも対応する構文を持つように設定することができます。

この位置で指定する引数は、プログラミング上、とても便利に利用できます。実際、他のプログラムから呼び出して利用するサブルーチンを作成する場合、位置で引数を指定する方法はとても便利です。

Stataはコマンド、またはdoファイルの後に続ける..情報を引数として、‘1’, ‘2’のように番号付けします。引数はスペース区切りや、引用符で区切られているものとします。基本的には1つの単語を一つの引数と認識しますが、ダブルクォーテーションで囲んでいる情報は一まとめにして認識します。

それでは引数を利用するプログラムを実際に見てみましょう。プログラムをさ作成する場合、あまりインタラクティブモードで作成することはありませんが、ここでは練習と割り切って、コマンドウィンドウに次のように入力します。

```
. program listargs
  1.   display "The 1st argument you typed is:      `1
  2.   display "The 2nd argument you typed is:      `2
  3.   display "The 3rd argument you typed is:      `3
  4.   display "The 4th argument you typed is:      `4
  5.   end
```

displayコマンドは単純にダブルクォーテーションの中身を表示するコマンドです。詳細は[P] displayを参照してください。

実際にプログラムを実行します。

```
. listargs
The 1st argument you typed is:
The 2nd argument you typed is:
The 3rd argument you typed is:
The 4th argument you typed is:
```

listargsと入力すると、画面にはプログラムの内容だけを表示します。ここでは何も引数を利用していませんので、当然の結果となります。今度は引数としてthis is a testという文字列を入力します。

```
. listargs this is a test
The 1st argument you typed is:  this
The 2nd argument you typed is:  is
The 3rd argument you typed is:  a
The 4th argument you typed is:  test
```

最初の引数は‘this’, 二番目は‘is’とう形になっていることが分かります。空白が引数のセパレータとなっています。しかし、文字列をダブルクォーテーションで囲むと、それて一つの情報として扱います。

```
. listargs "this is a test"
The 1st argument you typed is:  this is a test
The 2nd argument you typed is:
The 3rd argument you typed is:
The 4th argument you typed is:
```

ここでは‘this is a test’を一つの引数として設定しました。文字列をダブルクォーテーションで囲むと、Stataはそれを一つの塊と認識します。ここでは‘1’に‘this is a test’が入ります。

ダブルクォーテーションは複数の箇所で使うこともできます。

```
. listargs "this is" "a test"
The 1st argument you typed is:  this is The 2nd argument you typed i
s:  a test The 3rd argument you typed is:
The 4th argument you typed is:
```

最初の引数は‘this is’で、二番目の引数は‘a test’です。

18.4.1 位置の引数に名前を付ける

位置で指定する引数‘1’, ‘2’, ‘3’,...の代わりに、名前を付けることができます。例えば、n, a, bや、numb,alpha,betaのように、単純な数字の代わりに意味のある名前を付けることができます。プログラム中に‘1’, ‘2’,...という形式で記述されている場合、その意味を簡単に理解することはできません。

このような場合はargsコマンドを利用して位置で指定する引数に、分かりやすい名前を付けることができます。

```
program progname
        args argnames
        ...
end
```

例えば、プログラムで4つの引数を利用する場合、それらにvarname, n, oldval, newvalという名前を付けたとします。

```
program progname
        args varname n oldval newval
        ...
end
```

varname, n, oldval, newvalは新しいマクロになり、argsは‘1’, ‘2’, ‘3’, ‘4’はその内容をこれらのマクロにコピーします。‘1’, ‘2’, ‘3’, ‘4’の内容が変わることはありません。そして、新しい名前の代わりに、位置で引数を指定することも可能です。ただし、プログラムが正しい引数の情報を取得したか、ということを確認する機能はありません。例えば、ここで示すプログラムが2つの引数だけで足りてしまう場合、‘oldval’と‘newval’には何も入りません。 逆に5つの引数が存在する場合、5番目の引数に名前は付きませんので、そのままローカルマクロ‘5’として操作します。

xの範囲aからbまでの間に存在するn個のデータを作成するコマンドを作ってみましょう。例えば複雑な数式による関数をグラフ化するような場面を想定してください。xの範囲を詳細に設定するよりも、グラフする範囲だけを設定できれば便利です。（実際、Stataにはrangeコマンドがありますが、ここではそれを利用せずにプログラムを書いてみましょう。）

実際にプログラムを書く前にプログラムの概略を先に示します。xの範囲[a,b]にn個のデータを作成します。

1. clear
   メモリー中のデータをクリアします。
2. set obs n
   データセットの領域としてn行確保します。nが100の場合は、set obs 100とします。
3. gen x = (_n-1)/(n-1)*(b-a)+a
   _nは最初から内蔵されている変数で_n=1の時は1行目、2の場合は2行目を指します。詳細は[U] 13.4 System variables (_variables)を参照してください。

例えば次のような形でプログラムを作成できます。

```
program rng                                  // arguments are n a b clear
        set obs ‘1’
        generate x = (_n-1)/(_N-1)*( ‘3’ - ‘2’ )+ ‘2’
end
```

ここでは単に引数を順番で示しています。‘1’はn, ‘2’はa, そして‘3’はbに対応しています。次のように書き換えることで、簡単に分析の流れを確認することができます。

```
program  rng
        args n a b clear
        set obs ‘n’
        generate x = (_n-1)/(_N-1)*( ‘b’ - ‘a’ )+ ‘a’
end
```

18.4.2　位置の引数を増加させる

プログラムの引数がk個あったとすると、同じ処理をk回繰り返す事を意味します。kの大きさは問題ではありません。例としてsummarizeコマンドについて考えてみます。summarize mpgを実行すると、mpgの要約統計量を求めます。同じように、summarize mpg weightとすると、それぞれの要約統計量を求めます。

```
program ...
        local i = 1
        while `" `"i"' "' != "" {
                logic stated in terms of `"i"'  local ++i
        }
end
```

同様に、logicの部分に `"i"' だけを参照する場合は次のようにします。

```
program ...
        local i = 1
        while `" `"i"' "' != "" {
                        logic stated in terms of `"i++"'
        }
end
```

ここで `"i"' の用法について考えてみます。このwhileループのプログラムではダブルクォーテーションで囲んで、`" `"i"' "'となっています。例えば、iに1が入っているとします。つまり、`i'も1になります。この時、最初の変数の名前は`1'となります。引用符で囲んだ`" `"i"' "'は最初の変数の名前を表現しています。 つまり、このwhileのプログラムは変数名が空であるのか、もし、何か存在していれば、処理を実行する形になっています。`i'が2の時、`" `"i"' "' は2番目の変数名になります。そして変数名が空でなければ、処理を継続します。逆に名前は空である場合、処理を完了します。

仮にk個の変数を処理するサブルーチンを書く場合は、サブルーチン側では処理する変数の個数に関する情報を取得する必要があります。よって、最初に変数の数を数えて、その数kを元に処理を実行することになります。

```
program progname
        local k = 1                 // count the number of arguments while `" `"k"' "' != ""
        {
                            local ++k
        }
        local  --k                  // k contains one too many
                                    // now pass through again
        local i = 1
        while `i' <= `k' {
                code in terms of `"i"' and `k' local ++i
        }
end
```

この例ではwhileループを利用して、コマンドのループを実行します。Stataにはこの他にも、構文が分かりやすく、処理速度の速い2つのループコマンドforeachとforvaluesがあります。詳細は[P] foreachと[P] forvaluesを参照してください。これらのコマンドも機能としては実質的にwhileコマンドと同じです。

```
        local i = 1
        while `i' <= `k' {
                code in terms of `"i"' and `k' local ++i
        }
```

このプログラムは‘i’ = 1から ‘k’まで同じ処理を繰り返します。代わりに、次のように書くこともできます。

```
forvalues i = 1(1)‘k’ {
        code in terms of ‘‘i’’ and ‘k’
}
```

同様に、サブセクションの先頭で、whileループを使って次のように書くこともできます。

```
program ...
        local i = 1
        while ″ ‘‘i’’ ″ != ″″ {
                logic stated in terms of ‘‘i’’ local ++i
        }
end
```

次のようにも書けます。

```
program ...
        foreach x of local 0 {
                logic stated in terms of ‘x’
        }
end
```

詳細は[P] foreachと[P] forvaluesをご覧ください。

argsと位置を示す引数の数を使ってカウントを増加させることができます。例えば、varname(変数名n)を取得するサブルーチンを作成するものとします。取得する変数名の数は最小1個から最大20個までとします。変数の合計個数をｎで除して、その結果を最初の変数として保存します。プログラムの内容に意味がありませんが、引数の受け渡し方に注目してください。

```
program progname
        args varname n local i 3
        while ″ ‘‘i’’ ″ != ″″ {
                logic stated in terms of ‘‘i’’ local  ++i
        }
end
```

18.4.3 マクロシフトの利用

各引数について同じ処理を繰り返すコードとしてマクロシフトというものがあります。

```
program ...
        while ″ ‘1’ ″ != ″″ {
                logic stated in terms of ‘1’ macro shift
        }
end
```

macro shiftは‘1’, ‘2’, ‘3’,... と左方向にシフトします。よって、‘1’ はなくなり、‘2’は‘1’になり、そして‘3’は‘2’になります。

外側のwhileループはマクロ‘1’の内容が空になるまで繰り返されます。

macro shiftコマンドは古いコマンドで、もはや利用することはあまりありません。このコマンドの代わりに前のセクションで紹介したforeachやforevaluesを利用して、‘‘i’’を参照するループを作成します。

新しいコマンドを用意した理由を簡単に説明します。macro shiftは位置マクロ‘1’，‘2’を消失させてしまうので、ユーザはtokenizeで番号をリセットする事になります。そして、引数の数が多い場合(Stata/MPやStataSEなどの場合)、macro shiftコマンドの実行速度が非常に遅くなってしまいます。

❑ テクニカルノート

しかし、macro shiftにも他のコマンドには無い便利な機能があります。

番号マクロを利用する際に、‘*’を使って先頭以外の変数を指定することができます。例えば、プログラムにおいて先頭の変数を従属変数として認識し、二番目以降を独立変数として認識するものとします。そこで、最初の変数を‘lhsvar’、右辺の変数を‘rhsvars’として保存するものとします。プログラムは次のようになります。

```
program progname
        local lhsvar ″‘1’″
        macro shift 1
        local rhsvars ″‘*’″
        ...
end
```

今、一つのマクロに複数の変数名が入っているものとし、それを分割するような場面を想定します。‘varlist’には一般的なコマンドを実行した結果、変数が入っているものとします。(参照[U] 18.4.4 標準的なStataコマンドの構文)。その内容を‘lhsvar’と‘rhsvars’に分解します。tokenizeは番号マクロをリセットするコマンドです。

```
program progname
        ...
        tokenize ‘varlist’ local lhsvar ″‘1’″
        macro shift 1
        local rhsvars ″‘*’″
        ...
end
```

❑

18.4.4　標準的なStataの構文を解釈する

位置で指定する引数‘1’，‘2’，...を‘0’に変更します。

Stataの標準的な構文に従っていれば、コマンド全体を‘0’として操作できます。Stataの標準的な構文は次の通りです。

[by varlist:] command [varlist] [=exp] [using filename] [if] [in] [weight]
[, options]

詳細は[U] 11 Language syntaxをご覧ください。

syntaxコマンドは標準的な構文に分割するコマンドです。プログラムに入力するコマンドに長さの制約はありません。syntaxコマンドは‘0’　としてコマンド全体を把握し、構文の構成要素を確認します。コマンドの構文に誤りがなければ、情報を内部的に解釈します。しかし、コマンドの構文に合致しない場合、syntaxコマンドはその原因となるメッセージを返します。

オプションとexpと条件inを利用したプログラムの例について考えています。
in rangeプログラムの例を次に示します。

```
program ...
        syntax varlist(min=2) [if] [in]
        ...
 end
```

構文要素の設定方法に関する詳細は[P] **syntax**をご参照ください。コマンド文は一目で分かりやすい長さになっています。大括弧[ ]の部分はオプションの情報をを示しており、これらをsyntaxコマンドで利用することもできます。

syntaxコマンドは目的のコマンドの前につけて利用します。仮にユーザが複数の変数名やオプションでifやinを使った場合、syntaxコマンドは新しいローカルマクロを定義します。

`varlist'	複数の変数をリストします。
`if'	条件文を付ける
`in'	範囲を設定する

実際に次のような例を用いて動作を確認します。

```
program tryit
        syntax varlist(min=2) [if] [in]
        display "varlist now contains | `varlist' |"
        display  `"if now contains | `if' |"'
        display "in now contains | `in' |"
 end
```

さらに、

```
. tryit mpg weight
varlist now contains |mpg weight| if now contains ||
in now contains ||
. tryit mpg weight displ if foreign==1
varlist now contains |mpg weight displ|
if now contains |if foreign==1|
in now contains ||
. tryit mpg wei in 1/10
varlist now contains |mpg weight|
if now contains ||
in now contains |in 1/10|
. tryit mpg
too few variables specified r(102);
```

3番目の例では変数weightとweiに省略しました。もちろん、コマンドを実行した結果の部分では変数名をそのまま表示します。

プログラムが次のステップに変数リストを引き渡す場合、tokenizeコマンドで位置のマクロを`1', `2',...のようにリセットします。

```
tokenize `varlist'
```

[P] **tokenize**をご参照ください。tokenizeは`varlist'の先頭の語を`1',二番目の語を`2'にリセットします。

18.4.5　簡易コマンドの分解

[U] 19 簡易コマンドで紹介する簡易コマンドも引数を利用します。Stataコマンドの慣習上、簡易コマンドにはコマンドの末尾にiを付けます。例えば、mycmdiというコマンドが2つの変数をとるものとします。先頭のalphaは正の整数、2番目の変数はbetaとします。次のようなプログラムになります。

```
program mycmdi
        gettoken n 0 :0, parse(" ,")              /* get first number */ gettoken x 0 :0, parse(" ,")    /
        * get second number */ confirm integer number `n' /* verify first is integer */
        confirm number `x'                        /* verify second is number */
        if `n'<=0 error 2001                      /* check that n is positive */
        place any other checks here
        syntax [, Alpha Beta]                     /* parse remaining syntax */
        make calculation and display output
            end
```

詳細は[P] gettokenを参照してください。

18.4.6　標準的でない構文の分解

標準的ではない構文や位置の引数を分解する場合、プログラミングをかなり丁寧に行うことになります。実際に利用するコマンドはgettokenになります。

gettokenはユーザが指定した文字列からマクロの内容を取りだします。そして、新たにマクロを定義して残りの文字列を操作することも可能にします。例を用いて説明します。

`0'がすべての文字列を含むと、"this is what the user typed"となります。gettokenを実行した後は、次のようになります。

```
        新しいマクロ`token'の中身        "this"
        `0'の中身                        "this is what the user typed"
    または
        新しいマクロ`token'の中身        "this"
        新しいマクロ`rest'の中身         " is what the user typed"
        `0'の中身                        "this is what the user typed"
    または
        新しいマクロ`token'の中身        "this"
        `0'の中身                        "is what the user typed"
```

gettokenの構文は次の通りです。

gettoken emname*1* [emname*2*] : emname*3* [, parse(pchars) quotes match(lmacname) bind]

ここでemname*1*, emname*2*, emname*3*, lmacnameはローカルマクロ名です。(Stataはグローバルマクロも操作できますが、あまり利用頻度は高くありません。詳細は[P] gettokenを参照してください。)

gottokenはemname*3*から最初のトークンを取りだし、emname*1*に保存します。そしてemname*2*を指定している場合、emname*3*の残りの文字列をemname*2*に保存します。emname*1*, emname*2*, emname*3*が同じマクロでもかまいません。gettokenのコーディング例を示します。

```
gettoken emname1   :0 [, options]
gettoken emname1 0 :0 [, options]
```

‘0’にはユーザが入力したすべての情報がはいります。最初のコーディングはトークンが分かっている場合のもので、2番目のコードはトークンの存在だけを前提にしたものです。

gettokenのオプションは次の通りです。

parse(″string″)	文節を切り分ける文字 デフォルトはparse(″ ″)で、空白になっています。 一般的に空白をダブルクォーテーションで囲んだparse(‘″″ ″’)を利用することがよくあります。 (‘″″ ″’はスペースをダブルクォーテーションで囲んだ合体ダブルクォーテーションの文字列です。)
quotes	外側にダブルクォーテーションをつけても切り離さないmatch(lmacname)　括弧と大括弧で囲むlmacnameは“(”，“[”または空白で囲みます。その種別はemname*1*がカッコ、またはカギカッコのどちらで囲まれているかに依存します。 emname*1*は外側のカッコまたは大カッコを外したものです。

gettokenはemname*3*の先頭にクォーテーション、またはダブルクォーテーションがある場合、どちらも文字列をダブルクォーテーションで囲みます。parse(‘″″ ″’)を指定すると、中身の文字だけを取りだすことができます。

quoteはトークンを定義している元の情報からダブルクォーテーションを外さないようにします。例えば、“″this is″ a test”を分解する場合、quoteを使用しなければ、トークンは“this is”となり、quoteを利用した場合は“″this is″”となります。

match()はトークンの定義において、カッコ、カギカッコの指定を行います。カッコまたはカギカッコの外のレベルは取り除きます。“(2+3)/2”を分解する時にmatch()を利用すると、トークンは“2+3”になります。実際、match()は数式と一緒に利用しますが、変数名リストや時系列の変数リストを取りだす場合に良く利用されます。

18.5　スカラーと行列

マクロに加え、プログラミング用にスカラーと行列が用意されています。詳細は[U] 14 行列表現、[P] scalarそして[P] matrixを参照してください。

スカラーの計算に関しては、マクロとスカラーのどちらも利用しても同じです。ただし、マクロには数値も保存できます。Stataのスカラーは計算時間が若干速く、そしてマクロよりも精度は上です。ただ、処理速度の違いは実質的には無いに等しいとお考えください。マクロの場合、有効数字12ケタで、スカラーは16ケタです。繰り返し計算でなければ、どちらの場合もあまり深刻な差はありません。

スカラーに保存できる文字列はマクロのそれよりも長くなります。スカラーにはバイナリの文字列も保存できます。詳細は[U] 12.4.14 プログラム向けのノートをご参照ください。

StataにはMataと呼ばれる強力な行列プログラミング言語があります。その解説には別冊マニュアルも用意されています。Mataで作成したサブルーチンはStataのプログラムから呼び出すことができます。詳細はMataリファレンスマニュアル、特に[M-1] Adoをご参照ください。

18.6　メモリー上のデータを一時点に編集する

特別な目的のためにメモリー上にあるデータを編集することがあります。高性能なソフトウェアならば、必ず、編集後に元に戻す機能を有しています。preserveはまさにそのためのコマンドです。

```
code before the data need changing
preserve
code that changes data freely
```

preserveコマンドを実行すると、Stata/MPとStata/SEはユーザの操作しているデータをメモリ上に作成します。Stata/BEではディスク上に作成します。Stata/MPとStata/SEでは、このデータがディスク上に保存されないように、max_presermemで使用するメモリ量を設定します。詳細は[P] **preserve**をご覧ください。そしてプログラムが処理を終えた時点で、Stataはデータを復元し、コピーを削除します。詳細は[P] **preserve**を参照してください。

preserveを使用しない場合は、フレームを利用して変更する必要のあるデータのコピーを作成、新しいフレームでデータを操作し、操作後にフレームを削除します。操作例は[D] **frame prefix**を参照してください。

18.7　テンポラリオブジェクト

プログラムの中で、変数を確保するための一時的な変数、または一時的なスカラー、そして一次的なファイルを利用する場合があります。これらの一時的なブジェクトは基本的に計算中にだけ必要で、計算が完了した時点では不要になります。

Stataにはこれらの一時的なオブジェクトを作成するコマンドが用意されています。データセットにおいて変数を一時的に確保する場合はtempvar、スカラーと行列の場合はtempname、そしてファイルの場合はtempfileを利用します。詳細は[P] **macro**を参照してください。次に示す構文はすべて同じです。

{tempvar | tempname | tempfile} macname [macname ...]

このコマンドを実行すると、ローカルマクロを作成できます。

18.7.1　テンポラリな変数

計算途中でデータに変数sum_yとsum_zを追加することになったとします。当然、次のようなコマンドを利用したくなります。

```
...
generate sum_y = ...
generate sum_z = ...
...
```

しかし、データセットには既に同じ名前のデータが存在するので、上記のコマンドは実行できません。
仮に同名の変数を追加する場合は、既存の変数を削除する必要があります。よって、

```
...
tempvar sum_y
generate `sum_y' = ...
tempvar sum_z
generate `sum_z' = ...
...
```

または

```
...
tempvar sum_y sum_z
generate `sum_y' = ...
generate `sum_z' = ...
...
```

ここでは必ずしも‘sum_y’と‘sum_z’を削除する必要はありません。しかし、Stataはtempvarで命名したいかなる変数も自動的に削除します。tempvarコマンドを実行した場合、変数名は引用符付きで参照します。そしてそれがマクロであることを示します。しかし、引用符をつけずにtempvar sum_yと入力すると、それ以降、変数‘sum_y’を参照します。tempvarは一時的な変数は作成しません。代わりに、tempvarは後で変数を作成する場合に利用するネームを作成します。そしてtempvarはそのネームをユーザが設定したローカルマクロに保存します。

tempvarに関する詳細は[P] **macro**を参照してください。

18.7.2　テンポラリなスカラーと行列

tempnameはtempvarと同じような機能を持っています。例えば、次のようなコードがあるものとします。

```
tempname YXX XXinv
matrix accum `YXX' = price weight mpg
matrix `XXinv' = invsym(`YXX'[2..., 2...])
tempname b
matrix `b' = `XXinv'*`YXX'[1..., 1]
```

上記のコードはpriceにweightとmpgを回帰させた時の係数を確保します。行列コマンドの詳細は[U] 14 Matrix expressionsおよび[P] matrixを参照してください。

一次変数と同じように、プログラムの終了時に一時的なスカラーと行列も自動的に削除されます。

18.7.3 テンポラリファイル

一時的なファイルが必要になった場合、preserveコマンドでデータをメモリに確保することもできますが、一次ファイルを利用する場合は次のようにします。詳細は[U] 18.6 メモリー上のデータを一時点に編集するを参照してください。

複雑なプログラミングを行う場合、Stataでは次のようにして一次ファイルを作成することができます。例を次に示します。

```
preserve                          /* save original data */ tempfile males femal
es
keep if sex==1
save "`males'"
restore, preserve                 /* get back original data */ keep if sex==0
save "`females'"
```

一次変数、スカラー、行列と同じように一次ファイルを丁寧に削除する必要はありません。Stataはプログラムの終了時に自動的に削除します。

18.7.4 テンポラリフレーム

現在のフレームのデータセットに影響を与えず、一時的にメモリ上のデータセットを追加することができます。フレームのテンポラリネームを入手したり、データをコピー、ロードして操作することができます。操作が終わると、フレームとデータは自動的にメモリから削除されます。例を次に示します。

```
tempname frame
frame copy default `frname'
fname `frname' {
    commands which modify the data in frame `frname'
}
...
```

成否に関わらずプログラムが終了すると、作成されたフレームは自動的にメモリから削除されます。

18.8 他のプログラムの計算結果にアクセスする

Stataではある計算結果をその次に実行するコマンド、またはプログラムで連携して利用できるようになっています。この機能に関する詳細はコマンドごとに、リファレンスマニュアルの*Stored results*という項目で解説しています。結果の保存場所には3種類あります。

1. summarizeのようなr-classコマンドは計算結果をr()に保存します。ほとんどのコマンドはr-classコマンドです。
2. regressのようなコマンドはe-classコマンドで、計算結果をe()に保存します。e-classコマンドはStataのモデル推定コマンドです。
3. s-classコマンドの例として分かりやすいものがありませんが、s()に結果を保存します。利用頻度の低いコマンドですが、プログラミング時に文字列を分解する目的で使う事があります。

計算結果を保存しないコマンドのことはn-classコマンドと呼びます。より正確に言えば、generate newvar = ...のように、コマンドは結果の保存場所を基本的に必要とします。

例題1

平均の標準誤差を計算するプログラムを作成してみましょう。計算式は$\sqrt{s^2/n}$でs^2は分散です。(標準誤差はciコマンドで求めることができます。しかし、ここでは無いものと思ってプログラムを作成します。)[R] summarizeを参照すれば分かるように、平均はr(mean),分散はr(Var)、そしてデータの個数はr(N)に入ります。この情報を元にプログラムを作成します。

```
program meanse
        quietly summarize `1'
        display "       mean = " r(mean)
        display "SE of mean = " sqrt(r(Var)/r(N))
end
```

プログラムの実行結果を次に示します。

```
. meanse mpg
      mean = 21.297297
SE of mean = .67255109
```

r-classコマンドを実行した後で、return listコマンドを、または、e-classコマンドの後ではereturn listコマンドを実行します。Stataは保存結果を出力します。

```
. use https://www.stata-press.com/data/r18/auto
(1978 automobile data)
. regress mpg weight displ
  (省略)
. ereturn list
scalars:
                  e(N) =  74
               e(df_m) =  2
               e(df_r) =  71
                                e(F) =  66.78504752026517
              e(r2) =  .6529306984682528
            e(rmse) =  3.45606176570828
             e(mss) =  1595.409691543724
             e(rss) =  848.0497679157351
            e(r2_a) =  .643154098425105
              e(ll) =  -195.2397979466294
            e(ll_0) =  -234.3943376482347
            e(rank) =  3

macros:
            e(cmdline) : "regress mpg weight displ"
              e(title) : "Linear regression"
          e(marginsok) : "XB default"
                e(vce) : "ols"
            e(depvar) : "mpg"
                e(cmd) : "regress"
         e(properties) : "b V"
            e(predict) : "regres_p"
              e(model) : "ols"
          e(estat_cmd) : "regress_estat"
matrices:
                  e(b) :  1 x 3
                  e(V) :  3 x 3
               e(beta) :  3 x 3
functions:
             e(sample)
. summarize mpg if foreign
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
         mpg |         22    24.77273    6.611187         14         41
. return list scalars:
                  r(N) =  22
              r(sum_w) =  22
             r(mean) =  24.77272727272727
              r(Var) =  43.70779220779221
               r(sd) =  6.611186898567625
             r(min) =  14
             r(max) =  41
             r(sum) =  545
```

この例ではsummarizeコマンドの後でregressコマンドを実行しました。結果としてe(N)にはregressコマンドで利用したデータの個数74が入り、r(N)にはsummarizeコマンドで利用した22というデータの個数が入ります。r(N)とe(N)が異なる所がポイントです。

仮にtabulateのような他のr-classコマンドを実行すれば、r()の内容は変わります。しかし、e()の内容は変わりません。逆にprobitのようなe-classコマンドを実行すると、e()の内容は変わりますが、r()の内容は変わりません。e()の内容は他の推定コマンドを実行するまでは変わりません。r()の内容に影響されることもありません。例えば、e-classまたはn-classコマンドをサブルーチンで利用し、その結果を受けてe-classコマンドをサブルーチンで実行すると、r()の内容は変わります。Stataにはr-classコマンドが多いので、r()の内容は変わりやすいとご理解ください。

❑ テクニカルノート

なんらかのコマンドを実行した後でr()に保存されている計算にアクセスする方法を必ず覚えておきましょう。後の計算のために変数番号‘1’の平均と分散を取得する場面を考えます。最初に誤った例を示します。

```
summarize ‘1’
...
...r(mean) ... r(Var) ...
```

正しくは、

```
summarize ‘1’
local mean = r(mean) local var = r(Var)
...
... ‘mean’ ... ‘var’ ...
```

または

```
tempname mean var summarize ‘1’
scalar ‘mean’ = r(mean)
scalar ‘var’ = r(Var)
...
... ‘mean’ ... ‘var’ ...
```

❑

r(),e()のどちらも保存する情報はスカラー、マクロまたは行列です。たとえば、ereturn listまたはreturn listを実際に実行してみると、regressコマンドの場合は3種類の情報を保存することが分かります。それに対し、summarizeはスカラーだけを保存します。(regressは関数をe(sample)に保存します。これは他のe-classコマンドについても同じです。詳細は[U] 20.7 サブサンプルの利用を参照してください。)

e(name)またはr(name)の種類に環形なく、e(name)またはr(name)で情報を参照できます。計算結果の参照方法に関しては[U] 13.6 Stataのコマンドから結果にアクセスするをご参照ください。ほぼすべての情報が解説されています。ただし、保存された情報の参照方法は他にもあります。つまり、直接r(name)やe(name)を参照するのではなく、マクロの置換文字‘’を利用して‘r(name)’や‘e(name)’のようにしてリファレンス機能を利用する方法です。当然、結果はマクロ置換と同じものになります。中身を参照し、そして置換を行います。

```
. display ″You can refer to ″ e(cmd) ″ or to ‘e(cmd)’ ″ You can refer to regress or to r
egress
```

例えば、このコマンドで‘e(cmd)’の中身はregressなので、regressと入力したことと同じ事になります。

```
. ‘e(cmd)’
      Source |       SS       df       MS              Number of obs =        74
-------------+------------------------------           F(2, 71)      =     66.79
       Model |  1595.40969     2  797.704846            Prob > F      =    0.0000
(以下省略)
```

ereturn listでe(cmd)はマクロとして表示されています。このようにシングルクォーテーションの中にマクロ名を記述うすると、置換が実行されます。

この操作と同じ事を保存したスカラーやマクロでも実行できます。e(N)はスカラーの74です。これを利用して“display e(mss)/e(N)” や“local meanss = e(mss)/e(N)”のような形で利用でます。‘e(N)’は文字列の“74”で置換されますので、“local val‘e(N)’ = e(N)”のような形で利用できます(val74というマクロを作成します)。保存された計算結果へのアクセスを次にまとめます。

1. プログラム式の中で引用符を付けずにr(name)やe(name)で数値を参照できます。(s-class s(name)の場合はシングルクォーテーションが必要です。)
   1.1 nameが存在しない場合、欠損値(.)を返します。空の保存結果を参照した場合、それはエラーにはなりません。
   1.2 nameがスカラーの場合、倍精度の数値情報を返します。
   1.3 nameがマクロの場合、その中身が数値であるか否かをチェックします。数値の場合は、そのまま数値を返し

ます。文字列の場合はnameを返します。

1.4 nameが行列の場合、行列を返します。

2. どのようなコードであれ、‘r(name)’, ‘e(name)’, ‘s(name)’は引用符がついているので置換を実行します。

2.1 nameが存在しない場合、置換は実行されません。空の保存結果を参照した場合、それはエラーにはなりません。結果のウィンドウには‘r(name)’, ‘e(name)’, ‘s(name)’などの情報は表示しません。

2.2 nameがスカラーの場合、数値を文字属性で表示します。表示桁数は12ケタとなります。

2.3 nameがマクロの場合、その内容をそのまま返します。

2.4 nameが行列の場合、matrixという単語を返します。

一般的にスカラーや行列の場合は引用符を付けずにr(name) 、e(name)として参照します。マクロの場合は‘r(name)’, ‘e(name)’, ‘s(name)’のようにします。しかし場合によってはこれが逆になるケースもあります。例えば、r(example)に時点の数が入っている場合、r(example)はマクロとして保存され、スカラーにはなりません。この場合、プログラムでは引用符無しでr(example)を参照することで、目的の情報を取得できます。r(example)がスカラーの場合、操作も分かりやすいと思います。しかし、実際にはデータの形式は問題ではありません。

データの持ち方には注意しましょう。例えば、r(N)としてデータの個数がｽｶﾗｰで保存されているものとします。ここではオプションn(#)ではなく、他の方法について考えてみます。仮に、n(r(N))としてしまうのは誤りです。なぜならn()オプションの場合、カッコの中身は数値になります。したがって、ここではn(‘r(N)’)と入力することになります。

18.9 推定コマンドの計算結果にアクセスする

推定結果はe()に保存されます。これらの情報へのアクセスはここまで説明した方法と同じです。詳細は、[U] 18.8 他のプログラムの計算結果にアクセスするを参照してください。簡潔に言えば、

1. regress, logisticなどの推定コマンドはe()に入ります。
2. 推定コマンドはe(cmd)に入ります。例えば、regressは “regress”、poissonは“poisson”という文字列をe(cmd)に保存します。
3. 推定コマンドは実行したコマンド文全体をe(cmdline)に保存します。例えば、reg mpg displというコマンドを実行すると、“reg mpg displ”をe(cmdline)に保存します。
4. 推定コマンドはデータの個数をe(N)に、そして推定に利用した標本の情報をe(sample)に保存します。例えば、summarize if e(sample)とすると、推定に利用したデータの要約統計量を求めることができます。
5. 推定コマンドは係数ベクトルと分散共分散行列をe(b)とe(V)に保存します。これらの情報は行列です。演算は一般的行列と同じ要領で行います。

```
. matrix list e(b) e(b)[1,3]
        weight       displ       cons
y1   -.00656711   .00528078   40.084522


. matrix y = e(b)*e(V)*e(b)'
. matrix list y symmetric
y[1,1]
            y1
y1  6556.982
```

6. 推定コマンドは係数とその標準誤差を_b[*name*]と_se[*name*]というマクロで取り扱えるように設定します。詳細は[U] 13.5 係数と標準誤差にアクセスするを参照してください。
7. 推定コマンドはe()にスカラー、マクロ、行列による関連情報を保存します。この機能に関する詳細はコマンドごとに、リファレンスマニュアルの*Stored results*という項目で解説しています。

推定コマンドはr()にも結果を保存します。行列r(table)には出力された係数テーブルが保存されます。このテーブルには係数、標準誤差、検定統計量、p値、信頼区間が含まれます。

例題2

regressコマンドの後で実行するコマンドをプログラムする場合は、次のようなコードを入れておくといいでしょう。

```
if "`e(cmd)'" != "regress" {
        error 301
}
```

このコードはregressコマンドの後で保存された結果を利用しているか、確認するためのものです。Error 301は“直

近の推定結果が見当たらない”という事を示すStataのエラーです。

◁

18.10 結果を保存する

プログラムで何か計算を行った場合、その結果を他のプログラムで参照することができます。プログラムはインタラクティブに利用するだけでなく、他のコマンドのサブルーチンとして利用できます。

計算結果の保存場所を簡単にまとめます。

1. プログラムの中では取得する情報によってrclassはr(),eclassはe(),scalssはs()を使い分けます。
2. 次のコードは、

```
return scalar name = exp     (returnのないscalarと同じ構文です)
return local  name ...       (returnのないlocal同じ構文です)
return matrix name matname   (matnameをr(name)に保存します)
```

   計算結果をr()に保存します。

3. 次のコードは、

```
ereturn name = exp           (ereturnのないscalarと同じ構文です)
ereturn local  name ...      (ereturnのないlocal同じ構文です)
ereturn matrix name matname  (matnameをe(name)に保存します)
```

   計算結果をe()に保存します。係数ベクトルと分散行列e(b),およびe(V)は保存できません。代わりにereturn postを利用します。

4. 次のコードは、

```
sreturn local  name ...  (sreturnの無いlocalと同じ構文です)
```

   計算結果をs()に保存します(s-class はマクロのみ)。

プログラムにはr-class,e-class,s-classが用意されています。

18.10.1　計算結果をr()に保存する

[U] 18.8 他のプログラムの計算結果にアクセスするでは平均値と標準誤差の取得方法を紹介しました。 より便利方法としてr()の保存内容を利用する方法がありまので、ここで紹介します。

```
program meanse, rclass
        quietly summarize `1'
        local mean = r(mean)
        local sem  = sqrt(r(Var)/r(N))
        display "       mean = " `mean'
        display "SE of mean = " `sem'
        return scalar mean = `mean'
        return scalar se = `sem'
end
```

meanseを実行してr(mean)とr(se)に値を保存します。

```
. meanse mpg
      mean = 21.297297
SE of mean = .67255109
. return list scalars:
       r(se)        = .6725510870764975
       r(mean)      = 21.2972972972973
```

ここではプログラムステートメントにrclassオプションを追加しました。そして、プログラムの最後に2つのreturnコマンドを加えました。

プログラムの最後にreturnステートメントを追加しましたが、その位置が限定されている訳ではありません。もう少し、簡潔に書くと、次のようになります。

```
program meanse, rclass
        quietly summarize `1'
        return scalar mean = r(mean)
        return scalar se = sqrt(r(Var)/r(N))
        display "       mean = " return(mean)
        display "SE of mean = " return(se)
end
```

return()関数はr()関数とほぼ同じですが、return()はこのプログラムがもたらす計算結果を参照します。つまり、プログラム実行時に保存されている値でなく、summarizeコマンドで実際にこれから計算される値を参照します。returnコマンドは、その計算結果をただちにr()に保存するものではありません。Stataはプログラムが終了するまでの間、計算結果をreturn()に保存します。そしてプログラム終了後にreturn()をr()にコピーします。プログラムの実行中に計算結果を参照する場合はreturn()関数を利用します。(return()はプログラムが終了した時点でr()と同じように動作します。そして、マクロ置換を実行する場合は‘return()’と記述します。)

18.10.2　計算結果をe()に保存する

e()に計算結果を保存する方法はr()の場合とほぼ同じです。つまり、programステートメントにeclassオプションを追加します。そして、returnに代わって、ここではereturn...を利用します。しかし、次に述べる相違点に留意してください。

1. r()の場合とは違って、e()に推定結果が保存されしだい、ereturn scalar, ereturn local, ereturn matrixコマンドが利用可能になります。推定コマンドの結果はメモリーを大量に消費しますので、Stataではなるべく不要な情報でメモリーを浪費することは避けねばなりません。したがって、計算結果の処理はプログラムの最後に、整理して記述するように心掛けてください。

2. 推定を実行し、保存する準備ができたら最初に、既存の推定結果を削除し、係数ベクトルをe(b)に、そして分散共分散行列をe(V)に、推定に用いた標本をe(sample)に保存します。一連の処理は推定値の推定手法によって異なります。

   2.1 Stataの尤度推定量mlを利用して計算した場合、自動的にこれらの値はメモリにセットされますので、ステップ3をご参照ください。

   2.2 既存の推定量から結果をコピーした場合、e(b),e(V),e(sample)は既に用意されていますので、これらを敢えてクリアする必要はありません。ステップ3をご参照ください。

   2.3 目的の処理を行うコードを自作する場合、ereturn postコマンドを推定後に利用します。詳細は[P] ereturnを参照してください。“ereturn post ‘b’ ‘V’, esam-ple(‘touse’)”のようなコードを書いたとすると、‘b’は係数ベクトル、‘V’は分散行列、そしてデータを利用した場合、‘touse’は1を含む変数となります。データを利用していない場合、それは0となります。ereturn postは既存の推定値をクリアし、係数ベクトル、分散行列、変数をe(b), e(V), e(sample)に保存します。

   2.4 (2.3)に加えて、既存の推定量を使って推定値を計算するものの、それ以外のe()の情報を残しおくようなケースがあるかもしれません。そのような場合は次のようにします。

```
tempvar touse
tempname b V
matrix ‘b’ = e(b)
matrix ‘V’ = e(V)
quietly generate byte ‘touse’ = e(sample)
ereturn post ‘b’ ‘V’, esample(‘touse’)
```

3. ereturn scalar, ereturn local, ereturn matrixを利用して目的の情報をe()に保存することが可能です。

4. コードを利用してe(cmdline)を保存します。

```
ereturn local cmdline ‘″‘0’″’
```

   必ずしもこのようなコーディングが必要な訳ではありませんが、ここでは例として示しておきます。

5. ereturn local cmd ″cmdname″という形でコマンドを書きます。Stataはこのコマンドが実行されるまで、推定は完了していないものと解釈します。したがって、推定を完了させるためには必ずこのコマンドを実行することを覚えておきましょう。仮に、早い段階でe(cmd)を設定し、ユーザが*Break*キーを押した場合、実際の推定の状況に関係なく、推定は完了したものと解釈します。

例えば、次のような構文の推定コマンドを作成したとします。

myest depvar var_1 var_2 [if exp] [in range], optset*1* optset*2*

ここで、optset*1*は推定結果の表示方法、optset*2*は推定結果をコントロールするものとします。推定コマンドの重要な動作として、引数の無いまま、推定コマンドを実行すると、既存の推定結果を表示する、というものがあります。例題を示します。

```
program myest, eclass
        local options "optset1"
        if replay() {
                if "`e(cmd)'"!="myest" {
                error 301                   /* last estimates not found        */
                syntax [, `options' ]
        }
        else {
                syntax varlist [if] [in] [, `options' optset2]
                marksample touse
Code contains either this,
                tempnames b V
                commands for performing estimation
                assume produces `b' and `V'
                ereturn post `b' `V', esample(`touse')
                ereturn local depvar "`depv'"
or this,
                ml model ... if `touse' ...
and regardless, concludes,
                perhaps other  ereturn commands appear here
                ereturn local cmdline `"" `0' ""'
                ereturn local cmd "myest"
        }
                                                                /* (re)display results ...      */
code typically reads
        code to output header above coefficient table
        ereturn display                         /* displays coefficient table     */
or
         ml display                             /* displays header and coef. table */
end
```

e()に保存される情報を次に示します。もちろんe()に保存する情報の自作もできます。

e(N)（スカラー）
　データの個数

e(df_m)（スカラー）
　モデルの自由度

e(df_r)（スカラー）
　推定値が非漸近値である場合の自由度の分母

e(r2_p)（スカラー）
　計算可能な場合の疑似決定係数。(線形回帰における決定係数が計算可能な場合、それはスカラーのe(r2)に入ります。)

e(F)（スカラー）
　定数項のみのモデルに対する検定を行います。ただし、結果が非漸近的な場合に限ります。

e(ll)（スカラー）
　対数尤度。

e(ll_0)（スカラー）
　定数項だけのモデルの対数尤度。

e(N_clust)（スカラー）
　クラスタの数。

e(chi2)（スカラー）
　定数項のみのモデルに対する検定を行います。ただし、結果が非漸近的な場合に限ります。

e(rank)(スカラー) e(V)のランク

e(cmd)(マクロ)
推定コマンド名。

e(cmdline)(マクロ)　　入力したコマンド文

e(depvar)（マクロ）
被説明変数の名前。

e(wtype)とe(wexp)（マクロ）
加重推定を行った場合、e(wtype)には加重の種類(fweight, pweightなど)が入ります。e(wexp)には加重の推定式が入ります。

e(title)（マクロ）
推定結果のタイトル。

e(clustvar)（マクロ）
クラスタ変数名。

e(vcetype)（マクロ）
推定結果の表で標準誤差の上に表示する文字列。 Robust, Bootstrap, Jackknife, ""などを表示します。

e(vce)（マクロ）
vce()で指定したvectype。

e(chi2type)（マクロ）
e(chi2)の計算方法によってLRまたはWaldなどの検定統計値を表示します。

e(properties)（マクロ）
一般的にはb Vを含む

e(predict)（マクロ）
predictで利用したコマンド。空白の場合、デフォルトの_predictを表示。

e(b)とe(V)（行列）
係数ベクトルと対応する分散行列。ereturn postコマンドを実行した時に保存します。

e(sample)（関数）
この関数はereturn postのesample()オプションで指定した場合に定義できます。データを利用した場合は1を含む変数、利用しない場合は0となります。ereturn postはその変数から値を取得し、e(sample)を作成します。

18.10.3 計算結果をs()に保存する

他のクラスがスカラー、マクロ、行列を保存できるのに比べ、s()はマクロだけしか保存できません。

s()の利用場面は限定的です。つまり、入力した情報を解析する前に、その情報の分解を行うサブルーチンとしてのみ利用します。

具体的にs()の利用例を示します。例えば、標準的ではない構文のコマンドをユーザが作成したものとし、その情報を分解する場面を想定してください。複雑で不規則な構文を解釈し、メインルーチンに情報を返却しなければなりません。しかも、解釈すべき構文はユーザの入力内容を含むものとします。例えば、ユーザがsummarizeコマンドの結果に対して、r(mean)/sqrt(r(Var))という計算を実行するものとします。

文字列の分解ステップでr()の情報を取得するサブルーチンをコールする場合、その前にr(mean)とr(Var)の内容を消してしまうことのないように、注意しなければなりません。つまり、文字列の分解が完了するまで、r()の内容には注意を払っておく必要があります。つまり、r-classコマンドの対象となる情報に、他のサブルーチンが干渉することのないようにします。このようなケースではr()よりも、むしろ、s()に値を保存するs-classを利用します。s-classは文字列の分解専用に用意されたマクロです。

s-classルーチンを作成する場合、プログラム行に対するsclassオプションを設定し、計算結果に対するsreturn localマクロを利用します。

s-classの計算結果はsrerturn()コマンドを実行すると、s()に保存されます。それを考慮して保存先を決定します。また、s()は自動的にクリアされることはありません。ですから、必要に応じて、意識的にsreturn clearコマンドを利用してください。繰り返しますが、s-classサブルーチンの利用頻度はそう高くありません。

collectコマンド群は結果をs()に保存する数少ない例の1つです。このコレクションはr()とe()から結果を収集しますので、s()に保存することでr()とe()の結果を空白のままにすることができます。

18.11 adoファイル

adoファイルについては[U] 17 adoファイルで解説しました。

試しに‘gobbledygook’と入力すると、Stataは最初に内蔵コマンドを検索します。該当するコマンドが存在する場合、そのコマンドを実行します。見つからない場合、gobbledygookを定義済みプログラムとして検索します。該当するプログラムが存在する場合、そのプログラムを実行します。それが存在しない場合、gobbledygook.adoファイル

を色々なディレクトリで検索します。adoファイルが見つからない場合、“unrecognized command”というエラーメッセージを表示します。

adoファイルが存在する場合、Stataはパス情報を付けて‘run gobbledygook.ado’というコマンドを実行します。問題なく実行できたら、Stataはgobbledygookをadoファイルと認識します。adoファイルとして実行できない場合、Stataは“unrecognized command”というエラーを返します。(adoファイルに誤りがあります。)プログラムが定義されている場合、Stataはその内容を実行します。

プログラムを自動的に実行させることも可能です。例えば、次に示す内容を含むhello.adoを作成したものとします。

```
begin hello.ado
program hello
        display "hi there"
end
end hello.ado
```

ファイルをカレントディレクトリ、または、personalディレクトリに保存したと考えます(参照:[U] **17.5.2 個人的に利用するadoファイル** ディレクトリ)。その場合、helloと入力するだけでプログラムを実行できます。

```
. hello
hi there
```

helloがこのように実行できると、Stataコマンドの一種と考えてしまうかもしれません。

personal adoファイルを配置すべきフォルダには2つあります。一つはカレントディレクトリです。操作中のプロジェクトに限って利用する場合は、このような形で利用します。つまり、カレントディレクトリにadoファイルを配置します。もう一つのフォルダは*personal adoファイルディレクトリ*です。Windows版の場合はC:¥ado¥personal、Unixの場合~/ado/personal、そしてMac版の場合~/ado/personalです。それでは確認してみます。

Stataのコマンドウィンドウに次のように入力します。

```
. personal
```

❑ テクニカルノート

Stataはc-classの変数c(adopath)で定義されているディレクトリにおいてadoファイルを検索します。それは次のようにフォルダになります。

BASE;SITE;.;PERSONAL;PLUS;OLDPLACE

大文字で書かれたこれらのコードネームはディレクトリを指します。具体的な情報はsysdirコマンドで確認できます。Windowsでsysdirコマンドを実行した時の例を示します。

```
. sysdir
   STATA:  C:¥Program Files¥Stata18¥
    BASE:  C:¥Program Files¥Stata18¥ado¥base¥
    SITE:  C:¥Program Files¥Stata18¥ado¥site¥
    PLUS:  C:¥ado¥plus¥
PERSONAL:  C:¥ado¥personal¥
OLDPLACE:  C:¥ado¥
```

Windowsを利用している場合でもStataのインストール先によっては上記の情報は異なります。よってコードネームを利用して説明を続けることにします。物理的な位置を示すディレクトリでなく、論理的な位置情報で説明するためにコードネームを利用します。

c-classの変数であるc(adopath)にadoファイルの検索パスは設定されていますので、Stataは最初にBASEとSITEを対象に検索します。もちろん、StataはBASEフォルダだけを検索する訳ではなく、その中に含まれているサブディレクトリの中も検索します。拡張子が.styleがある場合は、最初にこの中身を検索します。ここでは仮にStataがgobbledygook.adoを検索するものとします。Stataは最初にBASE (C:¥Program Files¥Stata18¥ado¥base)を検索し、ファイルが存在しない場合、BASEのサブディレクトリgを検索し、それからSITEを検索します。このような形で順次、検索を行います。Stataがgobbledygook.styleを検索する場合、最初にBASEを検索し、ファイルが存在しない場合はBASEのstyleサブディレクトリ(C:¥Program Files¥Stata18¥ado¥base¥style)を検索し、それからSITEを検索します。このような形で順次、検索を行います。

なぜこのような複雑なことをやっているのでしょうか。Stataには数多くのadoファイル、ヘルプファイル、そしてそれ以外の多くのファイルが用意されています。OSによっては同一のディレクトリにファイルが多すぎると、問題になるものがあります。そして全てのOSで間違いなく、処理速度が低下します。その問題を回避するため、adoディレクトリを31個に分けました(*a-z*、アンダースコア、jar、py、resource、style)。例えばStataコマンドciはadoファイルとして用意され、BASEのサブディレクトリに入っています。

adoファイルを作成する場合、個人的にadoディレクトリを決めます。ただし、ディレクトリ中に配置するファイル数は250までとします。

❑

❑ テクニカルノート

gobbledygook.adoファイルを見つけ、実行すると、Stataは自動的にロードするプログラムの全体サイズを計算します。そのサイズがadosize (参照[P] **sysdir**)を越えてしまった場合、Stataはトータルサイズを満たすまで、古いプログラムファイルをメモリから削除します。ここで言う古い、とはプログラムの最後の実行時刻の事を指します。メモリから削除することでメモリの領域を確保しますが、ユーザの操作に影響を与えることはありません。プログラムは必要に応じて自動的にロードします。

ただ、計算パフォーマンスに影響を与えます。プログラムのロードには若干の処理時間を要しますので、一度メモリから削除したプログラムを再度、ロードするには若干、待ち時間が生じるケースもあります。逆に言えば、adosizeを大きくすると、メモリ領域と引き換えに待ち時間の発生を防ぐことができます。set adosizeコマンドでadosizeのパラメータを変更します。

詳細は[P] sysdirを参照してください。adosizeのデフォルト値は1,000です。1,000という数値は自動的にプログラムをロードするメモリ領域が1,000Kであることを意味しています。様々な実験の結果、この値が適当であるとの判断によるものです。

❑

18.11.1 バージョン

最初のprogramコマンドの行の次に、adoファイルhello.adoを作成した時のStataのリリースを宣言することをお勧めします。

```
begin hello.ado
program hello
        version 18.0
        display "hi there"
end
end hello.ado
```

versionコマンドの詳細は[U] 16.1.1 バージョンを参照してください。一般的なdoファイルの場合、versionコマンドは一番上に書きます。しかし、adoファイルの場合はprogramコマンドの後ろに書きます。なぜなら、最初にadoファイルをロードし、それから実行するという順番になっているからです。Stataがadoファイルで定義したプログラムを実行する場合、コマンドの翻訳作業を最初に行います。

adoファイルの先頭にversionコマンドを書くことは、doファイルで同じことをするよりも、少し、深い意味を持っています。doファイルに比べ、adoファイルは継続的に利用する可能性があります。したがって、いろいろな機能をその中で利用する可能性があるので、機能の更新がプログラムの本来の目的に影響しないようにする必要があります。

18.11.2 adoファイルにおけるコメント文

adoファイルにおけるコマンドの付け方はdoファイルのそれと同じです。/* comment */としてコメントを囲むか、行頭にアスタリク(*)を入力します。また//でコメント文を作成することもできます。詳細は[U] 16.1.2 doファイルにおけるコメントと空白行を参照してください。

物理行よりも論理行の方が長い場合、doファイルと同じ方法で処理されます。すなわち、デリミタをセミコロン(;)に変更したり、物理行の行末で///を利用することによって、新しい行をコメントアウトできます。

18.11.3 adoファイルのデバック

メモリの管理はStataが行っているので、adoファイルのデバックは少しトリッキーだと感じるかもしれません。

例えば、プログラムhelloの出力を“Hi, Mary”に変更することを考えてみましょう。最初にdoファイルエディタでhello.adoを開き、内容を編集します。

```
begin hello.ado
program hello
        version 18.0
        display "hi, Mary"
end
end hello.ado
```

adoファイルを上書き保存し、実行します。

```
. hello
hi there
```

Stataは古いadoファイルを実行します。つまり、メモリが更新されていない事が分かります。Stataはadoファイルを高速に実行するため、メモリの容量が不足するまでは、最初の情報を確保しています。Stataでメモリから古いhelloプログラムを削除し、改めてディスクからロードする必要があります。

ディスク上のプログラムを変更しても、そのままではメモリの内容をすぐに更新しません。したがって、次に示すコマンドを実行してメモリの内容を破棄し、改めて読み込むようにします。

```
. discard
. hello
hi, Mary
```

discardコマンドはメモリ上に確保されているプログラムを一度破棄し、改めて読み込むコマンドです。もちろん、一度、Stataを終了し、改めてStataを起動すれば、わざわざdiscardコマンドを実行する必要はありません。Stataはセッションが終了したら、メモリをクリアします。

18.11.4 ローカルサブルーチン

adoファイルには複数のプログラムを記述できます。複数のプログラムを記述した場合、メインプログラムとサブルーチンという形で位置づけします。例えば、次のように入力します。

```
                                                        ------begin decoy.ado
program decoy
        ...
        duck ...
        ...
end
program  duck
        ...
end
                                                        --------end decoy.ado
```

duckはdecoyのローカルサブルーチンです。decoy.adoをロードしたからと行って、直接、duckと入力すると“unrecognized command”というエラーが発生します。ここではローカルという言葉に注意してください。例えば、次に示すようなadoファイルduck.adoを作成したものと考えます。

```
                                                        ------begin duck.ado
program duck
       ...
end
                                                        --------end duck.ado
```

decoyの中でduckをコールすると、それは同じファイルにあるローカルなduckを実行します。さらに例を使ってローカルサブルーチンについて説明します。次のようなdecoy.adoを作成したものと考えてください。

begin decoy.ado

```
program decoy
        ...
        manic ...
        ...
        duck ...
        ...
end
program duck
        ...
end
```

end decoy.ado

そしてmanic.adoを次のように作成します。

begin manic.ado

```
program manic
        ...
        duck ...
        ...
end
```

end manic.ado

このdecoyを実行した時の流れを次に示します。

1. decoy.adoのdecoyを実行します。decoyはmanicをコールします。
2. manic.adoにあるmanicを実行します。manicはduckをコールします。
3. duck.adoにあるduckを実行します。duckは所定の処理を実行し、情報を返します。
4. manicにコントールが戻り、情報を返します。
5. decoyにコントロールを返します。decoyがduckをコールします。
6. decoy.adoのduckが所定の処理を実行し、情報を返します。
7. decoyにコントロールが戻ります。

manicがduckをコールした時に、グローバルadoファイルduck.doを実行します。しかし、decoyがduckをコールした時は、ローカルプログラムであるduckを実行します。

Stataはこのような流れでコマンドを処理します。

18.11.5 adoコマンドの作成例

ここでは新しいStataコマンドの作成例を示します。具体的には、線形回帰で利用する影響メジャーを計測するコマンドを作成します。興味深い統計量ですが、特に興味の無い場合でも、ここではプログラミング技術の習得と割り切って取り組んでください。

Belsley, Kuh, and Welsch (1980, 24)は線形回帰モデルのデータの影響力を次のように計測することを提案しました。

$$\frac{\mathrm{Var}\left(\hat{y}_i^{(i)}\right)}{\mathrm{Var}\left(\hat{y}_i\right)}$$

分子はi番目のデータを除いた状態で回帰したフィット値の分散です。分母はすべてのデータを利用してフィットした値の分散です。次に示す計算式を用いてこの分散比を求めます。

$$\text{FVARATIO}_i \equiv \frac{n-k}{n-(k+1)}\left\{1-\frac{d_i^2}{1-h_{ii}}\right\}(1-h_{ii})^{-1}$$

ここでnはサンプルサイズ、kは推定する係数の数、$d_i^2 = e_i^2/e'e$であり、e_iはi番目の残差です。h_{ii}はハット行列の対角要素です。この式の要素はパラメータを推定した後で、すべてのコマンドを使って取得できます。つまり、回帰のパラメータを推定した後にFVARATIO_iは計算できます。例えば、次のように入力します。

```
. regress mpg weight displ
. predict hii if e(sample), hat
. predict ei if e(sample), resid
. quietly count if e(sample)
. scalar nreg = r(N)
. generate eTe = sum(ei*ei)
. generate di2 = (ei*ei)/eTe[_N]
. generate FVi = (nreg - 3) / (nreg - 4) * (1 - di2/(1-hii)) / (1-hii)
```

FViの式における3は推定するパラメータの個数kです。つまり、定数項、weightとdisplの係数になります。そして4はk + 1を示すものです。

❑ テクニカルノート

上記のプログラムの動作は分かりますか。predictコマンドはh_{ii}とe_iを作成します。しかし、e_iの二乗和であるe'eの計算にはトリッキーな方法を利用しています。Stataのsum()関数で和を求めています。eTeの最初のデータには、e_1^2が入ります。二番目には$e_1^2+e_2^2$、三番目には$e_1^2+e_2^2+e_3^2$という形で入ります。最後のデータには、$\sum_{i=1}^{N} e_i^2$が入ります。ここでe_i^2はe'eの事です。我々はpredictコマンドにおいてe(sample)を設定したので、推定サンプルに制約をかけることができた訳です。したがって、hiiとeiiは欠損値になります。しかし、その欠損値の存在がsum()に影響することはありません。なぜなら、sum()は欠損値をゼロとして扱うからです。ここではStataの明示的なデータ読み取り機能を利用して、最後のeTe[_N]だけを参照します。(詳細は[U] 13.3 関数と[U] 13.7 行番号の指定方法を参照してください。)このようにして個別に値を取得し、最後に計算公式を使って、目的の値を求めます。

この影響メジャーの統計量を今後も利用することを考慮するなら、プログラム化しておくと便利です。先頭に2 行追加して、先ほど実行したコマンドをそのまま書き写します。❑

```
begin fvaratio.ado, version 1

program fvaratio
        version 18.0
        predict hii if e(sample), hat
        predict ei if e(sample), resid
        quietly count if e(sample)
        scalar nreg = r(N)
        generate eTe = sum(ei*ei)
        generate di2 = (ei*ei)/eTe[_N]
        generate FVi = (nreg - 3) / (nreg - 4) * (1 - di2/(1-hii)) / (1-hii)
        drop hii ei eTe di2
end

end fvaratio.ado, version 1
```

プログラムに移す際には先頭にprogram fvaratioと付け、最後にendを追加するだけです。定義したコマンドがfvaratioなので、ファイル名は必ずfvaratio.adoにし、カレントディレクトリ、または、personal adoディレクトリに保存します。(詳細は[U] 17.5.2 個人的に利用するadoファイルディレクトリを参照してください。)

以上の設定によりfvaratioと入力すると、Stataはファイルを見つけ、ロードし、実行することができます。ここではコマンドをプログラムにコピーし、さらに「drop hii ...」というコマンドを追加します。これにより、一時的に作成した変数を削除できます。

実際にプログラムを実行してみましょう。

```
. regress mpg weight displ
. fvaratio
```

結果としてデータに新しい変数FViが追加されます。

しかし、このプログラムは一般形になっていません。このプログラムでは説明変数の数を2つに限定しており、定数項を考慮してkを3としています。Stataではregressのような統計関連コマンドは対象とする情報をe()の中に保存する仕組みになっています。実際、[R] **regress**にある*Stored results*の項目をご覧いただくと、e(df_m)にモデルの自由度が入ることが分かります。定数項のあるモデルでは自由度はk − 1になります。また、回帰に利用したデータセットのサンプルサイズはe(N)に保存します。ここの情報を見れば、データの個数が分かり、スカラーとして取り出すことができます。これらの事を考慮してプログラムを書き換えます。

```
-------------------------------------------------------------------------------
                                                      begin fvaratio.ado, version 2
program fvaratio
        version 18.0
        predict hii if e(sample), hat
        predict ei if e(sample), resid
        gen eTe = sum(ei*ei)
        gen di2 = (ei*ei)/eTe[_N]
        gen FVi = (e(N)-(e(df_m)+1)) / (e(N)-(e(df_m)+2)) *     /// changed this
                  (1 - di2/(1-hii)) / (1-hii)                    // version
        drop hii ei eTe di2
end
-------------------------------------------------------------------------------end fvaratio.ado, version 2 ----------
```

FViの計算式において3の代わりに(e(df_m)+1)とし、4の代わりに(e(df_m)+2)、そしてサンプルサイズはe(N)としました。

また、プログラムの残差部分についても残差平方和はe(rss)として、eTeの計算過程は削除します。

```
-------------------------------------------------------------------------------
                                                      begin fvaratio.ado, version 3
program fvaratio
        version 18.0
        predict hii if e(sample), hat predict ei if e(sample), resid
        gen di2 = (ei*ei)/e(rss)                 // changed this version
        gen FVi = (e(N)-(e(df_m)+1)) / (e(N)-(e(df_m)+2)) *     ///
                  (1 - di2/(1-hii)) / (1-hii) drop hii ei di2
end
-------------------------------------------------------------------------------
                                                        end fvaratio.ado, version 3
```

この結果、プログラムはコンパクトにない、処理速度も向上しました。これで一般的なプログラムとして、誰でもが利用できるようになりました。変数に関するプログラミング方法を改良することで、汎用的なものになりました。しかし、このプログラムには次に示すような注意点があります。

1. 欠損値がある場合、算出される答えは正しいものの、欠損値の数を知らせるメッセージを表示します。(プログラムが一時的な変数を作成する場合にこの種のメッセージを表示します。)
2. 作成される変数の名前は常にFViとなり、その都度、コントロールすることはできません。また、過去の試行により、FViが既に存在する場合、FViの重複を知らせるエラーメッセージを表示します。その際は、FViをdropコマンドで削除して、再度、fvaratioと入力します。
3. hii,ei,di2などの変数が既に存在する場合も、同じ意味のエラーを表示し、プログラムは実行できません。

これらの問題を解決する方法を次に示します。問題の1番はquietly{}でプログラムを囲むことで解決できます。

```
-------------------------------------------------------------------------------
                                                  begin fvaratio.ado, version 4
program fvaratio
        version 18.0
        quietly {                               // new this version
                predict hii if e(sample), hat predict ei if e(sample), resid
                gen di2 = (ei*ei)/e(rss)
                gen FVi = (e(N)-(e(df_m)+1)) / (e(N)-(e(df_m)+2)) *     ///
                          (1 - di2/(1-hii)) / (1-hii)
                drop hii ei di2
        }                                       // new this version
end
-------------------------------------------------------------------------------
                                                    end fvaratio.ado, version 4
```

quietly()の中にプログラムを入れると、メッセージの表示を抑制できます。quietlyは各コマンドの前に、それぞれ付けることもできます。

結果が常にFViに出力されるという問題の2番はsyntaxコマンドを利用すれば、解決できます。最後に、hii, ei, di2という形で、良く利用する名前を採用してしまう問題の3番について考えます。

他の名前に変更する、という考え方が一般的かと思います。例えば、MyHiiVaRという名前に変更したとしても、これが何を意味するのか、却って混乱を招くことも考えられます。先々の事を考えると、やはり、分かりやすい名前にする方が良さそうです。そのための解決方法をこれからご説明します。Stataにはローカルマクロに値を保存するtempvarコマンドというものがあります。詳細は[U] 18.7.1 テンポラリ 変数を参照してください。

```
-------------------------------------------------------------------------------
                                                  begin fvaratio.ado, version 5
program fvaratio
        version 18.0
        tempvar hii ei di2                                  // new this version quietly {
                predict `hii' if e(sample), hat   // changed, as are other lines predict `ei' if e(sampl
                e), resid
                gen `di2' = ( `ei' * `ei' )/e(rss)
                gen FVi = (e(N)-(e(df_m)+1)) / (e(N)-(e(df_m)+2)) *         ///
                          (1 - `di2'/(1-`hii')) / (1-`hii')
        }
end
-------------------------------------------------------------------------------
                                                    end fvaratio.ado, version 5
```

プログラムの先頭で、テンポラリ変数を宣言します。(これはquietlyの外でも、中でもどちらでもかまいません。また、かならず先頭である必要性もありません。宣言の位置に制約はありませんが、先頭で宣言したほうが分かりやすいプログラムになります。)テンポラリ変数を参照する場合、hiiと入力して直接参照するのではなく、引用符を利用して`hii'のように間接的に引用します。このようにすれば、プログラムの最後にdropコマンドでテンポラリ変数を削除する必要はありません。Stataはプログラムの実行が完了すると同時にそれらを自動的に削除します。

❑ テクニカルノート

次にテンポラリ変数を引用符で囲んだ理由を説明します。tempvarはテンポラリ変数名を格納するためのローカルマクロを作成します。hiiは新しいローカルマクロで、`hii'とすることで、その変数の中身を参照します。

❑

以上の点を考慮してプログラムをさらに改良します。ただし、新しく作成される変数名だけは、まだ指定できません。改良例を次に示します。

```
begin fvaratio.ado, version 6

program fvaratio
        version 18.0
        syntax newvarname                                   // new this version
        tempvar hii ei di2
        quietly {
                predict `hii' if e(sample), hat predict `ei' if e(sample), re
                sid gen `di2' = ( `ei' * `ei' )/e(rss)
                gen `typlist' `varlist' = ///               changed this version
                        (e(N)-(e(df_m)+1)) / (e(N)-(e(df_m)+2)) *       ///
                        (1 - `di2' /(1- `hii' )) / (1- `hii' )
        }
end

end fvaratio.ado, version 6
```

プログラムの行数が少し増えましたが、先の問題点を改良しました。このプログラムではsyntaxコマンドを新たに利用しています。詳細は[U] 18.4.4 Stataの標準的な構文を分解するを参照してください。

‘syntax newvarname’というコマンドは、ユーザが新しい変数名を決める場合に利用します。仮にプログラムで‘syntax [newvarname]’とした場合、新たな変数名の入力はオプションになります。つまり、カギカッコなしで‘syntax newvarlist’とした場合には少なくとも一つの変数名を入力しなければなりません。つまり、syntaxコマンドはユーザの入力をチェックする機能を持っています。このチェックにより入力情報に過不足が生じた場合、syntaxコマンドはエラーメッセージを表示し、プログラムを中止します。条件を満たしている場合は、そのままプログラムを実行し、入力した情報をローカルマクロとして保存します。newvarnameを利用している場合、ユーザの入力情報はローカルマクロvarlistに保存し、変数タイプ(float, double,...)はtyplistに入ります。変数タイプをユーザが設定していない場合は、デフォルトの設定を適用します。

結果としてかなりプログラムを改良できました。しかし、さらに2点ほど改良の余地があります。今、ご説明したように‘syntax newvarname’を利用することで、変数名だけでなく、保存のタイプも設定できます。計算途中で利用する中間変数である‘hii’と‘di2’について、ここでは、なるべく精度の高い形で保存することにします。保存の精度に関する仕様をなるべくいいものに設定するのではなく、できるだけ良い精度で保存する、というアプローチを取り事にします。どのような計算であっても、その前段の計算が倍精度であれば、その精度を活かして計算を行うことができます。編集したプログラムを次に示します。

```
begin fvaratio.ado, version 7

program fvaratio
        version 18.0
        syntax newvarname
        tempvar hii ei di2
        quietly {
                predict double `hii' if e(sample), hat       // changed, as are
                predict double `ei' if e(sample), resid      // other lines
                gen double `di2' = ( `ei' * `ei' )/e(rss)
                gen `typlist' `varlist' = ///
                        (e(N)-(e(df_m)+1)) / (e(N)-(e(df_m)+2)) *       ///
                        (1 - `di2' /(1- `hii' )) / (1- `hii' )
        }
end

end fvaratio.ado, version 7
```

さらにプログラムを改良してみましょう。fvaratioはregressコマンドの後に実行することを想定したプログラムです。例えば、ユーザがそれとは知らずにlogisticコマンドの後で利用してしまった時のことを感がたら、どのような対策が必要でしょうか?直近に実行したコマンドの情報はe(emd)に入ります。詳細は[U] 18.9 推定コマンドの計算結果にアクセスするおよび[U] 18.10.2 e()に保存するを参照してください。例えば、次のように改良します。

```
                                                    begin fvaratio.ado, version 8
program fvaratio
        version 18.0
        if "`e(cmd)'"!="regress" {                      // new this version error 301
        }
        syntax newvarname
        tempvar hii ei di2
        quietly {
                predict double `hii' if e(sample), hat
                predict double `ei' if e(sample), resid
                gen double `di2' = (`ei'*`ei')/e(rss)
                gen `typlist' `varlist' = ///
                        (e(N)-(e(df_m)+1)) / (e(N)-(e(df_m)+2)) *       ///
                        (1 - `di2'/(1-`hii')) / (1-`hii')
        }
end
                                                    end fvaratio.ado, version 8
```

errorコマンドは所定のエラーメッセージを表示し、Stataのプログラムを停止します。Error 301は “last estimates not found”となります。詳細は[P] errorをご覧ください。(また、コマンドとしてerror 301を入力してみるのも良いでしょう。)

これで完全なプログラムになりました。

❑ テクニカルノート

影響力を計測するFVARATIOのように、必ずしも、すべての面において改良を図る必要はありません。バージョン1はあまりにも利用方法が限定的です。しかし、バージョン2のようにすれば一般的な用途に役立ちます。バージョン3はさらに使いやすくなっていますが、バージョン4まで改良を重ねる必要性は高くはありません。

Stataのプログラム言語に精通することが重要ですが、最後のバージョン4をいきなり書けるようになることを目指す必要はありません。徐々に工夫するよう心掛けることが重要です。より頻繁に、複数のユーザが利用するような場合は、プログラムに工夫を重ねてください。一般に配布しているadoファイルはどちらかと言えば、限定的な用途に限られています。プログラムに詳しくないユーザの利用を想定すると、adoファイルにも継続的に改良を施す必要があります。ですから、自分だけで利用するプログラムを作成する場合は、あまり深入りせずに、基本的な機能を持つプログラムを作りましょう。

❑

18.11.6 システムヘルプの作成

adoファイルを作成する場合、その利用方法を解説するヘルプファイルも作成してください。このヘルプファイルは単純なテキストファイルで、コマンド名.sthlpという形式の名前を付けます。そしてadoファイルと同じディレクトリに配置します。このようにしておけば、helpコマンドと作成した新しいコマンド名を入力(またはメニューのヘルプを利用)して、ヘルプファイルを参照できます。ヘルプを用意しない場合は、“help for ... not found”というメッセージを表示します。

ヘルプファイルの例を参照する場合は“sysdir” と入力して公式なadoファイルのディレクトリを調べてください。BASEというディレクトリの中に複数のヘルプファイルが入っています。

```
. sysdir
   STATA:  C:¥Program Files¥Stata18¥
    BASE:  C:¥Program Files¥Stata18¥ado¥base¥
    SITE:  C:¥Program Files¥Stata18¥ado¥site¥
    PLUS:  C:¥ado¥plus¥
    PERSONAL: C:¥ado¥personal¥
    OLDPLACE:C:¥ado¥
```

C:¥Program Files¥Stata18¥ado¥baseの下にあるa, b, ...というフォルダの中に.sthlpファイルが入っているので、それらを参照してください。

ヘルプファイル自体はテキストファイルですが、情報はStata Markup and Control Language(SMCL)という言語で書かれています。その言語形式について詳しく調べる必要はありません。例えば、次のような行があるものとします。

```
Also see help for the finishup command
```

これらはヘルプ画面上にそのまま表示されます。次のような情報の場合は、

```
Also see {hi:help} for the {help finishup} command
```

これは次のようになります。

Also see **help** for the **finishup** command

finishupはハイパーテキストリンクとなり、これをクリックすると、finishupのヘルプページへジャンプします。

SMCL形式に関する詳細は[P] smclを参照してください。Stataヘルプファイルの SMCLコードの例を確認したい場合は、viewsource examplehelpfile.sthlpと入力してください。**ヘルプ** > **Stataのコマンド**と操作し、examplehelpfileと入力して**OK**ボタンをクリックするか、help examplehelpfileと入力すると、Stataはヘルプファイルに相当する画面を表示します。

Stataのドキュメントと同じような形でヘルプファイルを用意しておけば、他のユーザにとって、プログラムはより理解しやすいものになります。そこで、ガイドラインとなるようなものを次に示します。

1. 最初の行は次のようにします。

```
{smcl}
```

Stataのヘルプファイルが SMCL フォーマットであることを宣言します。

2. 2行目は次のようにします。

```
{* *! version #.#.# date}{...}
```

* はコメントを示し、{...}は空行を作りません。ヘルプファイルを編集したら、バージョン番号を更新し、コメント行に日付を入力します。

3. 次の行にはプルダウンメニューを表示するクリックアクセスツールバー(Dialog, Also See, Jump To)の内容を記述します。

```
{vieweralsosee "[R] help" "help help "}{...}
{viewerjumpto "Syntax" "examplehelpfile##syntax"}{...}
{viewerjumpto "Description" "examplehelpfile##description"}{...}
{viewerjumpto "Options" "examplehelpfile##options"}{...}
{viewerjumpto "Remarks" "examplehelpfile##remarks"}{...}
{viewerjumpto "Examples" "examplehelpfile##examples"}{...}
```

4. タイトルを入力します。

```
{title:Title}

{phang}
{bf:yourcmd} {hline 2} Your title
```

5. 2行の空白行を用意し、構文のタイトル、構文ダイアグラム、そしてオプションテーブルの情報を入力します。

```
{title:Syntax}

{p 8 17 2}
syntax line

{p 8 17 2}
second syntax line, if necessary

{synoptset 20 tabbed}{...}
{synopthdr}
{synoptline}
{syntab:tab}
{synopt:{option}}brief description of option{p_end}
{synoptline}
{p2colreset}{...}

{p 4 6 2}
clarifying text, if required
```

6. 2行の空白行を用意し、Descriptionタイトルと情報を入力します。

```
{title:Description}

{pstd}
description text
```

コマンドの機能を手短に説明します。あまり細かいユーザが混乱します。ユーザが知りたいことは、このコマンドは自分が探しているものか、否か、という事です。

7. コマンドでオプションが利用できる場合は、2行の空白行を用意し、オプションタイトルと解説を入力します。

```
{title:Options}

{phang}
{opt optionname} option description

{pmore}
continued option description, if necessary

{phang}
{opt optionname} second option description
```

オプションとして入力したのはオプションテーブルに表示されますので、アルファベット順に入力します。オプションの文節は、オプション名のとなりにスペースを空けて力します。オプションとして複数の文節を利用する場合、それらの分析は{pmore}で設定します。1つの空白行を用いてオプションを分けます。

8. オプションとして2行の空白行を用いて、Remarksタイトルと情報を入力できます。

```
{title:Remarks}

{pstd}
text
```

必要なだけ、情報は入力できます。Stataの公式なヘルプシステムにはあまり情報は入っていません。なぜなら、マニュアルの方で詳しく解説しているからです。*Stata Journal*やStataのウェブサイトで公開されている、新しいリリースにおいて追加された機能の情報は大変有用ですが、マニュアルほど参照される機会は多くないので、ヘルプファイルにも収録しています。

9. オプションとして2行の空白行を用いて、Examplesタイトルとその情報を入力できます。

```
{title:Examples}

{phang}
{cmd:. first example}

{phang}
{cmd:. second example}
```

理論的な考え方を補足する意味でご利用ください。具体的な例題を示すことで大いに理解を深めることができます。

10. オプションとして2行の空白行を用いて、Authorタイトルとその情報を入力できます。

```
{title:Author}

{pstd}
Name, affiliation, etc.
```

ここは慎重に利用してください。問い合わせ先として電話番号などは書かないほうがいいでしょう。なるべくなら、Eメールアドレスを書いておくことにしましょう。

11. オプションとして2行の空白行を用いて、Refernceタイトルと情報を入力できます。

```
{title:References}

{pstd}
Author. year.
Title.Location: Publisher.
```

{hi:highlighting}をあまり多用せず、慎重に利用しましょう。情報を入力する際、{cmd:...}と入力すると、このマニュアルにあるように、タイプライター体の書体を用いて情報を出力します。

❑ テクニカルノート

複数の関連するコマンドのヘルプを、一つの.sthlpファイルに書いておくと便利な場合があります。例えば、xyzのヘルプファイルにxyzとabcの両方の情報を入力します。そして、help abcとしたときにxyz.sthlpを表示するようにabc.sthlpに次のように記述します。

```
begin abc.sthlp
.h xyz
end abc.sthlp
```

.sthlpの中に‘.h refname’のように書いておくと、Stataはrefnameのヘルプファイルを表示します。

❑

❑ テクニカルノート

多くのプログラムを作成するような場合は、プログラム用のインデックスを作成しておくと良いでしょう。例えば、エントリーとしてcontents.sthlpを作成します。名前は自由に決めることができます。もう少し具体的に言えば、personalのadoファイルディレクトにuser.sthlpを作成します。詳細は[U] 17.5.2 個人的に利用するadoファイルディレクトリを参照してください。実際にuserの内容を確認する場合は、help userと入力します。

大規模なサイトでUnix版をご利用の場合はディレクトリSITE(一般的には/usr/local/ado)にsite.sthlpを作成しておくと良いでしょう。site.sthlp用に作成したコマンドを作成する場合はhelp siteとして参照します。

❑

18.11.7 ダイアログボックスをプログラミングする

Stataコマンドを使って新しいコマンドやヘルプファイルを作成する方法を紹介しましたが、インテフェースやダイアログを設計することもできます。Stataにはダイアログボックスのプログラミング言語が用意されていますので、独自のダイアログボックスを作成できます。実際、Stataプログラムの多くのダイアログボックスは、独自の言語を用いてデザインされたものです。

オリジナルのダイアログボックスが必要な場合は、[P] **Dialog programming**を参照してください。マニュアルにはダイアログボックスの作成とプログラミングに関する情報が詳細に解説されています。

18.12 外部のプログラムや他のプログラム言語と通信するためのツール

プログラミング上級者の中には、他のプログラムからStataとコニュニケーションしたり、逆にStataから外部のプログラムを呼び出したり、ライブラリを利用したいと考えるケースがあるかもしれません。実際、Stataは次のような機能をサポートしています。

外部プログラムに同期、または非同期で連動するシェルを作成する。詳細は[D] **shell**を参照してください。

C, C++, FORTRAN などで記述されているライブラリ内のコードをコールする。詳細は[P] **plugin** を参照してください。

Java で記述されているコードをコールする。詳細は[P] **java intro**を参照してください。

StataとPythonのコードを相互作用させる。詳細は[P] **PyStata intro**を参照してください。

H2Oと相互作用させる。詳細は[P] **H2O intro**を参照してください。

外部プログラムからOLEを利用してコマンドを送信したり、その処理結果を受診する。詳細は[P] **Automation**を参照してください。

18.13 プログラマへの追加情報

プログラムやadoファイルにおいては、すべてのStataコマンドを利用できます。そのコマンドの中の、プログラマ用に開発された専用のコマンドも含まれています。*詳しくは、Stata Index*の先頭にある目次の中の*Programming*の項目をご参照ください。

StataのMata言語に関する詳細は*Mata Reference Manual*をご参照ください。

18.14 参考文献

Baum, C. F. 2009. *An Introduction to Stata Programming*. College Station, TX: Stata Press.

Belsley, D. A., E. Kuh, and R. E. Welsch. 1980. *Regression Diagnostics: Identifying Influential Data and Sources of Collinearity*. New York: Wiley.

Drukker, D. M. 2015. Programming an estimation command in Stata: Global macros versus local macros. *The Stata Blog: Not Elsewhere Classified*. http://blog.stata.com/2015/11/03/programming-an-estimation-command-in-stata-global-macros-versus-local-macros/.

Gould, W. W. 2001. Statistical software certification. *Stata Journal* 1: 29-50.

Haghish, E. F. 2019. Seamless interactive language interfacing between R and Stata. *Stata Journal* 19; 61-82.

Herrin, J. 2009. Stata tip 77: (Re)using macros in multiple do-files. *Stata Journal* 9: 497-498.

19 直接入力コマンド

目次

19.1 概要

メモリ上に展開されたデータではなく、キーボードから直接入力したデータを元に瞬時に計算を行うコマンドのことを直接入力コマンドと呼びます。直接入力コマンドはStataを高度な電卓として利用することを意味しています。

個別標本ごとのデータが手元になく、目的の統計量の算出に必要な、集約した値だけが分かっているものとします。例えば、平均値の標準誤差や95%信頼区間を求めるために、必ずしもすべての標本データを揃える必要はありません。平均、標準偏差、そしてデータの個数が分かれば十分です。もちろん、すべてのデータを入力し、検定を実行してもかまいません。しかし、検定に必要な要約統計量だけが分かっているのであれば、より簡単に検定を実行できます。例えば、10回コインを投げ、2回表が出たとします。もちろん、この10回のコイン投げのデータを個別に入力(表を1、裏を0)してもかまいません。

しかし、直接入力コマンドはもっと簡単に解析を行います。直接入力コマンドの特徴をここで示しておきます。

1. データをメモリに展開しません。データは操作せずに、その場で計算を実行します。
2. 直接入力コマンドの構文は次のような形で決まっています。すなわち、コマンド名の後ろに、要約統計量を入力するだけで済みます。入力すべき情報は基本的に要約統計量です。そして入力する順番は統計的に見て、自然な順番となっています。
3. 直接入力コマンドの語尾はすべてiです。しかし、*i*で終わるものがすべて直接入力コマンドである、という訳ではありません。一般的に直接入力コマンドに対しては、それと対になる、メモリ上のデータを操作する非直接入力コマンドというものがあります。Stataではすべての統計コマンドに対して、元データから要約統計量を計算でき、その入力作業が甚だしく面倒なものでない限り、直接入力コマンドを用意しています。
4. マニュアルにおいて、直接入力コマンドの用法は、非直接入力コマンドと同じ箇所に記載されています。ですから、信頼区間を計算する場合、要約統計量を使う直接入力コマンドであれ、元データがすべて揃っている非直接入力コマンドであれ、マニュアルの[R] **ci**の項目を参照してください。メモリ上のデータを利用する場合はci,直接入力コマンドの場合はciiを利用することとなっているはずです

19.1.1 例

例題1

信頼区間の計算例を示します。発表論文で一般的に、変数の平均、標準偏差、データの個数などが記載されている場合があります。ご存知のようにこれらの情報があれば、信頼区間を簡単に計算できます。例えば、走行距離の平均値が24、標準偏差は6、そして標本数が97台であるとき、信頼区間は次のようになります。

```
. cii 97 24 6
    Variable |        Obs        Mean    Std. err.       [95% conf. interval]
-------------+---------------------------------------------------------------
             |         97          24    .6092077        22.79073    25.20927
```

平均の標準誤差は0.61で95%信頼区間は[22.8, 25.2]です。上記のようにコマンドcii(ciコマンドの直接入力形)の後ろに、標本数、平均、標準偏差の順番で入力します統計値の入力順は [R] ciで確認してください。

平均は22であるという仮説検定をttestコマンドの直接入力形で実行します。

```
. ttesti 97 24 6 22 One-sample t test

------------------------------------------------------------------------------
         |        Obs        Mean    Std. err.   Std. dev.   [95% conf. interval]
---------+--------------------------------------------------------------------
       x |         97          24    .6092077           6    22.79073    25.20927
------------------------------------------------------------------------------
    mean = mean(x)                                                t =   3.2830
    H0: mean = 22                                 Degrees of freedom =       96

    Ha: mean < 22               Ha: mean != 22                 Ha: mean > 22
 Pr(T < t) = 0.9993         Pr(|T| > |t|) = 0.0014          Pr(T > t) = 0.0007
```

最初の3つのコマンドはcciコマンドで必要な要約統計量であり、4番目の数値が仮説検定に必要となる帰無仮説の値です。 詳細は[R] ttestを参照してください。

例題2

例えば、10回コインを投げ、2回表が出たとします。ciコマンドを利用して99%信頼区間を求めます。

```
. cii 10 2, level(99)

                                                    -- Binomial exact --
    Variable |        Obs        Mean    Std. err.       [99% conf. interval]
-------------+---------------------------------------------------------------
             |         10          .2    .1264911        .0108505    .6482012
```

ここではciiコマンドと3つの数値を用いて、2回表が出る確率の信頼区間を計算しました。直接入力コマンドは一般的にコマンドの引数として、必要な情報を入力します。ciコマンドは最初の2つの数値を二項実験の試行回数と成功回数と認識します。詳細は[R] ciを参照してください。

正確確率検定の検定にはbitestコマンドを利用します。

```
. bitesti 10 2 .5
Binominal probability test
        N   Observed  k   Expected   k  Assumed p    Observed  p
------------------------------------------------------------------
                   210          5       0.50000       0.20000
  Pr(k >= 2)            =  0.989258  (one-sided test)
  Pr(k <= 2)            =  0.054688  (one-sided test)
  Pr(k <= 2 or k >= 8) = 0.109375    (two-sided test)
```

この出力の詳細については[R] bitestを参照してください。

◁

例題3

Stataのtabulateコマンドは変数間の関係を表にして示すコマンドです。直接入力形コマンドtabiは次のような形式で利用します。

```
. tabi 5 10 ¥ 2 14
```

row	col 1	2	Total
1	5	10	15
2	2	14	16
Total	7	24	31

```
          Fisher’s exact =                 0.220
  1-sided Fisher’s exact =                 0.170
```

tabiコマンドは列区切り(改行)として‘¥’を利用する点が、他のコマンドとは若干異なります。

◁

19.1.2 直接入力コマンドの一覧

コマンド	参考文献	説明
bitesti	[R] **bitest**	二項確率検定
c c i	[R] **Epitab**	疫学テーブル
c s i		
i r i		
mcci		
cii	[R] **ci**	平均、比率、度数の信頼区間
esizei	[R] **esize**	平均比較の効果量
prtesti	[R] **prtest**	比率の検定
sdtesti	[R] **sdtest**	分散比の検定
symmi	[R] **symmetry**	対称性と周辺均一性の検定
tabi	[R] **tabulate twoway**	度数の二元表
ttesti	[R] **ttest**	t検定(平均値の検定)
twoway pci twow	[G-2] **graph twoway pci**	スパイクとラインを利用したプロット
ay pcarrowi two	[G-2] **graph twoway pcarrowi**	矢印を利用したプロット 二元の散布図
way scatteri	[G-2] **graph twoway scatteri**	プロット
ztesti	[R] **ztest**	z検定(平均値の検定)

19.2 displayコマンド

displayは直接入力コマンドではありませんが、電卓としても利用できます。

```
. display 2+5
7
. display sqrt(2+sqrt(3^2-4*2*-2))/(2*3)
.44095855
```

詳細は、[R] **churdle**をご覧ください。

19.3 検出力、精度および標本サイズに関するコマンド

power, ciwidth, gsboundsおよびgsdesignは入力した数値から直接的に何かを算出する訳ではないので、直接入力コマンドではありません。しかし、コマンドラインに入力した数値情報で計算を行い、メモリ上のデータは操作しませんので、基本的な動作は直接入力コマンドと同じです。

powerとciwidthコマンドは検出力と精度および標本サイズの計算を実行します。詳細は*Stata Power, Precision and Sample-Size Reference Manual*を参照してください。

gsboundsとgsdesignはグループ逐次デザインにおける停止境界と標本サイズを計算します。詳細は*Stata Adaptive Design: Group Sequential Trials Reference Manual*を参照してください。

20 推定と推定後のコマンド

目次

20.1 定型的な推定コマンドの用法

regress, logit, suregなど、Stataの統計モデルをフィットさせるコマンドは基本的に同じ用法に則っています。最初に一推定式のコマンド構文を示します。

command varlist [*if*] [in] [weight] [, options]

複数式のコマンド構文は次のようになります。

command (*varlist*) (*varlist*) ...(*varlist*) [*if*] [in] [weight] [, options]

一推定式、連立推定式のどちらの構文にも柔軟性はあります。例えば、heckmanは2つの推定式によるものですが、一般的にはセンサードデータに対する調整機能を持った線形モデルと理解できるので、実際のコマンドとしては一推定式のタイプとなっています。重要なことは、ほとんどの推定コマンドはここにあげた構文のどちらか一方、または、これら以外の構文も取りえる事です。

一推定式の場合、varlistの先頭には被説明変数を書き、その後ろに若干の例外はありますが、独立変数を入力します。例えば、mixedコマンドではランダム要因を設定するために、特別な変数プリフィックスを利用します。

プリフィックスコマンドは推定コマンドの前に配置して、推定コマンドの機能を変更します。書き方は次のようになります。

prefix: command ...

対応するコマンドは、[u] **11.1.10 Prefix commands**で一覧できます。推定コマンドとプリフィックスコマンドの対応については、コマンドの構文セクションをご覧ください。

一推定式、連立推定式のどちらの場合でも次に示す機能をサポートしています。

1. if expやin rangeなど、サブサンプルを利用するためのStataの標準的な機能を利用できます。サブサンプルだけのデータセットを改めて作成する必要はありません。
2. 直近の推定結果を表示する場合は、推定コマンドだけを再入力します。例えば、regressコマンドでの推定後、その推定結果を再表示する場合は、regressとだけ入力します。推定の直後に、この操作を行う必要はありません。推定後に他のコマンドを実行し、さらにデータを変更したり、一部削除した後でregressコマンドを実行すれば、最新の推定結果を表示します。Stataは直近の推定コマンドを記憶しています。詳細は[P] **discard**を参照してください。
3. 推定時にlevel()オプションを利用するか、または、再表示の際にこのオプションを利用すると、係数の信頼区間を表示できます。デフォルトは95%信頼区間です。デフォルトの信頼区間は自由に設定できます。詳細は[R] **level**を参照してください。
4. 限界効果として微分値のdy/dxまたは弾力性df(y)/dxを表示する場合は推定後のコマンドmarginsを利用します。詳細は[R] **margins**を参照してください。
5. 推定後にmarginsコマンドを利用して周辺平均、調整済み予測値、予測限界効果を求めることができます。詳細は[U] **20.17 条件付き限界効果と平均限界効果** 、そして [R] **margins**をご参照ください。
6. 因子変数のレベル間で多重比較を行う場合は、推定後にpwcompare を利用します。セルの平均、周辺平均、限界定数項、勾配、周辺勾配などの統計値を比較することが可能です。詳細は[U] **20.18 多重比較の計算**, [R] **margins**および [R] **margins, pwcompare**を参照してください。
7. 因子変数のレベルや交差項を比較する場合は、推定後にコマンドconrastを利用します。このコマンドは主効果、交差項の効果、単純効果、そしてネストした効果を算出します。殆どのコマンドにおいて利用可能です。[U] **20.19 主効果と交差項との比較**, [R] **contrast**そして[R] **margins, contrast**を参照してください。
8. marginsコマンドの実行後に、その結果をグラフするためのコマンドmarginsplotを利用できます。また、pwcompareやcontrastコマンドによる分析結果はmarginsで再生できるで、それらの分析結果から間接的にグラフを作成することも可能です。詳細は[R] **marginsplot**をご覧ください。
9. モデルに関連付けされている統計量を取得する場合はestatを利用します。推定後に取得可能な統計値は利用するコマンドによって異なりますが、その詳細は例えば、[R] **regress**コマンドの続きにある[R] **regress postestimation**を参照してください。

   推定量の分散共分散行列を取得する場合は、推定後にestat vceコマンドを利用します。相関行列か、または、共分散行列(VCE)として取得できます。(推定した係数や共分散行列をベクトルや行列の形で取得し、それらをStataの行列機能で操作きます。詳細は[U] **14.5 Stataコマンド** で作成した行列にアクセスするを参照してください。)
10. 予測値、残差、影響統計量などの値を取得する場合は推定後にpredictコマンドを実行します。カスタム化した予測値の点推定値、標準誤差などを求める場合はpredictnlコマンドを利用します。詳細は[R] **predict**および[R] **predictnl**を参照してください。
11. ダイナミックおよびスタティックな予測を、任意の信頼区間とともに求める場合はforecastコマンドを推定後に利

用します。例えば、同時推定式を推定した時なども、このコマンドで複数の予測値を計算できます。同時方程式ですから、推定式1でy2からy1を予測し、同時に推定式2でy1からy2を予測します、forecastコマンドは比較用の予測シナリオを利用できるよう、様々な機能をサポートしています。詳細は[TS] **forecast**を参照してください。

12. 標準的な構文を利用して、コマンド(たとえばgenerate)行の中で、推定値や標準誤差を参照できます。詳細は[U] **13.5 係数と標準誤差** にアクセスするを参照してください。推定式の推定結果から値を取りだす場合はe(resultname)を利用します。例えば、サブサンプルにおけるデータ数はe(N)で取り出せます。推定後にereturn listと入力すれば、その内容を参照できます。取り出し可能な情報に関する詳細は推定コマンドの解説を参照してください。

    e()の中でも特にe(sample)は便利な機能を持っています。推定においてサブサンプルが利用された時は1を、それ以外の場合は0を返します。よって、サブサンプルにおける推定に制約をかえる場合はif e(sample)をコマンドの最後に追加します。例えば、summarize if e(sample)のようにします。

13. 推定したパラメータの検定(線形モデルへのワルド検定)はtestコマンド、非線形モデルでの検定にはtestnlコマンド、そして尤度比検定にはlrtestコマンドを用意しています。推定値の線形結合にに関する点推定と信頼区間の計算にはlincomコマンド、非線形結合に関しては同様のコマンドnlcomを用意しています。

14. testコマンドやlincomコマンドにおいて、係数に関する情報をどのように入力したのか、その情報を検定結果画面に表示する場合はcoeflegendオプションを利用します。詳細は[R] **test**および[R] **lincom**をご参照ください。

15. カテゴリカル変数に対してカテゴリカル変数ごとにモデルをフィットし、その結果を表示する場合はstatsbyプリフィックスコマンド(参照[D] **statsby**)を利用します。

16. collectコマンドを使用して推定結果を収集し、それらの結果からカスタマイズされたテーブルを作成できます。[TABLES] **Intro**をご参照ください。

17. 継続的に推定結果を参照するような場合は、推定後にestimatesコマンドを利用して、推定結果をファイルに保存します。保存した情報はestimatesコマンドで呼び出す事もできます。詳細は[R] **estimates**.を参照してください。

18. _estimatesコマンドで推定結果を一時保存し、その次に他の推定を実行します。その段形で、比較のためにメモリ上の結果を呼び出すことができます。プログラムマにとっては役に立つ機能です。詳細は[P] **_estimates**をご参照ください。

19. SUR推定により求めた連立式の係数推定値について、同時パラメータベクトルや、分散共分散行列を求める場合はsuestコマンドを利用します。係数の相等性を調べる場合にはこのコマンドをご利用ください。詳細は[R] **suest**を参照してください。

20. ハウスマンのモデルの選択コマンドを利用する場合はhausmanコマンドを用います。詳細は[R] **hausman**を参照してください。

21. 推定値の標準誤差についてHuber/White/robustの値を求める場合はvce(robust)オプションを利用します。また、データの独立性の仮定を緩める場合はvce(cluster clustvar)を利用します。

    ほとんど全ての推定コマンドでvce(vcetype)のオプションを用いて、vce(opg), vce(bootstrap), vce(jackknife)などの形で代替の分散推定量を推定できるようになっています。

    vce(bootstrap)とvce(jackknife)が利用可能な場合は、プリフィックスコマンドのbootstrapやjackknifeではなく、これらを利用することをお勧めします。

原則として、上記の要約項目と後述する詳細情報はベイズ分析コマンドには当てはまりません。ベイズ分析コマンドの詳細は*Stata Bayesian Analysis Reference Manual*をご覧ください。

20.2　標準的な構文

推定コマンドでは一般的にif expと in rangeを利用できます。また、推定コマンドではby varlist:も利用できます。

例題1

74台の自動車について、燃費(mpg),車重(weight)， そして国産車と外車の区分を示すダミー変数(foreign)がデータとして手元にあるものとします。mpgをweightとweightの二乗、およびforeignに回帰させるモデルを次のようにして推定します。

```
. use https://www.stata-press.com/data/r17/auto2
(1978 automobile data)
. regress mpg weight c.weight#c.weight if foreign
```

Source	SS	df	MS		
Model	428.256889	2	214.128444	Number of obs	= 22
Residual	489.606747	19	25.7687762	F(2, 19)	= 8.31
				Prob > F	= 0.0026
Total	917.863636	21	43.7077922	R-squared	= 0.4666
				Adj R-squared	= 0.4104
				Root MSE	= 5.0763

mpg	Coefficient	Std. err.	t	P>|t|	[95% Conf.	Interval]
weight	-.0132182	.0275711	-0.48	0.637	-.0709252	.0444888
c.weight#c.weight	5.50e-07	5.41e-06	0.10	0.920	-.0000108	.0000119
_cons	52.33775	34.1539	1.53	0.142	-19.14719	123.8227

weightの二乗項は因子変数の用法により、c.weight#c.weightとして用います。詳細は[U] 11.4.3 因子変数を参照してください。

国産車、外車ごとに個別に推定する場合は、by varlist:のプリフィックスを利用します。

```
. by foreign: regress mpg weight c.weight#c.weight

-> foreign = Domestic
      Source |       SS       df       MS      Number of obs   =        52
-------------+------------------------------   F(2, 49)        =     91.64
       Model |  905.395466     2  452.697733   Prob > F        =    0.0000
    Residual |  242.046842    49  4.93973146   R-squared       =    0.7891
-------------+------------------------------   Adj R-squared   =    0.7804
       Total |  1147.44231    51  22.4988688   Root MSE        =    2.2226

------------------------------------------------------------------------------
         mpg |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
      weight |  -.0131718   .0032307    -4.08   0.000    -.0196642   -.0066794
             |
c.weight#    |
c.weight |   1.11e-06   4.95e-07     2.25   0.029     1.19e-07    2.11e-06
             |
       _cons |   50.74551   5.162014     9.83   0.000     40.37205    61.11896
------------------------------------------------------------------------------

-> foreign = Foreign
      Source |       SS       df       MS      Number of obs   =        22
-------------+------------------------------   F(2, 19)        =      8.31
       Model |  428.256889     2  214.128444   Prob > F        =    0.0026
    Residual |  489.606747    19  25.7687762   R-squared       =    0.4666
-------------+------------------------------   Adj R-squared   =    0.4104
       Total |  917.863636    21  43.7077922   Root MSE        =    5.0763

------------------------------------------------------------------------------
         mpg |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
      weight |  -.0132182   .0275711    -0.48   0.637    -.0709252    .0444888
             |
c.weight#    |
c.weight |   5.50e-07   5.41e-06     0.10   0.920    -.0000108    .0000119
             |
       _cons |   52.33775    34.1539     1.53   0.142    -19.14719    123.8227
------------------------------------------------------------------------------
```

どの推定コマンドでもif expとin rangeは利用できますが、by varlist:が利用できるものは限定的です。by()を利用する場合、Stataのメモリには制約がかかり、直近の推定値のみ記憶します。したがって、これからご説明する機能を利用した場合、直近の推定値を利用することになります。つまり、外車に絞ってweightとweightの二乗にmpgを回帰した結果をサンプルとして推定した結果だけを記憶します。

ただし、statsbyプリフィックス(詳細は[D] **statsby**)を利用すれば、グループごとの推定結果から統計値を取得することも可能です。

```
. statsby, by(foreign): regress mpg weight c.weight#c.weight
(running regress on estimation sample)

      Command:  regress mpg weight c.weight#c.weight
           By:  foreign

Statsby groups
-+- 1 -+- 2 -+- 3 -+- 4 -+- 5
- -
```

statsbyは最初に国産車のデータで回帰を行い、次に外車に対し回帰を実行します。そしてデータセットに推定値を上書き保存します。もちろん、元データがディスクに保存されている場合に限りこのように動作します。強制的に既存のデータセットに上書きする場合はオプションのclearを利用します。

メモリの内容は次のようになります。

```
. list
```

	foreign	_b_weight	_stat_2	_b_cons
1.	Domestic	-.0131718	1.11e-06	50.74551
2.	Foreign	-.0132182	5.50e-07	52.33775

これは2つのモデルの推定値です。statsbyはc.weight#c.weightの係数値を、汎用的な形式として_stat_2というように命名します。もちろん、標準誤差やその他の統計値も取得できます。詳細は[D] statsbyを参照してください。

◁

20.3　前出の結果を再表示する

引数なしで推定コマンドを実行すると、直近に推定した結果を再表示します。

▷ **例題2**

次のように操作してmpgをweightとdisplacementに回帰させます。

```
. use https://www.stata-press.com/data/r17/auto2, clear
(1978 automobile data)
. regress mpg weight displacement
```

Source	SS	df	MS		
Model	1595.40969	2	797.704846	Prob > F	= 0.0000
Residual	848.049768	71	11.9443629	R-squared	= 0.6529
				Adj R-squared	= 0.6432
Total	2443.45946	73	33.4720474	Root MSE	= 3.4561

mpg	Coefficient	Std. Err.	t	P>|t|	[95% conf.	interval]
weight	-.0065671	.0011662	-5.63	0.000	-.0088925	-.0042417
displacement	.0052808	.0098696	0.54	0.594	-.0143986	.0249602
_cons	40.08452	2.02011	19.84	0.000	36.05654	44.11251

上記の推定後にデータの要約統計量を調べたり、データの表示、仮説検定の実行など色々な調査を行ったものとします。そこで、再度、推定結果を表示したい、というなりましたら、単純に引数なしで回帰のコマンドだけを入力します。

```
. regress
```

Source	SS	df	MS		
Model	1595.40969	2	797.704846	Prob > F	= 0.0000
Residual	848.049768	71	11.9443629	R-squared	= 0.6529
				Adj R-squared	= 0.6432
Total	2443.45946	73	33.4720474	Root MSE	= 3.4561

mpg	Coefficient	Std. Err.	t	P>|t|	[95% conf.	interval]
weight	-.0065671	.0011662	-5.63	0.000	-.0088925	-.0042417
displacement	.0052808	.0098696	0.54	0.594	-.0143986	.0249602
_cons	40.08452	2.02011	19.84	0.000	36.05654	44.11251

再表示の際はオプションも指定できます。例えば、後のコマンドでの使用のために、推定された各係数を参照する内部変数を表示するには、次のように入力します。

```
. regress, coeflegend
(省略)
```

詳細は[U] 20.12 係数の操作をご覧ください。

こうしたスタイルの用法は全ての回帰コマンドに共通したもので、stcoxやlogitでも同様の操作を行えます。

◁

20.4 推定結果の操作

Stataは直近に推定した結果をメモリ上に確保します。Stataを利用している最中に、推定結果の再表示、比較、モデル間での検定などを行うために推定結果を一時的にメモリに確保する場合はestimatesコマンドを利用します。推定結果はディスク上に保存することもできますが、その方法については次のセクションで説明します。推定結果は最大300個までメモリ上に確保できます。

▷ **例題3**

自動車のデータをそのまま利用して、推定結果をメモリ上に保存すう方法を次に示します。ここではquietlyコマンドを利用して推定結果をコンパクトに表示します。

```
. quietly regress mpg weight displ
. estimates store r_base
. quietly regress mpg weight displ foreign
. estimates store r_alt
. quietly qreg mpg weight displ
. estimates store q_base
. quietly qreg mpg weight displ foreign
. estimates store q_alt
```

ここでは4つのモデルを推定し、その結果を r_base, r_alt, q_base, q_altという名前で保存します。名前を忘れてしまった時は、次のように操作します。

```
. estimates dir
```

Name	Command	Dependent variable	Number of param.	Title
r_base	regress	mpg	3	Linear regression, base model
r_alt	regress	mpg	4	Linear regression, alternate model
q_base	qreg	mpg	3	Quantile regression, base model
q_alt	qreg	mpg	4	Quantile regression, alternate model

目的のモデルの推定結果は次のようにして呼び出します。

```
. estimates replay r_base
```

Model **r_base**

Source	SS	df	MS		
				Number of obs	= 74
				F(2, 71)	= 66.79
Model	1595.40969	2	797.704846	Prob > F	= 0.0000
Residual	848.049768	71	11.9443629	R-squared	= 0.6529
				Adj R-squared	= 0.6432
Total	2443.45946	73	33.4720474	Root MSE	= 3.4561

mpg	Coefficient	Std. err.	t	P>|t|	[95% conf.	interval]
weight	-.0065671	.0011662	-5.63	0.000	-.0088925	-.0042417
displacement	.0052808	.0098696	0.54	0.594	-.0143986	.0249602
_cons	40.08452	2.02011	19.84	0.000	36.05654	44.11251

全てのモデルの結果をまとめて表示する場合は次のようにします。

```
. estimates table _all
```

Variable	r_base	r_alt	q_base	q_alt
weight	-.00656711	-.00677449	-.00581172	-.00595056
displacement	.00528078	.00192865	.0042841	.00018552
foreign		-1.6006312		-2.1326005
_cons	40.084522	41.847949	37.559865	39.213348

estimatesコマンドは係数だけを表示しますが、他の統計値を表示させる方法もあります。

目的の推定結果をアクティブにする方法もあります。これにより、あたかも最初に推定をした時点に戻ることになります。

```
. estimates restore r_alt
(results r_alt are active now)
. regress
```

Source	SS	df	MS		
				Number of obs	= 74
				F(3, 70)	= 45.88
Model	1619.71935	3	539.906448	Prob > F	= 0.0000
Residual	823.740114	70	11.7677159	R-squared	= 0.6629
				Adj R-squared	= 0.6484
Total	2443.45946	73	33.4720474	Root MSE	= 3.4304

mpg	Coefficient	Std. err.	t	P>|t|	[95% conf.	interval]
weight	-.0067745	.0011665	-5.81	0.000	-.0091011	-.0044479
displacement	.0019286	.0100701	0.19	0.849	-.0181556	.0220129
foreign	-1.600631	1.113648	-1.44	0.155	-3.821732	.6204699
_cons	41.84795	2.350704	17.80	0.000	37.15962	46.53628

◁

estimatesコマンドには様々な用法があります。 詳細は、[R] **estimates**を参照してください。特にHausman検定など、モデル間での検定を行う場合に、estimatesコマンドを利用します。

20.5 推定結果の保存

estimatesコマンドで推定結果をファイルとして保存することもできます。

```
. estimates save alt file alt.ster saved
```

まさに推定を行った後でアクティブな推定結果を保存したり、メモリに読み込んだ結果を改めて保存できます。ファイルとして保存しておけば、後日、改めてその推定結果を参照できます。

```
. estimates use alt
. regress
```

Source	SS	df	MS		
				Number of obs	= 74
				F(3, 70)	= 45.88
Model	1619.71935	3	539.906448	Prob > F	= 0.0000
Residual	823.740114	70	11.7677159	R-squared	= 0.6629
				Adj R-squared	= 0.6484
Total	2443.45946	73	33.4720474	Root MSE	= 3.4304

mpg	Coefficient	Std. err.	t	P>|t|	[95% conf.	interval]
weight	-.0067745	.0011665	-5.81	0.000	-.0091011	-.0044479
displacement	.0019286	.0100701	0.19	0.849	-.0181556	.0220129
foreign	-1.600631	1.113648	-1.44	0.155	-3.821732	.6204699
_cons	41.84795	2.350704	17.80	0.000	37.15962	46.53628

前出のメモリ上での保存と、ファイルに保存するのとでは大きな違いがあります。すなわち、後者においてはe(sample)がありません。e(sample)に関してはまだ解説していませんが、これは推定時に利用したデータの情報を管理するものです。例えば、モデルの推定直後に次のように操作します。

```
. summarize mpg weight displ foreign if e(sample)
```

画面に推定に利用した標本に関する要約統計量を表示します。これはestimates restoreコマンドの直後の状態でも行うことができます。しかし、estimates useコマンドの後では、同じようにはなりません。その理由はメモリ上にデータが確保されていないためです。たとえデータを開いていたとしても、次のような状態になります。

```
. summarize mpg weight displ foreign if e(sample)
```

Variable	Obs	Mean	Std.dev.	Min	Max
mpg	0				
weight	0				
displacement	0				
foreign	0				

つまり、推定に利用したデータがメモリ上には無い、という前提でStataが動作している事になります。

詳しい情報は他のセクションで解説します。例えば、estimates describeとすると、推定結果をもたらした元のコマンドを表示します。詳細は[R] **estimates**をご覧ください。

20.6　変数選択のツール

Stataのlassoコマンドは共変量の選択と連続、二値、カウントアウトカムのモデルのフィットを行います。lasso機能の詳細は[LASSO] Lasso introをご覧ください。

stepwise、fp、mfpコマンドは実際に推定を行う主体ではなく、推定コマンドと共に用いて変数の選択の面でのアシストを行うコマンドです。

Stataのプリフィックスコマンドの一つであるstepwiseは、ステップワイズ法による変数の選択機能を提供します。すべての推定コマンドをサポートしている訳ではありませんが、それでも多くの推定コマンドと共に用いることができます。サポートするコマンドの一覧詳細は、[R] stepwiseを参照してください。

fpおよびmfpは、フラクショナル多項式関数の変数選択をアシストするコマンドです。詳細は、[R] fp、および[R] mfpをを参照してください。

20.7　サブサンプルの利用

推定においてサンプルの一部(サブサンプル)を利用する場合は、推定コマンドにおいてif expや in rangeなどの条件文を利用します。

サブサンプルを利用した上で推定を実行すると、Stataはそのサブサンプルを記憶し、その情報をe(sample)に保存します。記憶したサブサンプルを利用する場合はコマンドの最後にif e(sample)というコマンド分を追加します。推定のサブサンプルという言い方は、アクティブな推定結果を計算するために利用したデータセットのことを指します。したがって、サブサンプルは全データもしくは、その一部ということになります。

```
. regress mpg weight 5.rep78 if foreign
```

Source	SS	df	MS		
Model	423.317154	2	211.658577	Number of obs	= 21
Residual	372.96856	18	20.7204756	F(2, 18)	= 10.21
				Prob > F	= 0.0011
Total	796.285714	20	39.8142857	R-squared	= 0.5316
				Adj R-squared	= 0.4796
				Root MSE	= 4.552

mpg	Coefficient	Std. err.	t	P>|t|	[95% conf.	interval]
weight	-.0131402	.0029684	-4.43	0.000	-.0193765	-.0069038
rep78						
Excellent	5.052676	2.13492	2.37	0.029	.5673764	9.537977
_cons	52.86088	6.540147	8.08	0.000	39.12054	66.60122

```
. summarize mpg weight 5.rep78 if e(sample)
```

Variable	Obs	Mean	Std. dev.	Min	Max
mpg	21	25.28571	6.309856	17	41
weight	21	2263.333	364.7099	1760	3170
rep78					
Excellent	21	.4285714	.5070926	0	1

上記の回帰では21個のデータを利用しており、次のコマンドではsummarize...if e(sample)コマンドを利用することによって、21個のデータの平均を求めています。最初の回帰に比べデータの個数が少ないのは2つの理由によります。回帰ではif foreignコマンドを利用しているのでデータが限定され、さらに5.rep78で欠損値が発生します。理由はさておき、e(sample)はデータが利用されている行では真、利用されていない行では偽となります。

Stataコマンドではif文が利用できますので、if e(sample)はコマンドの最後尾に付けます。

e(sample)が存在する場合、Stataはsummarizeコマンドと同じ結果を、ショートハンドコマンドで算出します。

```
. estat summarize, label
Estimation sample regress                       Number of obs =         21
```

Variable	Mean	Std. dev.	Min	Max	Label	
mpg	25.28571	6.309856	17	41	Mileage (mpg)	
weight	2263.333	364.7099	1760	3170	Weight (lbs.)	
rep78					Repair Record	1978
Excellent	.4285714	.5070926	0	1		

詳細は[R] estat summarizeを参照してください。

20.8　信頼区間の設定

推定の信頼区間を設定する場合はlevel()オプションを利用します。また、推定結果を再表示する場合も同じです。

例題4

より狭い信頼区間90%を利用してモデルをフィットする場合は次のようにします。

```
. regress mpg weight displ, level(90)
```

Source	SS	df	MS		
Model	1595.40969	2	797.704846	Number of obs	= 74
Residual	848.049768	71	11.9443629	F(2, 71)	= 66.79
				Prob > F	= 0.0000
Total	2443.45946	73	33.4720474	R-squared	= 0.6529
				Adj R-squared	= 0.6432
				Root MSE	= 3.4561

mpg	Coefficient	Std. err.	t	P>|t|	[90% conf.	interval]
weight	-.0065671	.0011662	-5.63	0.000	-.0085108	-.0046234
displacement	.0052808	.0098696	0.54	0.594	-.0111679	.0217294
_cons	40.08452	2.02011	19.84	0.000	36.71781	43.45124

推定後に、引数を付けずに推定結果を再表示すると、デフォルトの95%信頼区間を表示します。逆に、最初に95%信頼区間で推定し、後でregress, level(90)とすると、90%信頼区間を用いて結果を表示します。

信頼区間をデフォルトで90%に設定する場合はset level 90とします。詳細は[R] **level**を参照してください。

信頼区間は小数点以下2ケタの単位で、実数値を用いて10.00から99.99までの範囲で設定できます。例えば、次のように入力します。

```
. regress mpg weight displ, level(92.5)
```

Source	SS	df	MS	Number of obs	=	74
				F(2, 71)	=	66.79
Model	1595.40969	2	797.704846	Prob > F	=	0.0000
Residual	848.049768	71	11.9443629	R-squared	=	0.6529
				Adj R-squared	=	0.6432
Total	2443.45946	73	33.4720474	Root MSE	=	3.4561

mpg	Coefficient	Std. err.	t	P>|t|	[92.5% conf.	interval]
weight	-.0065671	.0011662	-5.63	0.000	-.0086745	-.0044597
displacement	.0052808	.0098696	0.54	0.594	-.0125535	.023115
_cons	40.08452	2.02011	19.84	0.000	36.43419	43.73485

20.9 係数テーブルのフォーマット

係数テーブルのフォーマットを変更する場合はsformat(), pformat(), cformat()などのオプションを利用します。sformat()オプションは検定統計値のフォーマットを変更します。pformat()はp値、そしてcformat()は係数、標準誤差、信頼区間のフォーマットを変更します。小数点以下の桁数を固定幅フォーマットの%fを用いて変更します。

```
. regress mpg weight displ, cformat(%6.3f) sformat(%4.1f) pformat(%4.2f)
```

Source	SS	df	MS	Number of obs	=	74
				F(2, 71)	=	66.79
Model	1595.40969	2	797.704846	Prob > F	=	0.0000
Residual	848.049768	71	11.9443629	R-squared	=	0.6529
				Adj R-squared	=	0.6432
Total	2443.45946	73	33.4720474	Root MSE	=	3.4561

mpg	Coefficient	Std. err.	t	P>|t|	[95% conf.	interval]
weight	-0.007	0.001	-5.6	0.00	-0.009	-0.004
displacement	0.005	0.010	0.5	0.59	-0.014	0.025
_cons	40.085	2.020	19.8	0.00	36.057	44.113

例えば、(%6.3f)は全体で6文字、小数点以下を3桁に固定します。フォーマットに関する詳細は[U] 12.5.1 数値形式を参照してください。

推定結果を再表示する場合にもフォーマットオプションを利用できます。モデルを再度推定する必要はありません。

```
. regress, cformat(%7.4f)
```

Source	SS	df	MS	Number of obs	=	74
				F(2, 71)	=	66.79
Model	1595.40969	2	797.704846	Prob > F	=	0.0000
Residual	848.049768	71	11.9443629	R-squared	=	0.6529
				Adj R-squared	=	0.6432
Total	2443.45946	73	33.4720474	Root MSE	=	3.4561

mpg	Coefficient	Std. err.	t	P>|t|	[95% conf.	interval]
weight	-0.0066	0.0012	-5.63	0.000	-0.0089	-0.0042
displacement	0.0053	0.0099	0.54	0.594	-0.0144	0.0250
_cons	40.0845	2.0201	19.84	0.000	36.0565	44.1125

20.10 分散共分散行列の表示

アクティブメモリにある推定結果から分散共分散行列を表示する場合はestat vceコマンドを利用します。

例題5

例題2でコマンドregress mpg weight displacementを実行しました。分散共分散行列はモデルの推定後に表示させることができます。

```
. estat vce
Covariance matrix of coefficients of regress model
```

e(V)	weight	displace~t	_cons
weight	1.360e-06		
displacement	-.0000103	.00009741	
_cons	-.00207455	.01188356	4.0808455

オプションcorrを利用してestat vceコマンドを実行すると、相関行列を出力します。

```
. estat vce, corr
Correlation matrix of coefficients of regress model
```

e(V)	weight	displa~t	_cons
weight	1.0000		
displacement	-0.8949	1.0000	
_cons	-0.8806	0.5960	1.0000

詳細は[R] estat vceを参照してください。

Stataの行列コマンドで直接e(V)として分散共分散行列を操作することも可能です。

```
. matrix list e(V)
symmetric e(V)[3,3]
                  weight   displacement          _cons
      weight     1.360e-06
displacement     -.0000103      .00009741
       _cons    -.00207455      .01188356      4.0808455
. matrix Vinv = invsym(e(V))
. matrix list Vinv
symmetric Vinv[3,3]
                  weight   displacement          _cons
      weight      60175851
displacement     4081161.2      292709.46
       _cons     18706.732      1222.3339      6.1953911
```

詳細は[U] 14.5 Stataコマンドで作成した行列にアクセスするを参照してください。

20.11 予測値の算出

このセクションでは予測値に関する操作方法について説明しますが、predictコマンドで算出可能なその他の統計量についても同じ要領で算出できますので留意してください。詳細は[R] predictを参照してください。

回帰であれ、その他の分析であれ、Stataはモデルをフィットすると、その結果である係数や変数名を内部的に保存します。predictコマンドはこのようにして保存した情報を操作するものです。

例題6

mpgをweightとweightの二乗に線形回帰します。

```
. regress mpg weight c.weight#c.weight
```

Source	SS	df	MS		
Model	1642.52197	2	821.260986	Number of obs	= 74
Residual	800.937487	71	11.2808097	F(2, 71)	= 72.80
				Prob > F	= 0.0000
				R-squared	= 0.6722
				Adj R-squared	= 0.6630
Total	2443.45946	73	33.4720474	Root MSE	= 3.3587

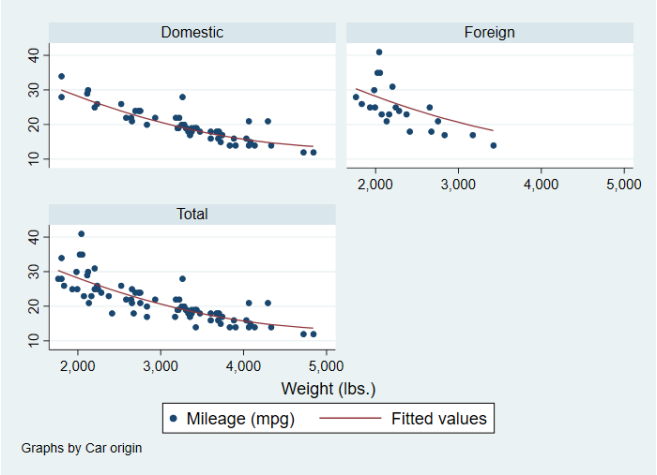
mpg	Coefficient	Std. err.	t	P>|t|	[95% conf.	interval]
weight	-.0141581	.0038835	-3.65	0.001	-.0219016	-.0064145
c.weight# c.weight	1.32e-06	6.26e-07	2.12	0.038	7.67e-08	2.57e-06
_cons	51.18308	5.767884	8.87	0.000	39.68225	62.68392

モデル推定の後にpredictコマンドを実行すると、次の計算を行うことになります。

$$-0.0141581\text{weight} + 1.32 \times 10^{-6}\text{weight}^2 + 51.18308$$

(係数は紙面の値ではなく、できるだけ精度の高い有効桁数で内部的に保存されます。)ここでは予測値を格納する変数名をfittedとし、predict fittedコマンドで計算を実行します。そして、フィット値と実現値をdomesticとforeignの種別ごとに散布図としてプロットします。

```
. predict fitted
(option xb assumed; fitted values)
. scatter mpg fitted weight, by(foreign, total) c(. l) m(o i) sort
```



predictコマンドは予測値以外にも様々な値を計算する機能を持っています。線形モデルを推定した場合、predictコマンドを利用して残差、標準化残差、影響統計量などを算出可能です。予測値以外の値を計算する場合はオプションを利用します。例えば、残差を新しい変数rに保存する場合は、次のようにします。

```
. predict r, resid
```

オプションはpredictコマンドの後ろに付けますが、利用できるオプションは推定コマンドによって異なります。predictコマンドのオプションは推定コマンドの項目に記載されています。regressコマンドの後で利用するpredictコマンドに関する詳細は[R] **regress**を参照してください。

20.11.1 predictコマンドの用法

predictコマンドは線形回帰モデル専用という訳ではありません。すべての推定コマンド実行した後で利用可能です。

例題7

合衆国以外の国で製造された車であるかどうかを、車重と燃費で判定するためのロジスティック回帰モデルをlogisticまたはlogitコマンドで推定します。詳細は[R] **logistic**と[R] **logit**を参照してください。ここではlogitコマンドを利用します。

```
. use https://www.stata-press.com/data/r17/auto2, clear
(1978 automobile data)
. logit foreign weight mpg
Iteration 0:   log likelihood =  -45.03321
Iteration 1:   log likelihood = -29.238536
Iteration 2:   log likelihood = -27.244139
Iteration 3:   log likelihood = -27.175277
Iteration 4:   log likelihood = -27.175156
Iteration 5:   log likelihood = -27.175156
Logistic regression                             Number of obs     =         74
                                                LR chi2(2)        =      35.72
                                                Prob > chi2       =     0.0000
Log likelihood = -27.175156                     Pseudo R2         =     0.3966
```

foreign	Coefficient	Std. err.	z	P>|z|	[95% conf.	interval]
weight	-.0039067	.0010116	-3.86	0.000	-.0058894	-.001924
mpg	-.1685869	.0919175	-1.83	0.067	-.3487418	.011568
_cons	13.70837	4.518709	3.03	0.002	4.851859	22.56487

logitモデルを推定後に、オプション無しでpredictコマンドを利用すると、外車であることを示す確率を算出します(詳細は[R] **logit**を参照してください。)車ごとに外国で製造された予測確率を求める場合は次のようにします。

```
. predict probhat
(option pr assumed; Pr(foreign))
. summarize probhat
```

Variable	Obs	Mean	Std. dev.	Min	Max
probhat	74	.2972973	.3052979	.000729	.8980594

```
. list make mpg weight foreign probhat in 1/5
```

	make	mpg	weight	foreign	probhat
1.	AMC Concord	22	2,930	Domestic	.1904363
2.	AMC Pacer	17	3,350	Domestic	.0957767
3.	AMC Spirit	22	2,640	Domestic	.4220815
4.	Buick Century	20	3,250	Domestic	.0862625
5.	Buick Electra	15	4,080	Domestic	.0084948

20.11.2　標本内での予測(内挿)

predictコマンドには、保存済みの数値情報を読み込んで計算を行うような機能は用意されていません。既存のメモリ上にある係数値とデータのみを利用します。先の例で次のように入力します。

```
. predict probhat
```

predictコマンドは計算可能な範囲のデータをすべて用いて演算を実行します。

例えば、推定時にif文を用いてデータに制約をかけたり、欠損値の影響で利用されないデータがあるような状況を想定してください。そのようなケースで、推定に利用したデータに対してだけ、予測値計算を実行する場合はコマンドの最後にif e(sample)コマンドを入力します。

```
. predict probhat if e(sample)
```

20.11.3　標本外での予測(外挿)

predictコマンドはメモリ上の係数値とデータを用いて計算を行います。もちろん、推定時に利用したデータだけに限定した話ではなく、それ以外のデータに対しても予測計算を実行できます。

データの一部であるサブセットに対してフィットしたモデルを使って外挿を行う場合は次のようにします。

```
. logit foreign weight mpg if rep78 > 3
. predict pall
```

predictコマンドの最後にif e(sample)を付けずにコマンドを実行すると、全標本に対して予測値を計算します。

アクティブな推定結果と説明変数のデータがあれば、どのようなデータに対しても予測計算を実行できます。

例題8

先ほど推定したロジットモデルの推定結果を利用して、異なるデータセットでの予測計算を行います。データセットにはmpgとweightが確かに入っています。モデルは既に前のデータで推定しているので、ここでは予測だけ計算します。

```
. use otherdat, clear (Different cars)
. predict  probhat                    Stata remembers the previous model
(option pr assumed; Pr(foreign))
. summarize probhat foreign
    Variable |       Obs        Mean    Std. dev.       Min        Max
-------------+--------------------------------------------------------
     probhat |        12    .2505068    .3187104   .0084948   .8920776
     foreign |        12    .1666667    .3892495          0          1
```

例題9

外挿によって予測値を求める方法は幾通りかあります。前の例ではあるデータセットで推定を行い、他のデータセットで予測値を求めました。次に、既存のデータセットに対してデータを追加する状況を想定します。データにお違いはdifbarsによって分かり、最初からあったデータには0、追加されたデータには1が入ります。

```
. logit foreign weight mpg if difcars==0
same output as above appears
. predict probhat
(option pr assumed; Pr(foreign))
. summarize probhat foreign if difcars==1
same output as directly above appears
```

モデル推定の後で、少しデータを追加したとします。最初のモデル推定では元データだけを利用しました。ここでは推定後にデータを直接入力して増やします。

```
. use https://www.stata-press.com/data/r17/auto2
(1978 automobile data)
. keep make mpg weight foreign
. logit foreign weight mpg
same output as above appears
. input
                make        mpg     weight    foreign
 75. "Merc.Zephyr" 20 2830 0                  we type in our new data
 76. "VW Dasher" 23 2160 1
 77. end
. predict  probhat                           obtain all the predictions
(option pr assumed; Pr(foreign))
. list in -2/l
```

	make	mpg	weight	foreign	probhat
75.	Merc.Zephyr	20	2,830	Domestic	.3275397
76.	VW Dasher	23	2,160	Foreign	.8009743

20.11.4 予測値の標準誤差、検定、信頼区間

predictコマンドを利用して標本内で、データとモデルのパラメータから時計値を計算することが出来ます。ユーザ独自の予測値を計算する場合は、generateコマンドを利用します。予測値に対する標準誤差、ワルド検定、そして信頼区間を計算する場合はpredictnlコマンドを利用します。例えば、予測値の標準誤差を求める場合は次のようにします。

```
. drop probhat
. predictnl probhat = predict(), se(phat_se)
. list in 1/5
```

	make	mpg	weight	foreign	probhat	phat_se
1.	AMC Concord	22	2,930	Domestic	.1904363	.0658387
2.	AMC Pacer	17	3,350	Domestic	.0957767	.0536297
3.	AMC Spirit	22	2,640	Domestic	.4220815	.0892845
4.	Buick Century	20	3,250	Domestic	.0862625	.0461928
5.	Buick Electra	15	4,080	Domestic	.0084948	.0093079

この出力と、先に計算した予測値(確率)はまったく同じものになります。唯一の違いは最後の列に、変数名、phat_seという標準誤差の値が用意されていることだけです。

predictnlコマンドによりprobhatが再度作成されますので、できれば、最初にdrop probhatとして列を削除しておけば、もっと分かりやすかったかもしれません。predictnlコマンドの行でpredict()を利用しているのはlogit推定後のデフォルト推定で点推定値と標準誤差を計算するためです。詳細は[R] **predictnl**を参照してください。

20.12 係数の操作

推定コマンドを実行した後で、係数と標準誤差を操作する場合は_b[name]と_se[name]という変数を利用します。詳細は[U] **13.5 係数と標準誤差にアクセスする**を参照してください。

例題10

線形回帰モデルの利用例を説明します。ある会社における男性と女性社員の賃金に関する考察を行います。各社員の賃金、教育水準、そして在職年数を変数として利用します。次のように入力します。

```
. regress lnearn ed tenure i.female female#(c.ed c.tenure)
```

(*省略*)

#の用法の詳細は[U] **11.4.3 因子変数**を参照してください。

次に全社員が男性だと仮定して、女性社員の賃金を計算してみます。

```
. generate asif = _b[_cons] + _b[ed]*ed + _b[tenure]*tenure
```

例題11

自動車の燃費を分析します。ここでは以前のデータよりも少し新しい情報を利用します。前回同様、燃費mpgをweight、weightの二乗、foreignとweightの交差項、ギア比(gear_ration)、そしてforeignとgear_ratioの交差項に回帰させます。二乗項と交差項は因子変数の記法を用いて作成します。詳細は[U] **1.4.3 因子変数**を参照してください。

```
. use https://www.stata-press.com/data/r17/auto2, clear
(1978 automobile data)
. regress mpg weight c.weight#c.weight i.foreign#c.weight gear_ratio
> i.foreign#c.gear_ratio
```

Source	SS	df	MS	Number of obs	=	74
				F(5, 68)	=	33.44
Model	1737.05293	5	347.410585	Prob > F	=	0.0000
Residual	706.406534	68	10.3883314	R-squared	=	0.7109
				Adj R-squared	=	0.6896
Total	2443.45946	73	33.4720474	Root MSE	=	3.2231

mpg	Coefficient	Std. error	t	P>|t|	[95% conf.	interval]
weight	-.0118517	.0045136	-2.63	0.011	-.0208584	-.002845
c.weight#c.weight	9.81e-07	7.04e-07	1.39	0.168	-4.25e-07	2.39e-06
foreign#c.weight Foreign	-.0032241	.0015577	-2.07	0.042	-.0063326	-.0001157
gear_ratio	1.159741	1.553418	0.75	0.458	-1.940057	4.259539
foreign#c.gear_ratio Foreign	1.597462	1.205313	1.33	0.189	-.8077036	4.002627
_cons	44.61644	8.387943	5.32	0.000	27.87856	61.35432

回帰分析や自動車の話にある程度の知識がないと、この分析結果を理解するのは少し難しいかもしれません。有意水準の話は一旦、脇におき、符号に注目しましょう。燃費は車重に反比例しますが、車重の二乗項には比例します。また、外車の場合、車重とともに燃費は国産車に比べ大きく下がります。ギア比と燃費は比例しますが、外車の場合は、より燃費効率があがります。

果たして、外車の方が燃費が良いと言えるのでしょうか?残念ながら、これを見ただけでは分かりません。外車の場合、車重が増えると国産に比べより燃費は落ち、ギア比は燃費が良くなる方に影響しています。一つの方法として、外車が国内で生産された場合に、どの程度の燃費が予測できるかということを考えてみます。計算に必要なデータはすべて手元に揃っています。国産車の燃費は次式で計算できます。

$$-0.012\text{weight} + 9.81 \times 10^{-7}\ \text{weight}^2 + 1.160\text{gear\_ratio} + 44.6$$

この式を用いて外車の仮想的な燃費を計算し、実際の燃費と比較します。係数をメモリ上で利用する場合は_b[]の形式で指定します。次のように入力してみましょう。

```
. generate asif=_b[weight]*weight + _b[c.weight#c.weight]*c.weight#c.weight +
>  _b[gear_ratio]*gear_ratio + _b[_cons]
```

_b[weight]はweightの係数推定値、_b[c.weight#c.weight]はc.weight#c.weightの係数値を指しています。

Hondaについて、予測値asifと実際の燃費を比較します。

```
. list make asif mpg if strpos(make,"Honda")
```

	make	asif	mpg
61.	Honda Accord	26.52597	25
62.	Honda Civic	30.62202	28

ここでは条件文を付けてHondaの車だけをピックアップしました。strpos()は最初の引数である変数名の中に、2番目の引数である文字列が検索するコマンドです。これにより変数makeの中に“Honda”という文字列を含むデータを探すことができます。詳細は[FN] **String functions**を参照してください。

リストを見ると、Hondaの仮想的な値が、実際の値を若干、上回っていることが分かります。念のために申し上げれば、この回帰モデルは燃費の変動を表現する完全なモデルではありませんし、Hondaを例にして用いたのも特別な意味はありません。実際、この予測値も平均予測値のRMSE内に入っています。

そこで、外車のデータ全体を利用して仮想的な燃費データを予測します。

```
. summarize mpg asif if foreign
```

Variable	Obs	Mean	Std. dev.	Min	Max
mpg	22	24.77273	6.611187	14	41
asif	22	26.67124	3.142912	19.70466	30.62202

この結果を見ますと、外車の平均的燃費はasifに比べ若干低いことが分かります。最後に、仮想的な燃費よりも現実の燃費の方が優れている外車をピックアップしてみましょう。

```
. list make asif mpg if foreign & mpg>asif, sep(0)
```

	make	asif	mpg
55.	BMW 320i	24.31697	25
57.	Datsun 210	28.96818	35
63.	Mazda GLC	29.32015	30
66.	Subaru	28.85993	35
68.	Toyota Corolla	27.01144	31
71.	VW Diesel	28.90355	41

6車種が該当することが分かりました。

◁

20.13 係数に関する仮説検定

20.13.1 線形な検定

推定後に、推定量の分散共分散行列を用いた、線形仮説検定(Wald検定)を行う場合はtestコマンドを利用します。

例題12

自動車データを利用して次のような回帰モデルを推定します。

```
. use https://www.stata-press.com/data/r17/auto2, clear
(1978 automobile data)
. generate weightsq=weight^2
. regress mpg weight weightsq foreign
```

Source	SS	df	MS		
Model	1689.15372	3	563.05124	Number of obs	= 74
Residual	754.30574	70	10.7757963	F(3, 70)	= 52.25
				Prob > F	= 0.0000
Total	2443.45946	73	33.4720474	R-squared	= 0.6913
				Adj R-squared	= 0.6781
				Root MSE	= 3.2827

mpg	Coefficient	Std.err.	t	P>|t|	[95% conf.	interval]
weight	-.0165729	.0039692	-4.18	0.000	-.0244892	-.0086567
weightsq	1.59e-06	6.25e-07	2.55	0.013	3.45e-07	2.84e-06
foreign	-2.2035	1.059246	-2.08	0.041	-4.3161	-.0909002
_cons	56.53884	6.197383	9.12	0.000	44.17855	68.89913

(注: testコマンドには複数の構文と機能が用意されていますので、用法については[R] test必ずを参照してください。)weightとweightsqの結合有意性検定を行う場合はtestコマンドを利用します。

```
. test weight weightsq
 ( 1)  weight = 0
 ( 2)  weightsq = 0
       F(  2,     70) =    60.83 Prob > F =           0.0000
```

係数値がゼロ、という仮説に限定されている訳ではありません。例えば、推定値が-2に等しいことを検定する場合は次のようにします。

```
. test foreign = -2 ( 1)  foreign = -2
       F(  1,     70) =     0.04
            Prob > F =     0.8482
```

testコマンドでは基本的な代数計算の機能もサポートしていますので、複雑な帰無仮説も設定できます。極端な例を次に示します。

```
. test 2*(weight+weightsq)=-3*(foreign-(weight-weightsq)) ( 1)  - weight + 5*w
 eightsq + 3*foreign = 0
       F(  1,     70) =     4.31
            Prob > F =     0.0416
```

testコマンドは条件式を計算し、できるだけ簡単な形式で結果を表示します。今回、複雑な条件を設定しましたが、有意水準1%では棄却できないことが分かります。
検定結果を他の検定結果と結合させる場合はオプションのaccumulateを利用します。

```
. test foreign+weight=0, accum
 ( 1)  - weight + 5*weightsq + 3*foreign = 0 ( 2)  weight + foreig
n = 0
       F(  2,     70) =     9.12
            Prob > F =     0.0003
```

testコマンドがいつでも利用できる訳ではありません。testコマンドは線形式による仮説検定にのみ利用可能です。非線形な条件式の場合、エラーとなります。

```
. test weight/foreign=0
not possible with test
r(131);
```

非線形式による仮説検定については[U] 20.12.4 非線形なワルド検定を参照してください。

20.13.2 検定の実行方法

testコマンドによるワルド検定は、モデルを推定した時に得られる推定量の分散共分散行列を利用するものです。最尤法推定の場合、情報行列を利用するのか、または、制約のある式を別途推定し、尤度比検定を行います。(詳細は[U] 20.12.3 尤度比検定を参照してください。)testコマンドは情報行列を利用しますので、係数テーブルに出力される漸近的なz統計値と同じ結果となります。

例題13

*Consumer Reports*で紹介されている乗用車の修理記録データを吟味してみましょう。

```
. tabulate rep78 foreign
```

Repair record 1978	Car origin Domestic	Foreign	Total
Poor	2	0	2
Fair	8	0	8
Average	27	3	30
Good	9	9	18
Excellent	2	9	11
Total	48	21	69

Poor, Fair, Average, Good, Excellentを1-5の数値に対応させます。このデータを被説明変数として、オーダードロジットモデルに最尤法でフィットします(nologオプションは繰り返し計算の状況を非表示にします)。

```
. ologit rep78 price foreign weight weightsq displ, nolog
Ordered logistic regression                     Number of obs      =         69
                                                LR chi2(5)         =      33.12
                                                Prob > chi2        =     0.0000
Log likelihood = -77.133082                     Pseudo R2          =     0.1767
```

rep78	Coefficient	Std. err.	z	P>|z|	[95% conf.	interval]
price	-.000034	.0001188	-0.29	0.775	-.0002669	.000199
foreign	2.685647	.9320404	2.88	0.004	.8588817	4.512413
weight	-.0037447	.0025609	-1.46	0.144	-.0087639	.0012745
weightsq	7.87e-07	4.50e-07	1.75	0.080	-9.43e-08	1.67e-06
displacement	-.0108919	.0076805	-1.42	0.156	-.0259455	.0041617
/cut1	-9.417196	4.298202			-17.84152	-.992874
/cut2	-7.581864	4.234091			-15.88053	.7168028
/cut3	-4.82209	4.14768			-12.95139	3.307214
/cut4	-2.793441	4.156221			-10.93948	5.352602

ここでforeign以外の推定値が同時にゼロであるか、仮説検定を行います。次のコマンドを入力して、線形な仮説を検定します。

```
. test weight weightsq displ price ( 1)  [rep78]weight = 0
 ( 2)  [rep78]weightsq = 0
 ( 3)  [rep78]displacement = 0
 ( 4)  [rep78]price = 0
                       chi2(  4) =      3.63
                     Prob > chi2 =      0.4590
```

最尤法推定の場合、情報行列を利用するのか、または、制約のある式を別途推定し、尤度比検定を行います。この結果を後述の尤度比検定の結果と見比べてください。詳細は[U] 20.13.3 尤度比検定 を参照してください。結果は若干、異なっています。

◁

20.13.3 尤度比検定

最尤法推定の場合、制約のあるモデルと無いモデルをそれぞれ推定し、それぞれestimates storeコマンドで保存します。そしてlrtestコマンドを実行します。詳細は[R] lrtestを参照してください。

▷ **例題14**

前出の[U] 20.13.2 検定の実行方法で、rep78のロジットモデルをフィットし、foreign以外の説明変数の有意性検定を行いました。

尤度比検定を行う場合、まず、フルモデルをフィットし、次にestimates store full_model_nameとします。full_model_nameは単に推定結果を示す名前です。

```
. ologit rep78 price foreign weight weightsq displ
  (省略)
. estimates store myfullmodel
```

このコマンドはmyfullmodelという名前で画面上の推定結果を保存します。

次に制約のあるモデルを推定します。制約のあるモデルを推定したら、名前は付けずに、‘lrtest myfullmodel .’と入力します。これにより直近に推定した制約モデルと制約の無いモデルを比較します。

```
. ologit rep78 foreign
Iteration 0:   log likelihood = -93.692061
Iteration 1:   log likelihood = -79.696089
Iteration 2:   log likelihood = -79.034005
Iteration 3:   log likelihood = -79.029244
Iteration 4:   log likelihood = -79.029243
Ordered logistic regression                     Number of obs      =         69
                                                LR chi2(1)         =      29.33
                                                Prob > chi2        =     0.0000
Log likelihood = -79.029243                     Pseudo R2          =     0.1565
```

rep78	Coefficient	Std. err.	z	P>|z|	[95% conf. interval]	
foreign	2.98155	.6203644	4.81	0.000	1.765658	4.197442
/cut1	-3.158382	.7224269			-4.574313	-1.742452
/cut2	-1.362642	.3557343			-2.059868	-.6654154
/cut3	1.232161	.3431227			.5596532	1.90467
/cut4	3.246209	.5556657			2.157124	4.335293

```
. lrtest myfullmodel .
Likelihood-ratio test
Assumption:. nested in myfullmodel
LR chi2(4) = 3.79
Prob > chi2 = 0.4348
```

同じことをtestコマンド(ワルド検定)で行うと、カイ二乗値は3.63となり、p値は0.4590となります。直近に推定した制約モデルは、もちろん、estimates storeコマンドで保存することもできますが、アクティブメモリにある推定結果は単純にドット(.)で指定できます。また、lrtestの際に入力する制約モデルと制約なしモデルの順序は決まっていません。lrtestコマンドは自由度を利用して両者を自動判別します。

◁

これと同じような形で動作するコマンドがさらに2つあります。ハウス検定を行うhausmanコマンド、SUR推定におけるsuestコマンドです。ハウスマン検定を行うhausmanコマンド、SUR推定におけるsuestコマンドです。これらのコマンドについてはそれぞれ、[R] **hausman**、[R] **suest**を参照してください。

20.13.4　非線形なワルド検定

アクティブな推定結果に対して非線形な仮説検定を実行する場合はtestnlコマンドを利用します。testnlはtestコマンドと同様、モデルを推定した時に得られる推定量の分散共分散行列を利用します。すべての推定コマンドの実行後に利用可能なコマンドです。詳細は[R] **testnl**を参照してください。

▷ **例題15**

次のモデルをフィットします。

```
.regress price mpg weight foreign
  (省略)
```

そして、

```
. testnl (38*_b[mpg]^2 = _b[foreign]) (_b[mpg]/_b[weight]=4)
  (1)  38*_b[mpg]^2 = _b[foreign]
  (2)  _b[mpg]/_b[weight] = 4
                          chi2(2) =       0.04
                Prob > chi2 =       0.9806
```

ここでは線形モデルの推定後に検定を実行しました。この非線形の検定は、どの推定コマンの後であれ、実行できます。

◁

p値の概念は、古典的な仮説検定の基本です。 その解釈と実際の使用に関する有用な議論については、Wasserstein and Lazar (2016) を参照してください。また、古典的な仮説検定の代替案として、[U] 27.33 **ベイズ統計分析**を参照してください。

20.14 係数の線形な組み合わせ

推定後に推定値を利用した線形式を計算し、点推定、標準誤差、tまたはz値、p値、そして信頼区間を算出します。オプションを利用して結果をオッズ比、罹患率比、相対リスク比で表示することも可能です。

▷ **例題16**

次の線形回帰モデルをフィットします。

```
. use https://www.stata-press.com/data/r17/regress, clear
. regress y x1 x2 x3
```

Source	SS	df	MS	Number of obs	=	148
				F(3, 144)	=	96.12
Model	3259.3561	3	1086.45203	Prob > F	=	0.0000
Residual	1627.56282	144	11.3025196	R-squared	=	0.6670
				Adj R-squared	=	0.6600
Total	4886.91892	147	33.2443464	Root MSE	=	3.3619

y	Coefficient	Std. Err.	t	P>|t|	[95% conf.	interval]
x1	1.457113	1.07461	1.36	0.177	-.666934	3.581161
x2	2.221682	.8610358	2.58	0.011	.5197797	3.923583
x3	-.006139	.0005543	-11.08	0.000	-.0072345	-.0050435
_cons	36.10135	4.382693	8.24	0.000	27.43863	44.76407

推定値x2とx1の差分について調査します。次のように入力します。

```
. lincom x2 - x1
 ( 1)  - x1 + x2 = 0
```

y	Coefficient	Std. Err.	t	P>|t|	[95% conf.	interval]
(1)	.7645682	.9950282	0.77	0.444	-1.20218	2.731316

◁

ある共変量グループと他のグループのオッズ比を求める場合にもlincomコマンドを利用します。

例題17

低体重児のロジスティックモデルを推定します。

```
. use https://www.stata-press.com/data/r17/lbw3
(Hosmer & Lemeshow data)
. logit low age lwd i.race smoke ptd ht ui Iteration 0:     log like
lihood =        -117.336
Iteration 1:    log likelihood =    -99.3982
Iteration 2:    log likelihood = -98.780418
Iteration 3:    log likelihood = -98.777998
Iteration 4:    log likelihood = -98.777998
Logistic regression                                     Number of obs      =        189
                                                        LR chi2(8)         =      37.12
                                                        Prob > chi2        =     0.0000
Log likelihood = -98.777998                             Pseudo R2          =     0.1582
```

low	Coefficient	Std.err.	z	P>|z|	[95% conf.	interval]
age	-.0464796	.0373888	-1.24	0.214	-.1197603	.0268011
lwd	.8420615	.4055338	2.08	0.038	.0472299	1.636893
race						
Black	1.073456	.5150753	2.08	0.037	.0639273	2.082985
Other	.815367	.4452979	1.83	0.067	-.0574008	1.688135
smoke	.8071996	.404446	2.00	0.046	.0145001	1.599899
ptd	1.281678	.4621157	2.77	0.006	.3759478	2.187408
h t	1.435227	.6482699	2.21	0.027	.1646414	2.705813
ui	.6576256	.4666192	1.41	0.159	-.2569313	1.572182
_cons	-1.216781	.9556797	-1.27	0.203	-3.089878	.656317

変数raceの1は白人、2は黒人、そして3はその他の人種を示しています。

黒人喫煙者の白人非喫煙者(リファレンスグループ)に対するオッズ比を求める場合は次のコマンドを利用します。

```
. lincom 2.race + smoke, or
 ( 1)  [low]2.race + [low]smoke = 0
```

low	Odds ratio	Std. err.	z	P>|z|	[95% conf.	interval]
(1)	6.557805	4.744692	2.60	0.009	1.588176	27.07811

lincomコマンドは$\exp(\beta_{2.\text{race}} + \beta_{\text{smoke}}) = 6.56$を計算します。

20.15　係数の非線形な組み合わせ

lincomは、例えば、2.race + smokeのような係数の線形関係や、指数関数における指数部の線形関係に利用するコマンドです。非線形な関係式の場合はnlcomを利用します。

例題18

先の例におけるサンプルデータをそのまま利用します。ここでは黒人と、白人および黒人以外の人種の係数の比(標準誤差、ワルド統計値、信頼区間でも可)について考察します。

```
. nlcom _b[2.race]/_b[3.race]
       _nl_1:  _b[2.race]/_b[3.race]
```

low	Coefficient	Std. err.	z	P>|z|	[95% conf.	interval]
_nl_1	1.316531	.7359262	1.79	0.074	-.1258574	2.75892

ワルド検定における帰無仮説は、非線形式はゼロに等しい(両側検定)というものです。残念ながら、これでは比が1に等しいことを検定することにはなりません。比が1に等しいことを調べる場合は比から1を引いた状態で、非線形式のためのnlcomコマンドを利用します。

```
. nlcom _b[2.race]/_b[3.race] - 1
       _nl_1:  _b[2.race]/_b[3.race] - 1
```

low	Coefficient	Std. err.	z	P>|z|	[95% conf.	interval]
_nl_1	.3165314	.7359262	0.43	0.667	-1.125857	1.75892

検定結果は、比から1を引いた値が0と有意に異なることは主張できない事を示しています。

このnlcomコマンドを利用する際には係数値を正しく指定する必要があります。例えば、黒人のための係数は_b[2.race]とし、lincomコマンドのように手短に、2.raceとすることはできません。試しに次のように入力してください。

```
. nlcom 2.race/3.race
```

エラーとなり、検定できません。

lincomやnlcomコマンドを利用する時に、係数の指定形を確認する場合はcoeflegend オプションを利用します。現在の推定値の指定形は次のようにして確認します。

```
. logit, coeflegend
Logistic regression                             Number of obs     =        189
                                                LR  chi2(8)      =      37.12
                                                Prob  >  chi2     =     0.0000
Log likelihood = -98.777998                     Pseudo R2         =     0.1582
```

low	Coefficient	Legend
age	-.0464796	_b[age]
lwd	.8420615	_b[lwd]
race		
Black	1.073456	_b[2.race]
Other	.815367	_b[3.race]
smoke	.8071996	_b[smoke]
ptd	1.281678	_b[ptd]
h t	1.435227	_b[ht]
ui	.6576256	_b[ui]
cons	-1.216781	_b[_cons]

20.16 周辺平均、調整済み予測値、予測マージンの計算

predictコマンドは推定結果(係数推定値と分散共分散推定量:VCE)を利用してデータの統計値を計算する場合に利用します。lincomとnlcomは係数間の線形、非線形な関係式の推定に利用し案す。そして、marginsコマンドはこれら2つのコマンドを組み合わせた分析と、応答のマージンを推定します。

marginsコマンドを利用すると、次に挙げる標本や母集団について、統計的な情報を得ることが可能です。

- 推定に利用した標本と他の標本
- 共変量を固定した標本
- トリートメントの各レベルの標本
- 複雑なサーベイ標本で代表する母集団
- 例えば、標本中の5番目の人
- 標本における共変量の値が平均値に相当する人
- 標本における共変量の値が中央値に相当する人
- 標本における共変量の値が25パーセンタイルに相当する人
- 標本における共変量の値がある関数の値に相当する人
- 標準化した母集団
- データ数の均一な(balanced)実験計画
- 上記の標本を任意に組み合わせたもいの
- 上記の標本の任意の比較

marginsコマンドはすべての共変量の値を固定した応答の条件付きの期待値や、算出した応答の平均値をこれらの標本について算出します。ここで言う応答とは、推定したパラメータによる関数値のことで、線形式における被説明変数、確率、ハザード、生存時間、オッズ比、リスク差(risk difference)などが相当します。たとえばlogitの実施後、確率の推定値や線形予測値などを求めたくなります。

marginsは共変量を目的の水準に固定したり、水準が変化した時に応答がどの程度変化するか(限界効果)、ということが分かる点にあります。もちろん、これらの統計値は標準誤差、検定統計量、信頼区間付きで提供されます。すなわち、設定した共変量の値や標本ごとに計算できます。これらの値のことを予測マージン、あるいはサーベイ統計量と呼ぶ事があります。

実際の観測値にかぎらず、すべて、または、一部の共変量の値を固定して、フィットしたモデルから計算した応答を利用して算出する統計値のことを、一般的にマージンと呼びます。

研究分野によって、マージンという用語の意味は異なりますが、基本的にフィットしたモデルから算出できます。ここでは一般的なマージンの利用例を紹介します。詳細は[R] **margins**を参照してください。

20.16.1 周辺平均を求める

共変量のデータ数がどのグループ(水準)においても等しい(balanced)時、線形推定量を使って期待周辺平均を推定する場合に'margins'という概念を用いてきました。'margins'に関する研究は長年、主にANOVAやMANOVAの分野で行われてきましたが、実験系以外の研究分野ではあまり利用されてきませんでした。詳細は[R] **margins**の*Obtaining margins as though*

the data were balanced や、［R］ anovaのexample 4を参照してください。

周辺平均の推定値は“最小二乗平均”と呼ばれることもあります。

心臓血圧の変化データに関する分散分析の例を元に周辺平均について考えてみましょう。患者に投与した薬剤の種類と、疾患の種類によってまとめたデータを次に示します。(Afifi and Azen 1979)

```
. use https://www.stata-press.com/data/r17/systolic
(Systolic blood pressure data)
. tabulate drug disease
```

Drug Used	Patient's Disease 1	2	3	Total
1	6	4	5	15
2	5	4	6	15
3	3	5	4	12
4	5	6	5	16
Total	19	19	20	58

```
. anova systolic drug##disease
```

Number of obs =	58	R-squared	=	0.4560	
Root MSE =	10.5096	Adj R-squared	=	0.3259	
Source	Partial SS	df	MS	F	Prob>F
Model	4259.3385	11	387.21259	3.51	0.0013
drug	2997.4719	3	999.15729	9.05	0.0001
disease	415.87305	2	207.93652	1.88	0.1637
drug#disease	707.26626	6	117.87771	1.07	0.3958
Residual	5080.8167	46	110.45254		
Total	9340.1552	57	163.86237		

薬剤をランダムに投与した結果、集計したデータの個数は均一にはなりませんでした。例えば、drug3の患者数は12,一方、drug4の患者数は16となっています。薬剤によって、投与した患者数が異なっています。しかし、各水準の患者数は同じであると仮定して、そのまま周辺平均の値を推定します。

```
. margins drug, asbalanced
Adjusted predictions                            Number of obs     =         58
Expression   :Linear prediction, predict()
At : drug    (asbalanced)
     disease (asbalanced)
```

	Margin	Delta-method std. err.	t	P>|t|	[95% conf.	interval]
drug						
1	25.99444	2.751008	9.45	0.000	20.45695	31.53194
2	26.55556	2.751008	9.65	0.000	21.01806	32.09305
3	9.744444	3.100558	3.14	0.003	3.503344	15.98554
4	13.54444	2.637123	5.14	0.000	8.236191	18.8527

このコマンドは全ての患者に対してdrug4を利用し、ある水準における患者の疾患の分布は均一であるといます。データ数がすべて疾患において等しいとしたときの、drug4による心臓血圧の変化の期待平均は13.54となりました。データはすべて均一であると設定してい

ますので、疾患の種類に関係なくdrug4による変化の期待値は13.54であると解釈します。

患者数が均一ではなく、観測したデータのように各水準で分布しているとして平均を推定する場合はasbalancedオプションを外します。

```
. margins drug
Predictive margins                              Number of obs     =         58
Expression   : Linear prediction, predict()
```

	Margin	Delta-method std. err.	t	P>|t|	[95% conf.	interval]
drug						
1	25.89799	2.750533	9.42	0.000	20.36145	31.43452
2	26.41092	2.742762	9.63	0.000	20.89003	31.93181
3	9.722989	3.099185	3.14	0.003	3.484652	15.96132
4	13.55575	2.640602	5.13	0.000	8.24049	18.871

この表では、すべての患者がdrug4を利用し、疾患によって患者数が一部、異なル場合、血圧変化分の期待値は13.56となっています。

このように標本数が均一であるという条件を外した場合、推定した平均値のことを周辺平均の期待値、とは呼びません。この表のmarginsの値は線形な応答関係という考え方で計算し、調整予測値(adjusted prediction)と呼びます。

結果として2つの計算手法のうち、どちらもdrug4による変化率の平均とはなっていません。なぜなら、marginsコマンドは全ての標本がdrug4を利用したものとして、平均値を計算するからです。各患者がそれぞれのグループのdrugを服用したという情報を設定しなければ、正しい比較になりません。よって、ここではdiseaseの値を標準化しなければなりません。

drug4の観測データの平均を求める場合は、標本をdrug4だけに限定するため、次に示すようにover()オプションを利用します。

```
. summarize systolic if drug==4
```

Variable	Obs	Mean	Std. dev.	Min	Max
systolic	16	13.5	9.323805	-5	27

```
. margins, over(drug)
Predictive margins                              Number of obs     =         58
Expression   : Linear prediction, predict()
Over         : drug
```

	Margin	Delta-method std. err.	t	P>|t|	[95% conf.	interval]
drug						
1	26.06667	2.713577	9.61	0.000	20.60452	31.52881
2	25.53333	2.713577	9.41	0.000	20.07119	30.99548
3	8.75	3.033872	2.88	0.006	2.643133	14.85687
4	13.5	2.62741	5.14	0.000	8.211298	18.7887

この表の最後の行の値はsummarizeコマンドによる観測データの平均値と一致しています。

普通、marginsという視点で分析する場合はmarginsを利用した最初の2つの方法を指します。最後の観測値の平均の表を見ると、drugの平均は8.75と13.5で、drugの効果を示す値となっています。もちろん、標本数の分布は異なるものとしています。

繰り返しますが、margins drug, asbalancedはdrug3とdrug4の各患者数は均一であるとして計算しています。一方、margins drugコマンドは、drug3とdrug4の各患者数は観測値によるとします。drugに関係なく標本数に共通の分布を仮定する場合、最初の2つの推定結果には、drugによる効果は含まれません。

20.16.2 調整済み予測値の計算

すべての共変量をある値に固定して計算したmarginsのことを調整済み予測値と呼ぶことにします。marginsコマンドでは、設定すべき共変量の値をフレキシブルに設定できます。例えば、高血圧のモデルについて考えます。説明変数はsex, age group, BMIとします。年齢の効果には性差があるものとして、age groupとsexの交差項を利用します。また、BMIの効果もage groupとsexで異なるものとしますので、full factorialモデルを推定します。

```
. use https://www.stata-press.com/data/r17/nhanes2
. logistic highbp sex##agegrp##c.bmi
```

Logistic regression

Number of obs	=	10,351
LR chi2(23)	=	2521.83
Prob > chi2	=	0.0000
Log likelihood = -5789.851	Pseudo R2	=	0.1788

highbp	Odds ratio	Std. err.	z	P>|z|	[95% conf.	interval]
sex						
Female	.4012124	.2695666	-1.36	0.174	.107515	1.497199
agegrp						
30-39	.8124869	.6162489	-0.27	0.784	.1837399	3.592768
40-49	1.346976	1.101181	0.36	0.716	.2713222	6.687051
50-59	5.415758	4.254136	2.15	0.032	1.161532	25.2515
60-69	16.31623	10.09529	4.51	0.000	4.852423	54.86321
70+	161.2491	130.7332	6.27	0.000	32.9142	789.9717
sex#agegrp						
Female#30-39	1.441256	1.44721	0.36	0.716	.2013834	10.31475
Female#40-49	6.29497	6.575021	1.76	0.078	.8126879	48.75998
Female#50-59	4.377185	4.43183	1.46	0.145	.6016818	31.84366
Female#60-69	1.790026	1.502447	0.69	0.488	.3454684	9.27492
Female#70+	.1958758	.2165763	-1.47	0.140	.0224297	1.710562
bmi	1.18539	.0221872	9.09	0.000	1.142692	1.229684
sex#c.bmi						
Female	.9809543	.0250973	-0.75	0.452	.9329775	1.031398
agegrp#c.bmi						
30-39	1.021812	.0299468	0.74	0.462	.9647712	1.082225
40-49	1.00982	.0315328	0.31	0.754	.9498702	1.073554
50-59	.979291	.0298836	-0.69	0.493	.9224373	1.039649
60-69	.9413883	.0228342	-2.49	0.013	.8976813	.9872234
70+	.8738056	.0278416	-4.23	0.000	.8209061	.930114
sex#agegrp#c.bmi						
Female#30-39	1.000676	.0377954	0.02	0.986	.9292736	1.077564
Female#40-49	.9702656	.0382854	-0.76	0.444	.8980559	1.048281
Female#50-59	.9852929	.0380345	-0.38	0.701	.9134969	1.062732
Female#60-69	1.028896	.0330473	0.89	0.375	.9661212	1.09575
Female#70+	1.12236	.0480541	2.70	0.007	1.032019	1.220609
_cons	.0052191	.0024787	-11.07	0.000	.0020575	.0132388

男女比とbmiの値を平均に固定した状態で、年齢グループagegrpごとの(高血圧である)確率を求めます。計算にはatmeansオプションを利用します。

```
. margins agegrp, atmeans
Adjusted predictions                            Number of obs     =     10,351 Model VCE    : OIM
Expression   : Pr(highbp), predict()
At           : 1.sex           =     .4748333 (mean)
               2.sex           =     .5251667 (mean)
               1.agegrp        =     .2241329 (mean)
               2.agegrp        =     .1566998 (mean)
               3.agegrp        =     .1228867 (mean)
               4.agegrp        =     .1247222 (mean)
               5.agegrp        =     .2763018 (mean)
               6.agegrp        =     .0952565 (mean)
               bmi             =      25.5376 (mean)
```

	Margin	Delta-method std. err.	z	P>|z|	[95% conf.	interval]
agegrp						
20-29	.1611491	.0091135	17.68	0.000	.1432869	.1790113
30-39	.2487466	.0121649	20.45	0.000	.2249038	.2725893
40-49	.3679695	.0144456	25.47	0.000	.3396567	.3962823
50-59	.5204507	.0146489	35.53	0.000	.4917394	.549162
60-69	.5714605	.0095866	59.61	0.000	.5526711	.5902499
70+	.6637982	.0154566	42.95	0.000	.6335038	.6940927

出力結果の表には各共変量の平均値を最初に表示します。この値を利用して確率を算出します。共変量の下には、agegrpの平均値を、実際の計算に利用する、しないに関係なく表示します。agegrpの値は1,2,3,4,5,6であることが分かります。これらの情報を出力するのは、周辺平均との違いを簡単に確認できるようにするためです。周辺平均を表示する場合は、例えば、margins sex agegrpとします。

モデルから算出した値によれば、30代で高血圧である確率は25%ほどですが、50台になると、52%に急激に増加します。70代以上だと67%となっています。一般的にモデルの推定値を示すよりも、このように周辺平均の形で表現すると、直感的に理解できます。

20.16.3 予測マージンの計算

先のセクションでは共変量を平均値に固定して、高血圧になる確率を計算しましたが、ここでは平均値に固定せず、観測値ごとに確率を計算し、その平均値を求めることにします。全てのデータの確率から算出した平均値は次のようになります。

```
. margins
Predictive margins                              Number of obs     =     10,351
Model VCE    : OIM
Expression   : Pr(highbp), predict()
```

	Margin	Delta-method std. err.	z	P>|z|	[95% conf.	interval]
_cons	.4227611	.0042939	98.46	0.000	.4143451	.4311771

例えば、agegrpが1の各患者の確率を求め、そこから平均を算出した結果が次のよの一番上の値になります。デフォルトではmarginsコマンドの次に入力した変数に対して、予測マージンを計算します。agegrpに対する予測マージンの計算結果を次に示します。

```
. margins agegrp
Predictive margins                              Number of obs     =     10,351 Model VCE       :
OIM
Expression   : Pr(highbp), predict()
```

	Margin	Delta-method std. err.	z	P>|z|	[95% conf.	interval]
agegrp						
20-29	.2030932	.0087166	23.30	0.000	.1860089	.2201774
30-39	.2829091	.010318	27.42	0.000	.2626862	.3031319
40-49	.3769536	.0128744	29.28	0.000	.3517202	.4021871
50-59	.5153439	.0136201	37.84	0.000	.4886491	.5420387
60-69	.5641065	.009136	61.75	0.000	.5462003	.5820127
70+	.6535679	.0151371	43.18	0.000	.6238997	.683236

予測マージンを直感的に理解する場合、 “すべての標本が、ある一つの年齢グループに属する場合、その予測値の平均はどうなるか?”と考えます。予測マージンのもう一つの考え方は、標本における他の共変量の分布を使って、各年齢グループにおける効果を標準化した値と考えるものです。この出力結果のマージンはagegrpの水準だけが変化しているので、比較可能です。年齢以外の共変量は、実際のデータを用いて計算するので、このマージンは手元の標本を表現する値であると考られます。

[U] **20.16.2 調整済みの予測値の計算**で推定した調整済み予測値と、上記の予測マージンの値は異なります。その理由はよ確率を計算するロジスティックモデルが非線形関数であることによります。詳細は[R] **margins**の*Example 3: Average response versus response at average*を参照してください。

ここまでの分析手順は比較的シンプルなものです。利用したデータは Second National Health and Nutrition Examination Survey (NHANES II)のデータセットであることに注意してください。NHANES IIのデータセットにはストラタ(層)やPSU(primary sampling unit)、そして加重などのようなサーベイデザインに関する情報が用意されています。Stataのサンプルデータは、殆どのケースにおいて既にサーベイデザインの情報を設定(svyset)した状態になっています。詳細は、[SVY] **svyset**を参照してください。したがって、モデルをフィットする際にはsvyプリフィックスを利用します。

```
. svy: logistic highbp sex##agegrp##c.bmi
(省略)
```

この状態でmargins agegrpコマンドを実行すると、点推定値は少しだけしか変化しませんが、標準誤差は大きく変わります。

ここまではマージンを考える変数を一つだけ考えてきました。次に、年齢グループに対する高血圧のマージンについて、性差を考慮して考えましょう。つまり、変数sexとagegrpの交差項に対してmarginsコマンドを実行します。サーベイデザインを考慮し、vce(unconditinal)オプションを利用します。

```
. margins sex#agegrp, vce(unconditional)
Predictive margins                              Number of obs     =     10,351 Expression   :
Pr(highbp), predict()
```

	Margin	Linearized std. err.	t	P>|t|	[95% conf.	interval]
sex#agegrp						
Male#20-29	.2931664	.0204899	14.31	0.000	.251377	.3349557
Male#30-39	.3664032	.0241677	15.16	0.000	.3171128	.4156936
Male#40-49	.3945619	.0240343	16.42	0.000	.3455435	.4435802
Male#50-59	.5376423	.0295377	18.20	0.000	.4773997	.5978849
Male#60-69	.5780997	.0224681	25.73	0.000	.5322756	.6239237
Male#70+	.6507023	.0209322	31.09	0.000	.6080109	.6933938
Female#20-29	.1069761	.0135978	7.87	0.000	.0792432	.1347091
Female#30-39	.1898006	.0143975	13.18	0.000	.1604367	.2191646
Female#40-49	.3250246	.0236775	13.73	0.000	.276734	.3733152
Female#50-59	.4855339	.03364	14.43	0.000	.4169247	.5541431
Female#60-69	.5441773	.0186243	29.22	0.000	.5061928	.5821618
Female#70+	.6195342	.0275568	22.48	0.000	.5633317	.6757367

ここではsexとagegrpの値を固定し、bmiには観測値をそのまま利用して確率を計算し、最後に平均を取りました。例えば、一番上のマージンの計算はsex=1,agegrp=1として各データの確率を計算し、最後に平均を計算したものです。したがって、このマージンの計算に関してはbmiの実現値の分布を反映しているものと考えます。

表の上6行分の結果は男性に関するものであり、下6行が女性に関する分析結果です。年齢グループごとに男女のマージンを比べてみましょう。例えば、一番若いグループでは、0.293/0.107 = 2.7となり、女性に比べ、男性は高血圧になる確率が3倍も高いことが分かります。2番目の30 代のグループでは約2 倍となっています。年齢があがるごとにその差は小さくなりますが、70 代でも男性の確率が高い傾向は見てとれます。

次に、bmiの値をコントロールすることで確率がどのように影響されているか、調べてみましょう。bmiを20(正常値の範囲)と30(太り気味と太りすぎの境界値)の2つの値に設定し、確率を再計算します。コマンドのオプションとして at(bmi=(20 30))を利用し、bmiをこの2つの値に設定します。

```
. margins sex#agegrp, at(bmi=(20 30)) vce(unconditional)
Adjusted predictions                            Number of obs     =     10,351
Expression   :Pr(highbp), predict()
1._at        : bmi             =          20
2._at        : bmi             =          30
```

	Margin	Linearized std. err.	t	P>|t|	[95% conf.	interval]
_at#sex# agegrp						
1#Male#20-29	.1302253	.0217288	6.41	0.000	.004011	.1825596
1#Male#30-39	.1714727	.0241469	7.10	0.000	.1222249	.2207205
1#Male#40-49	.1914061	.0366133	5.23	0.000	.1167329	.2660794
1#Male#50-59	.3380778	.0380474	8.89	0.000	.2604797	.4156759
1#Male#60-69	.4311378	.0371582	11.60	0.000	.3553532	.5069225
1#Male#70+	.6131166	.0521657	11.75	0.000	.506724	.7195092
1 # Female # 20-29	.0439911	.0061833	7.11	0.000	.0313802	.056602
1 # Female # 30-39	.075806	.0134771	5.62	0.000	.0483193	.1032926
1 # Female # 40-49	.1941274	.0231872	8.37	0.000	.1468367	.2414181
1 # Female # 50-59	.3493224	.0405082	8.62	0.000	.2667055	.4319394
1 # Female # 60-69	.3897998	.0226443	17.21	0.000	.3436165	.4359831
1#Female#70+	.4599175	.0338926	13.57	0.000	.3907931	.5290419
2#Male#20-29	.4506376	.0370654	12.16	0.000	.3750422	.526233
2#Male#30-39	.569466	.04663	12.21	0.000	.4743635	.6645686
2#Male#40-49	.6042078	.039777	15.19	0.000	.5230821	.6853334
2#Male#50-59	.7268547	.0339618	21.40	0.000	.657589	.7961203
2#Male#60-69	.7131811	.0271145	26.30	0.000	.6578807	.7684816
2#Male#70+	.6843337	.0357432	19.15	0.000	.611435	.7572323
2 # Female # 20-29	.1638185	.024609	6.66	0.000	.1136282	.2140088
2 # Female # 30-39	.3038899	.0271211	11.20	0.000	.2485761	.3592037
2 # Female # 40-49	.4523337	.0364949	12.39	0.000	.3779019	.5267655
2 # Female # 50-59	.6132219	.0376898	16.27	0.000	.536353	.6900908
2 # Female # 60-69	.68786	.0274712	25.04	0.000	.631832	.7438879
2#Female#70+	.7643662	.0343399	22.26	0.000	.6943296	.8344029

大変多くのマージンを出力していますが、6つの年齢グループ単位ごとに推定結果を見てください。最初の6つはBMIを20とした男性のマージンで、次の6つが同じBMIの女性のマージンです。3番目の6つのセットはBMIを30とした時、4番目のセットはBMIを30にした女性のマージンです。これらの推定結果を吟味すると、より詳細な事が分かります。高血圧である確率は、BMIが20である男性および女性のグループで低くなっています。また、良く見ると、BMIとの関係性が男女で異なることが分かります。女性に比べ、高血圧になる確率は約3倍(0.139/0.044)となっています。次の年齢グループでも大きな差0.171/0.076がありますが、その差は年齢が上がるごとに小さくなります。BMIが30のグループになると、よりリスクが高くなります。BMIの高いグループでは、性差に関係なく年齢の上昇にともなって、高血圧になるリスクが増え、50-60代の男性は女性に比べ約10%もリスクが高くなります。一般的にロジスティックモデルは確率0,1の両端で傾きが小さくなることを理由にこの差異を無視するのは早計です。我々はモデル推定にあたって、bmiの効果がsexとageの組み合わせで異なることを前提としましたので、むしろ望ましい結果と考えられます。

ここでの推定結果の出力画面の上部には “Predictive margins”ではなく、“Adjusted predictions”とあります。実際、我々は3つの共変量のそれぞれを固定して推定を行いました。つまり、marginsコマンドはデータの平均を取るのではなく、ユーザの指定した固定点でマージンを計算しているに過ぎません。実際、見出しの語句が変わっています。

上に述べたmarginsの計算結果や線形的な変数間の関係について、検定を用いて調べる場合は[R] **margins**マニュアルにある*Example 10: Testing margins-contrasts of margins*を参照してください。

マージンとmarginsコマンドに関する詳細な情報はマニュアルに用意されています。詳細は[R] **margins**の*Remarks and*

*examples*の項目を参照してください。

20.17 条件付き限界効果と平均限界効果

共変量が1単位変化した時の、応答の変化を計測するモノサシが限界効果です。簡単に言えば、限界効果とは微分値のことになります。ただし、これから紹介する例の中でも示すように、0と1の離散的な値を取るインジケータ変数に対しても限界効果は算出できます。研究者によっては連続変数の場合に限界効果、離散的な変数の場合にpartial effectと区別することもあります。しかし、ここでは両者とも限界効果と呼ぶことにします。一般的に限界効果とは、フィットした(非)線形モデルにおいて共変量の変化が、応答(被説明変数)をどの程度変化させるか、と考えます。多くの場合、被説明変数の種類としては確率、センサードデータ、生存時間、ハザードなどを利用します。

限界効果には、共変量のある値において計算する条件付き限界効果と、観測した共変量において計算した後、全体の平均をとる平均限界効果の2種類があります。

Stataにおける限界効果は、前者の微分値として計算するものになります。限界効果に関する詳細は[R] **margins**を参照してください。

20.17.1 条件付き限界効果の計算

全ての共変量の値を一定値に固定して計算した応答の微分値を限界効果と呼びます。一般的には共変量をその平均値に固定して限界効果を求めるケースが一般的で、この時、平均値回りの限界効果という用語を利用します。

女性就業者が組合員である確率を表現するプロビットモデルについて考えてみます。説明変数には高卒の場合に1をとる変数college、居住地区が南部である場合に1をとる変数south、調査時の職業における就業年数(tenure)、そしてsouthとtenureの交差項を利用します。分析の目的は、南部に住んでいるいることが、組合の加入にどの程度影響するか、という事を調べることです。分析に利用するのはU.S. National Longitudinal Survey of Labor Market Experience (参照[XT] **xt**)における1988以降のデータです。

```
. use https://www.stata-press.com/data/r17/nlsw88b, clear
(NLSW, 1988 extract)
. probit union i.collgrad i.south tenure south#c.tenure Iteration 0:      lo
g likelihood = -1042.6816
Iteration 1:    log likelihood = -997.71809
Iteration 2:    log likelihood = -997.60984
Iteration 3:    log likelihood = -997.60983
Probit regression                                 Number of obs      =      1,868
                                                  LR chi2(4)         =      90.14
                                                  Prob > chi2        =     0.0000
Log likelihood = -997.60983                       Pseudo R2          =     0.0432
```

union	Coefficient	Std. Err.	z	P>|z|	[95% conf.	interval]
collgrad						
not grad	.2783278	.0726167	3.83	0.000	.1360018	.4206539
1.south	-.2534964	.1050552	-2.41	0.016	-.4594008	-.0475921
tenure	.0362944	.0068205	5.32	0.000	.0229264	.0496624
south#						
c.tenure						
1	-.0239785	.0119533	-2.01	0.045	-.0474065	-.0005504
_cons	-.8497418	.0664524	-12.79	0.000	-.9799862	-.7194974

この結果から南部に住んでいる場合、明らかに加入の確率が下がります。ここではdydx()とmarginsコマンドのatmeansオプションを利用して、すべての共変量が平均値である人が南部に住んでいる場合、具体的にどの程度、組合加入の確率が下がってしまうのか、その値を求めることにします。

```
. margins, dydx(south) atmeans
Conditional marginal effects                      Number of obs      =      1,868 Model VCE        :
OIM
Expression   : Pr(union), predict() dy/dx wrt  : 1.sout
h
At           : 0.collgrad      =    .7521413 (mean)
               1.collgrad      =    .2478587 (mean)
               0.south         =    .5744111 (mean)
               1.south         =    .4255889 (mean)
               tenure          =    6.571065 (mean)
```

	dy/dx	Delta-method std. err.	z	|z|	[95% conf.	interval]
1.south	-.1236055	.019431	-6.36	0.000	-.1616896	-.0855215

Note: dy/dx for factor levels is the discrete change from the base level.

共変量の値がすべて平均値を取るような女性の場合、南部に住んでいるとすると組合への加入確率場12%下がります。限界効果の計算には、当然、変数i.southと交差項south#c.tenureの値を利用します。

marginsコマンドは各共変量の値(ここでは平均値)の下に、southが0の時の確率から、1の時の確率を引いた値を表示します。

見出し部分には共変量についての設定情報(ここでは平均値)を表示します。例えば、tenureの平均値は約6.6となっています。共変量をどの値に設定するかという事は分析者が決める事柄です。次にtenureを中央値に設定した時の限界効果を求めます。

```
. margins, dydx(south) atmeans at((medians) _continuous)
Conditional marginal effects                    Number of obs     =      1,868 Model VCE    :
OIM

Expression   : Pr(union), predict() dy/dx wrt : 1.south
At           : 0.collgrad      =    .7521413 (mean)
               1.collgrad      =    .2478587 (mean)
               0.south         =    .5744111 (mean)
               1.south         =    .4255889 (mean)
               tenure          =    4.666667 (median)
```

	dy/dx	Delta-method std. err.	z	P>|z|	[95% conf.	interval]
1.south	-.1061338	.0201722	-5.26	0.000	-.1456706	-.066597

Note: dy/dx for factor levels is the discrete change from the base level.

tenureの中央値は4.67で、限界効果は平均値の場合の6.6にくらべ2%ポイント小さくなっています。

条件付きの限界効果を調査する場合、共変量の値をいくつかの値に設定して、1回のコマンドで調査することもできます。実際、インジケータ変数collgrad(高卒なら1,大卒なら0)とtenureの組み合わせを複数用意してsouthの限界効果を調べてみましょう。

```
. margins collgrad, dydx(south) at(tenure=(0(5)25))
Conditional marginal effects                    Number of obs     =      1,868 Model
VCE          : OIM
Expression   : Pr(union), predict()
dy/dx wrt : 1.south
1._at        : tenure          =           0
2._at        : tenure          =           5
3._at        : tenure          =          10
4._at        : tenure          =          15
5._at        : tenure          =          20
6._at        : tenure          =          25
```

	dy/dx	Delta-method std. err.	z	P>|z|	[95% conf.	interval]
0.south	(base outcome)					
1.south _at#collgrad						
1#grad	-.0627725	.0254161	-2.47	0.014	-.112587	-.0129579
1#not grad	-.0791483	.0321151	-2.46	0.014	-.1420928	-.0162038
2#grad	-.1031957	.0189184	-5.45	0.000	-.140275	-.0661164
2#not grad	-.1256566	.0232385	-5.41	0.000	-.1712031	-.0801101
3#grad	-.1496772	.022226	-6.73	0.000	-.1932392	-.1061151
3#not grad	-.1760137	.0266874	-6.60	0.000	-.2283202	-.1237073
4#grad	-.2008801	.036154	-5.56	0.000	-.2717407	-.1300196
4#not grad	-.2282	.0419237	-5.44	0.000	-.310369	-.146031
5#grad	-.2549707	.0546355	-4.67	0.000	-.3620543	-.1478872
5#not grad	-.2799495	.0613127	-4.57	0.000	-.4001201	-.1597789
6#grad	-.3097656	.0747494	-4.14	0.000	-.4562717	-.1632594
6#not grad	-.3289702	.0816342	-4.03	0.000	-.4889703	-.1689701

Note: dy/dx for factor levels is the discrete change from the base level.

組合に加入するsouthの限界効果は、女性就業者の学歴と就業年数でかなり変化することが分かります。例えば、1番上の行にある未就業で高卒の女性の場合、南部に住んでいるとすると、6%ポイントしか加入確率が下がりません。逆に、1番下の行のデータになりますが、大卒で就業年数が25年の女性の場合、南部に住んでいるとすると33%近く、確率がさがります。この表を細かく考察することで、限界効果について詳しい情報を得ることができます。

20.17.2 平均限界効果の計算

平均限界効果を求める場合は、最初に各標本における限界効果を求め、次にその平均値を計算します。平均限界効果を計算する標本が母集団を代表している場合、計算した限界効果は母集団の特徴を示す値となります。

ここでも先の労働組合のデータを利用します。

```
. use https://www.stata-press.com/data/r17/nlsw88b
(NLSW, 1988 extract)
. probit union i.collgrad i.south tenure south#c.tenure
  (省略)
```

リグレッサの平均限界効果を求める場合は、次のコマンドを利用します。

```
. margins, dydx(*)
Average marginal effects                        Number of obs     =      1,868
Model VCE     : OIM
Expression    : Pr(union), predict()
dy/dx wrt : 1.collgrad 1.south tenure
```

	dy/dx	Delta-method std. err.	z	P>|z|	[95% conf.	interval]
collgrad						
not grad	.0878847	.0238065	3.69	0.000	.0412248	.1345447
1.south	-.126164	.0191504	-6.59	0.000	-.1636981	-.0886299
tenure	.0083571	.0016521	5.06	0.000	.005119	.0115951

Note: dy/dx for factor levels is the discrete change from the base level.

この計算結果を、前出の平均値周りの限界効果に比べると、かなり近い値であることが分かります。しかし、この結果は偶然ですので、注意してください。限界効果の値は共変量の分布に依存します。推定結果の表からは、母集団から標本を一つ選んだ場合、就業年数が1年増えるごとに組合加入確率が0.8パーセントポイントだけ増加することが分かります。大卒の場合は、平均で8.8パーセントポイント、加入確率は増えます。

この例では標本における共変量を既知とし、確定的な値として分析しました。しかし、標本は母集団からサンプリグしたものである、共変量の値は母集団における一標本の特徴を示すものであると、考えることもできます。母集団からの抽出という文脈を意識してモデルを推定する場合はvce(robust) やvce(cluster clastvar)オプションを利用し、限界効果の計算では vce(unconditional)オプションを使います。詳細は[R] **margins**における*Obtaining margins with survey data and representative samples*を参照してください。上記と同じような例による解説があります。

20.18　多重比較の計算

因子変数(factor variable)に対して多重比較を実行する場合はpwcompareコマンドを利用します。pwcompareコマンドは推定したセル平均、周辺平均、切片、マージナルインターセプト、傾き、マージナルスロープなどを比較する際に利用するコマンドです。また、pwcompareコマンドは多重比較において、有意性検定の結果を信頼区間付きで出力します。さらに、検定と信頼区間の計算においては色々なオプションを用意しています。

モデルにおいて因子変数を利用している場合、そのモデルの推定後にpwcompareコマンドを利用します。したがって、次のような推定コマンドのケースでは利用できません。

```
. regress yield fertilizer1-fertilizer5
```

次のように因子変数を利用している場合は、推定後に実行できます。

```
. regress yield i.fertilizer
```

異なる科学肥料を用いた小麦の収穫量に対し、線形回帰モデルをフィットし、その平均収穫量を比較します。多重比較に際してはTukeyのオプションを利用し、p値と信頼区間を求めます。

```
. use https://www.stata-press.com/data/r17/yield
(Artificial wheat yield dataset)
. regress yield i.fertilizer
```

Source	SS	df	MS		
Model	1078.84207	4	269.710517	Number of obs	= 200
Residual	9859.55334	195	50.561812	F(4, 195)	= 5.33
				Prob > F	= 0.0004
Total	10938.3954	199	54.9668111	R-squared	= 0.0986
				Adj R-squared	= 0.0801
				Root MSE	= 7.1107

yield	Coefficient	Std. err.	t	P>|t|	[95% conf.	interval]
fertilizer						
10-08-22	3.62272	1.589997	2.28	0.024	.4869212	6.758518
16-04-08	.4906299	1.589997	0.31	0.758	-2.645169	3.626428
18-24-06	4.922803	1.589997	3.10	0.002	1.787005	8.058602
29-03-04	-1.238328	1.589997	-0.78	0.437	-4.374127	1.89747
_cons	41.36243	1.124298	36.79	0.000	39.14509	43.57977

```
. pwcompare fertilizer, effects mcompare(tukey)
Pairwise comparisons of marginal linear predictions
Margins      : asbalanced
```

	Number of comparisons
fertilizer	10

	Contrast	Std. err.	Tukey t	Tukey P>|t|	Tukey [95% conf.	interval]
fertilizer						
10-08-22 vs 10-10-10	3.62272	1.589997	2.28	0.156	-.7552913	8.000731
16-04-08 vs 10-10-10	.4906299	1.589997	0.31	0.998	-3.887381	4.868641
(output omitted)						
29-03-04 vs 18-24-06	-6.161132	1.589997	-3.87	0.001	-10.53914	-1.78312

詳細は[R] **pwcompare**と[R] **margins, pwcompare**をご覧ください。

20.19 コントラスト、交互作用、そして主効果の計算

因子変数間の水準を比較し、検定を行う場合はcontrastコマンドを利用します。コントラストの同時検定(joint test)を実行し、主効果、交互作用、単純効果、そして入れ子型効果についてANOVA型の検定を実行します。すべての推定コマンド実行した後で、利用可能です。

contrastコマンドにはr.,ar.,p.などのコントラスト演算子が用意されています。この演算子は、例えば、r.varnameのように変数の前に付け、比較対照を設定します。この演算子はcontrastおよびmarginsコマンドとともに利用します。

各年齢グループにコレステロールレベルを回帰させた例を次に示します。

```
. regress chol i.agegrp
```

i.categoryの係数が、リファレンスカテゴリに対するコントラストの値となります。推定後、各年齢グループの平均を、その前のグループと比較する場合は、次のようなコマンドを利用して後方調整を実行します。

```
. contrast ar.agegrp
```

目的の分析例を次に示します。

```
. use https://www.stata-press.com/data/r17/cholesterol
(Artificial cholesterol data)
. regress chol i.agegrp
```

Source	SS	df	MS		
Model	14943.3997	4	3735.84993	Number of obs	= 75
Residual	7468.21971	70	106.688853	F(4, 70)	= 35.02
				Prob > F	= 0.0000
Total	22411.6194	74	302.859722	R-squared	= 0.6668
				Adj R-squared	= 0.6477
				Root MSE	= 10.329

chol	Coefficient	Std. err.	t	P>|t|	[95% conf.	interval]
agegrp						
20-29	8.203575	3.771628	2.18	0.033	.6812991	15.72585
30-39	21.54105	3.771628	5.71	0.000	14.01878	29.06333
40-59	30.15067	3.771628	7.99	0.000	22.6284	37.67295
60-79	38.76221	3.771628	10.28	0.000	31.23993	46.28448
_cons	180.5198	2.666944	67.69	0.000	175.2007	185.8388

```
. contrast ar.agegrp
Contrasts of marginal linear predictions
Margins      : asbalanced
```

	df	F	P>F
agegrp			
(20-29 vs 10-19)	1	4.73	0.0330
(30-39 vs 20-29)	1	12.51	0.0007
(40-59 vs 30-39)	1	5.21	0.0255
(60-79 vs 40-59)	1	5.21	0.0255
Joint	4	35.02	0.0000
Denominator	70		

	Contrast	Std. err.	[95% conf.	interval]
agegrp				
(20-29 vs 10-19)	8.203575	3.771628	.6812991	15.72585
(30-39 vs 20-29)	13.33748	3.771628	5.815204	20.85976
(40-59 vs 30-39)	8.60962	3.771628	1.087345	16.1319
(60-79 vs 40-59)	8.611533	3.771628	1.089257	16.13381

推定したセル平均に於いて、線形、二次、またはそれ以上の高次のトレンドの有無を直交多項式タイプのコントラストを利用して調べます。

```
. contrast p.agegrp, noeffects
Contrasts of marginal linear predictions
Margins      : asbalanced
```

	df	F	P>F
agegrp			
(linear)	1	139.11	0.0000
(quadratic)	1	0.15	0.6962
(cubic)	1	0.37	0.5448
(quartic)	1	0.43	0.5153
Joint	4	35.02	0.0000
Denominator	70		

一元配置モデルでもcontrastコマンドに制約がかかる訳ではありません。次のモデルをフィットします。

```
. regress chol agegrp##race
```

主効果と交互作用項の検定を行います。

```
. contrast agegrp##race
```

この結果はanovaコマンドのそれと同じものになります。contrastコマンドは因子変数を利用可能で、marginsコマンドが実行可能な、どのコマンドの後でも利用できます。次のように入力して試しみましょう。

```
. logistic highbp agegrp##race
. contrast agegrp##race
```

詳細は[R] **contrast** と[R] **margins, contrast**をご覧ください。

20.20 マージン、限界効果、コントラストのグラフ化

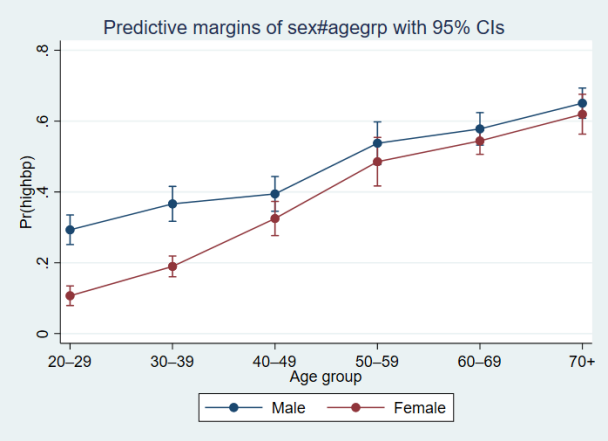
marginsコマンドの実行後の結果を用いてグラフを作成する場合はmarginsplotコマンドを利用します。また、pwcompareやcontrastコマンドで実行可能な分析はmarginsコマンドでも行えますので、それらの分析結果もmarginplotコマンドでグラフ化できます。

[U] **20.15.3 予測マージンの計算**にある例と同じものを実際に行ってみましょう。

```
. use https://www.stata-press.com/data/r17/nhanes2
. svy: logistic highbp sex##agegrp##c.bmi
. margins sex#agegrp, vce(unconditional)
```

グラフの作成は次のように行います。

```
. marginsplot, xdimension(agegrp)
Variables that uniquely identify margins: sex agegrp
```



詳細は[R] **marginsplot**をご覧ください。Mitchell (2012) は予測値とモデルを利用して、様々なグラフを作成しています。

20.21 ダイナミック予測とシミュレーション

予測モデルや、複数の内生変数を連立させた連立方程式から予測値を求める場合はforecastコマンドを利用します。regressやvarなどの推定コマンドで確率モデルを推定し、その結果を予測モデルに組み込みます。もちろん、変数間の関係を定義式で関係付けたり、モデルの外で任意に決まる外生変数を設定することもできます。forecastコマンドは連立モデルにおいて解を求め、予測値を計算するという目的にも利用できます。

forecastコマンドは時系列やパネルデータにおいて利用可能で、ダイナミック、または、スタティックな予測値を求めることができます。ダイナミックモデルとは、説明変数に被説明変数のラグ項が存在する場合、一期前の計算値をラグ項に利用して、将来値を予測するモデルです。一方、スタティック予測は一期前の実現値を説明変数のラグ項として利用します。したがって、既存のデータ範囲を越えて、将来予測を行うことはできません。しかし、モデル開発の段階では、スタティック予測もテストの意味合いで頻繁に利用します。

将来予測においては、外生変数に複数のパスを用意するという事を良く行います。パスとはシナリオのことで、外生変数の将来の変動をいくつかのパターンに分けて考えます。シナリオごとに将来予測値を計算することで、政策の違いがどのように将来に影響するか、ある一定の知見を得ることができます。

forecastコマンドは予測値に対する信頼区間も算出します。確率モデルにおいて、推定値の分散を考慮した予測値の計算を行います。確率モデルでは加法確率誤差を考慮する2つの方法があります。forecastコマンドにおいては誤差項が正規分布すると仮定して、乱数ジェネレータから乱数を取得する方法が一つあり、もう一つはスタティック予測の残差からランダムサンプリングする方法があります。

詳細は[TS] **forecast**を参照してください。

20.22 堅牢な分散推定値

Stataの多数のコマンドがロバストおよびクラスタロバストな分散を推定する機能を備えています。これらの推定をするには、ロバスト標準誤差の場合はvce(robust)、クラスタロバスト標準誤差の場合はvce(cluster clustvar)オプションを指定します。以下では、なぜそのようなオプションを指定するのか、vce(robust)の有無により標準誤差の解釈をどう変えるのか、そしてクラスタロバスト標準誤差に関連する重要な概念を俯瞰します。

推定値の分散を求めることは標準誤差を求めることであり、他の言い方をすれば、分散共分散行列を推定することです。分散共分行列の対角要素の平方根が標準誤差になります。分散共分散行列は分散の行列と考えることができます。推定のコマンドは分散の推定値を求めるものであり、それを利用して信頼区間や有意性検定を行います。

標準的な分散の推定量以外にも、様々な研究者が提案した分散が多数、存在します。中でも有名なものがHuberとWhiteの考案したものです。前者の提案した分散は,その計算式の形から、分散のサンドイッチ推定量と呼ばれ、後者は、その主張から分散の堅牢な推定量と呼ばれています。また、後者の推定量はサーベイ研究の視点からも、多く研究されています。

汎用的なモデルを利用して分散の推定量を求めます。次のようなモデルについて考えます。

$$y_i = \mathbf{x}_i \beta + \epsilon_i, \quad \epsilon_i \sim N\left(0, \sigma^2\right)$$

ここでは回帰分析を利用する理由は議論しません。回帰モデルを利用するにおいて、モデルの設定が正しと仮定してβの推定量と分散を求めるものとします。

$\widehat{\beta}$の標準誤差の計算はモデルが真であるという前提で行います。実現値のy_iと理論値$\mathbf{x}_i \beta$が一致することはありません。故に、βの値を正確に求めなければなりません。実現値y_iは理論値とノイズϵ_iの和となります。ノイズの分布はガウス分布であり、分散は一定とします。ノイズの存在がβの不確実性の源泉ですが、分布を仮定することで$\widehat{\beta}$の標本分布の計算が可能になります。

$\widehat{\beta}$の標準誤差は撹乱項、ϵに依存しますが、その値は回帰モデルが真のモデルであることを前提としています。しかし、我々は真のモデルのβとσ^2を知ることはできません。我々に分かることは、標本が無限に手に入る場合、βの推定量$\widehat{\beta}$は、βの真の値に収束し、分散はゼロに近づくということだけです。

モデルの推定について別の角度から考えてみましょう。次に示すモデルをフィットします。

$$y_i = \mathbf{x}_i \beta + e_i$$

$\mathbf{b}$の推定値は次のように求めることができます。

$$\hat{\mathbf{b}} = (\mathbf{X}' \mathbf{X})^{-1} \mathbf{X}' \mathbf{y}$$

ここではモデル式が真のモデルを示すという仮定や、e_iの分布を特には、設定しません。よって考え方の違いを強調するために、ここからはβとϵ_iを$\mathbf{b}$とe_iに変更します。そしてデータを用いて、上式による計算を行います。興味深いことに、確率分布を考えないこの方法でも、$\hat{\mathbf{b}}$の標準誤差を計算できます。ただし、標準誤差が何を表現するものであるか、という点については考え方を決めておく必要があります。

ここでは、データサンプリングと推定を繰り返し、何回も行った時に得られる$\hat{\mathbf{b}}$の標準誤差を示すものと定義します。

この考え方はモデルに依存した、一般的な標準誤差の考え方とは異なりますが、関連性はあります。両者とも$\mathbf{b}$とβの不確実性を示す統計量です。最初に紹介したモデルベースの考え方は誤差の変動に依拠するもので、標準誤差の利用には仮定が存在します。しかし、二番目の考え方では仮定はより緩いものなので、分析上、利用しやすいものになります。

点推定値についても、標準誤差と同じく二つの異なる考え方を示すことができます。$\hat{\mathbf{b}}$はモデルが真であるという仮定の下での推定値です。しかし、$\mathbf{b}$の推定値$\hat{\mathbf{b}}$はデータのサンプリングと推定を繰り返したときに収束する値と考えることができます。

$\mathbf{b}$の推定値は不偏と言えるでしょうか。“$\mathbf{b}$はβに等しいか”という事は推定したモデル真のモデルである場合に言えます。もし、真の場合は、$\hat{\mathbf{b}}$は$\mathbf{b}$の不偏推定値であると言えます。

しかし、$\mathbf{x}$とeが相関している場合は、どうでしょうか。問題は何もないと言えるでしょうか。結果から言えば、解釈に問題が生じます。つまり、$\mathbf{b}$は我々が欲するβの推定量とはなりません。しかし、実験や推定を繰り返すことができる場合、そこで得られた推定値は適切なものとなります。

したがって、パラメータの解釈と、推定における推定値の分散については、それぞれ異なる解釈を与えることができます。結果として、2通りの解釈はデータが単純なランダムサンプリングでなかったり、$(\mathbf{x}_i, \epsilon_i)$の分布がi.i.dで無

い場合、適切な統計的推測はどのように行うべきなのでしょうか。必然的に、我々には標準的なケースから逸脱した場合でも、適切な標準誤差を提供する方法が必要になります。

分散の堅牢な指定値はHuber (1967) および White (1980, 1982) (それぞれ、個別に)提案されました。もちろん、彼らの提案以外にも、Gail, Tan, and Piantadosi (1988)Kent (1982); Royall (1986); Lin and Wei (1989)などがあります。サーベイ論文の視点から、Kish and Frankel (1974), Fuller (1975), Binder (1983)などが提案されています。Stataの推定コマンドで、これらの標準准誤差の推定値はvce(robust)というオプションを利用して求めることができます。

20.22.1 標準誤差の解釈

vce(robust)を利用せずに、推定値の標準誤差を求めます。

```
. use https://www.stata-press.com/data/r17/auto7
(1978 automobile data)
. regress mpg weight foreign
```

Source	SS	df	MS	Number of obs	=	74
				F(2, 71)	=	69.75
Model	1619.2877	2	809.643849	Prob > F	=	0.0000
Residual	824.171761	71	11.608053	R-squared	=	0.6627
				Adj R-squared	=	0.6532
Total	2443.45946	73	33.4720474	Root MSE	=	3.4071

mpg	Coefficient	Std. err.	t	P>|t|	[95% conf.	interval]
weight	-.0065879	.0006371	-10.34	0.000	-.0078583	-.0053175
foreign	-1.650029	1.075994	-1.53	0.130	-3.7955	.4954422
_cons	41.6797	2.165547	19.25	0.000	37.36172	45.99768

次に、vce(robust)を利用します。

```
. regress mpg weight foreign, vce(robust) L
inear regression

                                                Number of obs     =         74
                                                F(2, 71)          =      73.81
                                                Prob > F          =     0.0000
                                                R-squared         =     0.6627
                                                Root MSE          =     3.4071

------------------------------------------------------------------------------
             |               Robust s
         mpg |      Coef.   td. err.      t    P>|t|     [95% conf. interval]
-------------+----------------------------------------------------------------
      weight |  -.0065879   .0005462   -12.06   0.000    -.007677   -.0054988
     foreign |  -1.650029   1.132566    -1.46   0.150    -3.908301    .6082424
       _cons |   41.6797   1.797553    23.19   0.000     38.09548    45.26392
------------------------------------------------------------------------------
```

どちらのケースも点推定値は同じです。異なる標準誤差を計算するvce(robust)オプションの利用例については、[R] **regress**にあるexampleを参照してください。

この違いをどのように理解すればよいでしょうか。最初にモデルを基本にして考えてみましょう。次の式を想定してください。

$$y_i = \mathbf{x}_i \beta + e_i$$

ここで$(\mathbf{x}_i, \epsilon_i)$は分散を$\sigma^2$とする独立で、同一な分布(i.i.d)から生成されているものとします。モデルを中心にして考えると、我々は$\mathbf{x}_i$とϵ_iは無相関であると、仮定しなければなりません。この仮定といくつかの統計的な条件が成り立っている場合、上記の回帰分析の結果、一致性のあるパラメータ推定値と標準誤差を得ることができます。次にこの仮定を弱めて、$(\mathbf{x}_i, \epsilon_i)$は独立だが、必ずしも同一ではないとします。この場合でも推定値に一致性はありますが、回帰の標準誤差は統計的推測には利用できません。つまり、同一の分布から生成された撹乱項で無いという事実に対して、堅牢な標準誤差の推定値を求める必要があります。二番目のモデルにおける標準誤差は我々の目的に適ったものです。つまり、データが同一の分布から作成されたものでないにも関わらず、係数に対して適切な統計的推定推測が可能となります。

次に、モデルに依存しないという視点から考えてみます。最初に$(\mathbf{x}_i, e_i)$がi.i.dであるというサーベイデザインを利用してデータを取得した場合、堅牢ではない標準誤差を利用してそのまま母集団パラメータ$\mathbf{b}$に対する統計的推測に利用できます。このようなケースでは、$\mathbf{x}_i$とϵ_iが無相関と仮定する必要はありません。実際にそれらが無相関である場合、母集団パラメータ$\mathbf{b}$とモデル推定値βは一致します。しかし、相関が存在する場合、母集団パラメータ$\mathbf{b}$とモデルによるパラメータβは一致しません。つまり、推定値は異なりますが、統計的推測のためには標準誤差が必要になります。つまり、データを収集する過程で$(\mathbf{x}_i, e_i)$が独立だが同一で無い場合、母集団パラメータ$\mathbf{b}$について適切な統計的推測を行うためには堅牢な標準誤差を求める必要があります。

20.22.2 誤差項の相関:クラスタに対する堅牢な標準誤差

通常の標準誤差に比べ、堅牢な標準誤差ではデータの独立性に関する仮定を緩めることができる点に特徴があります。したがって、データに相関がある場合に、vce(cluster clustvar)オプションを利用すると、それに対応した、正しい標準誤差を計算します。

自動車のデータを例にして考えてみましょう。同じ自動車メーカーの車種の間に相関が無い、と考えるのは無理があるように思えます。車種が違っても、製造メーカーが同じなら、技術、エンジン、製造ラインに関してなんらの相関があると考えられます。先の回帰におい

て、VW Dasherの残差は-2.80です。同じ会社のVW Rabbitについて、その残差は0近辺の値になると考える方が合理的でしょうか。(実際の残差は-2.32です)一方、 Chevrolet Malibu の残差は1.27となっています。この情報を使ってChevrolet Monte Carlo の残差の期待値を想像することができるでしょうか?(残差は1.53です)

データによってバラツキがあるので、結論を出すのは慎重にしなければなりません。実際、Datsun 210と510 (残差 +8.28と−1.01)、Cadillac EldoradoとSeville (残差 −1.99と +7.58)です。しかし、独立性に関しては少なからず仮定に対して疑問が生じる値も含まれています。車重と燃費の関係はメーカーごとに独立であると考えられます。しかし、メーカー単位で見ると、その関係には相関があると考えられます。

vce(robust)とvce(cluster clustvar)オプションはデータが独立であるという仮定を緩めた場合に利用するもので、後者はあるグループ単位では独立だが、そのグループ内では相関を持つ場合に利用します。

```
. regress mpg weight foreign, vce(cluster manufacturer) Linear regression
                                                Number of obs     =         74
                                                F(2, 22)          =      90.93
                                                Prob > F          =     0.0000
                                                R-squared         =     0.6627
                                                Root MSE          =     3.4071

                                   (Std.err. adjusted for 23 clusters in manufacturer)
------------------------------------------------------------------------------
             |               Robust
         mpg |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
      weight |  -.0065879   .0005339   -12.34   0.000    -.0076952   -.0054806
     foreign |  -1.650029   1.039033    -1.59   0.127    -3.804852    .5047939
       _cons |   41.6797   1.844559    22.60   0.000     37.85432    45.50508
------------------------------------------------------------------------------
```

ここで利用してデータの場合、vce(cluster clustvar)を使っても、標準誤差の変化はさほど大きくありません。VW やChevroletのような例は一部に過ぎません。もし、データ全体において同様の傾向が見られるようであれば、信頼区間はより広くなります。変数manufacturerには自動車メーカの名前が“Chev.”や“VW”という具合に入っています。この変数には“Chev. Malibu”や“VW Rabbit”など、変数から取りだした先頭の単語が入っています。

クラスタリングモデルの特徴を理解するため、[R] **regress**にregress,vce(robust)コマンドで推定したランダム効果モデル、一般的なOLS推定モデル、そしてGLSランダク効果推定の比較結果を掲載しました。ここではそ結果だけを簡単に紹介します。

14-46才の女性、4,711件のデータを利用します。女性一人当たり、平均で約6.057個のデータを取得しています。よって、データの個数は全体で28,534個あります。モデルは賃金関数で、賃金の対数を年齢、年齢の二乗項、そして現在の就業年数に回帰させます。これらの変数のうち、(例)としてに就業年数のパラメータに着目します。推定結果はそれぞれ、次のようになります。

推定量	点推定値	信頼区間
(不適切な)最小二乗法	0.039	[0.038, 0.041]
堅牢な分散(クラスタ)	0.039	[0.036, 0.042]
GLSランダム効果	0.026	[0.025, 0.027]

クラスタを考慮した堅牢な推定量とGLS推定によるランダム効果モデルを比較してみましょう。実際にハウスマンのspecification testを実行したところ、$\chi^2(3) = 336.62$という値を得ました。つまり、モデルの仕様が正しい、という帰無仮説は棄却されます。したがって、その推定値と標準誤差に問題がある、と理解します。以上の結果から、次のように考えることができます。

堅牢な標準誤差の推定値をここでは採用すべきだが、次にのべる点に留意する必要があります。つまり、堅牢な標準誤差の解釈とは、女性の賃金について繰り返し、データをサンプリングした時に、tenureの係数推定値の95%は上記に示した信頼区間[0.036, 0.042]に入る、と考えます。

堅牢な回帰の結果はこのように解釈します。一般的な回帰分析の理解とは異なりますので、注意してください。ハウスマン検定は観測できないデータに着目し、女性個人、および他の女性就業者と比較した異質性を考慮したモデルに対する検定で、[0.036, 0.042]という信頼区間を吟味するものです。

分散の堅牢な推定量の計算公式は次の通りです。

$$\widehat{\nu} = \hat{\mathbf{V}}\left(\sum_{j-1}^{N} \mathbf{u}'_{\ j}\mathbf{u}_{j}\right)\hat{\mathbf{V}}$$

ここで、$\hat{\mathbf{V}} = \left(-\partial^2 \ln L/\partial \boldsymbol{\beta}^{\ 2}\right)^{-1}$は一般的な分散の推定量です。そして$\mathbf{u}_j$(行ベクトル)はj番目のデータの、$\partial \ln L/\partial \boldsymbol{\beta}$に対するコントリビューションです。

先の例ではデータは独立であると考えました。ここではj番目のデータは独立ではなく、M個のグループ G_1, G_2, ... , G_Mに分類可能であるとします。そしてこのグループは独立であるとします。この時の分散の堅牢な推定量の計算公式は次の通りです。

$$\widehat{\nu} = \hat{\mathbf{V}}\left(\sum_{k-1}^{N} \mathbf{u}_k^{(G)\prime}\ \mathbf{u}_k^{(G)}\right)\hat{\mathbf{V}}$$

ここで $\mathbf{u}_k^{(G)}$はk番目のグループの$\partial \ln L/\partial \boldsymbol{\beta}$に対するコントリビューションです。ここでは$\mathbf{u}_j$, j = 1, ... , Nではなく、$\mathbf{u}_k^{(G)}$, k = 1, ... , Mとして、$\partial \ln L/\partial \boldsymbol{\beta}$の分解を行います。データの番号にjを利用すると、対数尤度関数は次のようになります。

$$\ln L = \sum_{j=1}^{N} \ln L_j$$

よって、$\mathbf{u}_j = \partial \ln L_j/\partial \boldsymbol{\beta}$となり、

$$\mathbf{u}_k^{(G)} = \sum_{j \in G_k} \mathbf{u}_j$$

vce(cluster clustvar)オプションを利用した場合、このような計算を行います。(この考え方はRogers [1993]が提案したものですが、Huber [1967]を一般化したものです。それをStataにはインプリメントしてあります。)

❑ テクニカルノート

上述の内容は漸近的には正しいのですが、$\widehat{\nu}$に対する有限標本調整は無視しています。最尤推定量の場合、vce(cluster clustvar)でなく、vce(robust) を利用した方が分散の推定量は$\widehat{\nu}^{*} = \{N/(N-1)\}\widehat{\nu}$で計算でき、より優れたものになります。オプションをvce(cluster clustvar)とした場合、計算式は$\widehat{\nu}^{*} = \{M/(M-1)\}\widehat{\nu}$となります。

線形回帰の場合、vce(cluster clustvar)を利用しない有限標本調整は$N/(N-k)$となります。ここでkは自由度です。vce(cluster clustvar)を利用した場合は$\{M/(M-1)\}\{(N-1)/(N-k)\}$となります。$\widehat{V}^{*}$の計算方法として、データにより2つの方法が提案されています。MacKinnon and White (1985)はregressコマンド用の方法を示しています。詳細は[R] **regress**を参照してください。Angrist and Pischke (2009, chap. 8)は堅牢な共分散行列の推定方法を提案しており、クロスセクションおよびパネルデータの場合のvce(robust)とvce(cluster clustvar)の、実用的な用法を示しています。

Halbert Lynn White Jr. (1950-2012)はカンザスシティにて生を受けました。経済学の学位をプリンストンとMITにて取得した後、ロチェスタ大学にて計量経済学の教育と研究に従事し、1979年からはUCサンディエゴ校にて同様の研究を行いました。彼は経済と法務に関するコンサルティング会社を共同設立し、特に計量経済学の手法をビジネスにおいて精密に適用した事でも有名です。堅牢な共分散行列を用いた彼の不均一分散に関する1980年の論文は、2012年のGoogle Scholarでの引用数は16,000にも及んでいます。また、1982年に発表した、定式化に問題のあるモデルにおける最尤推定に関する論文は、今では一般的になった疑似最尤法の開発に貢献したものと評価されています。これらの研究の後、ニューラルネットワークの利用、ノンパラメトリックモデル、そして金融マーケットにおける時系列モデルの研究に取り組みました。

彼は多くの賞を受賞すると共に、American Academy of Arts and Sciences and the Econometric Societyのフェローとなり、さらにはJohn Simon Guggenheim Memorial Foundationから特別研究員として認められました。もし、彼がその早すぎる死を迎えなければノーベル経済学賞を受賞したであろうと多くの研究者は考えています。
Whiteは熱心なジャズプレーヤーでもありました。UCサンディエゴ校のフェローでもある人気のトロンボニストJimmy Cheatamとの演奏は特に有名でした。

Peter Jost Huber(1934-)はスイス、アールガウ州に生まれました。ホモトピー論に関する博士論文により、ETH Zürichで数学の学位を取得し、ポスドクとしてその結果、1964年に堅牢な推定に関する論文を発表し、その論文はその後、推定の分野において重要な位置を示るようになりました。Huberはその後、Conell, ETH Zürich, Harvard, MIT, そしてBayreuthと、大西洋を行きつ戻りつしながら、研究業績を積み重ねています。彼の研究は統計学の他の分野にも及んでおり、しかも、理論および実用の両面においてなされており、その中には、回帰、探索的多変量分析、ラージデータセット、そして統計的コンピューティングなどが含まれています。Huberはさらにバビロニア天文学においても長い間、関心を寄せています。

20.23 スコアの計算

vce(robust)オプションが利用可能な推定コマンドでは、predictコマンドを利用して推定式水準(equation-level)のスコア変数を作成できます。scoreオプションを利用すると、predictコマンドは堅牢な推定値の計算で利用する重要な計算要素をデータセットとして算出します。先の[U] **20.22 堅牢な分散推定値**で説明したように、有限母集団修正を無視した場合、分散の堅牢な推定値は次のようになります。

$$\widehat{V} = \hat{V}\left(\sum_{j=1}^{N} \mathbf{u}_j' \mathbf{u}_j\right)\hat{V}$$

ここで$\hat{\mathbf{V}} = \left(-\partial^2 \ln L/\partial \beta^2\right)^{-1}$は一般的な分散の推定量です。和の形として求められる尤度関数について考えてみましょう。

$$\ln L = \sum_{j=1}^{N} \ln L_j$$

すると、 $\mathbf{u}_j = \partial \ln L_j/\partial \beta$ 一般的に関数L_jは$\mathbf{x}_j$と β の関数$L_j\left(\beta; \mathbf{x}_j\right)$です。尤度関数の関数形には色々なものがありますが、引数は線形モデル$\mathbf{x}_j \beta$となります。これらの場合、

$$\frac{\partial \ln L_j\left(\mathbf{x}_j \beta\right)}{\partial \beta} = \frac{\partial \ln L_j\left(\mathbf{x}_j \beta\right)}{\partial \mathbf{x}_j \beta} \frac{\partial\left(\mathbf{x}_j \beta\right)}{\partial \beta} = \frac{\partial \ln L_j\left(\mathbf{x}_j \beta\right)}{\partial \mathbf{x}_j \beta} \mathbf{x}_j$$

$u_j = \partial \ln L_j\left(\mathbf{x}_j \beta\right)/\partial\left(\mathbf{x}_j \beta\right)$と書きなおすと、単純に$u_j \mathbf{x}_j$という形になります。よって、分散の堅牢な推定値の計算式は次のようになります。

$$\widehat{v} = \hat{\mathbf{V}}\left(\sum_{j=1}^{N} u_j^2 \mathbf{x}_j{}' \mathbf{x}_j\right)\hat{\mathbf{V}}$$

ここで、u_jのことを推定式水準のスコアと呼ぶ事にし、これはpredictコマンドでscoreオプションを利用することで取り出すことができます。u_jは残差のように、次のような性質も持っています。

1. $\sum_j u_j = 0$
2. j = 1, 2, ..., Nに対するu_jと$\mathbf{x}_j$の相関は0

実際には、線形回帰において、u_jは標準化された残差です。

$$\frac{\partial \ln L_j}{\partial\left(\mathbf{x}_j \beta\right)} = \frac{\partial}{\partial\left(\mathbf{x}_j \beta\right)} \ln f\left\{\left(y_j - \mathbf{x}_j \beta\right)/\sigma\right\}$$
$$= \left(y_j - \mathbf{x}_j \beta\right)/\sigma$$

f(·)は標準正規密度関数です。

例題19

probitコマンドではvce(robust)オプションと、推定後にpredict、scoreコマンドを利用できます。推定式水準のスコアは分散の、堅牢な推定値を計算する際に重要な役割を果たします。ここではpredictとscoreの利用例を示します。

```
. use https://www.stata-press.com/data/r17/auto2
(1978 automobile data)
. probit foreign mpg weight
Iteration 0:   log likelihood =  -45.03321
Iteration 1:   log likelihood = -27.914626
Iteration 2:   log likelihood = -26.858074
Iteration 3:   log likelihood = -26.844197
Iteration 4:   log likelihood = -26.844189
Iteration 5:   log likelihood = -26.844189

Probit regression                               Number of obs     =         74
                                                LR chi2(2)        =      36.38
                                                Prob > chi2       =     0.0000
Log likelihood = -26.844189                     Pseudo R2         =     0.4039

------------------------------------------------------------------------------
     foreign | Coefficient  Std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
         mpg |  -.1039503   .0515689    -2.02   0.044    -.2050235   -.0028772
      weight |  -.0023355   .0005661    -4.13   0.000    -.003445   -.0012261
       _cons |   8.275464   2.554142     3.24   0.001     3.269437    13.28149
------------------------------------------------------------------------------
. predict double u, score
. summarize u
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
           u |         74   -6.64e-14    .5988325  -1.655439   1.660787
. correlate u mpg weight (obs=74)
(obs=74)

             |        u      mpg   weight
-------------+---------------------------
           u |   1.0000
         mpg |   0.0000   1.0000
      weight |  -0.0000  -0.8072   1.0000
. list make foreign mpg weight u if abs(u                )>1.65

     +-------------------------------------------------------------+
     | make          foreign    mpg   weight          u            |
     |-------------------------------------------------------------|
24.  | Ford Fiesta   Domestic    28    1,800   -1.6554395           |
64.  | Peugeot 604   Foreign     14    3,420    1.6607871           |
     +-------------------------------------------------------------+
```

高燃費なFord Fiestaは国産であり、燃費の悪いPeugeot 604 は外車であることが分かります。

◁

❑ テクニカルノート

推定のコマンドによっては複数のスコアを利用することがあります。例えば、$L_j(\mathbf{x}_j\boldsymbol{\beta}_1, \mathbf{z}_j\boldsymbol{\beta}_2)$のように2つの線形式を含む尤度について考えてみましょう。$\partial \ln L_j/\partial\boldsymbol{\beta}$は$(\partial \ln L_j/\partial\boldsymbol{\beta}_1, \partial \ln L_j/\partial\boldsymbol{\beta}_2)$と書くことができます。ここで各要素は$[\partial \ln L_j/\partial\boldsymbol{\beta}_1]\mathbf{x} = u_1\mathbf{x}$および$[\partial \ln L_j/\partial\boldsymbol{\beta}_2]\mathbf{x} = u_2\mathbf{x}$と書きなおせます。この時、2つの推定式水準スコア$u_1$と$u_2$が存在するとします。関数が増えればそれだけスコアも増えると考えます。

例としてStataのstreg, distribution(weibull)コマンドを紹介します。このコマンドはβとシェープパラメータlnpを推定し、後者は線形式(lnp)$\mathbf{z}$で、$\mathbf{z}$=1とした時の値と考えます。このコマンドの実行後にpredict, scoresコマンドを利用する場合、2つの新しい変数名、または、stub*としてstub1, stub2を自動作成する引数を必ず用います。最初の変数にはβ結合したu_1が入り、2番目の変数にはlnpと結合したu_2が入ります。

❑

❑ テクニカルノート

Stataの行列コマンドの使用方法については[P] **matrix** をご参照ください。堅牢な分散推定値を自分で計算し、Stataの値と比較して確認することができます。

```
. use https://www.stata-press.com/data/r17/auto2, clear
(1978 automobile data)
. quietly probit foreign mpg weight
. predict double u, score
. matrix accum S = mpg weight [iweight=u^2*74/73]
(obs=26.53642547)
. matrix rV = e(V)*S*e(V)
. matrix list rV
symmetric rV[3,3]
                    foreign:     foreign:     foreign:
                        mpg       weight        _cons foreign:mpg     .003
   52299
foreign:weight    .00002216    2.434e-07
 foreign:_cons   -.14090346  -.00117031     6.4474174
. quietly probit foreign mpg weight, vce(robust)
. matrix list e(V)
symmetric e(V)[3,3]
                    foreign:     foreign:     foreign:
                        mpg       weight        _cons foreign:mpg     .0035229
   9
foreign:weight    .00002216    2.434e-07
 foreign:_cons  -.14090346   -.00117031     6.4474174
```

結果は同じです。

行列のプログラミングを行うユーザにとってポイントがこのプログラムに含まれています。スコアが分かっていれば、一般的に計算した標準誤差から簡単に堅牢な標準誤差を計算できます。ですから、新しい推定コマンドをプログラミングしたような場合でも、vce(robust)オプションを用意することは比較的簡単です。

クラスタリングを無視すると、計算はとても簡単になります。クラスタリングが存在する場合、和の計算が必要になるので、少し手間がかかります。推定プログラミングにおいて、堅牢な標準誤差の計算が必要な場合は_robustコマンドを利用すれば手間は省けます。詳細は[P] **robust**を参照してください。

実際に_robustコマンドを利用する場合は最初に一般的な標準誤差(と係数ベクトル、共分散行列)と、スコアu_j(尤度関数が複数のstubを有する場合)を含む変数を計算します。その後、_robustを利用します。このコマンドは一般的に標準誤差を、堅牢な標準誤差に変換します。_robustコマンドはクラスタリングや、有限修正のあるデータに対して利用します。したがって、推定結果の画面では標準誤差のラベル部分に、堅牢な推定値である事を表示します。

もちろん、Stataの最尤法による最適化コマンドmlを利用すれば、話は簡単にです。単純にvce(robust)オプションをmlで利用するだけで済みます。mlコマンドは_robustコマンドを自動的に呼び出し、推定を行います。

❑

❑ テクニカルノート

推定コマンドによってはpredict,scoreコマンドにより推定式水準のスコア$\partial L_j/\partial \beta$ではなく、パンラメータ水準のスコア$\partial L_j/\partial \mathbf{x}_j \beta$を算出します。cmclogit, stcoxやマルチレベル混合効果コマンドでは、複数の応答を有するという共通した特徴があります。

堅牢な分散を計算する場合、必要な情報はパラメータ水準のスコア$\partial L_j/\partial \beta$です。殆どの場合、チェーンルールを利用して、推定式水準のスコアから目的のパラメータ水準の値を得られます。しかし、場合によってはチェーンルールが利用できない場合もあります。前述のケースでは、グループ水準で尤度を計算できますが、各観測値のコントリビューションに分割することはできません。つまり、チェーンルールが利用できないので、パラメータ水準のスコアを直接、計算しなければなりません。

仮に、推定式の各パラメータが定数項として取り扱える場合に、_robustオプションは利用できます。_robustコマンドをコールする前に、共分散行列において行と列の値を再設定します。各列と各行の推定式名はユニークであり、変数名はすべての_consとします。

❑

20.24　加重推定

加重の構文については[U] 11.1.6 加重を参照してください。Stataでは4種類の加重法を使用できます。Stataには4種類の加重方法が用意されています。fweightは度数による加重、 pweightはサンプリング加重、aweightは解析的加重、iweight,は重点加重です。構文はすべて同じです。次のようにコマンドウィンドウに入力します。

```
. regress y x1 x2
```

非加重の推定値を得ます。

```
. regress y x1 x2 [pweight=pop]
```

pweight推定値を算出します。

実際の用法は次のセクションで解説します。

20.24.1　度数加重

度数による加重は、整数値の情報を利用し、重み付けを行います。統計学的な興味という意味で、加重自体には大きな意味はありません。しかし、データ生成という視点から見ると、重要な意味を持ってきます。例えば、次のようなデータがあるとします。

Y	x1	x2
22	1	0
22	1	0
22	1	1
23	0	1
23	0	1
23	0	1

次の推定コマンドを実行します。

```
. regress y x1 x2
```

次のようにまとめたデータを用いて推定することもできます。

Y	x1	x2	pop
22	1	0	2
22	1	1	1
23	0	1	3

この場合は、次の推定コマンドを実行します。

```
. regress y x1 x2 [fweight=pop]
```

度数加重を利用する場合、データが指定した度数分だけ存在するものとして推定します。

❑ テクニカルノート

ユーザが作成したコマンドなど、コマンドによっては加重が利用できないものもあります。加重が利用できないコマンドについては、expand（参照 ［D］ **expand**）コマンドを利用して、度数加重を利用可能にする方法もあります。expandコマンドは実際にデータを複製し、データの個数を増やすことによって加重を可能にします。例えば、ユーザが作成したusercmdというコマンドはあるものとします。そのコマンドでusercmd y x1 x2 [fw=pop]としたいのですが、このusercmd コマンドは加重オプションをサポートしていません。そのような場合は、

```
. expand pop
. usercmd y x1 x2
```

という形でコマンドを実行します。ここで注意点を確認しておきます。このパターンでデータを用意した場合、加重オプションを利用しないコマンドを実行することになります。加重情報なしの複製済みデータと、度数加重を利用したデータは、まったく同じものとなります。

❑

20.24.2 解析的加重

解析的加重とはStata内で便宜的に利用する用語です。観測値が、複数のデータの平均値であるような特殊ケースにおいて利用します。次のようなモデルを例に考えます。

$$y_i = \mathbf{x}_i \boldsymbol{\beta} + \epsilon_i, \qquad \epsilon_i \sim N\left(0, \sigma^2\right)$$

このモデルを$\mathbf{x}_j$とy_jの平均値のデータセットにフィットします。例えば、本来の観測値が(3, 1), (4, 2), (2, 2)であるとします。しかし、研究者が分かるのはそれらの平均値だけであるとします。すなわち、平均値として {(3 + 4 + 2)/3, (1 + 2 + 2)/3} = (3, 1.67)という値だけが手元にあり、実際の3つのデータの情報は持っていないとします。いま、研究中のデータがすべてこのようなタイプのものであるとします。

このケースでregressコマンドを利用する場合はaweightオプションを利用します。

```
. regress y x [aweight=pop]
```

このようなケースでは、過去に誤った方法が利用されてました。研究者はここでいうセル平均でなく、確率加重標本を一般的に利用していました。Stataというソフトが存在する以前から、aweightの手法を利用して確率加重標本を求めていました。この点に関する詳細は[U] 20.24.3 サンプリング加重の項目で解説します。

ここで、aweightで解決できる統計モデルを示しておきます。

$$y_i = \mathbf{x}_i \beta + \epsilon_i, \qquad \epsilon_i \sim N\left(0, \sigma^2 / w_i\right)$$

w_iが解析的加重です。実際の計算はあたかもfweightを利用しているような形で加重を利用して線形回帰の計算を実行します。ただし、計算の前に合計がNになるように標準化します。

殆どすべてのコマンドについて、このaweightが利用できます。aweightを利用すると、次のような処理を行います。

1. 合計がNになるように標準化します。
2. そして、fweightと同じ計算式を利用します。

上記の線形回帰でaweightに焦点を当てていますが、aweightは回帰以外のコマンドで許可されています。 これらの重みは、分散や精度が異なる観測値を説明するためにより一般的に使用できます。その意味で、分析的重みを精度重みと呼ぶこともできます。

20.24.3　サンプリング加重

サンプリング加重pweightとは確率加重を利用したランダムサンプリングの際に利用します。実際、[pweight=...]で指定する変数には、サンプリングした標本が、母集団に含まれる全体数を入力します。標本としてサンプリングされる確率が1/3の場合、pweightは3になります。

研究者によっては、このケースでaweightを利用することもあるようです。しかし、残念ながら、そのような用法は誤りです。次のような回帰モデルについて考えてみましょう。

$$y_i = \mathbf{x}_i \beta + \epsilon_i, \qquad \epsilon_i \sim N\left(0, \sigma^2\right)$$

このモデルを平均値のデータに対するaweightでフィットすると、グループごとに不均一分散が生じてしまいます。誤差項ϵ_iは均一分散でなければなりません。つまり、分散はσ^2で、一定である必要があります。例えば、最初のデータは3つのデータの平均であるとします。

$$y_1 = \mathbf{x}_1 \beta + \epsilon_1, \qquad \epsilon_i \sim N\left(0, \sigma^2\right)$$
$$y_2 = \mathbf{x}_2 \beta + \epsilon_2, \qquad \epsilon_i \sim N\left(0, \sigma^2\right)$$
$$y_3 = \mathbf{x}_3 \beta + \epsilon_3, \qquad \epsilon_i \sim N\left(0, \sigma^2\right)$$

これらの平均を取ります。

$$\left(y_1 + y_2 + y_3\right)/3 = \left\{\left(\mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3\right)/3\right\} \beta + \left(\epsilon_1 + \epsilon_2 + \epsilon_3\right)/3$$

他のデータについても同じ考え方を適用すると、結果的にグループごとに分散が異なってしまいます。つまり、このデータの場合のモデルは次のようになります。

$$\bar{y}_i = \bar{x}_i \beta + \bar{\epsilon}_i, \qquad \bar{\epsilon}_i \sim N\left(0, \sigma^2 / N_i\right)$$

極端な例を用いて考えてみましょう。2個のデータの平均値と、100,000個のデータの平均値だけが手元にあるとします。残差の分散について考えた場合、100,000個の場合の分散が小さくなります。つまり、100,000個のデータにはより多くの情報が含まれていることを示唆しています。

次に確率加重を利用して同じモデル$y_i = \mathbf{x}_i \beta + \epsilon_i, \qquad \epsilon_i \sim N\left(0, \sigma^2\right)$にフィットを行います。この場合、データセット中の各データは、それぞれ異なる確率で標本に含まれることを示しています。よって、各データについて次の関係が成立します。

$$y_i = \mathbf{x}_i \beta + \epsilon_i, \qquad \epsilon_i \sim N\left(0, \sigma^2\right)$$

この場合、不均一分散の問題は生じません。aweightによる推定量をこのケースで利用するのは不適切です。

分析を行う際、引数を利用することで、加重の調整を自動的に行われます。加重に調整を行わない、ということはモ

デルが真であることを強く主張することになります。結果的に、サンプリグ加重を利用することで次の2つの項目をピックアップすることになります。

1. βの点推定値$\widehat{\beta}$の効率性
2. 標準誤差($\widehat{\beta}$の分散行列)

効率性と加重による調整は相反するものです。つまり、効率性の問題があるので、多くの研究者はpweight用のデータでaweightを利用してしまうのです。点推定値に対するpweightの調整と、aweightによる調整は同じことなります。

2番目の標準誤差については、aweightを利用すると、その推定結果は不適切なものになります。その理由は大きな加重により、そのデータがより正確な値であるとことを強調してしまうためです。pweightの場合、点推定値は観測値と残差ejと、分散σ^2の対になっているので、精度を上げることはできません。[U] 20.21 分散推定値の計算ではデータの収集と推定を繰り返すことによって得られる、分散推定量を紹介しました。pweightを利用する場合、モデルが真のモデルである場合に限る、という制約を外して考えることができます。もし、データの収集と推定を繰り返し行うような研究で、パラメータの変動を計測したいという場合は、標本のサンプリング確率を考慮する必要があります。

Stataで次のように入力します。

```
. regress y x1 x2 [pw=pop]
```

次のようなコマンドを実行しても結果は同じです。

```
. regress y x1 x2 [pw=pop], vce(robust)
```

pweightはvce(robust)オプションと同じ要領で計算を行います。つまり、加重した堅牢な標準誤差を算出します。ここでは例として、regressコマンドを利用しました。例えば、probitなど、他の推定コマンドについても同じ結果となります。もちろん、vce(robust)オプションを利用できないコマンドを少なからず存在しますが、それらは当然、pweightも利用できません。

pweightコマンドはサンプリグ確率がそれぞれ異なる、ランダムサンプリングの状態において利用すべきものです。そのような状態ではpweightを用いておけば十分です。ただし、サンプリングが独立ではなく、クラスタと呼ばれるグループ内でサンプリングされている場合、推定コマンドにおいてvce(cluster clustvar)オプションを利用します。

```
. regress y x1 x2 [pw=pop], vce(cluster block)
```

具体例を用いて考えてみましょう。

1. 堅牢な推定量とは、データをサンプリングし、推定する、という操作を繰り返し行った場合に観察される変動、という具合に考えることができます。このようなケースでは、推定にあたって、データを選択する際にグループ(クラスタ)について考慮する必要があります。もし、データの1と2が同じクラスタに存在している場合、1と2は同時に選ばれる可能性が高くなります。言い方を変えれば、1と2は同時に選択されない可能性も、ランダムサンプリングに比べて高くなります。
2. また、潜在的な相関関係についての考慮も必要です。つまり、ひとつのクラスタ内にあるデータを、独立なデータとして考えることはできません。この事はデータを調べてみれば、よりよく理解できます。例えば、クラスタ内における距離が近い場合、例えば、所得、経験、態度などが似ていたとしても驚には値しません。広範囲に調査した場合に比べ、例えば、年齢や性別などはにかよった値になります。

この2つのどちらの場合であっても、分散の堅牢性は同じ方法で計算します。

サンプリング加重は複雑なサンプリングデザインを用いて計算します。不均一なサンプリング確率、クラスタサンプリング、そして層別情報を利用します。Stataにはこのように複雑なサーベイデータを処理する機能が用意されており、それらは接頭語としてsvyを伴います。層別データに対して線形回帰を実行する場合は、例えば、svy:regressというコマンドを利用します。

svyコマンドを利用していない場合、pweightとクラスタリングオプションを利用すると、結果的にsvyコマンドを利用したケースと同じ計算結果になります。サンプリングデザインが単純なものである場合は、サーベイコマンドを利用しなくても正確な分析を行うことができます。もちろん、svyコマンドはサーベイデータに対して、より多くの分析を正確に行うための機能を備えています。サーベイデータの分析に関する詳細と、svyと非svyコマンドの違いに関する解説は[SVY] Surveyを参照してください。

Stataのすべてのコマンドがpweightに対応している訳ではありません。高度な計算手法や統計学上の問題によって、まだ、対応できていないものがあります。

20.24.4 重点加重

Stataのiweightコマンドは特別な目的のために利用するコマンドです。この加重コマンドを利用して他の加重コマンドにはない効果をデータに付加させることができます。例えば、regressコマンドの際にiweightを利用して、加

重を繰り返しながら最小二乗を実行することが可能になります。

iweightの効果はaweightのそれとほぼ同じですが、正規化は行いません。Stataのiweightコマンドは

1. 加重を正規化することなく、
2. fweightコマンドと同じように、計算式に加重を追加します。ただし、例外的な処理もありますので、*Methods and formulas*を参照してください。

iweightを利用するのは主に、特殊な加重を施す必要のあるプログラマに限られます。

20.25 推定後のコマンドの一覧

以下のコマンドは推定後に利用します。

[R] contrast	推定値を対比し、ANOVA型の結合検定を実施します。
[R] estat ic	赤池とシュワルツの情報基準(AICとBIC)
[R] estat summarize	推定時の標本に関する要約統計量
[R] estat vce	推定量の分散共分散行列(VCE)
[R] estimates	推定結果の格納
[TS] forecast	ダイナミック予測とシミュレーション
[R] hausman	ハウスマン検定
[R] lincom	点推定、標準誤差、検定、そして係数の線形関係についての統計的推測
[R] linktest	一推定式の定式化に関する検定
[R] lrtest	尤度比検定
[R] margins	周辺平均、予測マージン、限界効果
[R] marginsplot	マージンコマンドの結果をグラフ化する（プロファイルプロット、交互作用プロットなど)
[R] nlcom	点推定、標準誤差、検定、一般化予測値についての統計的推測
[R] predict	予測、残差、統計的推測、診断
[R] predictnl	点推定、標準誤差、検定、一般化予測値に対する統計的推測
[R] pwcompare	推定値のペアワイズな比較
[R] suest	SUR 推定
[R] test	簡単な帰無仮説や、線形結合した線形帰無仮説に対する仮説検定
[R] testnl	非線形帰無仮説に対するワルド検定

係数と標準誤差の操作に関する詳細は[U] 13.5 係数と標準誤差にアクセスするを参照してください。

ここに示したコマンドは汎用的なもので、一般的にはモデルの推定後に利用します。もちろん、推定コマンドによっては独自の、推定後に利用するコマンドを用意しています。

推定量について、その推定後のコマンドを調査する場合は、**統計** > **推定後の分析**を選択して推定事後ツール集を開きます。アクティブな推定結果に対する数多くの推定後分析機能が一覧でみられます。このツール集は、新たな推定コマンドが実行され、結果がメモリやディスクに保存されたときに自動で更新されます。詳細は[R] postestをご覧ください。

推定量について、その推定後のコマンドを調査する場合は**postestimation**というキーワードを利用して探します。リファレンスマニュアルの各推定量に関する情報の所に記載されているはずです。

20.26 参考文献

Afifi, A. A., and S. P. Azen. 1979. *Statistical Analysis: A Computer Oriented Approach*. 2nd ed. New York: Academic Press.

Angrist, J. D., and J.-S. Pischke. 2009. *Mostly Harmless Econometrics: An Empiricist's Companion*. Princeton, NJ: Princeton University Press.

Baum, C. F. 2009. *An Introduction to Stata Programming*. College Station, TX: Stata Press.

Binder, D. A. 1983. On the variances of asymptotically normal estimators from complex surveys. *International Statistical Review* 51: 279-292. https://doi.org/10.2307/1402588.

Buja, A., and H. R. Künsch. 2008. A conversation with Peter Huber. *Statistical Science* 23: 120-135. https://doi.org/10.1214/07-STS251.

Daniels, L., and N. Minot. 2020. *An Introduction to Statistics and Data Analysis Using Stata*. Thousand Oaks, CA: SAGE.

Deaton, A. S. 1997. *The Analysis of Household Surveys: A Microeconometric Approach to Development Policy*. Baltimore, MD: Johns Hopkins University Press.

Fuller, W. A. 1975. Regression analysis for sample survey. *Sankhyā, Series C* 37: 117-132.

Gail, M. H., W. Y. Tan, and S. Piantadosi. 1988. Tests for no treatment effect in randomized clinical trials. *Biometrika* 75: 57-64. https://doi.org/10.2307/2336434.

Hampel, F. R. 1992. Introduction to Huber (1964) "Robust estimation of a location parameter". In *Breakthroughs in Statistics. Volume II: Methodology and Distribution*, ed. S. Kotz and N. L. Johnson, 479-491. New York: Springer.

Huber, P. J. 1967. The behavior of maximum likelihood estimates under nonstandard conditions. In Vol. 1 of *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, 221-233. Berkeley: University of California Press.

——— .2011. *Data Analysis: What Can Be Learned from the Past 50 Years*. Hoboken, NJ: Wiley.

Kaufman, R. L. 2013. *Heteroskedasticity in Regression: Detection and Correction*. Thousand Oaks, CA: SAGE.

Kent, J. T. 1982. Robust properties of likelihood ratio tests. *Biometrika* 69: 19-27. https://doi.org/10.2307/2335849.

Kish, L., and M. R. Frankel. 1974. Inference from complex samples. *Journal of the Royal Statistical Society, Series B* 36:1-37. https://doi.org/10.1111/j.2517-6161.1974.tb00981.x.

Lin, D. Y., and L. J. Wei. 1989. The robust inference for the Cox proportional hazards model. *Journal of the American Statistical Association* 84: 1074-1078. https://doi.org/10.2307/2290085.

Lumley, T. S. 2020. Weights in statistics. Biased and Inefficient. https://notstatschat.rbind.io/2020/08/04/weights-in-statistics/.

MacKinnon, J. G., and H. L. White, Jr. 1985. Some heteroskedasticity-consistent covariance matrix estimators with improved finite sample properties. *Journal of Econometrics* 29: 305-325. https://doi.org/10.1016/0304-4076(85)90158-7.

McAleer, M., and T. Pérez-Amaral. 2012. Professor Halbert L. White, 1950-2012. *Journal of Economic Surveys* 26: 551-554. https://doi.org/10.1111/j.1467-6419.2012.00735.x.

Mitchell, M. N. 2012. *Interpreting and Visualizing Regression Models Using Stata*. College Station, TX: Stata Press. Pedace,

R. 2013. *Econometrics for Dummies*. Hoboken, NJ: Wiley.

Rogers, W. H. 1993. sg17: Regression standard errors in clustered samples. *Stata Technical Bulletin* 13: 19-23. Reprinted in *Stata Technical Bulletin Reprints*, vol. 3, pp. 88-94. College Station, TX: Stata Press.

Royall, R. M. 1986. Model robust confidence intervals using maximum likelihood estimators. *International Statistical Review* 54: 221-226. https://doi.org/10.2307/1403146.

Wasserstein, R. L., and N. A. Lazar. 2016. The ASA statement on p-values: Context, process, and purpose. American Statistician 70: 129-133. https://doi.org/10.1080/00031305.2016.1154108.

White, H. L., Jr. 1980. A heteroskedasticity-consistent covariance matrix estimator and a direct test for heteroskedasticity.
Econometrica 48: 817-838. https://doi.org/10.2307/1912934.

———. 1982. Maximum likelihood estimation of misspecified models. *Econometrica* 50: 1-25.
https://doi.org/10.2307/1912526.

Williams, R. 2012. Using the margins command to estimate and interpret adjusted predictions and marginal effects. *Stata Journal* 12: 308-331.

データ処理

21 レポート作成

目次

21.1 概要

Stataのレポート生成コマンドでは、フォーマット化されたテキスト、要約統計量、回帰結果およびグラフを含む、完全なWord, Excel, PDFおよびHTML文章を作成できます。

レポート作成コマンドには2つの種類があります。1つは、Stataのコマンドからの出力結果を負べ手含む、マークダウン形式でテキストのフォーマットを行うものです。もう一方は、Stataのコマンドで結果を保存し、文章内のテキストや表にそれらを挿入していくものです。

いずれの場合も、再現可能なレポートを作成することができます。レポートを作成するdoファイルまたはテキストファイルでコマンドを保存しましょう。こうして保存したファイルやコマンドを再実行することで、Stataの出力結果からレポートを再度作成できます。結果の再現可能性を確実にするためには、versionコマンドをファイル内に含める必要があります、詳細は、[u] 16.1.1 Versionをご覧ください。

文章は動的なものにすることも可能です。使用しているデータが変更されたときに、新しいデータセットでdoファイルを再実行するだけです。Stataのすべての結果が自動的に更新されます。

また、高度にカスタマイズした表を作成し、レポートに掲載することができます。詳細は、[R] table introと[TABLES] Introをご覧ください。

21.2 ダイナミックドキュメントコマンド

StataのダイナミックドキュメントコマンドはマークダウンテキストとStataの出力結果から、HTMLファイルやWordドキュメントにStataの出力結果を埋め込むことができます。ダイナミックタグがテキストファイル内でStataのコマンドを実行するために使用され、コードを実行して、結果を目的のファイルにエクスポートします。

Stataの出力結果を含むテキストファイルを作成するには、Stataのコマンドをダイナミックタグで囲み、dyntextコマンドで出力ファイルを作成します。例えば、次のように入力して回帰モデルにフィットして、

```
. sysuse auto
. regress mpg weight length i.foreign
```

出力結果を含む簡単なレポートにしてテキストファイルを作成するとします。さらに、レポートの先頭には、「回帰結果」とモデルの説明を入力します。このようなテキストファイルは次のように作成することができます。

```
begin dynex1.txt
Regression results
------------------
Linear regression of mpg on weight, length, and foreign.
<<dd_do>>
sysuse auto, clear
regress mpg weight length i.foreign
<</dd_do>>
end dynex1.txt
```

ダイナミックタグ、<<>dd_do>と<<>/dd_do>はStataに、その間にあるコマンドを実行させ、得られた結果をoutput.txtファイルに出力させるものです、結果が出力されたら、次のように入力します。

```
. dyntext dynex1.txt, saving(output.txt)
```

同様の回帰結果からHTMLドキュメントを作成することができます。ドキュメントのヘッダと変数名を太字にフォーマットするためにマークダウンを利用すると、次のようになります。

```
begin dynex2.html
Regression results
==================
Linear regression of **mpg** on **weight**, **length**, and **foreign**.
----
<<dd_do>>
```

```
sysuse auto, clear
regress mpg weight length i.foreign
<</dd_do>>
----
```

end dynex2.html

マークダウンフォーマットのテキストと回帰結果をHTMLファイル、dynex2.htmlを出力するには、次のように入力します。

```
. dyntext dynex2.txt
```

または、次のように入力すると、

```
. dyndoc dynex2.txt., docx
```

同じ結果を、Wordファイル、dynex2.docxとして作成します。

PDFファイルの場合は、まずWordドキュメントを作成して、docx2pdfを利用してWordドキュメントをPDFファイルに変換します。

テキストファイル、HTML, Wordドキュメントを含むダイナミックドキュメントコマンドの概要は、［RPT］**Dynamic documents intro**をご覧ください。［RPT］**Dynamic tags**ではダイナミックドキュメントに含まれるグラフなどの結果に関する情報を紹介しています。また、［RPT］**Dynodoc**では、変数や使用するデータセットが異なっても、同じようなレポートを作成する柔軟性のあるテキストファイルの作り方を説明しています。

21.3 pudtdocx, putpdf, putexcelコマンド

putdocx, putpdf, putexcelコマンドはそれぞれStataの結果を含む、カスタマイズされたWord, PDF, Excelファイルを作成します。前セクションで紹介したダイナミックドキュメントコマンドとは異なり、これらのコマンドはStataの出力結果を直接ドキュメント内に含むものではありません。その代わり、Stataの出力結果を表やテキスト内に配置します。ドキュメント作成の際に使用する一連のコマンドでは、テキストやグラフの内容やStataのコマンドから得られた統計的な結果の全体、もしくは特定の部分に関して、書式を指定することができます。

それでは、回帰した結果を含むWordドキュメントを作成してみましょう。

```
. sysuse auto
. regress mpg weight length i.foreign
```

ヘッダと得られた結果を説明するテキストも必要ですので、次のように入力します。

```
. sysuse auto
. putdocx begin
. putdocx paragraph, style(Heading1)
. putdocx text (“Regression results”)
. putdocx paragraph
. putdocx text (“Linear regression of mpg on weight, length, and foreign.”)
. regress mpg weight length i.foreign
. putdocx table regtable = e(table)
. putdocx save myreg
```

このコマンドで、ヘッダ「Regression results」と回帰についての文章のパラグラフを含むWordファイル、myreg.docxが作成されます。putdocx table regtable = e(table)コマンドはWord内に、regressコマンドで得られた結果の表を作成します。表は、モデル内の共変量ごとの係数、標準誤差、検定、および信頼区間を含みます。

PDFファイルを作成するときも、同様です。

```
. sysuse auto
. putpdf begin
. putpdf paragraph, font(“”, 20)
. putpdf text (“Regression results”)
. putpdf paragraph
. putpdf text (“Linear regression of mpg on weight, length, and foreign.”)
```

```
. regress mpg weight length i.foreign
. putpdf table regtable = e(table)
. putpdf save myreg
```

putdocxをputpdfに置き換えて、Wordではヘッダスタイルで設定されていた部分をフォントサイズを20ポイントとしています。

同じようにして、Excelファイルに結果を挿入することができます。

```
. sysuse auto
. putexcel set myreg
. regress mpg weight length i.foreign
. putexcel A3 = etable
```

これらのコマンドでは、ヘッダと回帰の結果表を含むmyreg.xlsxが作成されます。

putdocxコマンドのさらなる例と推奨される作業フローは、[RPT] **putdocx intro**をご覧ください。putpdfは[RPT] **putpdf**を、putexcelは[RPT] **putexcel**をご覧ください。

データ処理

22 データの入力とインポート

目次

22.1 概要

Stataにデータをインポートする場合、以下の方法を利用します。

[D] **edit**と[D] **input**	キーボードからデータ入力
[D] **import delimited**	デリミタで区切られるデータのインポート
[D] **import excel**	Excel ファイルのインポート
[D] **import sas**	SASファイルのインポート
[D] **import sasxport5**	SAS XPORT Version 5 形式のデータのインポート
[D] **import sasxport8**	SAS XPORT Version 8 形式のデータのインポート
[D] **import spss**	SPSSファイルのインポート
[D] **infile (free format)**	フォーマット情報の無いデータのインポート
[D] **infile (fixed format)**,[D] **infix (fixed format)**	フォーマット情報のあるデータのインポート
[D] **infile (fixed format)**	EBCDICデータのインポート
[D] **odbc**	ODBCデータのインポート
[D] **jdbc**	JDBCデータのインポート
[D] **import fred**	米連邦準備銀行経済データのインポート
[D] **import haver**	Haver Analytics形式のデータのインポート
[D] **import dbase**	dBASEファイルのインポート
[SP] **spshape2dta**	shapefileのStataが使用可能な形式への変換

データセットのフォーマットはどれも異なるので、それぞれに対応したコマンドを利用してください。

[D] **infile (fixed format)**と[D] **infix (fixed format)**は異なるコマンド名ですが、機能は同じです。両方の解説に目を通し、便利と思われる方をご利用ください。

キーボードからデータを入力する場合はeditまたはinputを利用します。editはデータエディタを開き、inputはコマンドラインから直接入力します。

この章を読み終えたら、[D] **import**を参照してください。データ入力の他の方法についての解説があります。

❑ テクニカルノート

文字列の保存は、ソフトウェアで最も一般的な方法であるUTF-8で行われます。ほかのソフトウェアからデータをインポートする際、特別な措置が必要な場合はあまりありません。しかし、13もしくはそれ以前のStataで作成したデータセットを含め、ASCII文字以外のいわゆる拡張ASCII文字が含まれるファイルに対しては、文字列をUTF-8に変換する必要があります。この、拡張ASCII文字の変換を行わない場合、ただしく表示されません。この変換を行うコマンドとして、unicode translateが用意されています。詳細は[D] unicode translate、[U] 12.4.2 Unicodeの取り扱い、[D] unicodeをご覧ください。

❑

22.2 インポート方法の選択

データのインポート方法を決める場合の目安を次に示します。各コマンドの基本的な利用方法と、参照すべき *リファレンス*マニュアルの情報を示しますので参考にしてください。

1. データが少ししかなく、直接、キーボードから入力する場合は[D] editを参照してください。簡単に入力できます。また、[D] inputも参照してください。
2. データセットがバイナリ形式か、または、他のソフトウェアの内部形式である時は、次に示す方法をご検討ください。
   a. データがスプレッドシートに入っている場合は、そのままStataのデータエディタにコピー&ペーストします。詳細は[D] editを参照してください。
   b. Excel形式のデータの場合、import excelコマンドを利用します。詳細は[D] import excelを参照してください。
   c. データがSAS形式の場合は、import sasコマンドを利用します。詳細は[D] import sasを参照してください。
   d. データがSAS XPORT Version 5またはVersion 8形式の場合は、import sasxport5またはimport sasxport8コマンドを利用します。詳細は[D] import sasxport5、または[D] import sasxport8を参照してください。
   e. データがSPSS形式の場合は、import spssコマンドを使用します。詳細は[D] import spssを参照してください。
   f. 米連邦準備銀行経済データ（FRED）のデータベースからインポートを行いたい場合は、import fredを利用します。詳細は[D] import fredを参照してください。
   g. データがHaver Analyticsの.dat形式(Haver Analyticsが提供する経済、金融データセット)で、StataのWindows版を利用している場合は、[D] import haverコマンドを利用します。
   h. dBaseファイルの場合、import dbaseを利用します。詳細は[D] import dbaseを参照してください。
   i. 他のソフトウェアを利用してデータをテキスト形式に変換します。例えば、ソフトウェアには一般的にタブ区切りやカンマ区切りのテキスト形式でデータを保存する機能が用意されています。詳細は[D] import delimitedを参照してください。
   j. データがデータベースやスプレッドシートソフトでサポートされている、ODBC形式で保存されている場合は、odbc loadコマンドを利用します。詳細は[D] odbcを参照してください。
   k. データがデータベースにあり、データベースベンダーがJDBCドライバーを持っている場合は、jdbc loadコマンドを使用してデータをインポートできます。詳細は[D] jdbcを参照してください。
   l. shapefileをStataで使用したい場合、spshape2dtaを利用してStataが扱える形式へ変換します。詳細は[SP] spshape2dtaを参照してください。
   m. この他に、Stata以外のフォーマットのデータを、Stata形式に変換するソフトウェアもあります。詳細は[U] 21.4 変換プログラムを参照してください。
3. 一行につき1つのデータがあり、タブまたはカンマ区切りでデータが用意されている場合はimport delimitedコマンドを利用します。詳細は[D] import delimitedを参照してください。これはテキストデータを読み込む、最も簡単な方法です。
4. データセットにフォーマット情報が用意されており、その情報に対応してインポートする必要がある場合はディスクショナリによるinfileまたは、infixコマンドを利用します。詳細は[D] infile (fixed format)または[D] infile (free format)を参照してください。
5. 文字列変数が存在しない場合はディスクショナリ無しでinfileコマンドを利用します。詳細は[D] infile (free format)を参照してください。
6. データの文字列変数がすべて句読点で囲まれている場合は、ディスクショナリ無しでinfileコマンドを利用します。詳細は[D] infile (free format)を参照してください。
7. デリミタ無しの文字列変数で、ブレークを含まない場合、ディスクショナリ無しでinfileコマンドを利用します。詳細は[D] infile (free format)を参照してください。

8. データが８形式の場合、[D] infile (free format)を参照してください。
9. 上記に該当する対応策が無い場合は[D] infile (fixed format)または[D] infile (free format)を参照してください。

22.2.1 データの直接入力

データ量が少ない場合、直接入力を行うのも一つの方法です。詳細は[D] editまたは[D] inputをご覧ください。そうでない場合データはディスクに保存されているものとします。

22.2.2 データのコピーと貼り付け

他のプログラムに入っているデータをStataで分析する場合、そのデータがクリップボードにコピーできることを確認します。コピー可能であれば、Stataのデータエディタを開き、**編集** > **貼り付け**と操作してStataにデータを取り込みます。

22.2.2.1 ビデオ例題

Copy/paste data from Excel into Stata

22.2.3 データセットがバイナリフォーマットの場合

Stataはテキストフォーマットのデータに対応していますので、データをそのまま画面に表示したり、プリンタに出力することができます。しかし、バイナリ形式の場合、必ずしもStataで読み込めるとは限りません。バイナリ形式のデータというものは一般的なものですが、ソフトウェアパッケージごとに異なるフォーマットを有しています。例えば、.dtaデータセットもバイナリ形式のファイルですが、Stataでしか読むことはできません。一般的なExcelファイルの.xls形式や.xlsx形式のファイルはStataでも読み込むことができます。しかし、OpenOfficeソフトの.ods形式のファイルをStataで読むことはできません。

データセットがバイナリ形式や、Stataの対応していないソフトウェアパッケージの内部形式である場合、最初にそれらのファイルをテキストファイルに変換するか、または変換ソフトを使ってStata形式に変換します。データセットが.xlsや.xlsxのExcel形式の場合はStataのインポート機能を利用して直接取り込む事ができます。詳細は[D] import excelを参照してください。データセットがデータベースに格納されていたり、ODBC形式の場合は[U] 21.4 ODBC sourcesを参照してください。データがデータベースにあり、データベースベンダーがJDBCドライバーを持っている場合は、[U] 22.5 JDBC sourcesを参照してください。データセットがSAS形式の場合は、import sasコマンドを利用して取り込みます。データセットがSAS XPORT Version 5形式、またはSAS XPORT Version 8形式の場合はStataのimport sasxport5コマンド、またはimport sasxport8コマンドを利用して直接取り込む事ができます。詳細は[D] import sasxport5および[D] import sasxport8を参照してください。データが米連邦準備経済データ（FRED）のデータベースからオンラインで入手できる場合、Stataのimport fredコマンドを利用して読み込む事ができます。データセットがSPSS .sav形式の場合は、Stataのimport spssコマンドを利用して読み込むことができます。詳細は[D] import spssを参照してください。詳細は[D] import fredを参照してください。データセットがHaver Analytics形式の場合はStataのimport haverコマンドを利用して直接取り込む事ができます。詳細は[D] import haverを参照してください。データセットがdBase形式の場合はStataのimport dbaseコマンドを利用して読み込む事ができます。詳細は[D] import dbaseを参照してください。shapefile形式のファイルをStataで使用したい場合、spshape2dtaで変換する事ができます。詳細は[SP] spshape2dtaを参照してください。データセットがEBCDIC形式の場合はStataのinfileコマンドを利用して直接取り込む事ができます。詳細は[D] infile (fixed format)を参照してください。

データファイルがバイナリ形式であるかどうか、簡単な判別法をご紹介します。例えば、ユーザがマイクロソフトWordの形式でデータを保存し、共有することを考えていると仮定します。したがって、データをワード形式で保存します。このワードを利用しているユーザ間では、Wordデータがテキストデータの役割を果たすことになります。つまり、ワードを持っていれば、画面上にデータを表示できます。そして、ワードを使って印刷も可能です。この場合、データセットはテキスト形式ではなくワードの内部形式ですので、ワードが無ければ表示したり印刷することはできません。つまり、データの内容を確認するためには必ずワードが必要になり、Stataで利用するにはテキスト形式(.txt)形式で保存する必要があります。

データセットがバイナリであることか、否かを確認する場合は次のように操作します。簡単なコマンドを使って調べることができます。OSに関係なく、Stataを起動し、次のように入力します。

```
. type myfile.raw
結果が出力
```

ファイルの内容全てを表示する必要はありません。確認できたら*Break*キーを押します。

象形文字のようなものが画面に表示されましたか。もし、そうだとすれば、ファイルはバイナリ形式です。詳細は[U] 21.4 プログラムの変換を参照してください。

データがそのまま画面上で確認できたら、それはテキスト形式ということになります。

テキストデータを読み込む場面を想定してください。データの形式によってユーザが利用すべきコマンドは異なります。形式ごとに利用すべきコマンドについて、以下で解説します。

22.2.4 データが単純な構造の場合

データをインポートする最も簡単なコマンドはimport delimitedです。詳細は[D] **import delimited**を参照してください。

import delimitedは単純な機能しか持っていません。データセットを開き、内容を確認し、それから読み込みます。import delimitedは単純であるが故に制約もあります。つまり、データセットは1行につき、ひとつの観測値だけを持ち、それらがタブまたはカンマで区切られていることが前提になります。import delimitedコマンドは次のようなデータの読み込みに適しています。

```
------------------------------------------------------------------ begin data1.csv -------
M,Joe Smith,288,14
M,K Marx,238,12
F,Farber,211,7
------------------------------------------------------------------ end data1.csv ---------
```

または(一行目は変数名)、

```
------------------------------------------------------------------ begin data2.csv -------
sex, name, dept, division
M,Joe Smith,288,14
M,K Marx,238,12
F,Farber,211,7
------------------------------------------------------------------ end data2.csv ---------
```

または(データはタブ区切り)、

```
------------------------------------------------------------------ begin data3.txt -------
M       Joe Smith       288     14
M       K Marx  238     12
F       Farber  211     7
------------------------------------------------------------------ end data3.txt ---------
```

文字の長さとタブの関係でデータの並び方少しおかしいように見えますが、問題はありません。ここでもヘッダーを付けることができます。しかし、import delimitedは次のようなデータの読み込みには適しません。

```
------------------------------------------------------------------ begin data4.txt -------
M       Joe Smith       288     14
M       K Marx          238     12
F       Farber          211     7
------------------------------------------------------------------ end data4.txt ---------
```

このデータの区切りはタブではなく、スペースです。

data3.txtとdata4.txtの構造の違いをStataで確認します。次にように入力すると、タブの存在を明示的に表示します。

```
. type data3.txt, showtabs
M<T>Joe Smith<T>288<T>14
M<T>K Marx<T>238<T>12
F<T>Farber<T>211<T>7
.type data4.txt, showtabs
M       Joe Smith       288     14
M       K Marx          238     12
F       Farber          211     7
```

22.2.5 形式情報を読み取る場合

データがある形式情報によってフォーマットされている場合、[D] infile (fixed format)や[D] infix (fixed format)を利用すると便利な場合があります。

infixコマンドとinfileコマンドはデータ定義(ディクショナリ)というものを利用しますが、それはいわば最後の手段です。

このセクションで説明する内容は、どうしてもインポートがうまく行かない場合に参照していただく内容です。必要性がなければ、読み飛ばしてください。手元のデータがある形式情報にしたがって構成されているからと言って、ここで紹介する方法を使うことなになる、という具合には理解しないでください。

データに形式情報があるといっても、それが、形式情報を理解しなければいけない、という事にはなりません。次のデータには形式情報が付属しています。

```
begin data5.raw
1    27.39     12
2     1.00      4
3   100.10    100
end data5.raw
```

各データは列ごとに整列していますが、そのように定めている形式情報を調べる必要性はありません。しかし、次のようになっている場合はどうでしょうか。

```
begin data6.raw
1 27.39 12
2  1.00  4
3100.10100
end data6.raw
```

このデータセットにも形式情報が付属しています。ここでは“3100.10100”の意味を正しく判断するために形式情報をユーザが理解する必要があります。2番目の変数は第4列から始まり、6文字で構成されることが分からなければ、100.10が本来のデータであることが把握できません。data6.rawのようなデータの場合、データの開始位置と終了位置を正しく把握していなければ、誤った形でデータをインポートしてしまいます。したがって、data6.rawのデータを読み込む場合は、infixまたはinfileコマンドを利用しなｋればなりません。

むしろ、形式情報の無いデータの読み込みは簡単です。形式情報が付属している場合は、その形式をStataに教える必要があります。ユーザに負担のかかる作業ですが、Stataで必要な設定を行ってください。コマンドは[D] infile (fixed format)または[D] infix (fixed format)を参照してください。

data4.rawに戻ります。

```
begin data4.raw
M       Joe Smith       288     14
M       K Marx          238     12
F       Farber          211     7
end data4.raw
```

一見しただけでは、データ定義を利用すべきか否か、判断することはできません。ハッキリと分からない場合は、使わないようにしましょう。

最後に、形式情報の無い、典型的なデータの例をご紹介します。

```
begin data7.raw
1 27.39           12
2 1 4
3 100.1 100
end data7.raw
```

ここでは複数の空白を利用して、一目でデータが識別できるようにしてあります。複数の空白を利用したことに深い意味はありません。

次のセクションではデータ定義を利用せずに、形式情報のあるデータと、形式情報の無いデータをインポートする方法について解説します。

22.2.6 文字列変数が無い場合

文字列変数がデータに含まれない場合は[D] infile (free format)を参照してください。

data7.rawに形式情報はありませんが、定義なしの形でinfileコマンドを使ってインポートすることができます。しかし、data4.rawの場合、文字列が含まれていますので、同じ方法は利用できません。

❑ テクニカルノート

先にデータ定義の利用に関する目安を述べましたが、ユーザによっては、それに関係なく、データ定義を利用するケースもあります。つまり、エディタを利用して変数リストや、変数ラベルを作成して、データディスクショナリを活用することに慣れているからこそ、可能であるとご理解ください。しかも、そのようなユーザならば、doファイルの中でinfileコマンドを利用してデータ定義を効率的に使いこなすことができます。データ定義を利用することのデメリットというものは、彼らには一切ありません。

しかし、無理をしてデータ定義を利用する必要はありません。あくまで、必要に応じて利用すれば十分です。データセットに形式情報がなく、文字列を含まない場合は、データディスクショナリを利用せずに簡単にインポートできます。もちろん、データディスクショナリを敢えて利用してインポートイすることもできます。

データ定義無しでinfileを利用することはストリームI/Oを、データ定義を利用する場合はレコードI/Oを使うことになります。両者に機能的な違いはもちろんありますが、それぞれ、データをインポートすることができます。データセットによって、ストリームI/Oとレコード I/Oを使い分けます。また、どちらの方法でもインポートできるものもあります。ここで説明した5つのパターンはそれぞれストリームI/Oを利用するものですが、どれを用いてもインポートできます。

❑

次は少なくとも一つの文字列変数を含むデータをインポートする方法を紹介します。

22.2.7 全ての文字列が句読点で囲まれている場合

データの文字列変数がすべて句読点(シングルまたはダブルクォーテーション)で囲まれている場合は、[D] infile (free format)を参照してください。

文字列変数の正式な定義は[U] 24 文字列の操作を参照してください。しかし、簡単に言えば、 “bob” や “joe” のようなものであり、数値変数の 1, 27.5, や -17.393というものとは明らかに異なります。区切り記号のない文字列は読み込むことができません。

デリミタを利用した文字列変数の例を次に示します。

```
begin data8.raw
"M" "Joe Smith" 288 14
"M" "K Marx" 238 12
"F" "Farber" 211 7
```

```
end data8.raw
```

または

```
begin data8.raw, alternative format
    "M" "Joe Smith" 288  14
    "M" "K Marx"    238  12
    "F" "Farber"    211   7
end data8.raw, alternative format
```

これらは両方ともdata4.rawのデータを句読点で囲ったり、区切りを小さくしたものです。定義無しのinfileコマンドを使ってインポートすることができます。

次にデリミタや形式情報を用いないdata4.rawの例を次に示します。

```
begin data9.raw
    M Joe Smith 288 14
    M K Marx 238 12
    F Farber 211 7
end data9.raw
```

このデータのインポートには少し問題があります。文字列を区切っているスペースは、あくまでスペースでしかなく、前後の関係を示すものではありません。つまり、Farbarの前にあるFはイニシャルなのか、それとも性別を示す情報なのか、それを判別することができません。

しかし、このようなケースは極めて稀な例ですから、あまり深く考える必要はありません。基本的には文字列データをdata8.rawのように句読点で囲むか、または、data4.rawのように明示的に分ければ、このような問題に悩まされることはありません。

22.2.8 デリミタの無い文字列に空白が無い場合

データが列に分かれていなかったり、デリミタを利用していなくても、文字列中に空白が存在しなければ問題はありません。例えば、ラストネームしかないようなデータを想定します。

```
begin data10.raw
    Smith 288 14
    Marx 238 12
    Farber 211 7
end data10.raw
```

この場合、データ定義なしでもデータをインポートできます。注意: Van Owenやvon Beethovenのようにラストネームに空白がある場合は注意してください。

デリミタの無い文字列で、空白が無い場合は[D] **infile (free format)**を参照してください。

22.2.9 EBCDICデータの場合

メインフレームで利用されているEBCDICコードのデータを処理するケースについて説明します。EBCDICコードはIBMのメインフレーム用OSで採用されているコードです。

EBCDICコードデータの場合、データを構成する各フィールドの開始と終了の位置に関する情報が必要です。そしてフィールドに含まれるデータの種類も把握する必要があります。EBCDICコードのデータも固定型形式情報を持ったテキストデータの読み込みと同じく、infileコマンドで対応できます。詳細は[D] infile (fixed format)を参照してください。データ定義フィールドに対する列情報を記述しinfileコマンドのebcdicオプションを利用します。

もしくは、filefilterコマンドを使って、EBCDICファイルをASCIIのテキストファイルに変換します。詳細は[D] filefilterをご覧ください。

22.2.10 まだ解決方法が見つからない場合

上記に該当する対応策が無い場合は[D] infile (fixed format)または[D] infix (fixed format)を参照してください。data4.rawを今一度、ご覧ください。

```
──────────────────────────────────────────────────────────────────── begin data4.raw ─────
    M       Joe Smith       288     14
    M       K Marx          238     12
    F       Farber          211     7
──────────────────────────────────────────────────────────────────── end data4.raw ───────
```

これはディスクショナリを利用したinfileまたはinfixコマンドでインポートできます。

22.3 メモリーが不足している場合

メモリーが不足してしまう場合は設定を調整する必要があります。詳細は[U] 6 メモリの管理と[D] memoryを参照してください。逆にメモリーを節約するアプローチもあります。

データをインポートする際に変数の種類を指定することもできます。Stataは整数値をフローティングタイプよりも、よりコンパクトに保存できます。また、小さな整数は、大きな整数よりも、よりコンパクトに保存します。詳細は[U] 12 データを参照してください。

上記のような対応策を利用してもデータ全体を読み込めない場合、データを部分的に取り込みます。infileとinfixはin rangeというモディファイヤを利用してデータの範囲を指定します。もちろん、rangeは読み込むデータの範囲のことです。infile ... in 1/100はデータの1行目から100行目までのデータだけをインポートします。

infile ... in 101/200の場合は101行目から200行目までのデータだけをインポートします。範囲の終点は、実際のデータ数よりも大きな値を設定することもできます。実際のデータは150行分しななくても、infile ... in 101/200とすることで、101行目から150行目までのデータを取り込むことができます。

この他に、if expモディファイヤを使ってデータの範囲を指定することもできます。例えば、男女同数のデータが含まれているとし、男性は0, 女性は1とコード化されているものとします。これを行うには、次のように入力します。infile ... if sex==0コマンドはsexの値がゼロであるデータを読み込み、1のデータは読み飛ばします。したがって、女性だけのデータを読み込む場合はinfile ... if sex==1とします。

データが大きすぎるので、データのランダムサンプリングを行う場面を想像してください。データ全体でなく、一部の標本だけを分析します。infileとinfixコマンドではif expが利用できるので、infile ... if runiform()<.1と入力します。runiform()は一様分布の乱数発生を行う関数です。詳細は[D] functionsを参照してください。このようにするると、データの約10%を標本としてインポートします。乱数機能runiform()を利用する場合は乱数シードの利用について考慮してください。詳細は[R] set seedを参照してください。

最後のアプローチは、いくつかの変数についてのみ、前データを読み込むというものです。データディスクショナリ無しでデータを読み込む場合、読み飛ばす変数は_skipで指定します。データ定義、または、infixコマンドを利用する場合、読み込む列を指定します。指定したもの以外のデータは読み飛ばします。

import excelコマンドを利用する場合、cellrange()オプションでExcelワークシートの一部だけを読み込むこともできます。詳細は[D] **import excel**をご覧ください。

22.4 ODBCソース

目的のデータセットがネットワーク上のデータベースや、共有状態にあるスプレッドシートに保存されている場合、ODBC経由でインポートできます。ODBC(Open Database Connectivity)は標準的なデータ交換プログラムです。StataにODBC経由でデータをインポートする場合はodbcコマンドを利用します。この方法でデータをインポートすできます。

もちろん、データソースとしてのデータベースがネットワーク上に存在しなければなりません。データベースからデータをインポートするodbcコマンドを利用する前提条件として、Stataを起動するPCと同一PC上にODBCソースを用意する必要があります。データベース本体ではなく、ODBCソースとしてのデータベースの定義を同一PCに用意しなければなりません。Windows PCの場合、ODBCソースはコントロールパネルの「データソース」で設定します。そして、Stataでodbc listコマンドを実行すると、すべてのODBCソースを画面表示します。

データベースが正常に稼働し、データソースをStataと同じPCに用意してあれば、odbc loadコマンドで目的のデータをインポートできます。このインポート処理の詳細は[D] **odbc**を参照してください。

22.5 JDBCソース

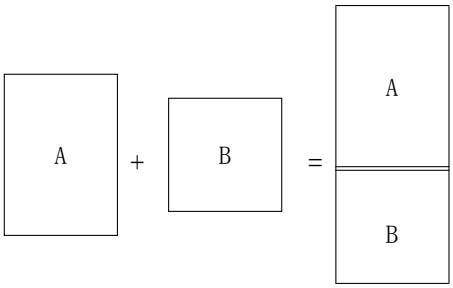
データセットがデータベースにある場合は、JDBCを介してデータをインポートできる場合があります。Java Database Connectivity（JDBC）は、プログラム間でデータを交換するための標準です。Stataは、長方形のデータを持つリレーショナルデータベースまたは非リレーショナルデータベース管理システムからデータをインポートするためのJDBC標準をサポートしています。

jdbcコマンドを使用してデータベースからデータをインポートするには、データベースベンダーがダウンロードしてインストールするためのJDBCドライバーを提供する必要があります。データベースが機能していて、ドライバーがStataによって検出される場合、データをインポートするために必要なのは、jdbc loadを使用した1回の呼び出しだけです。 このプロセスの詳細については[D] **jdbc**を参照してください。

23 データセットを組み合わせる

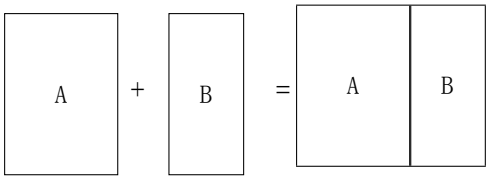
2つのデータセットを一つに組み合わせるような場面を想定してください。ここでは一つのデータセットをボックスで表現します。変数は隣り合って並び、データは縦方向に並んでいるものとします。

データセットを縦方向に結合する場合は [D] appendコマンドを参照してください。



appendコマンドは既存の変数にデータを追加します。用法はとても簡単です。appendコマンドは単にデータセットをつなぎ合わせるだけなので、その個数は問いません。例えば、患者のデータに、新たな患者のデータを追加するような場合に利用します。

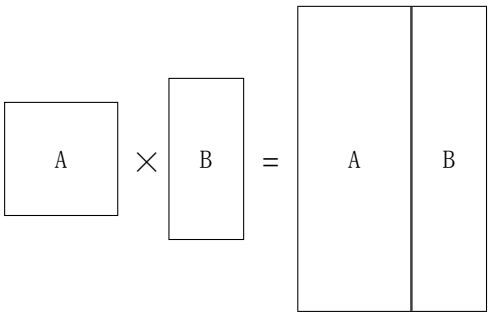
データセットを横方向に結合する場合は[D] mergeコマンドを参照してください。



mergeコマンドは既存のデータに変数を追加します。用法はとても簡単です。appendコマンドは単にデータセットを隣に並べるだけなので、その個数は問いません。例えば、既存のサーベイデータに、追加した質問の回答を追加するような場合に利用します。

これ以外にも、データを推定に統合する方法、正確にはデータを順序立てて別々のフレームにロードする方法があります。mergeではなくfrlinkを使用する場合の詳細は[D] frlinkをご覧ください。

同じようにデータセットを隣同士に並べる場合に[D] joinbyを利用します。グループに含まれる全ての組み合わせを自動作成します。



ここではjoinbyも利用できます。もちろん、全ての組み合わせデータがそれぞれの意味を持つ場合に限ります。例えば、Aに患者数が、Bにはその子供のデータが入っているような場合に相当します。例えば、joinby familyidコマンドは子供の親と子供の情報が一緒になってデータセットを作成します。また、2つのデータセットごとに組み合わせを作成するコマンドとして、利用頻度は少ないと思われますが、[D] crossというコマンドもあります。

Stataにおけるデータセットの結合はMitchell (2010, chap. 6)を参照してください。

23.1 参考文献

Golbe, D. L. 2010. Stata tip 83: Merging multilingual datasets. *Stata Journal* 10: 152-156.

Gould, W. W. 2011a. Merging data, part 1: Merges gone bad. The Stata Blog:Not Elsewhere Classified. http://blog.stata.com/2011/04/18/merging-data-part-1-merges-gone-bad/.

——. 2011b. Merging data, part 2: Multiple-key merges. The Stata Blog: Not Elsewhere Classified. http://blog.stata.com/2011/05/27/merging-data-part-2-multiple-key-merges/.

Mitchell, M. N. 2010. *Data Management Using Stata:A Practical Handbook*. College Station, TX:Stata Press.

24 文字列の操作

目次

先に[U] 12 データを一読してください。

24.1 説明

文字列とは英数字による、1セットの情報のことを指します。例えば、“Male”、“Female”、“yes”、 “no”、“R. Smith” and “P. Jones” のような情報の種類を*文字列*と呼びます。したがって、文字列の反対語は0, 1, 2, 5.7などの数値になります。この文字列で構成される変数を処理する事が、データ分析において必要になる事があります。文字列を利用しなければならない、現実的な4つのケースをここで紹介します。

文字列変数を構成する値が、なんらかの質的な情報を示す場合に利用します。例えば従業員名簿、病院における患者リスト、住所データにおける都市名などがこれに相当します。文字列はまさにこのような情報を記録するために利用します。

カテゴリの情報を示す場合にも文字列を利用します。“Male”と“Female”、“Yes” と “No” はまさに、その例です。しかし、文字列変数の用法としては不適切です。なぜなら、同じ情報であれば数値コードで表現でき、データの保存容量は小さくすることができます。また、データも効率的に処理できます。カテゴリを示す文字列から、カテゴリを示す数値を作成する方法は後述します。

また、単純な入力ミスによって文字列を入力してしまうことがあります。例えば、変数に1,5,8.2のような情報が入っている場合、データ読み込みの際のエラーにより、これらの数値を文字列として認識してしまうことがあります。この種の問題の解決方法は後述します。

最後に数値情報を、そのまま入力できないケースを紹介します。例えば、“15 Jan 1992”, “1/15/92”、そして“1A73”のような情報は数値としては入力できません。複雑な問題ですが、対応方法は後述します。

文字列にUnicodeが含まれる場合には、上記の情報と共に[U] 12.4.2 Unicode文字列の取り扱いをご覧ください。

24.2 カテゴリカルな文字列変数

カテゴリの情報を示す場合、変数には文字列が入ります。

例えば、性別を示す変数sexに文字列の“male”と“female”が入っているものとします。しかし、そのまま回帰を行うと、次のようなメッセージを表示します。

```
. use https://www.stata-press.com/data/r18/hbp2

. regress hbp sex
no observations
r(2000);
```

regressを始めとする、Stataの殆どの分析に関するコマンドは文字列に対応していませんので、数値データが無い、という意味のメッセージを表示します。上記のコマンドの場合は必要なデータは数値データです。数値データ以外のものは欠損値として認識します。しかし、次のコマンドを実行すると、推定結果を得るとができます。

```
. encode sex, generate(gender)
. regress hbp gender
```

Source	SS	df	MS	Number of obs =	1,128
				F(1, 1126) =	14.04
Model	.64448568	1	.644485682	Prob > F =	0.0002
Residual	51.673776	1,126	.045891454	R-squared =	0.0123
				Adj R-squared =	0.0114
Total	52.318262	1,127	.046422593	Root MSE =	.21422

hbp	Coefficient	Std. err.	t	P>|t|	[95% conf.	interval]
gender	.0491501	.0131155	3.75	0.000	.0234166	.0748837
_cons	-.0306744	.0221353	-1.39	0.166	-.0741054	.0127566

文字列変数sexをgenderという数値変数に、encodeコマンドを利用して変換しました。詳細は[U] 12.6.3 Value labelsと[D] encodeを参照してください。

24.3 偶然のエラーによる文字列

単純な入力ミスによって文字列を入力してしまうことがあります。

例えば、変数xに数値データが入っているものとします。しかし、データをインポートする時に、何らかの原因により文字列として読み込んでしまったとお考えください。listコマンドは正常に機能します。

```
. list x
```

	x
1.	2
2.	2.5
3.	17

(*省略*)

しかし、xの要約統計量を求めてみると、

```
. summarize x
```

Variable	Obs	Mean	Std.dev.	Min	Ma
x	0				

このようになってしまったら、念のためdescribeコマンドを実行します。

```
. describe

Contains data
Observations:         6
   Variables:         3

              Storage t    Display format      Value labe
Variable name     ype                          l              Variable label
------------------------------------------------------------------------------
x                 str4 fl  %9s
y                 oat flo  %9.0g
z                 at       %9.0g
------------------------------------------------------------------------------
Sorted by:
     Note: Dataset has chaned since last saved
```

xのフォーマットはstr4となっています。

したがって、summarizeコマンドが文字列の要約統計量を計算できなかったことになります。つまり、“Joe” + “Bill” + “Roger”などはそもそも計算できません。もちろん、文字属性の数値が入っている場合でも同じです。このようなケースではdestringコマンド([D] **destring**)を利用して、変数の属性を文字列から数値に変換します。

```
. destring x, replace
X: all characters numeric; replaced as double

. summarize x
```

Variable	Obs	Mean	Std. Dev.	Min	Max
x	6	13.08333	8..452317	2	20

destringコマンドの代わりに、generateコマンドとreal()オプションを利用する方法もあります。詳細は[FN] **String functions**を参照してください。

24.4 複雑な文字列

数値情報であっても、そのままでは入力できないケースを紹介します。

複数の情報を変数に含むような場合に、このような事になります。具体的に言えば、文字列が長くなり、また、そこに数値情報も含むような変数を利用する場合です。Stataの文字列は最大、20億の文字列に対応しており、バイナリ情報を保存することもできます。詳細は[U] **12.4 文字列**を参照してください。複雑な文字列の代表的なものに日付情報があります。“15 Jan 1992”には日、月、年の3つの情報が含まれています。日付情報を操作する場合は[U] **25 日付と時間の操作**を参照してください。

Stataには日付情報を処理するためのコマンドが用意されていますので、ここでは、それ以外の、やや複雑な情報を処理する例を示します。部品番号のデータが手元にあるものとします。

```
.list partno
```

	partno
1.	5A2713
2.	2B1311
3.	8D2712
	(*省略*)

この番号の最初の1ケタは部門番号、2ケタ目は製造工場番号です。次の3ケタが部品番号で、最後の1ケタが色を示すものとします。このようなケースではsubstr()コマンドとreal()を利用して情報を分割します。詳細は[FN] **String functions**を参照してください。

```
. generate byte div = real(substr(partno,1,1))
. generate str1 plant = substr(partno,2,1)
. generate int part = real(substr(partno,3,3))
. generate byte color = real(substr(partno,6,1))
```

文字列の一部を取りだす場合はsubstr()関数を、そして文字列を数値に変換する場合はreal()を利用します。詳細は[U] **12.4.2.1 Unicode文字列関数**をご覧ください。

文字列と数値の操作や、属性の変換に関する詳細はCox (2002)を参照してください。

24.5 参考文献

Cox, N. J. 2002. Speaking Stata:On numbers and strings. *Stata Journal* 2: 314-329.

Cox, N. J., and C. B. Schechter. 2018. Speaking Stata: Seven steps for vexatious string variables. *Stata Jourmnal* 19: 129-142.

25 日付と時間の設定

目次

25.1 概要

Stataの日付と時間に関する機能(関数や表示形式なども含む)の情報は[D] **datetime**を参照してください。

Stataでは21nov2006などの日付情報、13:42:02.213のような時間情報、そして両者を組み合わせた21nov2006 13:42:02.213のような情報を処理できます。もちろん、これらの情報を11/21/2006, November 21, 2006, そして1:42 p.mのような形式で入力することも可能です。
Stataは日付と時刻、そしてその組み合わせを整数で、4,102, 0, 82, 4,227や1,479,735,745,213のように表現します。話を少し整理してみましょう。

1. 日付と時刻を示す変数のことをStataでは日付時刻変数と呼びます。この変数は、21nov2006、11/21/2006、November 21, 2006、13:42:02.213、1:42 p.mのような表示形式をサポートしています。 日付と時間の情報を正確に示すように変数のケタ数は自動設定されます。
2. 以下で説明する関数を利用すると、元の文字列をStataの形式に変換し、変数として保存できます。
3. 変換された日付時間を表示する際は、Stataの内部形式ではなく、直感的に理解可能な 21nov2006、11/21/2006、November 21, 2006、13:42:02.213、1:42 p.m.のような形式で画面表示できます。

Stataは日付時刻を1960年1月1日のミリ秒単位(00:00:00.000)をゼロとします。つまり、1960年1月1日のミリ秒単位の開始値を0とします。

この場合、整数の1は01jan1960 00:00:00.001を意味します。

整数の-1は31dec1959 23:59:59.999を意味します。

この流れで考えますと、21nov2006 13:42:02.213の整数値は1,479,735,722,213になります。閏秒を考慮しない場合、少なくともこの値以上の整数になります。閏秒を考慮する場合、21nov2006 13:42:02.213は23秒遅れて1,479,735,745,213となります。Stataはどちらの形式にも対応しています。

日付時刻におけるミリ秒単位の情報を関数を使えば、簡単に変換できます。例えば、21nov2006 13:42:02.213のような場合、1,479,735,722,213または1,479,735,745,213と変換できます。

ミリ秒単位の情報であっても、あくまで01jan1960のミリ秒単位をゼロとして、整数でカウントする、ということだけご理解ください。

Stataは時刻部分についてのみ、別にカウントすることができます。日付時刻変数は1960年1月1日のミリ秒単位から整数をカウントアップしていきますが、その時の時刻部分だけを、毎日の開始時点から別個にカウントすることもできます。例えは、毎日午後2時にヒューストンからロンドンに向けて出発する航空機について考えてみましょう。午後2時は深夜0時(00:00:00.000)を始点とすると、ミリ秒単で50,400,000となります。

わざわざこのようにStataが処理する理由は日付と時刻の加算を考慮してのことです。航空機が21nov2006に出発するとして、その時の日付時間値はいくつでしょうか。2006年11月21日の00:00:00.000は閏秒を無視した場合、1,479,686,400,000ミリ秒です。よって、日付時間変数の値は1,479,686,400,000 + 50,400,000、つまり、1,479,736,800,000になります。

引き算の場合も同様に考えれば済みます。21jan1952 7:23 a.m.と21nov2006 3:14 p.mの時間差はどれほどでしょうか。
答え: (1,479,741,240,000 − (−250,706,220,000))/3,600,000 =
480,679.85 時間となります。

始点を1960年1月1日のミリ秒単として、閏秒を無視する場合の日付時刻変数のフォーマットは%tcです。

同じく始点を1960年1月1日のミリ秒単として、閏秒を考慮する場合の日付時刻変数のフォーマットは%tCです。

Stataには全部で9種類の%tフォーマットがあります。

一般的にはカレンダーの日付情報があれば十分です。例えば、従業員を2006年1月15日に採用したとします。このようなケースでは%tcを利用して日付を保存することもできますが、日数だけを管理するのであれば%tdの方が便利です。%tdの場合、1960年1月1日がゼロになります。そして1単位はミリ秒ではなく、1日となります。したがって、1は1960年1月2日となります。逆に−1は1959年12月31日となります。%tdの日付時刻変数の引き算は単純に経過日数を示すことになります。

金融データの場合は、%tqも便利かもしれません。%tqの場合、0は1960年の第一四半期、1は同年の第ニ四半期、−1は1959年の第四四半期になります。%tqの日付時刻変数の引き算は四半期単位の経過日数を示すことになります。

Stataの9種類の%tフォーマットを次に示します。

フォーマット	日付時間	単位	注釈
%tc	01jan1960	ミリ秒	閏秒を無視
%tC	01jan1960	ミリ秒	閏秒を考慮
%td	01jan1960	日	カレンダーの日付フォーマット
%tw	1960-w1	週	第52週を7日以上とする
%tm	jan1960	月	カレンダーの月次フォーマット
%tq	1960-q1	四半期	四半期フォーマット
%th	1960-h1	半年	半年=2四半期単位
%tv	0 A.D	年	1960は1960年
%tb	-	日	ユーザ定義

%tyと%tbを除く全てのフォーマットは1960年の1月を始点とします。0が意味するものは、フォーマットによって、ミリ秒、日、週、月、四半期、半年など最初の期を指します。したがって、1が意味するものは、フォーマットによって、ミリ秒、日、週、月、四半期、半年など2番目の期を指します。同様に、−1が意味するものは、フォーマットによって、ミリ秒、日、週、月、四半期、半年などの始期の1期前の時点を指します。

Stataの%tyフォーマットは西暦を数値情報として記録します。つまり、1960年を0とするのではなく、1960はあくまで1960, 2006もどうように2006と認識します。

25.2 日付と時刻の入力

日付と時刻は文字列として入力します。そして文字列を以下に示す%tのフォーマットで変換します。

フォーマット	文字列-数値変換関数
%tc	clock(string, mask)
%tC	Clock(string, mask)
%td	date(string, mask)
%tw	weekly(string, mask)
%tm	monthly(string, mask)
%tq	quarterly(string, mask)
%th	halfyearly(string, mask)
%ty	yearly(string, mask)

これらの関数の詳細は[D] Datetime translationを参照してください。

上の表においてstringは変換する文字列変数、maskはstringにおける日付時刻の並び方を明示的に示す情報です。例えば、%td関数date()におけるmaskは、M,D,Yを利用します。

date(string, "DMY")はstringの中に日付時刻の情報が日、月、年の順番で入っていることを示します。例えば、21nov2006、21 November 2006, 21-11-2006, 21112006などの文字列に対してdate()関数を利用できます。

一方、日付時刻の情報が日、月、年の順番で入っている場合にはdate(string, "DMY")を利用します。例えば、November 21, 2006, 11/21/2006, 11212006などの文字列に対してdate()関数を利用できます。

西暦が下2ケタ表示の場合は上2ケタに省略されている情報を入力します。例えば、月、日、年の順番で並んでいるデータがあるとします。ただし、年には下2桁の情報だけが入っているとお考えください。つまり、19が省略された形です。この場合はdate(string, "MD19Y")という形で関数を利用します。このように関数を利用すると、date()はNovember 21, 2006, 11/21/2006, 11212006だけでなく、Feb. 15 '98, 2/15/98, 21598という情報も変換できます。(この他にも2桁表示の情報を98ならは1998、06ならば2006のように変換する方法があります。この方法を利用する場合はオプションとして3番目の引数を利用します。詳細は[D] Datetime translation の*Working with two-digit years*をご参照ください。

次に%tdの利用例を示します。例えば、次のようなデータについて考えてみましょう。

begin bdays.raw

```
Bill  21 Jan 1952  22
May   11 Jul 1948  18
Sam   12 Nov 1960  25
Kay    9 Aug 1975  16
```

end bdays.raw

次のコマンドを実行してデータを読み込みます。

```
. infix str name 1-5  str bday 7-17  x 20-21 using  bdays
(4 observations read)
```

日付情報はスペースで区切られていますが、一つの変数として読み込みます。変数bdayに日付情報が入ります。

```
. list
```

	name	bday	x
1.	Bill	21 Jan 1952	22
2.	May	11 Jul 1948	18
3.	Sam	12 Nov 1960	25
4.	Kay	9 Aug 1975	16

データは問題なく取り込まれているように見えますが、変数bdayがこのままでは不十分です。変数bdayは日付のように表示されていますが、これは文字列にすぎません。bdayをStataが理解できるように%t変数に変換する必要があります。

```
. generate birthday = date(bday, "DMY")
. list
```

	name	bday	x	birthday
1.	Bill	21 Jan 1952	22	-2902
2.	May	11 Jul 1948	18	-4191
3.	Sam	12 Nov 1960	25	316
4.	Kay	9 Aug 1975	16	5699

新しい変数birthdayは%td変数です。しかし問題があります。新しい変数はStataには理解できますが、分析者が直感的に理解できません。次に%td変数に%tdフォーマットを設定します。

```
. format birthday %td
. list
```

	name	bday	x	birthday
1.	Bill	21 Jan 1952	22	21jan1952
2.	May	11 Jul 1948	18	11jul1948
3.	Sam	12 Nov 1960	25	12nov1960
4.	Kay	9 Aug 1975	16	09aug1975

新しい%td変数を利用して、2000年1月1日時点での年齢を表す変数を次のように作成します。

```
. generate age2000 = (td(1jan2000)-birthday)/365.25
. list
```

	name	bday	x	birthday	age2000
1.	Bill	21 Jan 1952	22	21jan1952	47.94524
2.	May	11 Jul 1948	18	11jul1948	51.47433
3.	Sam	12 Nov 1960	25	12nov1960	39.13484
4.	Kay	9 Aug 1975	16	09aug1975	24.39699

td()は%td形式の日付を作成する関数です。%tのフォーマットに対応して、tc(), tC(), tw(), tm(), tq(), th()などの関数が用意されています。詳細は[D] **datetime**を参照してください。

例題を用いて詳しく見て行きましょう。例えば、次のようなデータがあるものとします。

```
.use https://www.stata-press.com/data/r18/datexmpl2
.list
```

	id	timestamp	action
1.	1001	Tue Nov 14 08:59:43 CST 2006	15
2.	1002	Wed Nov 15 07:36:49 CST 2006	15
3.	1003	Wed Nov 15 09:21:07 CST 2006	11
4.	1002	Wed Nov 15 14:57:36 CST 2006	16
5.	1005	Thu Nov 16 08:22:53 CST 2006	12
6.	1001	Thu Nov 16 08:36:44 CST 2006	16

変数timestampは文字列ですが、これを%tcフォーマットに変換します。上記の表のデータについてclock()関数を利用します。clock()のマスクとしてD, M, Y, および h, m, sを利用します。曜日、月、年、時、分、秒の順番を仮定します。timestampには、これら以外の情報も入っていますので、直接clock()で変換することはできません。そこで、最初にclockで解読可能な形に変換します。

```
. generate str ts = substr(timestamp, 5, 15) + " " + substr(timestamp, 25, 4)
. list ts
```

	ts
1.	Nov 14 08:59:43 2006
2.	Nov 15 07:36:49 2006
3.	Nov 15 09:21:07 2006
4.	Nov 15 14:57:36 2006
5.	Nov 16 08:36:44 2006

新しい変数tsはclock(timestamp, "MD hms Y")を利用して変換できます。"MD hms Y"はtsの要素が月、日、時、分、秒、年の形である事を示しています。スペースに特別な意味はありません。clock(timestamp, "MDhmsY")としても同じです。スペースは単純に見やすいように、形を整える目的で利用します。

%tcは長い情報(文字列)を利用しますので、次に示すように、clock()関数を利用する場合は、double(倍精度)で保存します。

```
.generate double dt = clock(timestamp, "MD hms Y")
.list id dt action
```

	id	dt	action
1.	1001	1.479e+12	15
2.	1002	1.479e+12	15
3.	1003	1.479e+12	11
4.	1002	1.479e+12	16
5.	1005	1.479e+12	12
6.	1001	1.479e+12	16

見た目がかなり変わってしまいました。新しい変数dtには大変、大きな数値変数が入りましたが、データ精度を倍精度にしておきましたので、問題は生じません。デフォルトの表示形

式%tcだと、データの違いはよく分かりません。数値をもう少し詳しく表示させる場合は、%20.0gcなどのフォーマットを利用します。その形式だと、最初の値は1,479,113,983,000、二番目の値は1,479,195,409,000であることが分かります。しかし、ここでは大きな表示桁数は利用しません。代わりに%tcフォーマットで%tc変数を表示します。

```
. format dt %tc
. list id dt action
```

	id	dt	action
1.	1001	14nov2006 08:59:43	15
2.	1002	15nov2006 07:36:49	15
3.	1003	15nov2006 09:21:07	11
4.	1002	15nov2006 14:57:36	16
5.	1005	16nov2006 08:22:53	12
6.	1001	16nov2006 08:36:44	16

変数dtの内容をハッキリと表示できます。データ間の間隔を調べるには次のようにします。

```
. sort dt
. generate hours = hours(dt - dt[_n-1])
(1 missing value generated)
. format hours %9.2f
. list id dt action hours
```

	id	dt	action	hours
1.	1001	14nov2006 08:59:43	15	.
2.	1002	15nov2006 07:36:49	15	22.62
3.	1003	15nov2006 09:21:07	11	1.74
4.	1002	15nov2006 14:57:36	16	5.61
5.	1005	16nov2006 08:22:53	12	17.42
6.	1001	16nov2006 08:36:44	16	0.23

ここではdt間の引き算を行うと、その他いはミリ秒単位になります。その値を時単位に変換するのは簡単です。つまり、 60 × 60 × 1,000 = 3,600,000で割れば済みます。ただ、これもタイプミスをしてしまう可能性がありますので、Stataにはhours()関数が用意されています。この関数はミリ秒単位を時単位に変換します。hours()などの関数に関する詳細は[D] Datetimeを参照してください。

25.3 日付と時刻の表示

%td変数には%tdフォーマットを、%tc変数には%tcフォーマットという具合に対応させます。実際のコマンドは format varname %td, format varname %tcという具合になります。

%tc, %tC, %td, %tw, %tm, %tq, %th, %tyなどのフォーマットは一括して%tフォーマットと呼びます。これらのフォーマットコマンドに続けて、具体的な表示形式を入力します。

先の例で言えば、文字列変数timestampの中身は文字列の“Tue Nov 14 08:59:43 CST 2006”でした。元の変数から%tc変数を作成し、デフォルトの表示形式を設定しました。
%tcフォーマットはデフォルトで“14nov2006 08:59:43”と表示します。次に示すように%tcの後ろに具体的なフォーマットを記述することで、オリジナルの情報と同じ形式で変数を表示できます。

```
. format dt %tcDay_Mon_DD_HH:MM:SS_!C!S!T_CCYY
. list id dt action hours
```

	id	dt	action	hours
1.	1001	Tue Nov 14 08:59:43 CST 2006	15	.
2.	1002	Wed Nov 15 07:36:49 CST 2006	15	22.62
3.	1003	Wed Nov 15 09:21:07 CST 2006	11	1.74
4.	1002	Wed Nov 15 14:57:36 CST 2006	16	5.61
5.	1005	Thu Nov 16 08:22:53 CST 2006	12	17.42
6.	1001	Thu Nov 16 08:36:44 CST 2006	16	0.23

%tの表示形式に関する詳細は[D] **Datetime display formats**を参照してください。

25.4 日付と時刻の直接入力

場合によっては日付や時刻の情報を直接入力する場合があります。例えば、2000年1月1日現在の年齢を調べる場合は、次のようにします。

```
. generate age2000 = (td(1jan2000)-birthday)/365.25
```

次のコマンドも利用できます。

```
. generate age2000 = (14610-birthday)/365.25
```

14,460は2000年1月1日の%td値です。td(1jan2000)の方がタイプミスが少なそうです。

同様に、1960年1月1日の10:55を%tc形式で39,300,000と入力するよりは、tc(01jan1960 10:55)と入力した方が偶発的な誤りは避けることができそうです。詳細は[D] **Datetime**の*Conveniently typing SIF values*を参照してください。

25.5 日付と時刻情報の要素を取りだす

%t変数を作成できたら、次のコマンドで一部の情報を取りだすことができます。例えば、%td変数において利用できます。

year(date)	1980, 2002のような西暦4桁を取り出す。
month(date)	月情報 1, 2, ... , 12を取り出す
day(date)	日情報 1, 2, ... , 31を取りだす
halfyear(date)	上半期、または、下半期の情報、 1 または 2を取り出す
quarter(date)	四半期情報の1, 2, 3, 4 を取り出す
week(date)	年における週の情報を1, 2, ... , 52の中から取り出す
dow(date)	曜日の番号を0, 1, ... , 6 として取り出す。ただし、 0 = Sunday
week(date)	年における日の情報を1, 2, ... ,366の中から取り出す

%t変数で利用可能な変数はほかにもあります。詳細は[D] **Datetime**にある*Extracting time-of-day components from SIFs*と*Extracting date components from SIFs*を参照ください。

25.6 日付と時刻の変換

%t変数同士の変換コマンドについて説明します。例えば、cofd()関数は%td値を%tc値に変換します。17,126 (21nov2006)をcofd()で変換すると、1,479,736,920,000 (21nov2006 14:02)になります。1,479,736,920,000 (21nov2006 14:02)のdofc()は17,126 (21nov2006)を返します。

%tフォーマットの情報を変換する関数は他にもあります。詳細は[D] **Datetime**における*SIF-to-SIF conversion*を参照してください。

25.7 ビジネスカレンダ

%tcや%tdなどの内蔵型のコマンドに加え、Stataにはユーザが定義可能な%tbというビジネスカレンダに対応したフォーマットがあります。

ビジネスカレンダは、一般のカレンダから祝日や祭日を除いたものです。したがって、ビジネスカレンダとは平日のみが表示されたカレンダです。

November 2011

Su	Mo	Tu	We	Th	Fr	Sa
		1	2	3	4	X
X	7	8	9	10	11	X
X	14	15	16	17	18	19
X	21	22	23	X	25	X
X	28	29	30			

ビジネスカレンダでいう所のyesterdayは直近の平日であり、tomorrowは次の平日を指します。

date = 25nov2011と考えてください。dateが一般的な%tdフォーマットなら、

yesterday = date − 1 = 24nov2011

tomorrow = date + 1 = 26nov2011

dateがビジネス用の%tbフォーマットなら、

yesterday = date − 1 = 23nov2011

tomorrow = date + 1 = 28nov2011

この機能はラグやリードを利用する局面では特に重要です。詳細は[U] **13.10 Time-series operators**を参照してください。変数trading_dateが一般的な%td変数である場合、L.trading_dateは実際の昨日であり、F.trading_dateは翌日になります。しかし、trading_dateが%tbフォーマットで定義されている場合、L.trading_dateは1営業日前であり、F.trading_dateは次の営業日を指します。

bcal createコマンドは既存のデータセットからビジネスカレンダを作成する場合に利用します。データセット以外にもcalname.stbcalという名前のファイルを作成することも可能です。例えば、nyse.stbcalを作成できます。その時点から、Stataは新しいフォーマット%tbnyseを一般きなフォーマットとして認識します。詳細は[D] **Datetime business calendars**をご覧ください。

Stataはビジネスカレンダを一般のカレンダと同じように取り扱います。

25.8 参考文献

Cox, N. J. 2006. Speaking Stata:Time of day. *Stata Journal* 6: 124-137.

——. 2010. Stata tip 68: Week assumptions. *Stata Journal* 10: 682-685.

——. 2018. Stata tip 130: 106610 and all that: Data variables that need to be fixed. *Stata Journal* 18: 755-757.

Samuels, S. J., and N. J. Cox. 2012. Stata tip 105:Daily dates with missing days. *Stata Journal* 12: 159-161.

26 カテゴリカルデータと因子変数の操作

目次

26.1 連続、カテゴリカル、指標変数

Stataでは変数について3種類に区別できますが、ここでその違いを整理しておきます。

- 連続変数は何らの量や、大きさを示すものです。人間で言えば、年齢、身長、体重、社会経済的な情報であれば人口、土地面積、企業であれば利益や費用といったものが挙げられます。

  ここで言う連続変数とは、年齢など慣例によって、または人口などの定義による変数を含む広範なものを意味しています。任意のコードではなく、連続的な尺度の点として報告される値も含まれます。

- カテゴリカル変数は個体が属するグループを示すものです。人間であれば、人種、民族、都市であれば地理的な位置、企業であれば産業の種類によって分けることが可能です。したがって、カテゴリカル変数は文字列で表現される場合もあります。

- 指標変数はある条件が真であるか、否かを示すものです。例えば、ある日は退役軍人か否か、都市には公共交通システムがあるか否か、そして、企業ならば収益を挙げているか否か、といったことになります。

よって、指標変数はカテゴリカル変数の特殊形と考えることができます。例えば、就業しているか、失業しているかを記録した変数があるものとします。これはカテゴリカル変数と考えることができます。カテゴリカル変数はあるグループへの帰属を示すものです。対象となる人の就業状態をグループ分けする番号を付けることができます。しかし、この場合は指標変数も利用できます。つまり、就業しているという条件の真偽を示す変数と考えます。このような例では、より適切な範囲やカテゴリがあります。

このように対象物を2つのグループに分けることができる場合、カテゴリカル変数と指標変数のどちらも利用できます。対象物があるグループに属するという情報を示す場合はカテゴリカル変数、一方、対象物があるグループに属することを真とした場合、その真偽を示す形で利用する変数は指標変数となります。

よって、指標変数はカテゴリカル変数の一種と考えることができます。ポイントは真偽を示す情報であるということです。これに対し、カテゴリカル変数はグループ分けに際し、複数のグループを利用できます。考え方としては、グループが2つより多い場合にカテゴリカル変数という名前を利用すると考え、2グループにデータを分ける場合は指標変数という言葉を使います。

Stataには連続変数をカテゴリカル変数や指標変数に、カテゴリカル変数を指標変数に変換する機能が用意されています。

26.1.1 連続変数を指標変数に変換する

Stataはある条件式が真か偽を取るような論理式も扱うことができ、1または0でその真偽を示します。詳細は[U] 13 関数と数式を参照してください。例えば、年齢を示す連続変数のデータから、対象者が21才以上の場合に1をとる指標変数を作る場合は次のようにします。

```
. generate age21p = age>=21
```

年齢が21才以上の場合、変数age21pの値は1、21才未満の場合は0となります。

age21pは0と1の値しか取りませんので、データの形式をbyteにすると、効率的にメモリを利用できます。つまり、次のコマンドを実行します。

```
. generate byte age21p = age>=21
```

しかし、このままでは問題があります。このコマンドの場合、年齢に欠損値が存在すると、Stataはそれを無限大と解釈するので、変数に1が入ってしまいます。よって、欠損値の存在を考慮する場合は次のようなコマンドを実行すると安全です。

```
. generate byte age21p = age>=21 if age<.
```

このようにすれば、年齢に欠損値がある場合、age21pの値も欠損値になります。

❑ テクニカルノート

欠損値の問題を一旦脇において、 age21p = age>=21と同じ結果をもたらす次のコマンドについて考えてください。

```
. generate age21p = 1 if age>=21
```

このコマンドだけでは完結しません。このコマンドは21才以上の人にだけ、age21pの変数を1とし、それ以外の人のage21pの値は欠損値とします。

次のコマンドを続けて実行することで、正しく変数を作成できます。

```
. replace age21p = 0 if age<21
```

❑

26.1.2 連続変数をカテゴリカル変数に変換する

年齢に応じて人々を4つのカテゴリに分類する処理について考えてみましょう。グループ分けは21才以下、22才以上で38才以下、39才以上で64才以下、そして65才以上と分けるものとします。ここではグループ番号としては1, 2, 3, 4という数字を割り当てますが、これらの整数を割り当てる厳密な理由はありません。4つの整数からなる新しい変数の名前をagecatとします。該当するグループに分けられた人の最高年齢は例えば、グループ1では21才、2番目のグループでは38才、3番目のグループは64才、4番目のグループのそれは75才とします。この変数の作成方法の一例を以下に示します。方法は後述するように複数あります。

```
. use https://www.stata-press.com/data/r18/agexmpl
. generate byte agecat=21 if age<=21
(176 missing values generated)
. replace agecat=38 if age>21 & age<=38
(148 real changes made)
. replace agecat=64 if age>38 & age<=64
(24 real changes made)
. replace agecat=75 if age>64 & age<.
(4 real changes made)
```

ここではgenerateとreplaceコマンドを利用して、定義に従ってカテゴリカル変数を作成しました。念のため、最初のgenerateコマンドについてのみ解説します。ここでは0または1しか取らない新しい変数agecatをbyte型で作成しました。

同様の処理をrecode関数で作成することも可能です。

```
. use https://www.stata-press.com/data/r18/agexmpl, clear
. generate byte agecat=recode(age,21,38,64,75)
```

recode()は3つ以上の引数を利用できます。この関数は最初の引数(age)を、その後ろの引数と比較して真偽を決定します。最初の引数に指定した変数の値が、最初の値以上であれば、その次の引数で評価を行います。つまり、recode()は最初に1行ごとにageが21才以下か否かについて大小関係をチェックします。もし、真であれば、値は21になります。偽の場合、38才以下のチェックを行います。もし、真であれば、値は38になります。偽の場合、64才以下のチェックを行います。もし、真であれば、値は64になります。偽の場合、75才という値をとります。

一般的にこのようにして作成したカテゴリカル変数から表を作成します。

```
. tabulate agecat
```

agecat	Freq.	Percent	Cu
21	28	13.73	13.7
38	148	72.55	86.2
64	24	11.76	98.0
75	4	1.96	100.0
Total	204	100.00	

次に連続変数からカテゴリカル変数の作成を自動的に行ってくればコマンドを紹介します。autocode()はrecode()と同じ働きを持つ関数ですが、autocode()にはカテゴリ分けに必要な範囲の情報と、表のセル数を与える必要があります。

```
. use https://www.stata-press.com/data/r18/agexmpl, clear
. generate agecat=autocode(age,4,18,65)
. tabulate agecat
```

agecat	Freq.	Percent	Cu
29.75	82	40.20	40.2
41.5	96	47.06	87.2
53.25	16	7.84	95.1
65	10	4.90	100.0
Total	204	100.00	

ここでは例としてageを18から65までの等間隔の4つのカテゴリに分けるものとします。コマンドとしてtabulate agecatを実行して表を作成します。4つのカテゴリの境界値は 29.75, 41.5, 53.25,65才です。最初の列には29.75才以下の人数、2番目のカテゴリには29.75才よりも大きく、41.5才以下の人数、3番目のカテゴリには41.5才よりも大きく、53.25才以下の人数、そして最後のカテゴリには53.25才よりも上の年齢の人数が入ります。

❑ テクニカルノート

ここでは19-65才の範囲を任意にグループ分けしました。表から判別することはできませんが、データセットには18才未満の人と、65よりも高齢な人が含まれています。これらの人はそれぞれ最初のセルと最後のセルに含まれています。そこで、すべての人の年齢を加味してグループ分けをやり直してみましょう。すべてのデータを対象にしてsummarizeコマンドを実行し、最大最小値を調べて4つのカテゴリを作成します。そしてautocode()関数を利用して最大、最小値を18と65に置き換えます。

```
. use https://www.stata-press.com/data/r18/agexmpl, clear
. summarize age
```

Variable	Obs	Mean	Std. dev..	Min	Max
age	204	31.57353	10.28986	2	66

```
. generate agecat2=autocode(age,4,2,66)
```

sortコマンドでデータをageについて昇順に並べ替え、4つのカテゴリからなるカテゴリカル変数を作成します。ソートした状態でage[1]には最小値、age[_N]には最大値が入っていることに留意して次のコマンドを吟味してください。

```
. use https://www.stata-press.com/data/r18/agexmpl, clear
. sort age
. generate agecat2=autocode(age,4,age[1],age[_N])
. tabulate agecat2
```

agecat2	Freq.	Percent	Cum.
18	10	4.90	4.90
34	138	67.65	72.55
50	41	20.10	92.65
66	15	7.35	100.00
Total	204	100.00	

❑

26.2 因子変数を使った推定

Stataでは因子変数としてカテゴリカル変数を利用できます。詳細は[U] 11.4.3 因子変数を参照してください。カテゴリカル変数とは、例えば、性別やグループ、地域などを示す変数です。因子変数とはStataのカテゴリカル変数から作成される変数です。カテゴリカル変数の各水準(カテゴリ)を示す指標変数を作成する場合に、因子変数を利用します。また、交差項を作成することもできます。

以下の説明では水準という言葉を利用します。水準とはカテゴリカル変数が取る値のことです。例えば変数employedは水準として0と1の値を取ります。ここで、0は失業、1は就業などど考えます。ここで、変数employedは2水準の因子変数と呼びます。

因子変数によって作成されるリグレッサ(説明変数)のことは、指標変数と呼びます。厳密に言えば、これらは仮想指標変数となります。指標変数は実際に、データセット中に作成されることは基本的にありませんので、仮想という言葉を用いています。推定結果の表に指標変数名が出ているような状況でも、データセットには作成されません。

因子変数として利用する場合、カテゴリカル変数は非負の値でなければなりません。負の値が含まれる場合はrecodeコマンドを利用して編集します。詳細は[D] recodeを参照してください。文字列変数の場合、egen()関数を利用してコードを変更します。

```
. egen newcatvar= group(mystringcatvar)
```

その際、labelオプションを利用すると、egenコマンドはバリューラベルを数値コードに対して作成します。バリューラベルがあれば、分析結果はより分かりやすくなります。

```
. egen newcatvar= group(mystringcatvar), label
```

文字列によるカテゴリカル変数から数値コードを作成する場合はencodeコマンドを利用します。

```
. encode mystringcatvar, generate(newcatvar)
```

egen group(),labelと encodeの機能はどちらも同じです。ここではegen group(),labelの用法を紹介しました。詳細は、[D] egenと[D] encodeをご覧ください。

めったにありませんが、整数以外のカテゴリカル変数を処理する場合はegenコマンドを利用します。具体的な方法については[U] 26.1.2 Converting continuous variables to categorical variablesを参照してください。

❑ テクニカルノート

これまで文字列や数値変数から指標変数を作成した経験の無いユーザや、うまく指標変数がイメージできない方は次のように入力してみましょう。

```
. tabulate var, gen(newstub)
```

このように入力すると指標変数として newstub1, newstub2, ...を作成できます。詳細は[R] tabulate onewayを参照してください。

❑

以下では線形回帰のコマンドにおいて利用例を示しますが、それは解釈が容易であるが故です。例えば、ロジスティック回帰、ヘックマンの選択モデル、コックス比例ハザードモデルなどにおいても因子変数は利用できます。殆どすべの推定コマンドで利用できると言っても過言ではありません。

26.2.1 因子変数を利用する

因子変数は基本的に指標変数を組み合わせたものと考えることができます。つまり、変数の各水準ごとに指標変数を対応させた形と考えます。既存の変数から因子変数を作成する場合、変数名の前にi.を付けます。

```
. use https://www.stata-press.com/data/r18/fvex, clear
(Artificial factor variables' data)
. regress y  i.group age
```

Source	SS	df	MS		
Model	42767.8126	3	14255.9375	Number of obs	= 3,000
Residual	1348665.19	2,996	450.155272	F(3, 2996)	= 31.67
				Prob > F	= 0.0000
Total	1391433.01	2,999	463.965657	R-squared	= 0.0307
				Adj R-squared	= 0.0298
				Root MSE	= 21.217

y	Coefficient	Std. err.	t	P>|t|	[95% conf.	interval]
group						
2	-2.395169	.9497756	-2.52	0.012	-4.257447	-.5328905
3	.2966833	1.200423	0.25	0.805	-2.057054	2.65042
age	-.318005	.039939	-7.96	0.000	-.3963157	-.2396943
_cons	83.2149	1.963939	42.37	0.000	79.3641	87.06571

この例題において、変数groupは実際に1,2,3という値を取ります。そこで、次のように入力します。

```
. regress y  i.group age
```

以下のコマンドとは明らかに異なる結果を得ます。

```
. regress y  group age
```

単純にgroupを連続変数として回帰モデルで利用するのではなく、regressはgroupの各水準を示す指標変数をそれぞれ説明変数として利用します。推定結果の表の左側の列をご覧ください。変数groupの水準、2と3に対応する数字を表示しています。[U] 11.4.3 因子 変数で解説されている定義に従えば、2と3の係数はグループの2と3を示す仮想的な変数2.groupと3.groupの係数です。これらの値はそいぞれgroupの値が2、3を取る場合に1となります。

仮にgroupの値が1,2,3でなく、2,10,11,125の場合、推定結果の表のgroupの部分には10,11,125を表示します。それらの対応する仮想変数は10.group,11.group,125.groupです。

指標変数は必要なだけ利用できます。利用例を次に示します。

```
. regress y  i.group i.sex i.arm ...
```

26.2.2 ベースレベルの設定

先の例ではgroup = 1を基準としたため、regressコマンドはgroup = 1の係数を省略しました。理由はともかく、省略されたものも含め、推定結果を知りたいという場合は、baselevelsオプションを利用して、ベースレベルを明示的に表示することが可能です。実際、このオプションを次のように利用して推定を実行してください。

```
. regress, baselevels
```

Source	SS	df	MS	Number of ob	=	3,000
				F(3, 2996)	=	31.67
Model	42767.8126	3	14255.937	Prob > F	=	0.0000
Residual	1348665.19	2,996	450.15527	R-squared	=	0.0307
				Adj R-square	=	0.0298
Total	1391433.01	2,999	463.96565	Root MSE	=	21.217

y	Coefficient	Std. err.	t	P>|t|	[95% conf.	interval]
group						
1	0	(base)				
2	-2.395169	.9497756	-2.52	0.012	-4.257447	-.5328905
3	.2966833	1.200423	0.25	0.805	-2.057054	2.65042
age	-.318005	.039939	-7.96	0.000	-.3963157	-.2396943
_cons	83.2149	1.963939	42.37	0.000	79.3641	87.06571

デフォルトでは、因子変数のうち、一番小さな値をベースとします。[U] 11.4.3.2 ベースレベルで解説している手法を利用すれば、ベースレベルを任意に設定できます。例えば、group = 2をベースにする場合は次のようにします。

```
. regress y  ib2.group age
```

groupの一番大きな値をベースにする場合は、次のようにします。

```
. regress y   ib(last).group age
```

ベースレベルを変更しても、基本的にモデルの予測値が変わることはありません。また、ベースレベルを変更したからといって、係数の解釈が変わることもありません。上記の推定結果において、groupの係数はgroup = 1との差分を示すものです。group2についてはgroup1と-2.4程の、有意水準5%で、有意な差が認められます。group3については有意差はありません。

group=3をベースとして推定する場合は、次のようにします。

```
. regress y  ib3.group age
(省略)
```

group = 1とgroup = 2の係数はそれぞれ、-0.297と-2.692になります。したがって、group2とgroup1の違いは -2.692 - (-0.296) = -2.4となります。一見、推定結果は異なるように見えますが、よく見れば同じ情報を表現していることが分かります。同様にgroup = 2のp値は0.012でなく、0.805となっていますいますが、モデルの形が異なるので、相応の結果であると考えることができます。先の推定結果において0との有意性は、group1とgroup2の推定値が統計的に異なるか、という事に等しいことになります。ここに出力を表示していない、皆さんの画面を見れば、group2とgroup3の差異についても調べることができます。仮にib3.groupで推定した後、次のコマンドを実行してみます。

```
. test 2.group = 1.group
```

同じように0.012というp値を得ることになります。同様に、test 3.group = 1.groupというコマンドを実行すると、上記と同じく0.805という値を得ます。

26.2.3 ベースレベルを永久に固定する

ベースレベルを一時的に変更する場合は、上記のようにib演算子を利用します。詳細は[U] 11.4.3.2 ベースレベルを参照してください。設定を半永久に有効にする場合はfvsetコマンドを利用します。詳細は[U] 11.4.3.3 ベースレベルを永久的に設定するを参照してください。

26.2.4 主効果の有意性検定

ここでは次のデータを利用します。

```
. use https://www.stata-press.com/data/r18/fvex
. regress y i.group age
```

groupの指標変数i.groupの係数のことを一般的に主効果と呼びます。交差項は用いていないので、i.groupの主効果はgroupの水準の効果をそのまま読み取れば良いことになります。その時の主効果の有意性を検定する場合は次に示すようにcontrastコマンドを利用します。詳細は[R] contrastを参照してください。

```
. contrast group
Contrasts of marginal linear predictions Margins
            : asbalanced
```

	df	F	P>
group	2	4.89	0.007
Denominator	2996		

contrastコマンドの前に実行した推定で利用した因子変数名を入力します。そうしますと、当該因子変数における結合検定を実行します。ここではgroupの指標変数の係数が同時にゼロであることを検定しました。結果はご覧のとおり、帰無仮説を棄却しています。

26.2.5 因子変数として指標(ダミー)変数を利用する

次のデータをダウンロードし、回帰を実行します。

```
. use https://www.stata-press.com/data/r18/fvex
. regress y  i.group age
```

このモデルに変数sexを追加します。変数sexは0/1の値を取ります。このようなデータを持つ変数のことをStataでは指標変数と呼びますが、一般的にダミー変数と呼ばれています。次のように入力します。

```
. regress y  sex i.group age
```

明示的に次のように入力しても結果は同じです。

```
. regress y  i.sex i.group age
```

因子変数の先頭にi.を付けて指標変数としてモデルに組み込むと効率的です。

繰り返しになりますが、推定結果は両者ともに同じですが、単純にsexとするよりも、i.sexとした方が推定後のコマンドのことを考えると便利です。実際、i.を付けると連続変数でなく、指標変数であることを認識し、コマンドの入力も効率的に行えます。marginコマンドは、指標変数の場合に効果的に利用できるコマンドの例です。詳細は[R] marginsを参照してください。

次にregress y i.sex i.group ageと入力し、baselevelsオプションでベースレベルを明示的に出力する例を示します。一般的にbaselevelsオプションを利用することはありません。

```
. regress y  i.sex i.group age, baselevels
```

Source	SS	df	MS		
Model	214569.5	4	53642.3772	Number of obs	= 3,000
Residual	1176863.	2,995	392.942737	F(4, 2995)	= 136.51
				Prob > F	= 0.0000
Total	1391433.	2,999	463.965657	R-squared	= 0.1542
				Adj R-squared	= 0.1531
				Root MSE	= 19.823

y	Coefficient	Std. err.	t	P>|t|	[95% conf.	interval]
sex						
male	0	(base)				
female	18.44069	.8819175	20.91	0.000	16.71146	20.16991
group						
1	0	(base)				
2	5.178636	.9584485	5.40	0.000	3.299352	7.057919
3	13.45907	1.286127	10.46	0.000	10.93729	15.98085
age	-.3298831	.0373191	-8.84	0.000	-.4030567	-.2567094
_cons	68.63586	1.962901	34.97	0.000	64.78709	72.48463

すべての因子変数と同じように、デフォルトではsexの最初の水準をベースにします。
つまり、係数18.4はsex = 1をsex = 0と比較した場合のyの増分ということになります。このデータの場合、sex = 1は女性を、sex = 0は男性であることを示しています。

上記の推定結果のデーブルでは数字の0と1の代わりにmaleとfemaleという文字を表示しています。sexという変数用に値ラベルsexlabが用意されているので、Stataは推定結果のウィンドウに値ラベルを表示しています。ラベルの表示にはnofvlable, fvwrap(#), fvwrapon(word|width)というオプションがあります。これらのオプションを利用して因子変数の値ラベルの表示方法をコントロールします。詳細は[R] Estimation optionsを参照してください。

26.2.6 交差項を利用する

次のデータを用いてモデルを推定します。

```
. use https://www.stata-press.com/data/r18/fvex
. regress y  i.sex i.group age
```

groupの水準によって、男性と女性にそれぞれ異なる効果を及ぼすと考えられる場合は、sexとgroupのそれぞれの組合せに関して交差項を作成し、モデルに追加します。次の交差項を追加します。

```
sex = Male    と   group = 1
sex = Male    と   group = 2
sex = Male    と   group = 3
sex = Female  と   group = 1
sex = Female  と   group = 2
sex = Female  と   group = 3
```

このようにすれば、sexとgroupのすべての組合せがyに対して及ぼす効果を調べることができます。
実際にこれらの交差項を作成する場合には#演算子を利用します。次のコマンドを実行してください

```
. regress y   i.sex i.group i.sex#i.group age
```

#演算子の両側には因子変数を利用するという決まりがあります。したがって、それらの変数にi.を付ける必要はありません。したがって、次のように入力しても先ほどと同じ結果を得ることが可能です。

```
. regress y   i.sex i.group sex#group age
```

ただし、主効果の前にはi.sexとi.groupのようにi.を必ず付けてください。

次に推定結果を示します。ここではallbaselevelsオプションを利用しました。allbaselevelsオプションはbaselevelsと同じような機能を持ったオプションですが、allbaselevelsは主効果同様、交差項においてもベースレベルを表示します。出力結果の見方に慣れるまではallbaselevelsオプションを利用すると便利です。

```
. regress y  i.sex i.group sex#group age, allbaselevels
```

Source	SS	df	MS		
				Number of obs	= 3,000
				F(6, 2993)	= 92.52
Model	217691.706	6	36281.9511	Prob > F	= 0.0000
Residual	1173741.3	2,993	392.162145	R-squared	= 0.1565
				Adj R-squared	= 0.1548
Total	1391433.01	2,999	463.965657	Root MSE	= 19.803

y	Coefficient	Std. err.	t	P>|t|	[95% conf.	interval]
sex						
male	0	(base)				
female	21.71794	1.490858	14.57	0.000	18.79473	24.64115
group						
1	0	(base)				
2	8.420661	1.588696	5.30	0.000	5.305615	11.53571
3	16.47226	1.6724	9.85	0.000	13.19309	19.75143
sex#group						
male#1	0	(base)				
male#2	0	(base)				
male#3	0	(base)				
female#1	0	(base)				
female#2	-4.658322	1.918195	-2.43	0.015	-8.419436	-.8972081
female#3	-6.736936	2.967391	-2.27	0.023	-12.55527	-.9186038
age	-.3305546	.0373032	-8.86	0.000	-.4036972	-.2574121
_cons	65.97765	2.198032	30.02	0.000	61.66784	70.28745

sex#groupの項目についか確認してみましょう。sexとgroupの組合せは全部で6つあります。しかし、そのうち4つはbaseというラベルが付き、2つの組合せの係数だけが表示されています。確かに、sex#ageの組合せ自体は3×2ありますので、合計6個の推定値を推定します。そして定数項と連続変数ageの係数も推定しています。ここでベースとして認識された組合せを確認します。sexのベースである0と組合せを構成するもの、そしてgroupのベースと組合せを構成するものがベースと認識されています。sex = 0(male)とgroup = 1は両方のパターンに入りますので省略されています。これら以外に省略されているものは、age,groupのどちらかもベースを含むものです。

ここではsexとgroupの交差項を利用しました。仮にsex#groupの交差項の効果が、治療器具(treatment arm)にも依存すると考えられる場合、sex#group#armとして3変数の交差項を作成することも可能です。Stataは因子変数で8変数により交差項、それと別に、8つの連続変数による交差項の作成をサポートしています。

❑ テクニカルノート

交差項における仮想的な変数名は1.sex#2.groupおよび1.sex#3.groupのようになります。

❑

26.2.7 交差項の有意性検定

次のデータを用いてモデルを推定します。

```
. use https://www.stata-press.com/data/r18/fvex
. regress y   i.sex i.group sex#group age
```

ここで次のコマンドを実行することで、sex#groupのすべての交差項について有意性検定を実行できます。

```
. contrast sex#group
Contrasts of marginal linear predictions
Margins     : asbalanced
```

	df	F	P>F
sex#group	2	3.98	0.0188
Denominator	2993		

モデルに入力したものと同じ形式で-sex#group-と書くだけで有意性検定を行えます。有意水準5%で考えると、交差項は有意であると言えます。回帰式においてsexもgroupも有意ですから、その交差項が有意であっても驚くにはあたりません。

26.2.8 因子変数を追加する

次のモデルを推定してください。

```
. use https://www.stata-press.com/data/r18/fvex
. regress y   i.sex i.group sex#group age
```

このモデルはsexとgroupおよびその交差項を含むので、全要因を含むモデルと考えられます。もし、このモデルにageが含まれていないとしたら、モデルのことを全要因(full-factorial)モデルと呼ぶことができます。いずれにしろ、Stataでは全要因のモデルを簡単に入力する方法をサポートしています。実際には次のように入力します。

```
. regress y  sex##group age
```

A##Bとすると、StataはA B A#Bと解釈します。

A##B##Cとすると、StataはA B C A#B A#C B#C A#B#Cと解釈します。このパターンを繰り返します。最大、8種類の変数を使って全要因モデルを記述できます。

##はStataの省略記法です。もちろん推定結果に差は生じません。次にallbaselevelsオプションを利用しない推定例を示します。

```
. regress y  sex##group age
```

Source	SS	df	MS		
Model	217691.706	6	36281.9511	Number of obs	= 3,000
Residual	1173741.3	2,993	392.162145	F(6, 2993)	= 92.52
				Prob > F	= 0.0000
Total	1391433.01	2,999	463.965657	R-squared	= 0.1565
				Adj R-squared	= 0.1548
				Root MSE	= 19.803

y	Coefficient	Std.err.	t	P>|t|	[95% conf.	interval]
sex						
female	21.71794	1.490858	14.57	0.000	18.79473	24.64115
group						
2	8.420661	1.588696	5.30	0.000	5.305615	11.53571
3	16.47226	1.6724	9.85	0.000	13.19309	19.75143
sex#group						
female#2	-4.658322	1.918195	-2.43	0.015	-8.419436	-.8972081
female#3	-6.736936	2.967391	-2.27	0.023	-12.55527	-.9186038
age	-.3305546	.0373032	-8.86	0.000	-.4036972	-.2574121
_cons	65.97765	2.198032	30.02	0.000	61.66784	70.28745

26.2.9 二乗項と多項式の利用

変数の前にc.を付けて連続変数として設定すれば、#で二乗項を作成できます。
例えば、

```
. regress y  age c.age#c.age
```

yをageの二次関数にフィットします。同様に、

```
. regress y  age c.age#c.age c.age#c.age#c.age
```

これは3次式にフィットすることになります。

#をこのように利用することで、二乗項や三乗項を簡単に作成できます。Stataはageとc.age#c.age、さらにはc.age#c.age#c.ageの違いを明確に認識します。また、推定後の検定や診断のコマンドでも、この方法を利用することで表示結果をスマートなものにします。詳細は[R] **margins**にある*Requirements for model specification*を参照してください。

26.2.10 連続変数の交差項

#と##を利用して、連続変数とカテゴリカル変数の交差項を作る方法を解説します。連続変数を利用する場合は、sex#c.ageのように、変数の前にc.を付けます。

```
. regress y   i.sex age sex#c.age
. regress y  sex##c.age
. regress y  i.sex sex#c.age
```

最初のコマンドによるモデルの推定結果を次に示します。実際には2番目のコマンドと同じ結果になります。ここでは結果を誤解しないように、allbaselevelsのオプションを利用します。

```
. regress y   i.sex age sex#c.age, allbaselevels
```

Source	SS	df	MS		
Model	170983.675	3	56994.5583	Number of obs	= 3,000
Residual	1220449.33	2,996	407.35959	F(3, 2996)	= 139.91
Total	1391433.01	2,999	463.965657	Prob > F	= 0.0000
				R-squared	= 0.1229
				Adj R-squared	= 0.1220
				Root MSE	= 20.183

y	Coefficient	Std. err.	t	P>|t|	[95% conf.	interval]
sex						
male	0	(base)				
female	14.92308	2.789012	5.35	0.000	9.454508	20.39165
age	-.4929608	.0480944	-10.25	0.000	-.5872622	-.3986595
sex#c.age						
male	0	(base)				
female	-.0224116	.0674167	-0.33	0.740	-.1545994	.1097762
_cons	82.36936	1.812958	45.43	0.000	78.8146	85.92413

交差項(-0.022)の係数は男性と年齢の交差項に比較した場合の、女性であることの効果を示すものです。しかし、残念ながら一般的な有意水準で考えた場合、有意ではありません。

次に年齢の主効果を利用しないモデルを推定してみます。次に示すように男性と女性で、それぞれ年齢との交差項において異なる係数を得ることができました。

```
. regress y  i.sex sex#c.age
```

Source	SS	df	MS		
Model	170983.675	3	56994.5583	Number of obs	= 3,000
Residual	1220449.33	2,996	407.35959	F(3, 2996)	= 139.91
Total	1391433.01	2,999	463.965657	Prob > F	= 0.0000
				R-squared	= 0.1229
				Adj R-squared	= 0.1220
				Root MSE	= 20.183

y	Coefficient	Std. err.	t	P>|t|	[95% conf.	interval]
sex						
female	14.92308	2.789012	5.35	0.000	9.454508	20.39165
sex#c.age						
male	-.4929608	.0480944	-10.25	0.000	-.5872622	-.3986595
female	-.5153724	.0472435	-10.91	0.000	-.6080054	-.4227395
_cons	82.36936	1.812958	45.43	0.000	78.8146	85.92413

この推定結果から個々の係数を簡単に把握できますが、両者の相等性に関する情報は得られません。ここでの相等性の検定はlincomコマンドを利用します。

```
. lincom 1.sex#c.age - 0.sex#c.age
 ( 1)  - 0b.sex#c.age + 1.sex#c.age = 0
```

y	Coefficient	Std. err.	t	P>|t|	[95% conf.	interval]
(1)	-.0224116	.0674167	-0.33	0.740	-.1545994	.1097762

最初に触れたように、交差項を用いた場合、推定後の検定において個々の変数を参照する方法が複雑になることがあります。そのような場合はcoeflegendのオプションを利用して、推定結果を再表示します。

```
. regress, coeflegend
```

Source	SS	df	MS		
Model	170983.675	3	56994.5583	Number of obs	= 3,000
Residual	1220449.33	2,996	407.35959	F(3, 2996)	= 139.91
				Prob > F	= 0.0000
Total	1391433.01	2,999	463.965657	R-squared	= 0.1229
				Adj R-squared	= 0.1220
				Root MSE	= 20.183

y	Coefficient	Legend
sex female	14.92308	_b[1.sex]
sex#c.age male	-.4929608	_b[0b.sex#c.age]
	-.5153724	_b[1.sex#c.age]
female	82.36936	_b[_cons]

これを利用して次のように検定を実行します。

```
. lincom _b[1.sex#c.age] - _b[0b.sex#c.age]
```

lincom 1.sex#c.age-0.sex#c.ageと入力するより分かり易いかもしれません。カギカッコを用いた_b[]で推定値を指定する事と、モデルのベースレベルが明示的に示されています。ここでは推定値だけに関心がありますので、他の情報を気にする必要はありません。コマンドによってはカギカッコを省略できる場合もあります。もちろん、すべてのコマンドでカギカッコの表記をサポートしていますので、用法を理解しておきましょう。

26.2.11　カッコの利用

因子変数の変数はこれまで変数名の前に付けて利用してきましたが、カッコを使ってグループ化した変数群に対しても利用できます。例えば、次のように入力します。

```
. regress y  sex##(group c.age c.age#c.age)
```

直接入力する場合は次のようになります。

```
. regress y  i.sex i.group sex#group age sex#c.age c.age#c.age sex#c.age#c.age
```

カッコはネストさせた形で利用することもできます。カッコを使ってモデルを推定することで、直接入力以上の仕様をモデルに設定できる訳ではありませんが、効率的にモデルを設定でき、なにより、直感的に分かり易く入力できます。次のような例を考えます。

```
. regress y  i.sex i.group sex#group age sex#c.age c.age#c.age sex#c.age#c.age
. regress y  sex##(group c.age c.age#c.age)
```

二番目のモデルはより短く、見た目を分かり易くなっています。すべての共変量が、男性、女性ごとに異なる係数を持つという結果になりました。

26.2.12 シングルレベルの指標変数

米国各州ごとに人口10万人当たりの既婚者数を、その州の年齢の中央値に回帰するモデルを推定しました。

```
. use https://www.stata-press.com/data/r18/censusfv
(1980 Census data by state)
. regress marriagert  medage
```

Source	SS	df	MS		
Model	148.944706	1	148.944706	Number of obs	= 50
Residual	173402855	48	3612559.48	F(1, 48)	= 0.00
Total	173403004	49	3538836.82	Prob > F	= 0.9949
				R-squared	= 0.0000
				Adj R-squared	= -0.0208
				Root MSE	= 1900.7

marriagert	Coefficient	Std. err.	t	P>|t|	[95% conf.	interval]
medage	1.029541	160.3387	0.01	0.995	-321.3531	323.4122
_cons	1301.307	4744.027	0.27	0.785	-8237.199	10839.81

この推定結果から、年齢の中央値は、既婚者数に何ら影響しないことが分かります。しかし、アメリカ各州から結婚を機にネバダ州ラスベガスに移住する、という例があるかもしれません。つまり、このような人の移動が推定にバイアスをもたらしていると考えられます。そこで、人口の密集するネバダ州の指標変数を追加します。describeコマンドでデータを調べてみると、変数stateの値ラベルはstであることが分かります。そこで、label list stと入力して、ネバダ州の値を調べます。30という番号であることが分かりましたので、次のコマンドを実行します。

```
. regress marriagert medage i30.state
```

Source	SS	df	MS		
Model	171657575	2	85828787.6	Number of obs	= 50
Residual	1745428.85	47	37136.784	F(2, 47)	= 2311.15
Total	173403004	49	3538836.82	Prob > F	= 0.0000
				R-squared	= 0.9899
				Adj R-squared	= 0.9895
				Root MSE	= 192.71

marriagert	Coefficient	Std. err.	t	P>|t|	[95% conf.	interval]
medage	-61.23095	16.2825	-3.76	0.000	-93.98711	-28.47479
state						
Nevada	13255.81	194.9742	67.99	0.000	12863.57	13648.05
_cons	2875.366	481.5533	5.97	0.000	1906.606	3844.126

すべての係数が有意になり、より合理的な結果を得ることができました。

指標変数をもう少し追加して様子をみましょう。次はカルフォルニア州の指標変数を利用します。推定結果を分かりすくするためにbaselevelsオプションを利用しましたが、残念ながら、まだ直感的に分かり易いものにはなっていません。とりあえず、結果を確認してください。

```
. regress marriagert  medage i5.state i30.state, baselevels
```

Source	SS	df	MS		
Model	171657575	2	85828787.6	Number of obs	= 50
Residual	1745428.85	47	37136.784	F(2, 47)	= 2311.15
				Prob > F	= 0.0000
Total	173403004	49	3538836.82	R-squared	= 0.9899
				Adj R-squared	= 0.9895
				Root MSE	= 192.71

marriagert	Coefficient	Std. err.	t	P>|t|	[95% conf.	interval]
medage	-61.23095	16.2825	-3.76	0.000	-93.98711	-28.47479
state						
California	0	(base)				
Nevada	13255.81	194.9742	67.99	0.000	12863.57	13648.05
_cons	2875.366	481.5533	5.97	0.000	1906.606	3844.126

stateの係数をご覧ください。我々は5.stateとしてカルフォルニア州の係数を得ようとしたのでが、Stataはそれをベースレベルとして省略してしまいました。

基本的にStataは因子変数の指定をした変数はすべての関連していると理解します。ここでは次に示すように、コマンドにi5.stateとi30.stateを利用しました。

```
. regress marriagert  medage i5.state i30.state
```

このコマンドの場合、Stataは5と30の州の効果をモデルに取り込みます。よって、Stataは標準的な処理の流れとして、一番小さい番号の変数をベースカテゴリと認識します。

この表記を変更する場合は、次のように"base none"(bn)の演算子を利用して、ベースを利用しない状態でモデルを推定します。

```
. regress marriagert medage i5bn.state i30.state
```

bnは変数を問わず、1つの変数にだけ付けます。例えば、

```
. regress marriagert medage i5.state i30bn.state
```

推定結果として同じものを得ます。複数利用しても結果は同じになります。

```
. regress marriagert medage i5bn.state i30bn.state
```

念のため、推定結果を次に示しておきます。

```
. regress marriagert  medage i5bn.state i30.state, baselevels
```

Source	SS	df	MS		
Model	171681987	3	57227328.9	Number of obs	= 50
Residual	1721017.33	46	37413.4203	F(3, 46)	= 1529.59
				Prob > F	= 0.0000
Total	173403004	49	3538836.82	R-squared	= 0.9901
				Adj R-squared	= 0.9894
				Root MSE	= 193.43

marriagert	Coefficient	Std. err.	t	P>|t|	[95% conf.	interval]
medage	-60.80985	16.35134	-3.72	0.001	-93.7234	-27.8963
state						
California	-157.9413	195.5294	-0.81	0.423	-551.5214	235.6389
Nevada	13252.3	195.7472	67.70	0.000	12858.28	13646.32
_cons	2866.156	483.478	5.93	0.000	1892.965	3839.346

26.2.13 水準のサブグループを利用する

先ほどのコマンドは次のようなものでした。

```
. regress marriagert medage i5bn.state i30.state
```

ネームリストを利用して特定の水準だけを利用することもできます。例えば、

```
. regress marriagert   medage i(5 30)bn.state
```

i(5 30)bn.stateを投入することで5から30までが示す水準を回帰モデルに組み込んでいます。同様の水準の選択の方法は、交差項でも行えます。

```
. regress y i.arm i.agegroup arm#i(3/4)bn.agegroup
```

通常は行われませんが、主効果に含める水準と交差項に含める水準を別に指定することもできます。上記は、arm#agegroupの交差項を、agegroupの3と4に対応するものだけ作成し、すべての水準の効果をi.agegroupで考慮すしています。

26.2.14 因子変数と時系列演算子

因子演算子は時間演算子のL.(ラグ)、およびF.(リード)と組み合わせて利用できます。例えば、iL.group(またはLi.group)， cL.age#cL.age(またはLc.age#Lc.age)や、F.arm#L.groupなどとして利用できます。もちろん、tssetやxtsetなどの設定が前提とします。詳細は[U] 11.4.3.6 時系列演算子と因子変数の利用を参照してください。

26.2.15 空のセルの処理

例えば、次のようなデータの場合について考えます。

```
. use https://www.stata-press.com/data/r18/estimability, clear
(margins estimability)
. table sex group, nototals
```

	Group				
Sex	1	2	3	4	5
Male	2	9	27	8	2
Female	9	9	3		

このデータの場合、sex = Femaleとgroup = 4, sex = Femaleとgroup = 5のデータは欠損値となっています。この状態でモデルをフィットすると、次のような結果になります。

```
. regress y  sex##group
note: 1.sex#4.group identifies no observations in the sample.
note: 1.sex#5.group identifies no observations in the sample.
```

Source	SS	df	MS		
Model	839.550121	7	119.935732	Number of obs	= 69
Residual	1500.65278	61	24.6008652	F(7, 61)	= 4.88
				Prob > F	= 0.0002
Total	2340.2029	68	34.4147485	R-squared	= 0.3588
				Adj R-squared	= 0.2852
				Root MSE	= 4.9599

y	Coefficient	Std. err.	t	P>|t|	[95% conf.	interval]
sex Fe						
male	-5.666667	3.877352	-1.46	0.149	-13.41991	2.086579
group 2						
3	-13.55556	3.877352	-3.50	0.001	-21.3088	-5.80231
4	-13	3.634773	-3.58	0.001	-20.26818	-5.731822
5	-12.875	3.921166	-3.28	0.002	-20.71586	-5.034145
	-11	4.959926	-2.22	0.030	-20.91798	-1.082015
sex#group						
Female#2	12.11111	4.527772	2.67	0.010	3.057271	21.16495
Female#3	10	4.913786	2.04	0.046	.1742775	19.82572
Female#4	0	(empty)				
Female#5	0	(empty)				
_cons	32	3.507197	9.12	0.000	24.98693	39.01307

Stataは画面にsex = Femaleとgroup = 4, sex = Femaleとgroup = 5のセルが空になっています。つまり、推定値は存在しません。1.sex#4.groupと1.sex#5.groupの係数はありません。ここで1は女性を示しています。

セルが空白になっている所はデータが存在しないので、気にする必要はありません。しかし、推定後に線形や非線形の係数検定を交差項について実行する場合は、係数の指定方法に注意してください。詳細は[R] **margins**にある*Estimability of margins*を参照してください。

26.3 参考文献

Cox, N. J. 2018. Speaking Stata: From rounding to binning. *Stata Journal* 18: 741-754.

Cox, N. J., and C. B. Schechter. 2019. Speaking Stata: How best to generate indicator or dummy variables. *Stata Journal* 19: 246-259.

27 推定コマンドの概要

目次

27.1 はじめに

Stataには記述統計量の算出や統計モデルをフィットさせる様々な推定コマンドが用意されておりますが、ここでは代表的なコマンドの概要だけを簡単にご説明します。推定コマンドは全く考え方の異なるものから、目的に則して若干異なるもの、そして本質的に等しいものと分けることができます。

Stataにはsvy:、mi:、bayes:、fmm:など、推定コマンドに付いて計算方法を少し変える推定プリフィックスコマンドがあります。

この章で解説していない推定コマンドの共通項目は[U] 20 Estimation and postestimation commandsを参照してください。分散共分行列の計算(標準誤差の計算)方法を工夫した推定手法については、[U] 20.22 Obtaining robust variance estimatesを参照してください。また、[U] 20.13 Performing hypothesis test on the coefficientsもぜひ参照してください。本章では推定の手法や共分散行列の計算方法に関連する事柄は説明しません。ある統計学的な考え方にマッチするコマンドはどのようなものであるか、という事を解説します。

本章では、Stata 18が公式に備える推定コマンドのみ議論します。別途、Stataユーザが開発し、自主的に共有を行ったコマンドが存在します。search estimationや、ssc new、あるいはssc hotコマンドを実行して、そうしたコマンドを発見することもできます。詳細は、[R] sscでを参照してください。無論、ご自身でコマンドを開発することはいつでも可能です。

27.2 平均や比率などの統計量

ここで解説するコマンドは回帰モデルのフィットではなく、要約統計の推定に利用するものです。計算方法は比較的単純ですが、これらも推定のコマンドですので、[U] 20 Estimation and postestimation commandsで解説するような推定後のコマンドをサポートしています。

mean, proportion, ratioそしてtotalはそれぞれ平均、割合、比率、合計を推定するコマンドです。これらのコマンドではカテゴリカル変数を利用して標本をグループ化した時に、各グループにおける各統計量の計算もサポートしています。また、mean, proportion, ratioは標準化したデータにおける統計量の計算も、原データから直接行えます。

カテゴリカル変数でグループ化したデータの平均を計算する際、pwmeanには特別なイオプションが用意されています。また、pwmeanは全てのグループの平均の組み合わせについて、その有意性検定と信頼区間の計算をサポートしています。もちろん、多重比較を考慮して信頼区間の計算を行います。

27.3 連続アウトカム

27.3.1 ANOVA、ANCOVA

ANOVAとANCOVAは一般的な線形回帰モデルをフィットするため、[U] 27.3.2 線形回帰での議論に関連する内容ですが、違いを強調するために別項目として紹介します。対応するコマンドはanova, oneway, onewayです。

anovaは、ANOVAとANCOVAモデル(一元、二元および三元配置モデル)をフィットします。ネスト型、混合計画モデルや繰り返しのモデルに対しても利用できます。

onewayは一元配置のモデルで利用します。等分散性を検定するBartlett検定や多重比較の検定も同時に行えるところがポイントです。多重比較の検定は、anovaの後、pwcompareを実行することでも行えます。

lonewayはonewayとよく似たコマンドです。解析結果は完全に同じです。ただし、lonewayはより多くの水準に対応しています。onewayは水準数376です。lonewayは級内相関などanovaに比べより多くの統計量を出力します。

詳細は、[U] 27.22 多変量解析でを参照してください。

27.3.2 線形回帰

次のようなモデルについて考えましょう。

$$y_j = \mathbf{x}_j \beta + \epsilon_j$$

yは連続変数とします。ここではσ^2_ϵが観測値jに対して一定であるとします。このようなモデルを線形回帰モデル、推定量は最小二乗推定量(OLS)と呼びます。

Stataの線形回帰コマンドはregressです。regressコマンドは標準的な推定量の計算同様、分散の堅牢な推定量の計算もサポートしています。また、推定後に行うフィットの診断や検定、予測計算に関する機能もサポートしています。

以下に説明するコマンドも、線形回帰を実行しますが、それぞれ特殊な機能を用意しています。

1. aregは$y_j = \mathbf{x}_j \beta + \mathbf{d}_j \gamma + \epsilon_j$というモデルをデータにフィットさせるコマンドです。ここで$\mathbf{d}_j$はダミー変数を意味しています。aregの便利な所はダミー変数を自動作成して推定値βと、関連する統計量を計算するところにあります。ただし、γの推定値は報告しません。固定効果モデルを推定する場合は、[U] 27.51 連続アウトカムのパネルデータで解説するxtregコマンドを利用してください。aregコマンドの利用を考えている方にとっては、xtregを用いた方がメリットがあります。つまり、xtregは要約統計量や検定統計量をより詳細に提供します。繰り返しますが、aregの出力は、regressコマンドでダミー変数を利用したものと同じです。つまり、R^2はγの効果を含んだものになります。
2. wildbootstrap regressとwildbootstrap aregは、regressおよびaregと同じモデルを適合させますが、ロバストな推論のためにワイルドクラスターブートストラップのp値と信頼区間を提供します。詳細は [R] wildbootstrapを参照してください。
3. cnsregは係数に線形制約を課して推定を行うコマンドです。
4. eivregは変数の観測誤差を考慮した推定コマンドです。
5. rregは堅牢な回帰モデルの推定を行うコマンドです。ただし、堅牢な標準誤差を算出するコマンドとは異なりますので、ご注意ください。堅牢な標準誤差の計算に関する詳細は[U] 20.22 堅牢な分散推定値の堅牢な標準誤差の計算を参照してください。堅牢な回帰のポイントは標準誤差ではなく、点推定値に関心があることです。実際の方法は、外れ値をダウンウェイトするという考えに基づくものです。

27.3.3 誤差項が不均一分散の場合の回帰

$y_j = x_j \beta + \epsilon_j$とうモデルを想定します。ここで、$\epsilon_j$の分散は一定ではないものとします。

hetregressは、ϵ_jの分散が1つ以上の共変量の指数関数で表されるとする乗法的分散不均一性を伴うモデルでの推定を行います。分散不均一性は、最尤法もしくはHarveyの二段階一般化最小二乗法で推定できます。

ϵ_jの分散の従う関数が分からない場合、regressコマンドでvce(robust)オプションでバイアスのない推定が行えるのでおすすめです。Stataでrobustという場合は、不均一分散に対するWhiteの修正を意味します。

vwls(分散-加重最小二乗法)は$y_j = x_j \beta + \epsilon_j$の推定値を求めるコマンドです。ただし、$\epsilon_j$の分散はグループデータから計算できるか、または既存の情報から分かっているものとします。vwlsはカテゴリデータを取り扱う研究者や、物理学者にとって重宝するコマンドです。

分位回帰を実行するqregコマンドを不均一分散の状況で利用すると、どうなるでしょうか。ϵ_jが不均一分散の場合、中央値回帰(qregの機能)は$y_j = x_j \beta + \epsilon_j$の推定量となります。他の分位についても分位回帰をつかって、不均一分散のモデルを推定可能です。詳細はsqregとiqregコマンドをご覧ください。sqregは複数の分位をiqregは分位間の差を推定するコマンドです。

27.3.4 系列相関のある場合の推定

誤差項に相関があるという場合、観測値がグループ化でき、そのグループ内でデータ間に相関があると考えます。ただし、グループが異なる場合、誤差項には相関はないものとします。この場合、vce(cluster clustvar)オプションをregressコマンドで利用すれば正しい推定値を得る事が可能です。効率性はないものの、標準誤差は正確に計算でき、堅牢性もあります。詳細は[U] 20.22 堅牢な分散推定値の計算を参照してください。ただし、相関構造が既知でかつ正しい場合、xtregやxtglsコマンドを利用してより優れた推定値を得ることができます。詳細は[U] 27.15.1 パネルデータにおける線形回帰および[U] 27.16 時マルチレベル混合効果モデルを参照してください。

系列相関が存在する場合の推定に関しては[U] 27.14 時系列データによるモデルを参照してください。

27.3.5 センサード型および切断型の回帰

1. tobitはy_iが左側切断、または右側切断、もしくは両側切断のデータである場合に、線形回帰を行うコマンドです。例えば、y_iがy_i<1,000である場合、y_iを観測しないというルールがあったとしてもy_i<1,000ということだけは分かります。tobitはそのようなデータに対して利用する推定コマンドです。
2. intreg （インターバル回帰）はtobitを一般化したものです。開区間だけでなく、intregは閉区間のデータに対しても利用できます。$y_{0j} \leq y_j \leq y_{1j}$という関係が既知である場合、$y_{0j} \leq y_j \leq y_{1j}$において$y_j$そのものでなく、$y_{1j}$が観測できることがあります。例えば、サーベイデータで所得/月が$1,500-$2,500などと報告される場合がこれに相当します。intregはこのようなデータに対する回帰の際に利用します。intregは$y_{0j} = y_{1j}$であったとしても良いので、regressコマンドによる推定結果と同じものを得る事もあります。さらに、intregはy_{0j}が負の無限大で、y_{1j}は正の無限大でもかまいません。つまり、tobitと同じ結果になる事もあります。
3. truncregは母集団に制約を課した取得した標本に対する回帰モデルのフィットに利用します。基本的にtobitと似ていますが、独立した変数は取り扱いません。母集団はあくまで正規分布しているという仮定の下、誤差項は切断型正規分布に従うものとします。
4. churdleは、y_iに対する下限ll、上限ul、あるいはその両方があるような線形または指数ハードルモデルでの推定を行うコマンドです。従属変数は境界での離散的な観測値とその間の連続的な観測値の混合です。このy_iにおける境界での離散的な値と、その間での連続的な値は両方が実際的な値です。これに対し、tobitやintregなどに見られる打ち切りのあるモデルでは、境界の間の観測値が実際的な値であるのに対し、境界の観測値は実際の値がその先に存在することを示すのみです。ハードルモデルは、観測値が境界上のデータなのか、あるいはその間のデータかを決定する一つのモデルと、境界間のデータに対する一つのモデルの合計二つを使用します。

27.3.6 連立方程式の推定

推定式の撹乱項間に相関があるものの、内生性のある変数は利用していないようなケースについて考えます。

$$\begin{aligned} y_{1j} &= \mathbf{x}_{1j}\boldsymbol{\beta} + \epsilon_{1j} \\ y_{2j} &= \mathbf{x}_{2j}\boldsymbol{\beta} + \epsilon_{2j} \\ &\vdots \\ y_{mj} &= \mathbf{x}_{mj}\boldsymbol{\beta} + \epsilon_{mj} \end{aligned}$$

ここで、$\epsilon_{k.}$と$\epsilon_{.l}$には相関ρ_{kl}があるものとします。この相関はデータから推定します。このタイプの回帰モデルはZellnerのSURと呼ばれ、suregコマンドを利用します。特に、$\mathbf{x}_{1j} = \mathbf{x}_{2j} = \cdots = \mathbf{x}_{mj}$の場合、このモデルのことを多変量回帰と呼び、コマンドにはmvregを利用します。

推定式は線形である必要はありません。非線形の場合はnlsurコマンドを利用します。詳細は[U] 26.3.8 非線形回帰を参照してください。消費者の商品やサービスの購入を記述する一連の方程式を適合させるには、demandsysを参照してください。

27.3.7 確率フロンティアモデル

クロスセクションデータにおいて確率的生産/費用フロンティアモデルを推定する場合はfrontierコマンドを利用します。モデルの定義は次のようになります。

$$y_i = \mathbf{x}_i\boldsymbol{\beta} + v_i - su_i$$

ここで、

$$s = \begin{cases} 1 & \text{生産関数} \\ -1 & \text{費用関数} \end{cases}$$

u_iは非負の撹乱項で生産関数の場合は技術の非効率性を、費用関数においては費用の非効率性を示します。個体ごとの撹乱項v_iは正規分布を仮定しますが、非効率性を示す撹乱項は半正規、指数、切断正規分布のいずれかを仮定します。また、撹乱項において非負の分布である半正規または指数分布を選択した場合、frontierコマンドで共変量を条件とする不均一分散モデルの推定が可能となります。一方、非負の分布として切断型の正規分布を利用する場合、frontierで条件付き平均モデルを推定できます。切断型正規分布の平均は共変量の組み合わせによる、線形関数としてモデル化します。

パネルデータの確率フロンティアモデルに関しては[U] 27.15.1 連続アウトカムのパネルデータを参照してください。

27.3.8 非線形回帰

nlは$y_j = f\left(\mathbf{x}_j, \boldsymbol{\beta}\right) + \epsilon_j$における非線形最小二乗推定量です。$f\left(\mathbf{x}_j, \boldsymbol{\beta}\right)$は指数関数や対数正規分布など、任意の関数です。nlsurは実行可能な一般化非線形最小二乗法（feasible generalized nonlinear least squares）による非線形推定式の連立方程式のフィットを行います。ZellnerのSURの非線形版と見ることもできます。

非線形モデルのひとつにBox-Cox変換があります。boxcoxは次のモデルにおける係数の最尤推定値とBox-Cox変換パラメータを求めます。

$$y_i^{(\theta)} = \beta_0 + \beta_1 x_{i1}^{(\lambda)} + \beta_2 x_{i2}^{(\lambda)} + \cdots + \beta_k x_{ik}^{(\lambda)} + \gamma_1 z_{i1} + \gamma_2 z_{i2} + \cdots \gamma_1 z_{i1} + \epsilon_i$$

ここでは、$\epsilon \sim N\left(0, \sigma^2\right)$とします。この式はyをパラメータ$\theta$でボックスコックス変換することを意味しています。$x_1$, x_2, ... , x_kはパラメータλでボックスコックス変換した値です。z_1, z_2, ... , z_1は独立変数で、変換は行いません。上記の一般形に加え、boxcoxコマンドには$\lambda = \theta$, $\lambda = 1$, $\theta = 1$と条件で定義した3 つのモデルも存在します。

非線形混合効果モデルに関しては、[U] **27.16 マルチレベル混合効果モデル**を参照してください。

27.3.9 ノンパラメトリック回帰

上記で議論してきたモデルはすべて、アウトカムと共変量の関係性をある関数形で指定しました。ノンパラメトリック回帰は、関数形が不明な場合でも与えられた共変量のもとでのアウトカムの平均をモデル化することができます。一般的には、regressで行えたモデルについてはすべて、安心してnpregress kernelとnpregress seriesでモデル化できます。線形関係を仮定しない点のみが異なります。しかし、ノンパラメトリックモデルでは、パラメトリックモデルよりも多くの観測値が必要になります。

npregress kernelは2種類のカーネルベース推定量、局所線形（local-linear）推定量と局所一定(local-constant)推定量）を実装します。カーネルベース推定量は、バイアスと誤差分散とのトレードオフとなるバンド幅の最適化に依存します。しかし、これまで議論してきたパラメトリック推定量よりもかなり多くの観測量を必要とします。局所線形（local-linear）推定量および局所一定（local-constant）推定量は、平均に関する同等の推定量を与えますが、考慮すべき重要な違いがあります。

局所線形推定量からは、共変量のマージン効果を得ることができます。また、モデルが線形の場合、局所線形推定量は局所一定推定量が必ずしも復元しない線形な平均値を常に得ることができます。一方、アウトカムが非負なケースで、局所一定推定量は常に非負な予測値を与えます。局所線形推定量は負の予測値を与えることがあります。

npregress seriesはB-スプラインまたは多項基底を利用したノンパラメトリック連続推定量を用います。推定では、基底関数の要素の線形の組み合わせによって、未知の平均関数が近似されます。つまり、多項基底を考慮することができるということです。多項を利用して未知の平均関数を近似します。平均関数が複雑になるほど、平均を近似するために必要な多項($x, x^2, x^3, \ldots$)が増えていきます。同様に、平均関数の複雑さが増していくに連れて、B-スプラインやスプライン基底関数に必要な項も増えていきます。npregress seriesはバイアスと分散がトレードオフのバランスを取るような最良の項の数を指定することができます。項が指定されると、最小二乗法でフィットを行います。線形の近似関数とそれを利用した予測は推定を興味深いものにします。

ノンパラメトリック回帰の詳細は、[R] npregress introを参照してください。

27.4 二値アウトカム

27.4.1 ロジスティック、プロビット、補二重対数回帰

ここで議論するモデルの表現法はさまざまありますが、ひとつには以下があります。

$$\Pr\left(y_j \neq 0\right) = F\left(\mathbf{x}_j \beta\right)$$

ここでFは累積分布関数です。F()の代表的なものとして正規分布(プロビットモデル)とロジスティック分布(ロジット/ロジスティックモデル)があります。プロビット/ロジットモデルの推定には最尤推定を行うprobitとlogitコマンドを利用します。logitとよく似たコマンドにlogisticコマンドがあります。両者の推定値は同じです。表現方法が異なるだけです。この他に正確なロジスティック推定量というものがありますが、詳細は[U] 27.11 正確推定量を参照してください。

これ以外に補二重対数分布というものがあります。これはcloglogコマンドを利用して最尤推定値を求めるものです。

ロジットとロジスティックという言葉にこだわる必要はありません。一方から他方へ、表示形式を簡単に変更できます。医療科学分野ではロジットは最小カイ二乗推定量で、ロジスティックは最尤推定量を意味します。社会科学分野ではまた異なった考え方を用います。しかし、書籍ではロジットとロジスティックという言葉を使い分けていますのでご注意ください。StataにはlogitとlogisticというコマンドをSeparateに用意しています。どちらも最尤推定を用いて推定を行い、それぞれの形式で推定結果を表示します。

ここで具体的にlogitコマンドとlogisticコマンドの差異を説明します。logitは係数を表示し、logisticは指数を係数とする累乗(オッズ比)を表示します。これはどの情報を必要とするかという、目的の違いによるものです。また、logisticには推定後の検定や診断の機能が用意されています。実際、logisticとlogitでは、推定後に利用可能なコマンドが異なります。

logitやlogisticのどちらを利用するか、迷うようなケースではlogisticコマンドを利用します。ロジスティックコマンドの場合、オッズ比を一目で把握できます。

binregはGLMによるモデル推定で、個人水準またグループ化データで推定を行う場合に利用するコマンドです。分布族は二項分布を仮定し、各リンクは採用したリンク関数のパラメータを推定します。また、binregは目的とする生物統計モデルにしたがって、出力形式に応じたリンク関数の選択を行うオプションを用意しています。利用可能なリンク関数と表示形式のオプションを次に示します。

オプション	リンク関数	パラメータ
or	logit	オッズ比 = exp(β)
rr	log	リスク比 = exp(β)
hr	log complement	健康比 = exp(β)
rd	identity	リスク差 = β

reriは、相対リスクの加法モデルにおける二要素間の相互作用を評価するために使用することができます。reriは、相互作用による相対過剰リスク、起因比、およびシナジー指数を算出します。reriは、二項一般化線形モデルやロジスティックモデル、ポアソンモデル、負の二項モデル、Coxモデル、パラメトリック生存モデル、および区間打ち切りパラメトリックおよび非パラメトリック生存モデルをサポートしています。

ロジットに関連して、非対称なロジット推定量を求めるscobitというコマンドも用意されています。これはロジットリンク関数の検出力を向上させます。

hetprobitは不均一分散のプロビットモデルを推定するコマンドです。このモデルの誤差項の分散に関してもパラメータを用います。

また、biprobitコマンドはニ変量プロビットモデルの推定に利用するコマンドです。このモデルではアウトカムに相関を仮定します。biprobitは、(0,0)と(1,1)などアウトカムが部分的にしか観測できないようなデータに際しても、部分観測モデルをフィットします。

27.4.2 条件付きロジスティック回帰

条件付きロジスティック回帰の推定にはclogitコマンドを利用します。このモデルでは、データはグループに分けることができ、そのグループでは予め決まった数のイベントが発生するものとします。モデルはデータの共変量$\mathbf{x}_j$により発生するイベントのリスクを計測するものです。対応のあるケースコントロール研究(clogitは1:1, 1:kそしてm:kのマッチングに対応しています)、および、データがある集合ごとにグループ化でき、そのイベント数が決まったいる場合に、このモデルを利用します。 clogitはグループ効果のある固定型ロジスティック回帰の推定にも利用できます。

27.4.3 ROC分析

ROCは受信者操作特性(Receiver Operating Characteristics)の略です。ROCは特異度と感度の関係を表現するものです。すなわち、診断による偽陽性数と検出できなかった真の陽性のカウント数を利用します。ROCという用語はレーダーの操作ノブのROCというラベルが貼られていたことに由来します。そのノブを一方向に傾けると受信機の感度があがるになっているとします。それに伴い飛行機の存在をより感知しやすくなります。しかし一方で、飛行機以外のものを誤って飛行機と認識する(偽陽性)ことも多くなってしまいます。ノブを逆の方向に傾けると、ご認識の危険はなくなりますが、実際の飛行機を認識する能力が落ちてしまいます。したがって今日では統計手法を用いたアプリケーションでは、診断に際し境界値を設定し、その値以上ならば陽性、以下ならば陰性と判定します。

ROC分析はプロビットやロジスティック回帰など、被説明変数が二値のモデルに対して適用します。最初にモデルを推定し、陽性であることの予測確率を求めます。それが境界値を上回れば陽性、そうでなければ陰性となります。

ROC分析においては境界値の設定に関し、感度と特異度がトレードオフの関係にあります。

StataではROC分析用にroctab, roccomp, rocfit, rocgold, rocreg, rocregplotという6つのコマンドを用意しています。

roctabはROCカーブのノンパラメトリック推定を行うコマンドであり、面積に対してBamber and Hanleyの信頼区間を算出します。

roccompはROC面積について相等性の検定を行うコマンドです。ノンパラメトリックおよびパラメトリックな従法線ROCカーブを推定します。

rocfitは単一の分類子(classifier)に対する最尤推定を実行します。ここで言う分類子とは、真の状態に対する潜在的な従法線変数のインジケータのことを意味します。

rocgoldは至適基準としてのROC曲線と、手元の解析結果が等しいことを検定するコマンドです。多重比較を行う場合はシダックの手法を用いて有意水準を調整します。

rocregはROC回帰を実行するコマンドです。年齢や性別などの予後因子に対して感度と特異度を調整します。したがってROCコマンドの中では最も一般的に利用されるコマンドです。

rocregplotはrocregコマンドで推定したモデルのROC曲線を描画するコマンドです。共変量、分類変数、そして両者を組み合わせたものごとにROC曲線を描きます。

詳細は、[R] **roc**をご覧ください。

27.5 フラクショナルアウトカム

フラクショナル応答データは、着目するアウトカムがフラクション、比率、比などで測定されるときに表れます。こうしたアウトカムに広く用いられるモデリングは、ベータ回帰およびフラクショナル回帰です。

betaregは、アウトカムが厳密に0より大きく1未満の場合に対応するベータ回帰モデルのパラメータ推定に用いることができます。

fracregは、アウトカムが0以上1以下の場合に対応する、フラクショナルロジスティックモデル、フラクショナルプロビットモデル、フラクショナル不均一分散プロビットモデルのパラメータ推定に用いることができます。

両コマンドとも疑似最尤推定法を用います。従属変数が0より大きく1未満の場合、betaregのほうがfracregに比べて従属変数の条件付き平均の分布に対してよりフレキシブル性を有します。

27.6 順序アウトカム

順序アウトカムの分析には順序ロジット、順序プロビット、ゼロ拡大順序プロビット、ランク順序ロジットのほか、ランク順序プロビットのコマンドを用意しています。

oprobitとologitは最尤法による順序プロビットおよびロジットモデルを推定するコマンドです。これらのコマンドは、一般的に比例オッズモデルとして知られているプロビット/ロジットモデルを一般化したモデルの推定に利用します。アウトカムは小さいものから大きいものに順序がしたがっているものとします。このモデルは非観測な$z_j = \mathbf{x}_j\boldsymbol{\beta}$を仮定し、k番目のアウトカムが観測される確率は$\Pr(c_{k-1} < z_j < c_k)$とします。$c_0 = -\infty$，$c_k = +\infty$で、$c_1$，...，$c_{k-1}$は$\boldsymbol{\beta}$と同時にデータから推定します。

heteroprobitは不均一分散の順序プロビットモデルにフィットを行います。これは分散を独立変数の関数としてモデル化し、項目やグループを区別する一般化順序プロビットモデルです。

zioprobitはゼロ拡大順序プロビットおよびzioloitはゼロ拡大順序ロジットの推定を行います。順序プロビットまたは順序ロジットによる期待値より高い割合で最下級の度数が生起する順序アウトカムに用いられます。通常、最下級はゼロで表されます。アウトカムは2つのプロセスの結果となります。第一は、ネガティブアウトカムとしての大量のゼロの存在を記述するプロビットモデルによるプロセスです。第二は、先ほどのプロビットモデルのポジティブアウトカムを記述する、ポジティブアウトカムに対する条件付きの基での順序プロビットプロセスです。ゼロ拡大順序ロジットも同様ですが、過剰なゼロがロジットプロセスでモデル化され、順序ロジットプロセスが順序アウトカムを記述する点が異なります。 ロジットバージョンは、オプションでオッズ比を報告できます。

cmrologitはランク順序ロジットモデルをフィットする場合に利用します。このモデルはPlackett-Luceモデルとしても知られていますが、探索型ロジットモデルであり、選択ベースの結合型分析に利用します。

cmroprobitは順位データに対してprobitモデルをフィットさせるコマンドで、cmrologitよりもフレキシブルなコマンドです。その理由は順位に共変量を利用できるところにあります。

27.7 カテゴリカルアウトカム

カテゴリカルの分析には多項ロジスティック回帰、多項プロビット回帰、ステレオタイプロジスティック回帰、McFaddenの選択モデル(条件付き固定効果ロジスティック回帰)、ネストしたロジスティック回帰、多項プロビット選択モデル、混合ロジット選択モデルのコマンドを用意しています。

mlogitは多項ロジスティックモデル(多値ロジスティックモデル)を推定するコマンドです。mprobitは同様の機能を有しますが、プロビットモデルを一般化したものになっています。両コマンドともアウトカムの順序が不自然で、アウトカムの特性だけが分かっている場合に利用します。

slogitはデータが順序通りにはなっていない、(ologitと同じ)データに対して利用するコマンドです。しかし、実際に順序が正確か否か不明な場合はmlogitを利用します。

cmclogit はMcFaddenの選択モデル(条件付きロジスティック回帰)をフィットします。名前にMcFaddenの選択モデルとあるように、多項ロジスッティック回帰と同様、アウトカムの順序が不自然だが、選択されたアウトカムと選択されなかったアウトカムの特性だけが分かる場合に利用します。

条件付きロジスティック回帰と呼ばれているモデルはデータを一まとめにした形で用意し、その時々である選択肢を取るというものです。例えば、継続的にある個人を観察した時に、ある疾病にかかってしまったり、なんらかの不運に見舞われるというデータを分析する時に利用します。被説明変数に相当する何らかの選択肢を、観察時に必ずとることを前提とします。選択者の特性に関して考慮する必要はありません。結果的に、選択者の特性はモデルに影響しません。

選択モデルにおいてmlogitとcmclogitは、オッズ比はIIA(選択肢からの独立性)の仮定を満たしていることを前提とします。しかし、この仮定は実際のデータと照らしてみると成立していないことがあります。よって、ネスト型のロジットモデルではこの仮定を緩めて利用します。ランダム型の効用選択モデルの推定する際にnlogitを利用します。

cmmprobitはアウトカムの順序が不自然な並びになっており、リグレッサも変更される場合に利用します。cmmixlogitはmlogitの一般化してアウトカムの選択肢間での相関を許したものと考えることができます。mlogitと違い、cmmprobitおよびcmmixlogitはIIAを仮定していません。cmmixlogitにおける変量効果はIIAを緩和し、所与の共変量の条件下での異質性を直接的にモデル化するものとなります。

27.8 カウントアウトカム

ここでは従属変数はあるイベントの発生回数を示すモデルについて考えます。具体的にはポアソン回帰と負の二項回帰モデルを利用します。ポアソンモデルの定義を次に示します。

$$E(\text{count}) = E_j \exp\left(\mathbf{x}_j \boldsymbol{\beta}\right)$$

ここでE_jは曝露時間です。poissonコマンドはこのモデルをフィットします。詳細は[R] poissonを参照してください。この他に正確ポアソン推定量というものがありますが、詳細は[U] 26.10 正確推定量を参照してください。ivpoissonは説明変数に内生性のある共変量を含むポアソンモデルを推定もする場合に利用します。このコマンドはカウントの代わりに非負の、連続なアウトカムを用いる場合も利用できます。詳細は[R] ivpoissonをご覧ください。

負の二項回帰は、ポアソンの混合分布に従うデータでモデル推定する場合に利用するものです。データがポアソン回帰モデルでフィットでき、脱落変数が存在し、それがパラメータαのガンマ分布に従う場合に、負の二項モデルを利用します。負の二項回帰は$\boldsymbol{\beta}$とαを推定します。モデルをフィットするコマンドはnbregです。このモデルの派生形でαをモデル化するコマンドがあります。このモデルのコマンドはgnbregです。詳細は[R] nbregを参照ください。

アウトカム変数の値は、首尾よくある範囲内の値をとり観測される場合と、観測されずある範囲外の値をとることが分かる場合とがあります。そうした観測されないケースは打ち切りと切断の2種類に分類されます。アウトカムの観測データはあるが、本来の値までは観測できず、特定の範囲外の値であることのみが分かっている状態は打ち切りと呼びます。アウトカムの値が特定の範囲外であるため、観測データがない状態は切断と呼びます。cpoissonコマンドは打ち切りのあるアウトカムデータをモデルにフィットします。tpoissonとtnbregコマンドは切断のあるアウトカムデータをモデルにフィットします。

カウントモデルにおけるゼロインフレーションとは、通常のモデルよりも、カウントのゼロの出現回数が多いような状態を指します。超過分のゼロはプロビットや、ロジット過程(予備過程)によって説明可能です。もし、これらの過程で正のアウトカムを算出する場合、一般的なカウント形過程はイベントの発生を示し、それ以外の場合はカウントしてゼロを示します。そして、予備過程で負のアウトカムが発生した場合、超過分のゼロを作成するという仕組みになっています。このようなケースではzipとzinbを利用します。詳細は[R] zipと[R] zinbを参照してください。

27.9 一般化線形モデル(GLM)

一般化線形モデルの定義は次の通りです。

$$g\left\{E\left(y_j\right)\right\} = \mathbf{x}_j \boldsymbol{\beta}, \quad y_j \sim F$$

ここでg()はリンク関数、Fは指数分布族です。推定時にはこれらの選択を行います。コマンドはglmです。

GLMフレームワークの中には一般的には別名で知られているモデル、例えば、線形回帰、ポアソン回帰、指数回帰などが含まれます。ご存知のように、Stataにはこれらのモデルを推定する専用コマンドが用意されています。例えば、線形回帰にはregress、ポアソン回帰にはpoisson、指数回帰にはstregなどとなっています。

glmはデフォルトで最尤推定を用いますが、irlsオプションにより繰り返しのある再加重最小二乗法を用いることもできます。各分布族Fに対して対応するリンク関数g()が用意されていますが、IRLS推定でも推定結果は最尤推定のそれと同じものになります。ちなみにg()のことを正準リンクと呼ぶこともあります。分布族とリンク関数は

ユーザが設定するものですが、正準リンク以外のリンク関数をある分布族で選択した場合、推定結果は異なりますが、回帰係数の推定値と標準誤差の計算は適切に行われます。つまり、その推定結果は漸近的に最尤推定量と一致しますので、標本数が少ない場合は違いが目立つかもしれません。

例えば、二項分布族の正準リンクはlogitです。ですから、glm, irlsという形でコマンドを実行すると、その結果は最尤推定のlogitおよびlogisticと同じものになります。プロビットリンク関数を用いた二項分布族の場合、モデルはプロビットモデルになりますが、probitは正準リンクではありません。よって、glm, irlsの形式で推定した場合、標準誤差の推定値はStataの最尤推定を利用したprobitコマンドのそれとは微妙に異なります。

完全に一致することはありませんが、一般的にはIRLS推定値よりも最尤推定値の好まれますが、実質的な差異はありません。最尤法のprobitは漸近的性質を持っています。つまり、二項分布族におけるglm, irlsとprobitリンクの組合せは漸近的性質を持っているので、標本数が少ない場合は標準誤差は若干異なるものになります。

ここで、念のために申し上げますが、推定に専用のコマンドが用意されている場合は、そちらをご利用ください。glm, irlsのコマンドに相当するものはすでにStataに専用コマンドが用意されているケースが殆どです。そちらを利用した方が利用法を誤ったりするリスクを避けることができます。例えば、logit, probit, poissonなどは簡単に利用でき、しかも、推定後の検定や診断もスムーズに行えるようになっています。

それらのコマンドにおいて用意されていない分布族とリンク関数の組合せが必要な場合にglmをご利用ください。

また、glmには残差において不均一分散や自己相関がある場合でも一致性のある分散共分散行列の推定量を提供する機能が用意されているケースもあります。さらに言えば、glmの場合、glmならでは利用可能な分散共分散推定量があります。また、glmではVCEの計算において最尤推定を用いる手法も用意されています。

27.10 選択モデル

選択モデルは、アウトカムが選択肢となっているモデルです。例えば、消費者がいくつかの異なるブランドから、どのシリアルを選択するかをモデル化します。Stataの選択モデルコマンドは二種類、離散選択モデルと順序選択モデルがあります。一人一人が単一の選択肢を選ぶ、例えば、シリアルを1箱選んで購入する場合、データは離散選択データです。一人一人が選択肢を順序付けて、例えば、消費者がシリアルを好みによって順序立てていれば、そのデータは順序選択データです。

二値アウトカム、カテゴリアウトカム、パネルデータ、マルチレベルモデル、ベイズ推定などは、他のデータタイプとともに選択モデルの分析を行う際に、便利です、詳細は[CM] Intro4をご覧ください。以下では、選択データを利用するコマンドを説明しています。それぞれのコマンドは、選択肢固有の共変量、選択肢（前述の例でのシリアル）やケース（個人）毎に異なる共変量を許容します。さらに、これらのモデルでは、選択肢のサブセットから選択するようなアンバランスなデータにも対応しています。

27.10.1 離散選択モデル

離散選択データでは、Stataは、条件付きロジット（McFaddenの選択モデル）、多項プロビット、混合ロジット、パネルデータ混合ロジット、入れ子型ロジット回帰があります。これらのモデルの概要は[CM] Intro5をご覧ください。

cmclogitはMcFaddenの選択モデル、条件付きロジット回帰にフィットします。clclogitは選択肢からの独立性(IIA)、つまり選択肢を決定する確率の比は、選択肢が新たに追加されたり、削除されても変化しない、ということを前提としています。[CM] Intro8をご覧ください。

混合ロジットモデル、多項プロビットモデルおよび入れ子型ロジットモデルでは、様々な方法でIIAを緩和しています。

cmmxlogitは選択モデルに混合ロジット回帰でフィットします。このモデルでは、1つ以上の選択肢固有のランダム係数を許容します。このランダム係数によって、選択肢の相関を許容し、IIAの制約を緩和します。cmxtmixlogitはパネルデータを、このモデルをフィットします。

cmmprobitは多項プロビット選択モデルにフィットします。cmclogitと同様に、固定係数を推定します。このモデルは異なる選択肢の誤差項の相関を直接モデル化するため、IIAの制約を満たすを必要がありません。

nlogitは入れ子型ロジット選択モデルにフィットします。このモデルでは、選択肢の誤差項が相関しがちな、近しい選択肢を入れ子型グループとします。このモデルでは、同じ入れ子内の相関を報告します。

27.10.2 順序選択モデル

順序選択肢では、Stataは順序ロジットと順序プロビットモデルを提供します。これらのモデルの概要は[CM] Intro6をご覧ください。

cmrologitは順序ロジットモデルにフィットします。このモデルはPlackett-Luceモデル、exploded logitモデルまたは、選択ベースの結合型分析としても知られています。このモデルはIIAの制約を満たしていることが必要です。また、このモデルのユニークな点は、選択肢が固有ではないということです。これらの選択肢は選択肢固有の共変量によって区別されます。

cmroprobitは順序選択肢を利用した多項プロビット選択モデルの拡張である、順序プロビットモデルにフィットします。選択肢固有とケース固有変数をの両方を許容します。このモデルでは、IIAの制約を仮定しない代わりに、選択肢間の誤差の相関をモデル化します。

27.11 正確推定量

正確推定量とは、漸近的公式を利用して推定を行うのではなく、十分統計量の条件付き分布を利用して推定値を求める手法です。推定には条件付き分布を用いた最尤法を利用します。この方法の場合、標準誤差は推定できませんが、信頼区間は求めることができます。

バイナリデータにおけるロジスティックモデルをこの手法で推定する場合はexlogisticを利用します。

カウントデータにおけるポアソンモデルをこの手法で推定する場合はexpoissonを利用します。

小標本の場合、正確推定値の方が、一般的な、漸近的分布による推定値よりも優れています。ただし、正確推定値は推定値、検定、そして共変量の信頼区間の算出だけしか実行できません。ここでいう共変量とは観測可能なアウトカムを完全に予測できる変数のことを指します。

27.12 内生性のある共変量を伴うモデル

共変量がモデルで観測できない要素と相関しているとき、その共変量は内生性があると言います。内生性は、測

定誤差、リグレッサと相関のある変数の省略、同時性など数多くのケースを包含します。Stataはアウトカムの種類や内生性の原因となる相関のモデル化の種類に対応した、様々なコマンドを提供します。

内生性に対する解決策は操作変数を使用した分析法を頼りに行います。操作変数とは、モデルで観測できない要素と相関せず、アウトカムへの関連が内生性のある変数を介した形で存在する変数です。

ivregressは、操作変数を用いた二段階最小二乗法、制限情報最尤推定法、一般化モーメント法（GMM）の一種を用いて線形アウトカムモデルのフィットを行います。3つの推定法は、効率性、堅牢性のほか、誤差項に課す制約などに関する仮定に違いがあります。

ivprobitは、1つ以上の共変量に内生性のある場合のプロビットアウトカムモデルのフィットを行います。ivtobitは、ivregressとほぼ同様のフィットを割合や比率などのアウトカムに対して行います。ivpoissonは、1つ以上の共変量に内生性のある場合のポアソンアウトカムモデルのフィットを行います。カウントアウトカムのほか、非負な連続アウトカムにも利用できます。ivqregressは、1つ以上の共変量が内生的に決定される場合に、分位数回帰モデルをフィットするために使用されます。

ivregressに実装されているGMM推定量は、gmmで行える推定の特別な形です。ほかの関数形については、gmmに自作のモーメント評価プログラムを記述したり、代数を使用したモーメント条件を入力したりできます。詳細は[U] **27.24 一般化モーメント法(GMM)**をご覧ください。

拡張回帰モデルは、内生性のある共変量を伴う二値、順序、センサード型、連続の各アウトカムモデルのフィットを行います。eregressは内生性のある共変量を伴う線形モデルのフィットを、eintregは内生性のある共変量を伴う区間回帰を、eprobitは内生性のある共変量を伴うプロビットモデルを、eoprobitは内生性のある共変量を伴う順序プロビットモデルを、それぞれフィットします。これらのコマンドは内生性のあるサンプルセレクション（詳細は[U] **27.13 内生性サンプルセレクションモデル**を参照）、さらに処置効果（詳細は[U] **27.20 因果推論**を参照）を同時に扱うこともできます。

内生性のある共変量を伴う線形推定式の連立方程式モデルには、reg3で三段階最小二乗法（3SLS）推定法を用いた制約付きあるいは制約なし推定を行うことができます。[U] **27.25 構造方程式モデリング(SEM)**で議論する構造方程式モデリングおよび[U] **27.24 一般化モーメント法(GMM)**で議論するGMM推定量は、こうした連立方程式モデルに広く用いられています。

27.13 内生的サンプルセレクションモデル

サンプルとして誰が観測されるかに影響を与える観測できない要素が、アウトカムに影響を与える同じく観測できない要素と相関しているとき、内生性サンプルセレクションがあると言います。このとき、内生性サンプルセレクションモデルの構築が望まれます。下記に紹介するコマンドの一つを使用しましょう。

サンプルセレクションバイアスのある線形回帰モデルのことをHeckmanモデルと呼びます。$y_j = \mathbf{x}_j\boldsymbol{\beta} + \epsilon_j$はあるイベントが発生しない限り観測できないものとします。そのイベントの発生確率は$p_j = F\left(\mathbf{z}_j\boldsymbol{\gamma} + v_j\right)$とします。$\epsilon$と vには相関があり、 $\mathbf{z}_j$と$\mathbf{x}_j$には共通の変数が含まれるものとします。

heckmanは最尤推定、または、Heckmanの2段階法でモデルフィットを実行します。

この手法は近年研究は進み、線形回帰モデルではなく、プロビットモデルで利用されるようになりました。そのコマンドはheckprobitです。heckoprobitはサンプルセレクションバイアスのあるデータで順序プロビットモデルのフィットに利用します。heckpoissonは内生性サンプルセレクションの問題が内在するカウントデータに用いることができます。

Stataの拡張回帰モデルコマンドは内生性サンプルセレクションに加えて、内生性のある共変量および処置効果の場合をモデル化できます。対応コマンドは[U] **27.12 内生性のある共変量を伴うモデル**で議論されています。

27.14 時系列データのモデル

ARIMAモデルとは自己回帰和分移動平均モデルのことであり、コマンドはarimaです。これにより、撹乱項がARIMA型のモデルをカルマンフィルタと最尤法を利用して推定します。これらのモデルでは任意の共変量を利用できます。arimaコマンドでARMAモデルを推定するこもできます。

ARFIMAは自己回帰移動平均モデルで、部分的に和分関係があるモデルを指しますが、こちらは長期記憶過程に対応したモデルです。ARFIMAモデルはARMAとARIMAを一般化したものです。ARMAは短期記憶性だけを考えたもので、ショックが発生しても比較的早い段階でトレンドに回帰します。ARIMAモデルではショックは永続し、記憶性は維持されるものと考えます。ARFIMAモデルは確率過程の記憶を中期的なレンジで考えたものです。ARFIMAモデルの推定にはarfimaコマンドを利用します。ワンステップおよびダイナミック予測に加え、arfimaコマンドは部分的に和分関係を

示す変数の予測を行えます。

UCMとは非観測のコンポーネントモデルを意味し、時系列シリーズをトレンド、季節性、周期性の成分、そして独自成分に分解します。この分解に際しては外生変数が利用できます。UCMコマンドは柔軟かつ公式な計算手法でデータをスムージングし、分解します。UCMモデルの推定にはucmコマンドを利用します。

ARIMA, ARFIMA, UCMの推定したパラメータは、インプライドなスペクトル密度という観点で理解できます。psdensityコマンドを利用すると結果を変換できます。詳細は[TS] **psdensity**を参照してください。

バンドパスやハイパスフィルタを利用して時系列データをトレンドと周期成分に分解することも可能です。最もこれを行うためのtsfilterコマンドは推定コマンドではありません。Baxter-King, Butterworth, Christiano-Fitzgerald, そしてHodrick-Prescottフィルタが必要な場合はこのコマンドを利用します。

Prais-Windtenまたはコクラン-オーカット変換を利用して撹乱項のAR(1)モデルをす推定する場合は、praisコマンドを利用します。Hildreth-Lu searchの方法と同じく、2つの手法はどちらも2段階で、繰り返し計算の手法を利用します。

neweyはNewey-Westの分散推定値を利用する時のコマンドです。これは設定した次数における系列相関と不均一分散に対して堅牢な推定量を提供するコマンドです。

Stataには一変量および多変量のARCH, GARCHモデルの推定量を求めるコマンドが用意されています。これらのモデルはボラティリティの時間変化をモデル化したものです。ARCHモデルは分散のラグ項を利用して条件付きの不均一分散を考慮したモデルです。GARCHモデルはARCHモデルに撹乱項の、ラグ付き二次モーメントを加えたものです。ARCHは自己回帰条件付き不均一分散の略称です。GARCHはARCHを一般化したものという意味です。

一変量のARCHおよびGARCHモデルを推定する場合はarchコマンドを利用します。乗数型の条件付き不均一分散など、このコマンドには様々な拡張形が用意されてします。撹乱項は正規分布、またはt分布、もしくは一般化誤差分布に従うものとします。堅牢な標準誤差もオプションとして用意されています。

mgarchは多変量ARCHおよびGARCHモデルをフィットするコマンドです。分散共分散行列のオプションとしてvech、定数、条件付き可変相関モデルが用意されています。撹乱項の分布は多変量正規分布、または多変量t分布を仮定します。堅牢な標準誤差の計算もオプションで用意されています。

Stataには複数の時系列データによるVAR, SVAR, VECモデルを推定するコマンドも用意されています。VARとSVARは定常時系列データを扱うものであり、SVARはVARモデルに制約を課したモデルです。SVARはその制約下でのインパルス応答も行えます。VECは共和分関係を含んだVARモデルです。VARはベクトル自己回帰モデルの略称です。SVARは構造VARの略称です。そしてVECはベクトル誤差修正モデルの略称です。

VARモデルの推定はvar、SVARモデルにはsvar、そしてVECモデルの推定にはvecを利用します。これらのコマンドに関しては同じコマンドで検定や予測、そしてパラメータの解釈を行えます。varとsvarの詳細は[TS] **var intro**を、vecに関しては[TS] **vec intro**を、そしてインパルス応答関数と予測誤差の分散分解については[TS] **irf**をご参照ください。ラグ次数の選択、残差の分析、グレンジャーの因果性検定に関しては[TS] **var intro**(varとsvar)と[TS] **vec intro**を参照してください。

lpirfは、ローカルプロジェクションを介してインパルス応答関数を推定します。ローカルプロジェクションは、VARモデルの適合後に得られるモデルベースのインパルス応答推定に対する柔軟な代替手法を提供します。ローカルプロジェクションは、推定と仮説検定を簡素化し、対応する信頼区間はVARの推定値よりも小標本特性が向上する場合があります。

カルマンフィルタを用いて多変量の状態空間モデルを推定する場合はsspaceコマンドを利用します。時系列モデルにおいて状態空間モデルを利用することのメリットは多様なモデルを構築できる所にあります。例えば、ベクトル自己回帰移動平均(VARMA)モデル、ダイナミックファクタ(DF)モデル、構造時系列(STS)モデルなどを推定できます。ある種の確率的なダイナミック-プログラミング問題を解くこともできます。

ダイナミックファクタモデルのパラメータを推定する場合はdfactorコマンドを利用します。多変量時系列において状態空間モデルを利用すれば、観測可能な出力および観測不可能な出力に対してベクトル自己回帰モデルを推定できます。また、アウトカムの観測/非観測に関係なく外生的な共変量も利用できます。

時系列データはときどき、平均や分散のシフトで特徴づけられることがあります。そうした異常な特性はもはや線形自己回帰モデルで十分にとらえきることが難しくなります。マルコフスイッチングモデルおよび閾値モデルは、こうした時系列データのケースのフィットを行えます。

マルコフスイッチングモデルは、非観測な状態に応じた一連の変化に対して用いられ、状態に応じて過程が異なる変化をする場合をモデル化します。変化はマルコフ過程に従って起こります。状態間の遷移時間、および遷移と遷移の間の滞在時間はランダムです。一方、閾値モデルは、閾値で仕切られる領域の間の遷移するデータに用いることができます。Stataのmswitchコマンドはマルコフスイッチングダイナミック回帰(MSAR)モデルとマルコフスイッチング自己回帰(MSDR)モデルでのフィッティングを行います。MSDRでは、状態ベクトルが自己回帰ラグに依存しないため、MSARよりも高次のラグを含めることができます。thresholdコマンドは閾値モデルのフィットを行うことができます。

bayes: var コマンドを用いてVARモデルのベイズ推定を行うことができます。ベイズ推定を使用すると、実際によく利用されるモデルパラメータに関する外部情報を組み込むことができ、データのサイズに比べて多くのモデルパラメータが存在する場合に、より安定した推論が可能になります。[BAYES] **bayes: var**のAdvantages of Bayesian VAR modelsを参照してください。

27.15 パネルデータモデル

ここで扱うコマンドはすべてxtという文字で始まります。xtコマンドを使用するには事前にxtsetでデータ設定を行う必要があります。

27.15.1 連続アウトカムのパネルデータ

xtregは次のモデルをフィットします。

$$y_{it} = \mathbf{x}_{it}\beta + v_i + \epsilon_{it}$$

xtregはビトウィーン回帰推定量、ウィズイン回帰(固定効果)推定量、一般化最小二乗法 (GLS) ランダム効果(ビトウィーンとウィズインの行列加重平均)推定量の算出に利用します。また、最尤ランダム効果推定量の計算もサポートしています。wildbootstrap xtregはワイルドクラスターブートストラップP値と頑健な推定による信頼区間とウィズイン回帰推定量を算出します。

xtgeeはPA(population-averaged)モデルのフィットを行うコマンドで、オプションを利用すれば分散の堅牢な推定値も算出できます。xtgeeはいくつかの相関構造をサポートしています。特にunstrucutured という名前で呼ばれているデータの分析には適しています。ウィズイン-パネル相関のデータは何も制約を掛けることなく推定可能です。詳細は[U] 27.15.4 パネルデータにおける一般化線形モデルを参照してください。このモデルは線形回帰という制約に縛られることはありません。

xtfrontierはパネルデータで確率的生産/消費関数を推定するコマンドです。時間に対して不変なモデル、または、時変型の減衰モデルを選択してください。どちらのモデルにおいても非負の非効率項は切断型の正規分布に従うものとします。時間に対して不変なモデルの場合、非効率性は定数となります。時変減衰モデルの場合、非効率項は特定の時間関数を乗じた切断型正規確率変数としてモデル化されます。どちらのモデルでも級内撹乱項は正規分布すると仮定します。パネル特有の効果はランダムな非効率項となります。

xtheckmanは内生性を伴い、標本選択を行う変量効果モデルにパネルデータをフィットします。主要なアウトカムの方程式と選択方程式に変量効果が含まれ、相関を許容します。

xtivregはビトウィーン回帰2SLS推定量、ウィズイン回帰2SLS推定量、1階差分2SLS推定量、および2種のパネルデータランダム効果2SLS推定量を用いることができ、内生性のある共変量を含んだモデルでの推定ができます。

xteregressは内生性を伴う共変量の組み合わせ、標本選択、ランダムでない処置割り当てを伴うモデルにフィットします。

xtdidregressは観測データから差分の差分(DID)または三重差分(DDD)によって処置における平均処置効果(ATET)を推定します。連続アウトカムにおける二値または連続の処置の平均処置効果は時間とパネルの固定効果を含む線形モデルにフィットすることで推定されます。xthdidregressはxtdidregressの拡張です。異なる時間、異なる処置コホートの複数のATETを推定します。処置コホートは異なる時点で処置を受けたグループです。

xthtaylorコマンドは操作変数推定量を利用して次の形式のパネルデータにおけるランダム効果モデルを推定します。

$$y_{it} = \mathbf{X}_{1it}\beta_1 + \mathbf{X}_{2it}\beta_2 + \mathbf{Z}_{1i}\delta_1 + \mathbf{Z}_{2i}\delta_2 + u_i + \epsilon_{it}$$

個別効果u_iは説明変数$\mathbf{X}_{2it}$および$\mathbf{Z}_{2i}$と相関していますが、$\mathbf{X}_{1it}$および$\mathbf{Z}_{1i}$とは無相関です。ここで、$\mathbf{Z}_1$と $\mathbf{Z}_2$はパネルにおける定数です。

xtglsは次式における一般化最小二乗推定値を求めます。

$$y_{it} = \mathbf{x}_{it}\beta + \epsilon_{it}$$

ここでユーザはϵ_{it}の構造を決定する必要があります。仮にϵ_{it}がすべてのiとtに関して独立であると設定すると、xtglsの計算結果はregressコマンドで小標本の自由度修正を行った場合と同じ結果になります。

iとtに関して、それぞれ3つの分散構造が仮定できますので、結局、9種類のモデルから選択を行うことになります。iは不均一分散とクロスセクション方向の相関に関する仮定の設定を行うものです。また、tは自己相関、より正確に言えばAR(1)の自己相関の仮定に関するものです。

これら以外にもOLS係数とGLSによる分散共分散推定量を算出するコマンドが用意されています。xtpcseは線形クロスセクション時系列モデルにおいてパネル修正標準誤差(PCSE)を算出します。パラメータの算出にはOLSまたはPrais-Winsten回帰を用います。標準誤差と分散共分散推定値を計算する場合、デフォルトで撹乱項は不均一分散であり、パネル間で同時相関を有することが前提となっています。

Swamyのランダム係数線形回帰モデルを推定する場合はxtrcコマンドを利用します。このモデルではグループ間で切片が変化するだけでなく、係数も変化させることができます。

パネルのマルチレベル、ランダム係数、そして分散要素推定に利用するxtregを一般化したモデルおよびxtrcに関しては[U] 27.16 マルチレベル混合モデルを参照してください。xtrcはmixedの特殊形です。

27.15.2 センサード型アウトカムのパネルデータ

xttobitはランダム効果のあるtobitモデルを推定し、データごとのセンサード情報にも対応するコマンドです。

xtintregはランダム効果間隔モデルを推定し、データごとのセンサード情報にも対応するコマンドです。間隔回

帰は開区間と閉区間の両方に対応しています。

これらのモデルはマルチレベルデータへの一般化が可能です。詳細は[U] 26.15 マルチレベル混合モデルを参照してください。

27.15.3 離散アウトカムのパネルデータ

xtprobitは最尤法を利用してランダム効果プロビットモデルを推定します。また、GEEを利用したPAモデルの推定もサポートしています。これは二項分布族とプロビットリンク関数を用いて、交換型の誤差構造モデルをxtgeeで推定するものです。

xtlogitは最尤法を利用してランダム効果ロジスティックモデルを推定します。条件付き固定効果モデルを最尤法でフィットすることもできます。また、xtprobitはGEEを利用してPAモデルを推定します。

xtcloglogは最尤法を利用してランダム効果補間ログログ回帰モデルを推定します。

GEEを利用してPAモデルを推定することもできます。

xteprobitは内生性を伴う共変量の組み合わせ、標本選択、ランダムでない処置割り当てを伴う変量効果プロビットモデルにフィットします。

xtologitとxtoprobitは複数のアウトカムに対応したモデルを推定します。xtologitはランダム効果順序ロジスティックモデルを、xtoprobitはランダム効果順序プロビットモデルを推定します。

xteoprobitは内生性を伴う共変量の組み合わせ、標本選択、ランダムでない処置割り当てを伴う変量効果順序プロビットモデルにフィットします。

xtmlogitは最尤法によりランダム効果多項ロジスティック回帰モデルにフィットします。また、最尤法により条件付き固定効果モデルにもフィットします。

xtpoissonは2種のランダム効果ポアソン回帰モデルを最尤推定法を利用してフィットします。ランダム効果の分布に関して、ガンマ分布と正規分布の2種類を設定できます。xtpoissonは条件付き固定効果モデルのフィット、およびGEEを利用したPAモデルのフィットが行えます。

xtnbregはランダム効果非負二項回帰モデルを最尤推定法を利用してフィットします（ランダム効果の分布にはベータ分布を仮定します）。xtnbregはxtpoissonは条件付き固定効果モデルのフィット、およびGEEを利用したPAモデルのフィットが行えます。

xtprobit、xtlogit、xtcloglog、xtpoisson、xtnbregはxtgeeで適切な分布族とリンク関数を指定し、可換型の誤差構造を設定したものにほかありません。詳細は[U] 27.15.4 パネルデータにおける一般化線形回帰を参照してください。

上記のモデルはマルチレベルデータに一般化できます。詳細は[U] 27.16 マルチレベル混合効果モデル参照してください。

27.15.4 パネルデータにおける一般化線形回帰

[U] 27.7 一般化線形モデルでは次のようなモデルについて付いて考えました。

$$g\left\{E\left(y_j\right)\right\}=\mathbf{x}_j\beta,\quad y_j\sim F$$

ここでg()はリンク関数、Fは指数分布族で、推定に際し、両者を設定します。

一般化線形モデルをパネルデータで利用するには2つの方法があります。一般化線形混合モデル(GLMM)と一般化推定方程式(GEE)です。

GEEは相関構造を利用してウィズインパネルの相関をモデル化します。GEEはxtgeeコマンドで推定します。詳細は[XT] xtgeeを参照してください。

パネルデータを含む、マルチレベルデータにおける一般化線形モデルの詳細はマルチレベル混合効果モデルを参照してください。[U] 27.16 マルチレベル混合効果モデルを参照してください。

27.15.5 パネルデータにおける生存時間モデル

xtstreg変量効果パラメトリック生存時間モデルに対し最尤法によるフィッティングを行います。変量効果が与えられたときの応答の条件付き分布は、指数、ワイブル、対数正規、対数ロジスティック、ガンマのうちいずれかを想定します。選択した分布によって、xtstregは比例ハザード(PH)、または加速故障時間(AFT)パラメトリゼーションを用いてモデルのフィッティングを行います。他のパネルデータコマンドと異なり、xtstregではデータに対しxtsetとstsetによる設定が行われている必要があります。

これらのモデルはマルチレベルモデルでも利用できます。詳細は[U] 27.16 マルチレベル混合効果モデル参照してください。

27.15.6 パネルデータにおけるダイナミック自己回帰モデル

xtregarはϵ_{it}がAR(1)過程に従うと仮定した時に、ウィズイン推定量とGLSランダム効果推定量を求めます。

xtabondはダイナミックパネルデータモデル(ラグ付従属変数のあるモデル)で利用するもので、1ステップ、1ステップロバースト、そして、2ステップのArellano-Bond推定量を推定します。xtabondは先決的な共変量に対応するコマンドで、Sarganテスト及び、Arellano and Bondの提案した自己相関の検定をサポートしています。

xtdpdsysはxtabondの拡張形で、AR過程の係数が大きいものである場合、よりバイアスの小さな推定量を求めることができます。xtpdsysはxtabondよりも効率的な推定量を算出します。xtabondが撹乱項の一階の階差を利用したモーメント条件を利用するのに対し、xtpdsysコマンドは一階の階差とレベルに対するモーメント条件を利用します。

xtdpdはxtdpdsysの拡張形で、ダイナミックパネルデータモデルの、より広範囲なクラスのパラメータを推定します。xtdpdは級内誤差に系列相関があるモデルの推定に対応しています。一方、xtdpdsysとxtabondは系列相関は存在しないことを仮定しています。また、xtdpdは先決変数の構造が、xtdpdsysやxtabondで仮定されているものより複雑である場合に利用します。

27.15.7 ベイズ推定

bayesプリフィクスコマンドにより、パネルデータのランダム効果モデルのベイズ推定を行うことができます。[BAYES] **Bayesian estimation** の Bayesian panel-data commands を参照してください。実際によく利用されるモデルパラメータの外部情報を組込みたい場合や少ないデータから多くのパラメータを推定した場合など、ベイズ推定に関心を持つかもしれません。ベイズ推定は、変量効果の推定に関心がある場合に特に役立ちます。これは、各変量効果のモデルパラメーター推定に変動のすべての原因を組み込んだ事後分布全体を提供するためです。さらに、bayespredictコマンドを使用して、モデルパラメータ推定値の漸近正規性の仮定に依存することなく、予測とその変動性を計算できます。[BAYES] **Bayesian estimation** の Panel-data modelsを参照してください。

27.16 マルチレベル混合効果モデル

マルチレベルデータの場合、観測値をある共通した性質ごとにグループに分けてモデル推定します。例えば、データの観測対象が生徒である場合、グループとして学校というものが考えられます。患者ならば、グループは病院となり、農耕用トラクタならば製造工場をグループと考えます。その共通した性質がどんなものであれ、モデル化するアウトカムに対して、その共通要素が何らかの影響を及ぼすと考えるのは自然なことです。

学生と高校のケースで言えば、卒業後の進路や就職についてモデル化してみましょう。つまり、学校単位で比較した場合、優秀な学校ほど、生徒の進路に良い効果を及ぼすと考えられます。患者と病院のケースならば、ある病院の患者のその後の健康状態が良好ならば、その病院が特定の問題に対して、他に比べ、優れた医療を提供していると考えれます。農耕用トラクタと工場の例で言えば、優れた工場で生産されたトラクタは信頼性が高いと想像できます。

これらのようなデータは2レベルデータと呼びます。1stレベルは学生、患者、トラクタで、2ndレベルは高校、病院、工場となります。観測値はグループにネストしていると言います。つまり、高校の下に生徒、病院の下に患者、そして工場の下にトラクタというデータ構造を考えます。

アウトカムに対する効果が直接観測できない場合でも、その効果がグループ内のすべての観測値に対して等しく影響すると仮定することによって、効果をコントロールすることが可能です。グループごとの効果はモデルの残差および他のグループの効果と無相関であると仮定した時に、効果はある統計的分布から確率的に出現するとしてモデル化します。

マルチレベルを具体的に説明します。

少し複雑ですが3レベルモデルの例を考えます。すなわち、生徒の上に教師、そして上位に学校と考えます。同様に、患者、医師、病院という階層や、トラクタ、生産ライン、工場というものが考えられます。

上位階層データのほかに横方向のクロス型データというものも考えられます。つまり、労働者、職業、産業分野という形です。

Stataには次に示すようなマルチレベル分析用のコマンドがあります。一覧として示します。

コマンド	アウトカム変数	対応関係
mixed	連続変数	線形回帰
menl	連続変数	非線形回帰
metobit	センサード	トービット回帰
meintreg	センサード	区間回帰
meprobit	バイナリ	プロビット回帰
melogit	バイナリ	ロジスティック回帰
mecloglog	バイナリ	補二重対数回帰
meoprobit	順位カテゴリ	順位プロビット回帰
meologit	順位カテゴリ	順位ロジスティック回帰
mepoisson	カウント	ポアソン回帰
menbreg	カウント	負の二項回帰
mestreg	生存時間	パラメトリック生存時間回帰
meglm	多様なアウトカム	一般化線形モデル

上記の推定量はランダム切片とランダム係数を提供し、係数と分散要素にそれぞれ制約を設定できます。(非線形回帰の場合、制約は利用できません。)

詳細は[ME] *Stata Multilevel Mixed-Effects Reference Manual*と[ME] **me**を参照してください。

27.17 生存時間(故障時間)モデル

比例ハザードモデル、競合リスク回帰、そしてパラメトリックな生存モデル用のコマンドが用意されています。パラメトリックな生存モデルとしては指数関数、ワイブル関数、ゴンペルツ関数、対数正規関数、対数ロジスティック関数、一般化ガンマ関数をサポートしています。コックスのコマンドはstcoxです。コックス回帰は左または右打ち切り生存時間データにフィットします。区間打ち切り生存時間データに対してstintcoxはセミパラメトリックコックス比例ハザードモデルをフィットするのに使われます。パラメトリックモデルはstregで右側センサードデータの生存時間データへ、stintregで区間センサード生存時間データへ、それぞれフィットすることができます。

stcoxおよびstregは故障が1回または複数回発生するデータに利用します。一方、競合リスク回帰のコマンドstcregおよびstintregは故障が1回だけの場合に利用します。これらの4つのコマンドにおいては一般的な標準誤差の他に堅牢な標準誤差、ブートストラップ法やジャックナイフ法を利用した標準誤差をサポートしています。ただし、stcrregにおける堅牢な標準誤差は、いわゆる一般的な標準誤差に相当します。

コックスモデルとパラメータモデルはどちらも、次に示す一般化に対応しています。最初に、どちらも潜在的なランダム効果をサポートしています。2つ目として、両方ともベースラインハザード関数において層化に対応しています。その時のベースラインハザード関数は層の設定により大きく変わります。パラメトリックモデルでは補助的パラメータのモデリングも可能です。

数種類の故障が発生するような状況、言い換えれば、故障イベントが互いに競合しているようなケースでは比例ハザードモデルよりも競合リスク回帰の方が実用的です。競合リスクの存在する場合、競合リスク回帰は関心のある故障の発生に及ぼす共変量の効果を分析できます。故障の種類に関し独立性という強い仮定を置く必要はありません。

stcox, stcrreg, stregを利用する場合は事前の準備としてデータをstsetします。stsetを施しおくと、Stataは時間とセンサードデータを認識します。ユーザは単純にstcox,stcrreg,stregに対し、変数とオプションを設定するだけです。stcrregコマンドを利用する場合は、stsetで設定した故障に関し競合するイベントの設定を行います。stintcoxとstintregは区間センサード時間変数の指定を必要とするため、stsetでの設定は無視されます。

Stataは観測された生存時間データを基に、平均処置効果や処置群に対する平均処置効果を推定するコマンドが備わっています。詳しくは[U] 27.20 因果推論をご覧ください。

パネルデータ生存時間モデルについては[U] 27.15.5 パネルデータ生存モデルで議論しています。これらのモデルはマルチレベルモデルでも利用できます。詳細は[U] 27.16 マルチレベル混合効果モデル参照してください。

27.18 メタ分析

メタ分析は同じようなリサーチクエスチョンを用いた複数の異なった研究の結果を統合する統計手法です。メタ分析の目標は、研究結果を比較し、可能であれば、推定結果から統合された結論を導き出すことです。Stataはメタ分析を行う一連のコマンドを提供します。

メタ分析の主要な要素は研究固有の効果量とそれに対応する標準誤差です。これらは、meta setまたはmeta esizeを使ってメタ分析の設定を行うステップ([META] meta data)で指定します、詳細は[META] meta set と [META] meta esizeをご覧ください。

全体の効果量と信頼区間および異質性統計量を含む、基本的なメタ分析のサマリーは表形式（[META] **meta summarize**）、またはフォレストプロット（[META] meta forestplot）として表示できます。3つのメタ分析モデル、変量効果、固定効果、共通効果と制限付き最尤法やMantel-Haenszel法などの複数の推定モデルをサポートしています。

メタ分析において、異質性やbetween-studyの分散は頻繁に発生します。これらはフォレストプロットのサブグループを使用するか、GalbraithまたはL'Abbéプロットを使用してグラフィカルに探索できます。[META] **meta forestplot**や[META] **meta galbraithplot**や[META] **meta labbeplot**の subgroup()を参照してください。これらは、メタ回帰とサブグループ分析を通して観測されます。詳細は、[META] meta regress、[META] **meta summarize**のsubgroupオプションおよび[META] **meta forestplot**をご覧ください。meta summarizeまたはmeta forestplotと、cumulative()オプションを使用して累積メタ分析も可能です。Leave-one-outメタ分析は、これらのコマンドとleaveoneoutオプションを指定することで実行できます。

出版バイアス存在も、メタ分析における関心事項の一つです。これは、研究結果の有意さによって、それを出版するかを決定する際に、生じる典型的なものです。有意でない結果を示した研究は出版バイアスとなりがちです。輪郭強調ファンネルプロット（[META] meta funnelplot）、ファンネルプロットの左右対称性の検定（[META] **meta bias**）、およびtrim-and-fill法（[META] **meta trimfill**）を利用して出版バイアスを検出し、メタ分析結果への影響を確認します。さらに、meta funnelplotとmeta biasは所謂、小規模研究の効果量(small-study effect size)、または、小規模研究によってより大きな効果量が報告される傾向を調べることができます。

研究が複数の効果量を報告する場合、効果量間の依存性を考慮しながら、meta mvregressを使用して多変量メタ分析を実行できます。また、モデルに共変量を組み込み、多変量メタ回帰を実行して、効果量の多変量不均一性を調査することもできます。

さらに、効果の大きさの間の依存関係が階層的な構造や多層構造に由来する場合、meta meregressを使用してマルチレベルメタ分析を実行することができます。meta meregressを使用すると、ランダムな切片とランダムな傾斜を持つモデルを適合させることができます。ただし、ランダム切片のメタ分析モデルのみを適合させることに興味がある場合は、よりシンプルな構文を提供するmeta multilevelを使用することができます。

その他にも、metaコマンドでは、メタ回帰後の予測機能やバブルプロット（[META] **meta regress postestimation** および[META] meta mvregress postestimation, [META] meta me postestimation）などの様々な推定後ツールがあり

ます。

27.19 空間自己回帰モデル

StataのSp推定コマンドは空間自己回帰（SAR）モデル（同時自己回帰モデルとしても知られる）をフィットします。コマンドを用いて、従属変数、独立変数の空間ラグを追加できるほか、空間自己回帰誤差も設定できます。時系列分析において、ラグは直前の時間を表します。空間分析において、ラグは隣接エリアを表します。

SARモデルのモデル選択で重要になるのが、空間ラグの定式化です。空間ラグは空間重み行列を用いて指定します。潜在的に大きい行列であることから、Stataでは空間重み行列を作成、読み込み、保存するコマンドを提供しています。

空間モデルはある単位空間（エリア）から別の単位空間への溢れの効果を推定します。また、空間モデルでない推定のように直接効果を推定することもできます。直接効果は単位空間内の効果を示します。任意のSp推定コマンドの実行後、直接効果、間接効果、総合効果の推定値の確認がestat impactで行えます。推定結果の解釈が簡単に行えます。

SARモデルのデータセットは地理的なエリアやそのほかの単位ごとの観測値を含みます。単位ごとの距離を計測したデータが唯一必須になります。地理的なエリア上の空間データは、通常シェープファイルを基にします。Spシステムは、.dtaデータセット同士のマージを可能にするため、標準のshapefile形式をStataの.dtaファイルへ変換します。

Spシステムはshapefileを直接扱うこともできます。データは(x, y)座標を含むものであれば地理データ以外でも扱うことができます。ソーシャルネットワークの分析がその一例です。

[SP] **Intro**や、続く導入的なセクションを参照し、SARモデルの概要とExample付きチュートリアルを一読して、データの準備の仕方や空間重み行列の作成の仕方を確認してください。

実行可能なSp推定コマンドの一覧は以下です。

コマンド	説明	対応関係
spregress, gs2sls	GS2LSL推定量を用いたSAR	regress
spregress, ml	ML推定量を用いたSAR	regress
spivregress	内生的リグレッサを伴うSAR	ivregress
spxtregress, fe	パネルデータにおける固定効果SAR	xtreg, fe
spxtregress, re	パネルデータにおける変量効果SAR	xtreg, re
spxtregress, re sarpanel	代替的な変量効果SAR	

spregress, gs2slsおよびspivregressでは、従属変数の空間ラグ、空間自己回帰誤差項、共変量空間ラグがそれぞれ複数あるモデルのフィットも行えます。そのほかのSp推定コマンドは、従属変数の空間ラグ、空間自己回帰誤差項に関しては単一のモデルしか扱えませんが、共変量空間ラグについては複数あっても大丈夫です。

27.20 因果推論

多くの研究問題は因果関係に焦点を当てています。他の変数（しばしば処置変数と呼ばれる）に変化が加えられた場合、興味のある結果に何が起こるかを推論したいと考えています。適切な仮定の下では、Stataの多くの推定コマンドを因果推論に使用することができます。詳細については、[CAUSAL] Introを参照してください。Stataはまた、因果推論が研究の目標である場合に特に設計された推定コマンドも提供しています。

teffects、stteffects、eteffects、mediate、didregress、hdidregress、xtdidregress、xthdidregressの各コマンドは観測データから処置効果を推定します。

処置効果とは、同一の個体がある処置と、それと対になる処置を受けた場合の効果の差を計測するものです。ただし、ここで推定できるのは処置効果の平均値であり、個々の処置効果ではありません。なぜなら、観測したデータは処置のどちらか一方に限定されるためです。

teffects、stteffects、eteffects、telassoコマンドは、実際に二者択一の処置であるにも関わらず、両方の処理の効果を計測する手法を利用します。このようなアプローチは一般的に潜在的アウトカムのグレームワークと呼ばれています。観測データの分析に関連する主要概念への基礎的な導入は[CAUSAL] **teffects intro**に記述しています。さらに高度な導入は[CAUSAL] **teffects intro advanced**を参照し、潜在的なアウトカムを直感的な把握をしてください。[CAUSAL] **stteffects intro**では、先の2つの概念を生存時間へつなぎます。

mediateコマンドは、因果媒介分析においてポテンシャルアウトカムのアプローチを取ります。この推定により、治療が結果に直接的な効果を及ぼすだけでなく、媒介変数を介した間接効果も推定することができます。

didregress、xtdidregressコマンドは潜在的な結果のフレームワーク内でも理解できます。ただし、これらの推定量は、モデルにグループ効果と時間効果が含まれているという点で、teffects、stteffects、eteffectsとは異なります。 効果の推定にバイアスをかける可能性のある、観察不能なグループと時間のグループと時間の効果の制御を含めます。

hdidregressとxthdidregressは、時間や処置コホートによって変動する処置効果を推定します。処置コホートは、

異なる時点で処置を受けるグループです。これらのコマンドはdidregressとxtdidregressの拡張版ですが、これらとは異なり、1つではなく複数のATET（平均処置効果）を推定します。hdidregressとxthdidregressは、異質な処置効果の推定のためのコマンドです。

ここで、例として運動が血圧に及ぼす効果について考えてみましょう。潜在的アウトカムのフレームワークを利用して、全員が運動しない場合と、運動する場合の平均処置効果(ATE)を考察します。同様に、運動しない人と運動をした人の平均的な処置効果の差を、実際に運動した人から推定(ATET)します。最後に、仮に全員が運動していた場合の平均血圧と、運動しなかった場合の平均血圧を推定します。この時のパラメータは潜在アウトカム平均と呼びます。

teffectsはATE,ATET,POMを推定できます。teffectsに用意されているこれらの推定量は、潜在アウトカムフレームワークの構造をデータに、それぞれ異なる方法で課すものです。

- 回帰調整推定量は潜在的なアウトカム用のモデルを利用します。詳細は[CAUSAL] **teffects ra**をご覧ください。
- 処置の設定には逆確率加重推定量を利用してモデルを推定します。詳細は[CAUSAL] **teffects ipw**をご覧ください。
- 拡大逆確率加重推定量と逆確率加重回帰調整推定量を利用して、潜在的アウトカムと処置の設定モデルを推定します。これらの推定には二重の意味で堅牢な性質が備わっています。つまり、2つのモデルのうち片方だけの設定が正しければ、正確に処置効果の推定が可能になっています。詳細は[CAUSAL] **teffects aipw**、[CAUSAL] **teffects ipwra**を参照ください。
- 隣接マッチング(NNM)とプロペンシティスコアマッチング(PSM)推定量は、個体のアウトカムが極力、似ているものを比較します。もちろん、片方は処置を受けたもので、他方は受けていないものです。NNMはノンパラメトリックな手法で似たものを探します。一方、PSMは推定した処置確率を利用して似たデータを探します。詳細は[CAUSAL] **teffects nnmatch**、[CAUSAL] **teffects psmatch**を参照してください。

stteffectsはATE,ATET,POMを推定できます。stteffectsに備わるこれらの推定量は、潜在アウトカムフレームワークの構造をデータに、それぞれ異なる方法で課すものです。

- 回帰調整推定量は、潜在アウトカムのモデルを利用し、打ち切りは最尤法で調整されます。詳細は[CAUSAL] **stteffects ra**をご覧ください。
- 逆確率加重(IPW)推定量は処置割り付け、および打ち切りのある時間に対するモデルを利用します。詳細は[CAUSAL] **stteffects ipw**をご覧ください。
- 逆確率加重回帰調整(IPWRA)推定量は潜在アウトカムおよび処置割り付けに対するモデルを利用します。IPWRA推定量は、アウトカムモデルまたは別の打ち切りモデルにより打ち切りを調整します。推定には二重の意味で堅牢な性質が備わっています。すなわち、2つのモデルのうち、片方だけの設定が正しければ、正確に処置効果の推定が可能になっています。打ち切りモデルを指定した場合、処置割り付けモデルと打ち切りモデルの両方で推定量の設定が正しく行われないと、二重の意味での堅牢性は生まれません。詳細は[CAUSAL] **stteffects ipwra**をご覧ください。
- 加重回帰調整推定量はアウトカムと打ち切りになった時間をモデル化します。詳細は[CAUSAL] **stteffects wra**をご覧ください。

teffectsとstteffectsは多値トリートメントに対する処置効果を推定します。詳細は[CAUSAL] **teffects multivalued**を参照してください。

telassoは、モデルに含まれる可能性のある制御変数から選択するためにLassoを使用しながら、拡張された逆確率重み付けによって観測データからATE、ATET、およびPOMを推定します。

mediateは、処置効果を結果に対する直接効果と間接効果に分解します。直接効果は、処置が結果に直接的にどのように影響するかを示し、間接効果は、処置が媒介変数を介して結果に間接的にどのように影響するかを示します。mediateは、平均的な直接処置効果（ADTE）、平均的な間接処置効果（AITE）、処置に関する平均的な直接処置効果（ADTET）、対照群に関する平均的な間接処置効果（AITEC）、総平均処置効果（ATE）、およびPOMsを推定します。

didregressおよびxtdidregressは、差分の差分（DID）または三重差分（DDD）によって、観測データからATETを推定します。連続結果に対するバイナリまたは連続処理のATETは、線形モデルを、didregressで時間とグループの固定効果の線形モデル、xtdidregressで時間とパネルの固定効果の線形モデルに当てはめることによって推定されます。didregressは、各期間で異なるグループの個人が観察される繰り返しクロスセクションデータ用、xtdidregressはパネルデータ用です。hdidregressとxthdidregressは、didregressとxtdidregressを拡張して、時間と処置コホートごとに変動するATET（平均処置効果）を推定します。処置コホートは、異なる時点で処置を受けるグループです。

しかし、処置が内生的に決まるようなケース(潜在的アウトカムが条件付き独立でないの)でteffectsまたはstteffects、telasso、didregress、hdidregress、xtdidregress、xthdidregressを利用することはできません。処置が内生的に決まる場合は、内生的な処置効果モデルを利用して平均処置効果を推定します。これらのモデルでは、アウトカムに対して内生的に決定される、バイナリ型処置変数の効果を求めます。

eteffectsはATE,ATET,POMを推定できます。内生的処置効果モデルは、アウトカムに対し線形または非線形(プロビット、部分プロビット、指数)モデルのどちらかを用いてモデル化します。eteffectsはコントロール関数回帰調整推定量を備えます。

etregressとetpoissonもまた、内生的処置効果モデルでのフィッティングが行え、ATEおよびATETの推定に利用できます。詳細は[CAUSAL] **etregress**と[CAUSAL] **etpoisson**を参照してください。etregressはアウトカムに対する線

形モデルを利用して、内生的な処置効果モデルをフィットします。一方、etpoissonはアウトカムに対する非線形(指数)モデルを利用して、内生的な処置効果モデルをフィットします。

センサード型アウトカムの場合、eintregで外生的に割り付けられた処置の効果のほか、割り付けに内生性のある場合の処置の効果の推定も行えます。eregress、eprobit、eoprobitは、それぞれ連続アウトカム、二値アウトカム、順序アウトカムの場合の処置効果（外生・内生を問わない）を推定できます。これら4つのコマンドは、内生あるいは外生処置に加えて、内生性のある共変量を伴う場合、さらに内生性サンプルセレクションのある場合にも対応できます。詳細は[U] **27.13 内生性サンプルセレクションモデル**、または[U] **27.12 内生性のある共変量を伴うモデル**参照してください。

27.21 薬物動態データ

薬物動態データの分析には4つの推定コマンドを用意しています。コマンドの詳細は[R] **pk**を参照してください。

1. pkexamineは時間-濃度の個体データから薬物動態の評価基準を計算します。pkexamineで最大計測濃度、最大計測濃度における時点、最終計測時点、放出時点、半減期、濃度-時間曲線下の面積を計算できます。
2. pksummは各薬物動態の計測値の経験分布から4つのモーメントを計算し、それらの計測値の分布が正規分布に従うという帰無仮説の検定を行います。
3. 重複のある実験計画から得たデータを分析する際はpkcrossを利用します。また、pkcrossコマンドは薬剤の試験データを分析する際、処置、繰り越しの影響、順番などのデータが既知の場合、処置と繰越し効果の分離に関するオムニバス検定を実行できます。
4. pkequivは2通りの処置に関して生物学的同等性の検定を実行します。デフォルトではpkequivは2つの処置の平均値の差について、標準的な対称信頼区間を計算します。また、Fillerの提案したゼロと間隔に関する対称信頼区間を算出します。さらに、pkequivは生物学的同等性の間隔仮説検定を実行します。

27.22 多変量分析

多変量分析に関するコマンドは*Multivariate Statistics Reference Manual*を参照ください。

1. mvregは多変量回帰をフィットするコマンドです。
2. MANOVAとMANCOVAモデル(一元、二元、三元配置モデル)を推定する場合はmanovaを利用します。これはネスト型や混合計画モデルにも対応しています。詳細は[U] **26.5 ANOVA, ANCOVA, MANOVAそしてMANCOVA**を参照してください。
3. canonは正準相関とその負荷量を推定するコマンドです。正準相関は二変数群の関係を記述する手法です。
4. pcaは主成分と固有値、そして因子負荷量を算出するコマンドです。主成分分析は記述的なツールであり、係数と同じく標準誤差も関連していると考えられます。しかし、一方では主成分分析は単純な次元低下テクニックであると考えることもできます。
5. factorは因子モデルをフィットし、主因子、主コンポーネント因子、繰り返しのある主コンポーネント因子、そして最尤推定量を提供します。因子分析はこれら共通因子の線形結合により、元の変数$\mathbf{y}_{\square}$, i = 1, ... ,Lを示すモデルを推定します。元の変数$\mathbf{y}_i$を構成する共通因子$\hat{z}_k$, k = 1, ...,qを求めることを目的とした分析手法です。
6. pcaやfactorと関連しますが、バイナリデータに対してPCAや因子分析を利用する場合はtetrachoricコマンドを利用します。
7. factorやpcaにおいて直交、斜交回転を行う場合はrotateコマンドを利用します。回転を行うことで分析結果の解釈を容易にする場合があります。
8. 多次元スケーリングの標準的手法の一つであるプロクラステス分析を行う場合はprocrustesコマンドを利用します。変換と拡張に加え、直交、斜交回転をサポートしています。
9. mdsコマンドは変数群においてデータ間の相違性を表現する距離/非距離による多次元スケーリングを行います。相違性を表現する様々な手法をサポートしていますが、それはclusterコマンドにおける手法と同じものです。
10. caはコレスポンデンス分析のコマンドです。クロス表の分析や、行と列の関係性を探索的に、多変量を利用して行います。
11. mcaは複数コレンスポンデンス分析(MCA)と同時コレンスポンデンス分析(JCA)を行うコマンドです。
12. 平均、共分散、相関の検定に際し、多変量正規性の検定を行う場合はmvtestコマンドを利用します。
13. clusterはクラスタ分析用のコマンドです。階層的、区分的なクラスタリング手法が利用可能です。厳密に言えば、クラスタ分析は統計的推定というカテゴリには属しません。むしろ、探索的データ分析のテクニックの一種と考えられます。Stataのクラスタ分析には連続およびバイナリデータに関する同等性、相違性を表現する手法が豊富に用意されています。
14. discrimとcandiscは判別分析用のコマンドです。candiscは線形判別分析(LDA)を実行します。そして、discrimはLDA に加え、二次の判別分析(QDA), k次隣接分析(KNN), ロジスティック判別分析をサポートしています。この2つのコマンドはデフォルトの出力情報が異なります。discrimは区分サマリー、candiscは正準線形判別関数を出力します。しかし、両者とも、これらの出力を表示できます。

多変量線形モデルにおいて、観測変数と潜在変数を含める場合は、[U] 27.25 構造方程式モデリング (SEM) を参照してください。バイナリ、順序、名義尺度のアイテムおよびそれらの組み合わせに対して項目反応理論モデルを適合させる場合は、[U] 27.28 項目反応理論 (IRT) を参照してください。多変量時系列モデルについては、[U] 27.14 時系列モデルを参照してください。多変量メタ回帰モデルを適合させる場合は、[META] meta mvregressを参照してください。

27.23 最尤推定

Stataの多くの推定コマンドは、最尤推定を用いてモデルを適合させます。もしStataに希望するモデルに対するコマンドがなく、かつモデルの尤度を指定できる場合は、mlexpコマンドまたはmlコマンドのスイートを使用して最尤推定を行うことができます。mlexpはプログラミングを必要とせず、簡単に使用できるコマンドですが、各観測値の対数尤度を記述できる必要があり、全体の対数尤度は個々の対数尤度の総和である必要があります。一方、mlはパネルデータなど、これらの要件を満たさないモデルのクラスを適合させることができますが、一定のプログラミングが必要です。

27.24 一般化モーメント法(GMM)

一般化モーメント法(GMM)で推定する場合はgmmコマンドを利用します。コマンドラインを利用してモデルを入力する場合、nlまたはnlsurと同じように代用可能な表現を利用して、モーメント条件の式を入力します。モーメント条件を評価するプログラムを利用すれば、複雑な式の入力も簡単に行えます。すなわち、自分でパラメータベクトルに基づくモーメントを計算するプログラムを記述します。

gmmは一推定式または連立推定式のどちらにも対応しています。そして$E\left\{\mathbf{z}_i u_i(\beta)\right\}=0$という形でモーメント条件を入力します。ここで$\mathbf{z}_i$は操作変数ベクトルであり、$u_i(\beta)$は撹乱項です。モーメント条件を一般的な形式で書くと$E\left\{\mathbf{h}_i\left(\mathbf{z}_i;\ \beta\right)\right\}=0$のようになります。最初の記述法の場合、$u_i(\beta)$の式を書き、instruments()とxtinstruments()オプションを利用して$u_i(\beta)$を設定します。後者の場合、$\mathbf{h}_i\left(\mathbf{z}_i;\ \beta\right)$という形式で書きます。この形式の場合、自動的に操作変数を認識しますので、instruments()とxtinstruments()オプションを利用する必要はありません。

gmmはクロスセクション、時系列、パネルデータに対応しています。推定にあたってはiid誤差を実現するための加重行列とVCEを利用できます。それにより不均一分散にも対応できます。また、観測値にクラスタが存在したり、残差に不均一分散と自己相関が存在する(HAC)場合にも対応できます。HAC加重行列およびVCEの場合、gmmではバンド幅の設定や、自動バンド幅の選択アルゴリズムが利用可能です。

27.25 構造方程式モデリング(SEM)

SEMとは構造方程式モデリング(Structural equation modeling)の略です。SEMをフィットするコマンドにはsemとgsemの2つがあります。

semは標準的な線形SEMをフィットする場合に利用します。gsemはいわゆる一般化SEMをフィットするコマンドです。ここで一般化とは一般化線形応答モデルやマルチレベルモデリングを意味します。

もちろん、一般化線形モデルとはprobitやlogitなどの二値応答モデル、ポアソン分布や負の二項分布などのカウンド応答モデル、多項ロジットのようなカテゴリ応答モデル、順序プロビット、順序ロジットのような順序応答モデルなどのことを指します。ですから、線形応答モデルは一般化線形モデルに含まれることになります。

マルチレベルモデルは、例えば、医師に対する患者、病院における医師とその患者などのようなネストした効果を考慮したモデルです。また、職業と業種などの交互効果をモデルに含むことができます。

それではもう少し具体的にsemコマンドについて説明します。semは線形回帰から同時方程式、確認的因子分析モデル(CFA)、相関特異モデル(CUM)、潜在成長モデル、多重指標多重原因(MIMIC)モデルまで、フィットできます。推定結果は標準化/非標準化、直接/間接効果、適合度統計量、修正指標、スコア検定、ワルド検定、パラメータの線形/非線形検定、そして、パラメータの線形/非線形な関係式の検定をサポートしています。もちろん、信頼区間の情報も提供します。モデルを簡単に設定し、グループの横断的な推定と、グループの不変性に関する検定を簡単に行うことができます。推定と検定は元データや要約統計量を用いて行われます。さらに、semにはオプションとして全情報最尤推定(FIML)をサポートしていますので、欠損値のあるデータにも対応できます。

gsemはフィット可能なモデルをより拡張したものです。応答としては連続、序数、度数、カテゴリ、生存時間をサポートし、gsemはマルチレベルモデリングにも対応しています。潜在変数を任意のレベルに配置できます。よって、ランダム切片とランダム勾配のモデルも推定できます。もちろん、これらのランダム効果はネストさせたり、横断的に配置できます。

機能としてsemとgsemには重複する部分も多くあります。機能が同じ場合、semの方がより高速です。

そしてコマンドも簡易な場合があります。

gsemで利用可能な一般化応答変数は、異なる種類の応答を持つ計測モデルや、同じく異なる種類の応答を持つ潜在成長モデルで利用できます。

この他にもgsemはIRT(項目応答理論)モデル、マルチレベルCFAモデル、マルチレベル混合効果モデル、マルチレベル構造方程式モデルの推定をサポートしています。詳細は、[U] 27.28 **項目応答理論(IRT)**、[U] 27.26 **潜在クラスモデル**、[U] 27.27 **有限混合モデル**をご覧ください。

そして場合によっては、推定結果を指数型にしてオッズ比、罹患率比、相対リスク比として表示することも可能です。また、予測値、尤度比検定、ワルド検定を用意しています。sem,gsemのどちらの場合もモデルの設定はコマンド入力と、SEMビルダのパス図の両方をサポートしています。

SEMのことを詳しく知らなくても、線形回帰、ロジスティック回帰、順序ロジット回帰、順序プロビット回帰、ポアソン回帰、SUR、多変量回帰、同時方程式、観測誤差モデル、選択モデル、内生的処置効果モデル、トービットモデル、生存モデル、部分応答モデル、マルチレベル混合効果モデルなどを以前に利用した方は、この機会にSEMに取り組んでみましょう。

また、GMMに興味のある方もSEMをお試しください。semとgsemは、上記のモデルの多くを最尤法またはGMMでできる擬似最尤法でフィットできます。

ここに紹介していないモデルでもsemとgsemでフィットできます。semとgsemを利用することの長所をここで確認しておきましょう。これらのコマンドにより観測誤差を考慮した潜在変数の導入、異なる応答を持つ同時方程式の推定、例えば選択モデルなどの一般的なモデルのマルチレベル版を推定できます。

詳細は[SEM] **Stata Structural Equation Modeling Reference Manual**を参照してください。また、[SEM] **intro 5**をご一読ください。

27.26 潜在クラスモデル

潜在クラスモデル（LCM）は、母集団に潜む非観測なグループの特定および理解に用いられます。分析では母集団内の個体が観測できない、クラスと呼ばれるサブグループに分割ができると仮定します。クラスは1つ以上のカテゴリカル潜在変数で識別されます。LCMでは、クラスへの帰属（クラスメンバーシップ）を測定できるとされる観測可能な変数を1つ以上含めることがよくあります。モデルのパラメータは、クラスごとに異なる値を取ってもよいとされます。観測可能な変数のモデル化に加え、クラスへの帰属確率のモデル化も行えます。

LCMのフィット後、母集団内における各クラスの占有率や、個体の各クラスへの帰属確率を推定することもできます。

ここで、LCMとはカテゴリカル潜在変数を含むあらゆるモデルを指しています。文献によっては、LCMがカテゴリカル潜在変数と観測された二値あるいはカテゴリカル測定変数のみからなるモデルを指すという狭義な使い方をするものもありますが、本文では広義な使い方を行っています。LCMに深く関係のある分野としては、潜在クラス分析、潜在クラスタモデル、潜在クラスタ分析、潜在プロファイルモデル、潜在プロファイル分析、有限混合モデルがあります。こうした分析やモデルのフィットは、StataのLCMで行うことができます。詳細は[SEM] **intro 5**をご一読ください。

Stataではgsemコマンドをlclass()オプションと共に指定して潜在クラスモデルのフィットを行えます。詳細は[SEM] **Stata Structural Equation Modeling Reference Manual**を、特に[SEM] **Intro 1**、[SEM] **Intro 2**、[SEM] **Intro 5**、[SEM] **Example 50g**、[SEM] **Example 52g**を参照してください。

27.27 有限混合モデル(FMM)

有限混合モデル(FMM)は、オブザベーションの分類、クラスタの調整、観測できない異質性のモデル化に用いられます。有限混合モデルでは、観測されたデータが観測できない、クラスと呼ばれるサブ母集団（subpopulation）に振り分けられると仮定し、確率密度の組成または回帰モデルの混合物を使用したアウトカムのモデル化が行われます。

FMMにより、母集団を構成するとされる非観測なサブ母集団それぞれの分布の平均と分散を推定できます。サブ母集団の分布のほか、連続、二値、順序、カテゴリ、カウント、フラクショナル、生存のアウトカムに対する回帰モデルを混合したモデルを構築することも可能であり、パラメータは回帰モデルごとに別の値を取ることができます。また、全体の母集団に対する各サブ母集団の組成比（割合）を推定することもできます。さらに、サブ母集団への帰属確率をモデル化する共変量をモデルに組み込むこともできます。

Stataでは、fmmプリフィックスをサブ母集団と共に指定することでFMMのフィットを行えます。FMMでフィットを行えるモデルに関しては[FMM] **fmm estimation**を参照してください。

Stata Finite Mixture Models Reference ManualはStataで有限混合モデルでの分析を行うための機能を記述した完全マニュアルです。FMMの概要と導入的なexampleに関しては[FMM] **fmm intro**を参照してください。

27.28 項目応答理論(IRT)

項目応答理論(IRT)は、潜在特性の測定を目的とした類似の試験や操作に関する計画、分析、得点化、比較に用いられる手法です。潜在特性は観測できないため直接測定できませんが、ある操作によって評価することができます。操作の簡単な例としては、個々人の潜在特性上のレベルを測定するために設計された項目群があります。例えば、数

学の実力(潜在特性)の測定をしたい調査員が100問の問題(項目)からなる試験(操作)を設計する場合です。

操作の設計や得られたデータの分析では、調査員は個別項目がいかに潜在特性と関連しているか、または項目群全体でみたとき、それがどう潜在特性と関連しているかに注目します。IRT モデルではこれらの分析ができます。

バイナリ応答やカテゴリカル応答に対する様々なモデルでフィットを行うため、StataにはIRT推定を行うコマンド一式が備わっています。モデルの結合も可能です。搭載されたコマンドはを一覧として示します。

コマンド	説明	応答
irt 1pl	1パラメータロジスティックモデル	バイナリ
irt 2pl	2パラメータロジスティックモデル	バイナリ
irt 3pl	3パラメータロジスティックモデル	バイナリ
irt grm	段階反応モデル	カテゴリカル
irt nrm	名義反応モデル	カテゴリカル
irt pcm	部分採点モデル	カテゴリカル
irt gpcm	一般化部分採点モデル	カテゴリカル
irt rsm	評価スケールモデル	カテゴリカル
irt hybrid	ハイブリッドIRTモデル	組合せ

IRTの主要概念は項目特性曲線(ICC)にあります。ICCは、個人が項目（試験の一項目）に「成功」する確率と潜在特性の関係を表します。irtgraph iccにより、モデル上での各項目のICCがプロットできます。

irtgraph tccは試験全体での得点の期待値と潜在特性の関係を示すテスト特性曲線(TCC)をプロットできます。項目情報関数やテスト情報関数も、irtgraph iif、およびirtgraph tifによりプロットできます。

研究においては、異なるグループの潜在的特徴を計測しているかが重要になります。複数グループのIRTモデルでは全てのirtコマンドにgroup()オプションを追加して、グループごとに異なるパラメータを追加できます。

詳細は[IRT] irtをご覧ください。

27.29 動学的確率一般均衡(DSGE)モデル

DSGEモデルは、経済学の分野でポリシー分析や予測に用いられる時系列モデルです。マクロ経済の理論から導き出されたモデルで、複数の推定式を連立させた構成となります。大きな特徴の一つが、将来における変数の期待値が現在における変数の値に影響を与えるという点で、この点においてDSGEがほかの多変量時系列モデルと著しく異なっています。もう一つの大きな特徴が、理論から導出されたモデルであるという点で、パラメータが理論に沿った解釈が自然にできるようになっています。

dsgeとdsgenlコマンドはDSGEモデルのフィットを行えます。dsgenlは非線形DSGEモデルの、dgseは線形DSGEモデルのフィットを行います。詳細はStata Linearized Dynamic Stochastic General Equilibrium Reference Manual、特に[DSGE] Intro 1を参照してください。

DSGEモデルのベイズ推定は、bayes: dsgeおよびbayes: dsgenlを使用して利用できます。コマンド（[BAYES]bayes: dsge、[BAYES]bayes: dsgenl、および[DSGE]Intro9を参照）。ベイズ推定を使用すると、実際に利用できることが多いモデルパラメータに関する外部情報を組み込むことができます。[DSGE]Intro 9bを参照してください。 bayes: dsgeまたはbayes: dsgenlを使用して推定した後、[BAYES] bayesirfを使用してIRF分析を実行できます。

27.30 Lasso

Lassoはモデル選択と推定を同時に行います。lassoを用いて選択できるモデルの候補は、赤池情報規準やベイズ情報規準を利用する伝統的なモデル選択の手法と比べて、大幅に拡大します。大規模なモデルに対して、モデル選択と推定を同時に行うことで、lassoは機械学習の一般的なツールとなっています。

lassは連続値、二値、カウント、生存時間アウトカムにおいて、罰則付き最適化問題を解くものです。罰則がない状態では、lassoも伝統的な尤度法と同じ解答を行います。罰則はいくつかの変数を強制的にモデルから排除します。つまり、lassoにおいて罰則はモデル選択の特性を決定します。詳細は、[LASSO] lassoをご覧ください。

lassoに近いものでelastic netとsquare-root lassoがあります。elastic netとsquare-root lassoはどちらもlassoのように標本選択と推定を行うのが特徴です。lasso、elastic net、square-root lassoの違いは、モデルの罰則をどのように行うかにあります。lassoと比べelastic netの罰則では、変数同士が強く相関している場合に適した変数を導き出します。square-root lassoはlassoと同等ですが、罰則係数をより簡単に計算することができます。elastic netとsquare-roo lassoの詳細は、[LASSO] elasticnetと[LASSO] sqrtlassoをご覧ください。

Stataでは、lasso、elastic net、sqrtlassoを利用して、前述の変数を取り込む、標本外予測を行います。さらに、教師データや検証、予測に利用されたランダムなサブサンプルを、これらのコマンドに使用することもできます。使用しているデータのこのように標本に分割するには、splitsampleを使用します。

さらに、二重選択lassoやpartialing-out lasso、cross-fit partialing-out lassoで推定結果を得ることができます。これらの推定結果では、既知の共変量の組み合わせによる係数の検定を行うだけでなく、大量の制御変数を潜在的に含むlassoを利用したモデル選択を行います。以下の推定コマンドで連続値、二値、カウントアウトカムのモデルにフィットします。

コマンド	記述
dsregress	二重選択lasso線形回帰
dslogit	二重選択lassoロジスティックス回帰
dspoisson	二重選択ポワソン回帰
poregress	partialing-out lasso線形回帰
pologit	partialing-out lassoロジスティックス回帰
popoisson	partialing-out lassoポワソン回帰
poivregress	partialing-out lasso操作変数回帰
xporegress	cross-fit partialing-out lasso線形回帰
xpologit	cross-fit partialing-out lassoロジスティックス回帰
xpopoisson	cross-fit partialing-out lassoポワソン回帰
xpoivregress	crpss-fit partialing-out lasso操作変数回帰
telasso	Treatment-effects estimation using lassoラッソを使った処置効果

27.31 サーベイデータ

複雑なサーベイデータに統計モデルをフィットする場合はsvy コマンドを利用します。svyはプリフィックスコマンドです。したがって線形回帰を行う場合は次のようにします。

```
. svy: regress ...
```

プロビット回帰の場合は、

```
. svy: probit ...
```

しかし、これらのコマンドを実行する前にサーベイデザインの特徴をsvysetコマンドで設定しなければなりません。svyプリフィックスは多くの推定コマンドと組合せ利用できます。詳細は*Stata Survey Data Reference Manual*を参照してください。

svyプリフックスでは次のような分散推定手法をサポートしています。

- テイラー展開により線形化法
- ブートストラップ法
- BRR(平衡反復複製)法
- ジャックナイフ法
- SDR(連続差分複製)法

詳細は[SVY] **Variance estimation** を参照してください。

svyコマンドは次のようなサーベイデザインの特徴をサポートしています。

- 復元あり/なしのサンプリング
- 観測値レベルのサンプリング加重
- 推定段階レベルのサンプリング加重
- 層化
- サンプリング加重の調整
- クラスタリング
- 復元無しのマルチステージによるクラスタリング
- BRR及びジャックナイフ法による複製加重

詳細は[SVY] **Svyset**を参照してください。svyプリフィックスを用いた推定段階レベルのサンプリング加重の応用例については[ME] **meglm**のexample 6を参照してください。

すべての推定コマンドでサブ母集団の推定が可能です。

平均、比率、複数のサブ母集団における合計の作表と要約統計量、そして、平均と比率の標準化をサポートしています。

詳細は[SVY] **svyset**を参照してください。

27.32 多重代入

多重代入(MI)は推定における欠損値の存在を考慮した統計手法です。例えば、yをStataの推定コマンドでx_1, x_2, x_3に推定するとします。この時、y, x_1, x_2, x_3のいずれかに欠損値が存在すると、そのデータセットの一部は推定から除外されることになります。このように分析から対になっているデータを除外することをリストワイズ、または、ケースワイズ削除と呼びます。x_1, x_2, x_3の中に欠損値が存在する場合、該当する行のデータは一括で分析対象から外れます。欠損値がランダムに欠損している(MAR)か、または、欠損値が完全にランダムな場合(MCAR)、MIは推定に

利用できないこれらの情報を復元します。データが欠損する確率が、観測値に依存し、非観測なデータに依存しない場合、データはMARと考えます。一方、欠損の確率が観測データの関数で表現できない場合、それをMCARと呼びます。

MIという名前は欠損値の所に、ある値を作り出して代入するという文脈で命名されたものです。ただし、単純に欠損値のとこに代わりの値を代入するというものではありません。何回も代入を繰り返すという処理を行います。もちろん、真の値に近くなるような値を繰り返し作っていく、というものではありません。あくまで正しい統計的推測が行えるように欠損値を処理することを目的としたものです。

MIには3つのステップがあります。最初のステップはM個の欠損値に何らの値を代入します。2番目のステップは目的のモデルをそのデータセットにフィットします。最後に推定結果から得たパラメータを利用して、データセットを再び作成します。

利用するコマンドはmiです。サーベイ、生存分析、パネル、マルチレベルなどStataのほとんどの推定コマンドでmiを利用できます。詳細は[MI] Introを参照してください。

27.33 検出力と正確性、標本サイズの分析

標本サイズの決定は、研究計画を立てる際に重要になります。研究目的を達成するために必要なリソースの配分に役に立ちます。

研究においてパラメータの推定を行うために仮説検定を行いますが、検出力と標本サイズの(PSS)分析は研究リソースの最適な配分を調査して、目的の効果を検出できる確率を向上させます。さらに、データ収集プロセスが長期にわたる研究では、データが徐々に集まるごとに中間解析を実施することが一般的です。全データが収集された後に一度だけ解析を行うのではなく、中間解析を行うことで、第I種の誤りを保護しながら解析を進めることができます。中間解析を行うための一般的な枠組みとして、グループ逐次デザインがあります。グループ逐次デザインでは、治療が効果的であるか無効であるかを示す有力な証拠が見つかった場合、中間解析の一つで試験を早期終了することができます。これにより、研究者は効果的な治療法や無効な治療法に関する重要な情報をより早く得ることができます。

推定において信頼区間(CI)を利用する際、正確性と標本サイズの(PrSS)分析は、希望の信頼区間の正確性を達成するために必要な標本サイズを推定するために利用されます。

27.33.1 検出力と標本サイズの分析

PSS分析は仮説検定を行う研究計画を立てる際に利用されます。例えば、血圧降下剤の新薬を評価する実験計画を構築する場面を想定してください。新薬を利用する実験グループの平均血圧は、現行の薬を利用するコントロールグループの平均血圧と等しいものとします。2群の平均値の差を検定しますので、2群のt検定を実行します。しかし、臨床上の重要性に値する有意差を検出するための研究を行うためには、どの程度の標本が必要になるでしょうか。PSSを利用すると、このような疑問に答えることができます。

PSS分析は研究計画の作成段階で考えなければならない、その他の問題点に対する情報を提供します。例えば、用意した標本サイズによる検定の検出力はどの程度か、そして、限られた環境の中で関心のある効果をどのくらい検出できるのでしょうか。これらの疑問に的確に答えることができれば、過剰な研究投資や、脆弱な研究に資源を浪費してしまうことを回避する一助になります。

PSS分析の詳細は[PSS-2] Intro (power)を参照してください。

PSS分析にはpowerコマンドを用います。平均、分散、比率、相関や分割表の各比較、単回帰、重回帰、生存時間分析に対するPSS分析が行えます。一標本、二標本、対応の有無の各場合の平均、分散、比率、相関の分析をサポートします。分割表では、マッチングのある標本、2 × 2 × K分割表、2 × J分割表の分析も行えます。生存時間データでは、Cox比例ハザードモデルに対する一標本分析がサポートされています。すなわち、生存関数のパラメトリックまたはノンパラメトリックな二標本分析がサポートされています。

powerコマンドは、一標本、二標本の平均と比率の分析など、いくつかの分析におけるクラスタランダム化設計(CRD)に対応します。CRDでは、個々の患者ではなく、患者のグループまたはクラスタレベルのランダム化が行われます。その結果、通常はクラスタ内の患者に相関があり、この点はPSS分析で考慮されます。

powerコマンドには自作のPSS手法を追加することもできます。詳細は[PSS-2] *power usermethod*を参照してください。

powerコマンドは表、グラフ、検出力曲線をサポートしています。詳細は[PSS-2] power, tableと[PSS-2] power, graphを参照してください。

サポートしている分析手法の一覧と、そのコマンドの詳細は[PSS-2] powerを参照ください。

powerコマンドによる分析、ダイアログによるユーザインタフェースを利用した分析の両方をサポートしています。詳細は[PSS-2] GUI(power)を参照してください。

27.33.2 正確性と標本サイズの分析

PrSS分析は推定のために信頼区間を利用する研究の計画に利用されます。例えば、血圧降下剤の新薬を評価する

実験計画を構築する場面を想定してください。新薬を服用する実験群の血圧の平均値と、従来のものを服用する統制群の平均値の差の推定するとします。臨床上の意味のある信頼区間を得るためには、どの程度の標本が必要になるでしょうか。PrSSを利用すると、このような疑問に答えることができます。

PrSS分析は、研究の計画段階で生じる、その他の問題にも答えることができます。例えば、任意の標本サイズから得られる信頼区間の幅はどの程度か、限られた研究リソースから特定の信頼区間の幅を確保できる可能性がどの程度か、といったものです。このような問いに答えることで、研究対象が限られる中でもコストを抑えることができます。さらに、有意な信頼区間を得るためには、少なすぎる標本のままで分析を行ってしまうのを防ぐこともできます。

詳細は、[PSS-3] **Intro (ciwidth)**をご覧ください。

ciwidthはPrSS分析を実行するものです。このコマンドは信頼区間の平均や分散を用いたPrSS分析を提供しています。また、独立した2群の平均の差や対応のある2群の平均の差を用いたPrSS分析を提供します。

ciwidthコマンドは表形式と視覚的なアウトプット、標本サイズ曲線のいずれも提供することができます、詳細は[PSS-3] **ciwidth, table**と[PSS-3] **ciwidth, graph**をご覧ください。

対応している全ての分析手法とコマンドの説明は[PSS-3] **ciwidth**をご覧ください。

ciwidthコマンド群はインタフェースウィンドウをクリックすることでも利用できます、詳細は[PSS-3] **GUI (ciwidth)**をご覧ください。

27.33.3 グループ逐次デザイン

グループ逐次デザイン（GSD）は、研究者が治療が効果的または無効であるという有力な証拠を見つけた場合、試験を早期に終了させることができる適応型デザインの一種です。例えば、がん治療薬のスニチニブが腫瘍治療に効果的かどうかを検証するための研究を設計したいとします。また、データ収集が3年間にわたって行われると予想します。全データの収集が完了した後に単一の解析を行うのではなく、データが集まるごとに中間解析を実施することができます。例えば、6ヶ月ごとに中間解析を行い、各中間解析でスニチニブの効果があるかどうかを検証することができます。初めの中間解析で効果の強い証拠が見つかった場合、研究を終了し、規制当局への承認申請を早期に進めることができます。一方、効果のない証拠が見つかった場合、無効性により試験を中止し、追加の参加者に不適切な治療を行わせることを避けることができます。GSDでは、各中間解析でこれらの判断を行うための基準が提供されます。

GSDでは、各中間解析で効果と無効性を検証することができます。gsboundsコマンドを使用してGSDの停止境界を計算し、gsdesignコマンドを使用して中間解析のサンプルサイズも計算することができます。停止境界とサンプルサイズは、単一の平均または比率の検定、2つの平均または比率の検定、およびログランク検定に対して取得することができます。

また、gsdesignコマンドに独自のGSDメソッドを追加することもできます。詳細については、[ADAPT] gsdesign usermethodを参照してください。

gsboundsコマンドとgsdesignスイートのコマンドは、表形式の出力とグラフ形式の出力の両方を提供します。サポートされるメソッドの完全なリストとコマンドスイートの説明については、[ADAPT] gsdesignを参照してください。

27.34 ベイズ統計分析

ベイズ統計分析は、統計モデルにおける未知のパラメータに関する調査上の疑問に関する答えを、確率を用いて導く手法です。ベイズ統計分析は、モデルの全パラメータがランダムな量であり、事前情報に従うという前提を置きます。この仮定は、広く用いられているそれまでの古典的統計手法に見られる、全パラメータは未知であるがある定値を取るとする特徴と大きく異なります。

ベイズ統計分析は、観測データを条件とするパラメータの事後分布のモデリングと要約を基にします。事後分布はデータの確率分布とモデルパラメータの事前分布からなります。多くの事後分布は閉形式ではありませんが、Metropolis-Hastings(MH)法やGibbs法、そして時としてその組み合わせなどのマルコフ連鎖モンテカルロ(MCMC)法で近似が行われる必要があります。MCMCの収束は、推論が行なわれる前に吟味されます。MCMCの収束がされれば、フィットしたベイジアンモデルに基づいてシミュレーションされたデータの分布を様々な点から比較して、モデルの確認ができます。

ベイズ統計分析では、パラメータに関する周辺の事後分布が推論に用いられます。事後平均やメディアンなどの点推定量、equal-tailed確信区間や最高事後密度区間などの区間推定量で要約されます。

Stataはベイズ統計分析を実施するための連携したコマンド群を備えています。ベイズ推定で用いるコマンドはさまざまなベイズ回帰モデルのフィットを行えるbayesプリフィックスコマンドと、一般的なベイズモデルのフィットを行えるbayesmhコマンドから構成されます。両コマンドは、適応型メトロポリス・ヘイスティングス・サンプリング、ギブス・サンプリング、および2種の混合という3つのMCMCサンプリング手法を備えます。マルチレベルモデルを含む、あらゆるサポートされたベイズ回帰モデルからモデルの選択が可能であり、自作のベイズモデルを組むこともできます。詳細は[BAYES] **bayes**、[BAYES] **bayesmh**、[BAYES] **bayesmh evaluators**を参照してください。

MCMCの収束はbayesgraphと、bayesstats grubinを用いたGelman-Rubinの収束診断で視覚的に確認できます。モデルの確認はbayespredictとbayesstats ppvaluesを使用します。周辺の要約統計量はbayesstats summaryで取得でき、仮説検定はbayestestで実施できます。詳細は[BAYES] **Bayesian postestimation**を参照してください。VARモデル

（[BAYES]**bayes: var**）の後、ベイズ予測（[BAYES] **bayesfcast**）により事後推定コマンドも利用できます。また、IRF分析はVARおよび線形および非線形DSGEモデル（[BAYES]**bayes: dsge**および[BAYES]**bayes: dsgenl**）の後に利用できます。

コマンドに関する詳細と例題については[BAYES] **Bayesian commands** と *Overview example*を参照してください。

27.35 ベイズモデル平均化

単一のモデルを適合させる代わりに、ベイズモデル平均化（BMA）を使用することで、モデルの不確実性を考慮して複数の妥当なモデルの結果をベイズの原則に基づいて組合せることができます。BMAの導入に関する簡単な動機は、[BMA] IntroのRemarks and examplesを参照してください。

回帰の枠組みでは、モデルの不確実性は回帰モデルに含めるべき予測変数がどれであるかの不確実性を意味します。bmaregressコマンドは線形回帰のためのBMAを実行し、推論、予測、またはモデル選択に使用することができます。詳細は[BMA] bmaregressを参照してください。

bmaregressコマンドを使用した後、bmagraph pmpを使用して収束を確認し、bmagraph pmpおよびbmastats modelsを使用して、事前に仮定された情報と観測データに基づいてどのモデルがより可能性が高いかを調べることができます。つまり、より高い事後モデル確率を持つモデルを調べることができます。詳細については、[BMA] bmagraph pmpおよび[BMA] bmastats modelsを参照してください。

bmastats pipコマンドを使用すると、事前情報と観測データに基づいてモデルに含まれる確率が高い予測変数（重要な予測変数）を特定することができます。詳細については、[BMA] bmastats pipを参照してください。bmagraph varmapコマンドを使用すると、影響力のあるモデルと予測変数をより便利に視覚化することができます。詳細については、[BMA] bmagraph varmapを参照してください。また、bmastats jointnessを使用すると、考慮されるモデル間での予測変数の共通の重要性を探索することができます。つまり、変数がモデル内で一緒に現れるか別々に現れるかを調べることができます。詳細については、[BMA] bmastats jointnessを参照してください。bmastats msizeおよびbmagraph msizeは、BMAモデルの複雑さを探索するために、モデルのサイズの要約と分布を探索するために使用することができます。詳細については、[BMA] bmastats msizeおよび[BMA] bmagraph msizeを参照してください。

上記の機能を使用すると、モデル選択分析を実行するだけでなく、BMAを予測にも使用することができます。bmapredictコマンドは、予測平均や信頼区間などのさまざまな事後予測の要約を計算し、新しい結果をシミュレーションし、結果の複製を生成します。詳細については、[BMA] bmapredictを参照してください。その後、bmastats lpsコマンドを使用して、対数予測スコアを使用してBMAモデルの予測パフォーマンスを評価することができます。詳細については、[BMA] bmastats lpsを参照してください。

最後に、パラメータ平均化が適用可能な場合、回帰係数に対する推論を行うことができます。通常のベイズ分析と同様に、係数の推定値は単一の値ではなく、全体の分布です。BMA分析では、この分布はさらにモデルの不確実性を考慮に入れています。bmagraph coefdensityコマンドを使用して、この分布を探索することができます。詳細については、[BMA] bmagraph coefdensityを参照してください。より一般的には、bmacoefsampleコマンドを使用して、回帰係数や他のモデルパラメータのMCMCサンプルをシミュレーションし、その後、標準的なベイズツールを使用して推論を行うことができます。詳細については、[BMA] bmacoefsample、[BAYES] bayesstats summary、[BAYES] bayesgraphを参照してください。

推定後の機能の一覧は、[BMA] BMA postestimationを参照してください。

27.36 参考文献

Gould, W. W.2011. Use poisson rather than regress; tell a friend. The Stata Blog:Not Elsewhere Classified. http://blog.stata.com/2011/08/22/use-poisson-rather-than-regress-tell-a-friend/

28 便利なコマンド

ユーザが特に興味を持つであろう統計コマンドはさておき、全てのユーザにぜひ知っておいていただきたいコマンドを紹介します。

ヘルプを見つける	[U] 4 Stataのヘルプと検索機能
help, net search, search	
Stataを最新版に更新する	
ado, net, update	[U] 29 インターネットによるプログラムの更新
ado update	[R] ado update
OSのインターフェイス	
pwd, cd	[D] cd
ディスクのデータを使用して保存する	
save	[D] save
use	[D] use
compress	[D] compress
Stataにデータを入力する	[U] 22 データの入力とインポート
import	[D] import
edit	[D] edit
基本的なデータレポート	
describe	[D] describe
codebook	[D] codebook
list	[D] list
browse	[D] edit
count	[D] count
inspect	[D] inspect
table	[R] table
tabulate	[R] tabulate onewayと[R] tabulate twoway
summarize	[R] summarize

データ変更	[U] 13 関数と計算式の入力
append, merge	[U] 23 Combinning datasets
generate, replace	[D] generate
egen	[D] egen
rename	[D] rename, [D] rename group
clear	[D] clear
drop, keep	[D] drop
sort	[D] sort
encode, decode	[D] encode
order	[D] order
by	[U] 11.5 by varlist: 構文
reshape	[D] reshape
frames	[D] frames
データをグラフにする	
graph	*Stata Graphics Reference Manual*
作業内容を記録する	
log	[U] 15 出力を保存し印刷する—ログファイル
notes	[D] notes
その他	*Stata Reporting Reference Manual and Stata Customizable Tables and Collected Reference Manual*
便利な機能	
display	[R] display

29 インターネットによるプログラムの更新

目次

29.1 概要

Stataはインターネット上のファイルを開くことができます。次のコマンドを入力してください。

```
. use https://www.stata.com/manual/chapter28, clear
```

このコマンドはインターネットに接続し、Stata社のWebサイトからデータセットを開きます。データセットはHTML形式という訳ではありません。また、ブラウザと関連する形式でもありません。単純に、Stataのデータファイル、chapter28.dta をインターネットでアクセスできるサーバに保存してあるだけです。これだけで世界中の人々が同じデータを見ることができます。Webサイトを持っている場合、同じ事ができます。これで、共同研究者などとデータセットを簡単に共有できます。

では、次のコマンドを入力してみましょう。

```
. update query
```

何も悪いことは起こらないので、安心してください。updateコマンドはwww.stata.comからの短いファイルを読み、そのファイルを基にして使用中のStataが最新かどうか確認します。最新版を使用しているでしょうか。調べる方法は簡単ですね。最新ではない場合、Stataでのアップデート方法を次に説明します。実際の操作はupdateを入力するだけなので簡単です。

では、次のコマンドを入力してみましょう。

```
. net from https://www.stata.com
```

このコマンドはwww.stata.comのWebページに移動し、そのページでユーザがダウンロード可能なサイト情報一覧を表示します。これらの情報サイト先から入手できるものは公式ではありませんが、とても便利です。さらに簡単に目的の情報を探すには、次のように入力しましょう。

```
. search kernel regression, net
```

あるいは、以下でも同様です。

```
. net search kernel regression
```

これらはStata内にあるカーネル回帰に関する情報だけでなく、Web全体で検索します。その中には*Stata Journal*, *Statalist*, アーカイブ、個人的なユーザのサイトも含まれます。

Stataはインターネット上のファイルを読み込むことができ、それにより以下のことが行えます。Stataはインターネット上のファイルを開くことができます。

1. 世界中の共同研究者などとデータセット、doファイルなどを共有できます。これには、特殊な技能は必要ありませんが、Webサイトは必要です。
2. Stataは無料で簡単にアップデートできます。
3. Stataに、同じく無料で簡単に、新しい機能を検索して追加できます。

また、自分のWebサイトを作成すれば、Stataの新しい機能を公開できます。

29.2 データセット(と他のファイル)を共有する

データセットをインターネットで共有するのに、特別な技術などは必要ありません。単純に、ファイルをそのまま（バイナリ形式で）サーバーにコピーしてそのファイルの場所を共同研究者に連絡するだけです。これは、.dta、.do 、.adoファイルを始めとした全てのStataのファイルで実行できます。

受け取る側では、（.dtaならば）useコマンドで直接使用するか、コピーできます。

```
. use https://www.stata.com/manual/chapter28, clear
. copy https://www.stata.com/manual/chapter28.dta mycopy.dta
```

Stataにはファイルをコピーするコマンドがあり、useのようにインターネット上のファイルにも使用できます。詳細は[D] **copy**をご覧ください。

29.3 公式アップデート

アップデートのリリースに関しては特にスケジュールがあるわけではありませんが、通常、Stataはアップデートを月1回程度の頻度で提供します。それほど頻繁にアップデートをする必要はありませんが、可能な限り行うようにお勧めしています。ご利用いただいているStataが最新版かどうかを確認する方法は次の2つがあります。

メニューから選択	入力
ヘルプ > **アップデートのチェック**	. update query

もしアップデートが使用可能な場合、次の操作は以下のようになります。

クリック	入力
Install available updates	. update all

Stataをアップデートしたら何が変わったのか確認するには

次を選択	入力
ヘルプ > **更新情報**	. help whatsnew

29.3.1 アップデートに関するFAQ

1. 何か予期せぬことが起こり、Stataが使えなくなることはありますか。

   いいえ、ありません。アップデートファイルは、まずテンポラリ（一時的な）ファイルを保存する場所にコピーされます。その中でStataが必要な情報があることを確認してから実際の保存場所にコピーします。よって、アップデートはインストールするかしないか、どちらかだけです。

2. Stataを使用してるコンピュータからインターネットに接続できません。Stataを手動でアップデートする方法はありますか。

   はい、あります。Webブラウザでhttps://www.stata.com/support/updates/のページを開き、その中の手順に従ってください。

29.4 ユーザによる追加内容をダウンロードして管理する

実際にやってみましょう。

次のように選択

ヘルプ > Stataジャーナル/コミュニティ投稿コマンド

または入力します。

```
. net from https://www.stata.com
```

それから、リンクのうちの１つをクリックします。

29.4.1 ファイルのダウンロード

Stata社だけがStataの追加項目を開発しているわけではありません。Stataは熟練ユーザが多数集まるコミュニティにサポートされています。この重要な部分は*Stata Journal* (SJ)と*Stata Technical Bulletin* (STB)が担っています。*Stata Journal* は四半期ごとに出版される定期刊行物で、Stataユーザが興味を持つ内容を中心に査読済みの記事を掲載します。詳細と購読に関する情報は、*Stata Journal*のWebサイト、https://www.stata-journal.comをご覧ください。

*Stata Journal*は印刷と電子媒体で準備され、このJournalの中で紹介しているプログラムと共に提供されます。このジャーナルを読みたい場合、購読の手続きを取る必要がありますが、中のプログラムは無料です。

Stata Journalからプログラムをインストールする

1. Stata内でメニューから**ヘルプ > Stataジャーナル/コミュニティ投稿コマンド** と選択
2. *Stata Journal*をクリック
3. *sj10-4*をクリック
4. *st0015_6* をクリック
5. *click here to install*をクリック

または

1. net from https://www.stata-journal.com/softwareと入力
2. net cd sj10-4と入力
3. net describe st0015_6と入力
4. net install st0015_6と入力

上記のコマンドは、次のように省略できます。

```
. net from https://www.stata-journal.com/software/sj10-4
. net describe st0015_6
. net install st0015_6
```

または、次のように入力することもできます。

```
. net sj 10-4
. net describe st0015_6
. net install st0015_6
```

29.4.2 ファイルの管理

つい先ほどダウンロードしてインストールしたので、concordコマンドがStataにインストールされました。確認するには、次のようにコマンドを入力しましょう。

```
. help concord
```

このコマンドを実際に実行してみるのもいいでしょう。では、インストールした追加のプログラムをリストしてみます。おそらく、concordコマンドだけですが、このconcordコマンドをリストから削除します。

コマンドモード（コマンドウィンドウ）で次のように入力します。

```
. ado dir
[1] package st0015_6 from https://www.stata-journal.com/software/sj10-4
                         SJ10-4 st0015_6.  Concordance correlation ...
```

concordコマンド以外にも追加したプログラムがある場合は、ここにリストされます。では、*st0015_6*がインストールされたことが分かったので、さらに詳しい内容をdescribeコマンドで確認することもできます。それには、次のように入力します。

```
. ado describe st0015_6
  (output omitted)
```

*st0015_6*を削除するには、次のように入力します。

```
. ado uninstall st0015_6
package st0015_6 from https://www.stata-journal.com/software/sj10-4
     SJ10-4 st0015_6.  Concordance correlation ...
(package uninstalled)
```

同じ内容の操作は、マウスのクリックとカーソル移動で行うインターフェースでも行えます。**ヘルプ**メニューを開き、**Stataジャーナル/コミュニティ投稿コマンド**を選択します。表示されるビューワの画面の下にある*List*をクリックしましょう。その後は *st0015_6* をクリックすると、このプログラムパッケージに関する内容の詳細が確認でき、*click here to uninstall* をクリックすればプログラムを削除できます。

adoコマンドとそれに関するメニュー表示に関する詳細は[R] **net**をご覧ください。

29.4.3 ダウンロードするファイルを探す

便利なファイルやプログラムを探すには2つの方法があります。1つは単純にWebの情報をどんどん見ていくことです。効率は悪いですが、時に興味深い情報を見つけることもできます。Webのブラウジングを行いたい場合、操作としては次のように行ってください。

1. **ヘルプ > Stataジャーナル/コミュニティ投稿コマンド**
2. *Other Locations*をクリック
3. *links*をクリック

ここでは、Stata社のダウンロードサイトから他のリンクをたどってページを確認していることになります。Stata社では他のサイトのリストも準備しており、これらのサイトにはさらに多くのリンクがあります。もちろん、同じ操作をコマンドモード（コマンドウィンドウに入力）からもできます。

```
. net from https://www.stata.com
. net cd links
```

効率的な方法はファイルを検索することです。つまり、Stataのsearchコマンドを使用することになります。

```
. search concordance correlation
```

同様に、メニューから**ヘルプ > 検索...**と操作することもできます。どちらの場合でもst0015_6 のプログラムについて確認でき、クリックするとインストールできます。

29.4.4 ユーザ作成プログラムを更新する

ユーザ作成機能をインストールしている場合、定期的にアップデートやバグの修正があるか確認する必要があります。アップデートの有無を確認する場合、ado updateコマンドを使います。「ado update」と入力すると、更新項目があるかどうかを確認します。「ado update, update」と入力すると、アップデートを入手できます。詳細は[R] ado updateをご覧ください。

29.4.5 ビデオ例題

How to download and install user-written commands in Stata

29.5 自分のダウンロードサイトを作成する

自分のダウンロードサイトを作成する理由は大きく分けて2つあります。

1. あるデータセットを持っており、それを共同研究者と共有したい場合。そのデータセットを共同研究者が簡単にダウンロードできるように設定します。
2. Stataのユーザ作成プログラムを作成して、それをStataのユーザコミュニティと共有したい場合。

自分のダウンロードサイトを作成する前に、作成したコマンドをStatistical Software Components (SSC)アーカイブに提出するほうがいいでしょう。SSCアーカイブはWeb上に存在する最も大きなStataのユーザ定義プログラムの格納場所です。Stataはこのアーカイブを検索するコマンドを準備しており(詳細は[R] sscをご覧ください)、簡単に目的のプログラムを検索してインストールできます。

作成したコマンドをSSCに提出する詳細については、http://repec.org/bocode/s/sscsubmit.htmlをご覧ください。

自分のダウンロードサイトを作成したい場合の操作手順は[R] netに記載されています。それほど難しくないので、ご安心ください。

この章の最初で、共同研究者と共有したいデータセットがある、と仮定しました。そこでは、データセットをサーバに置いて、そのデータセットの在り処を連絡すればよい、と記載しました。

では、2つのデータセット、ds1.dtaとds2.dta、があると仮定して手順を確認しましょう。共同研究者にはデータセットをnetコマンドで自分のStataで利用できるようにするか、**ヘルプ**メニューの**Stataジャーナル/ユーザ作成コマンド**項目からデータをダウンロードしてもらうことにします。

まず、最初にデータセットを今までの手順と同じようにホームページにコピーします。そして、それ以外に3つのファイル、stata.tocという名前の、作成したWebサイトを説明するファイルと提供するそれぞれの“パッケージ”に関する説明を追加します。

```
──────────────────────────────────────────────────── begin stata.toc ─────
v 3
d My name and affiliation (or whatever other title I choose)
d Datasets for the PAR study
p ds1 The base dataset
p ds2 The detail dataset
──────────────────────────────────────────────────── end stata.toc ───────
```

```
------------------------------------------------------------------ begin ds1.pkg
v 3
d ds1.  The base dataset
d My name or whatever else I wanted to put
d This dataset contains the baseline values for ...
d Distribution-Date:26sep2022
p ds1.dta
-------------------------------------------------------------------- end ds1.pkg
------------------------------------------------------------------ begin ds2.pkg
v 3
d ds1.  The detail dataset
d My name or whatever else I wanted to put
d This dataset contains the follow-up information ...
d Distribution-Date:26sep2022
p ds2.dta
-------------------------------------------------------------------- end ds2.pkg
```

説明の中にあるDistribution-Date行は、パッケージに何か変更が施されるごとに更新する必要があります。この行はado updateで使用され、このパッケージを使用しているほかのユーザがアップデートの有無を確認する際に使います。

共同研究者が作成したサイトを訪れると、次のような画面が表示されます。

```
. net from http://www.myuni.edu/hande/~aparker
------------------------------------------------------------------------------
http://www.myuni.edu/hande/~aparker
My name and whatever else I wanted to put

------------------------------------------------------------------------------
Datasets for the PAR study PACKAGES you could -net des
cribe-:
    ds1             The base dataset
    ds2             The detail dataset

------------------------------------------------------------------------------
. net describe ds1

------------------------------------------------------------------------------
package ds1 from http://www.myuni.edu/hande/~aparker
------------------------------------------------------------------------------

TITLE
      ds1.  The base dataset
DESCRIPTION/AUTHOR(S)
      My name and whatever else I wanted to put
      This dataset contains the baseline values for ...Distribution-Date:2
      6sep2022
ANCILLARY FILES                                 (type net get ds1)
      ds1.dta

------------------------------------------------------------------------------

. net get ds1
checking ds1 consistency and verifying not already installed...
copying into current directory...
      copying  ds1.dta
ancillary files successfully copied.

. _
```

詳細は[R] netをご覧ください。

用語集

ASCII ASCIIはAmerican Standard Code for Information Interchangeを意味しています。コンピュータ内のテキストを文字で表示する規格です。数字、区切り記号、発音区別符号のない標準文字、スペースやタブ文字などの空白文字、および改行記号からなるプレーンASCIIと、発音区別符号のある文字や記号含む拡張ASCIIがあります。
Stata14前では、データセット、doファイル、adoファイルおよびその他のStataファイルはASCIIでエンコードされています。

binary 0 binary 0またはヌル文字はASCIIやUTF-8などの文字列の終端を表すために用いられてきました。
binary 0はchar(0)で確認できます、また¥0と表示される事があります。詳細は、[U] 12.4.10 strL変数とバイナリ文字列をご覧ください。

バイナリ文字列 バイナリ文字列は技術的にはテキストを含まない文字列を指します。しかしStataでは、binary 0を含む文字列、またはfileread()関数で読み込むことのできる内容、バイナリとしてマークされている文字列を含む文字列式を指しています。
Stataではstrl形式の変数、stringスカラー、Mata文字列がbinary stringとして保存されます。
詳細は、[U] 12.4.10 strL変数とバイナリ文字列をご覧ください。

BLOB BLOBはbinary large objestを意味するデータベース用語です。StataではBLOBはstrL形式で保存されます。このようにstrLはWordドキュメントやJPEG画像などを含めることができます。詳細は、***strL***をご覧ください。

バイト バイト、8ビット：1バイトはコンピュータのデータを記録するための単位です。各バイトは0から255までの値を格納します。Stataの保存形式byteとは異なるものです、byte形式では、-127から100までの値を保存できます。すべての文字列は、1また複数のバイトで表現される文字でエンコードされてた文字で構成されています。
例として、Stataで利用されているUTF-8でのエンコーディングでは、97の値のbyteに「a」が割り当てられています。195と161の値には「á」が割り当てられています。

構造情報 構造情報はStataのデータセットと、データセット内の各変数に関するメタデータの1つです。これらはプログラミングで使用されます。例えば、xtコマンドでは、パネル変数名と可能であれば時間変数が必要になります。これらの変数名はデータセットのcharacteristicsとして保存されます。詳細は、[U] 12.8 構造情報の概要を、技術的な情報は[P] charをご覧ください。

コードページ コードページは特定の言語で使用される文字セットを拡張ASCIIの値で指定するものです。例えば、最も一般的なコードページはWindows-1252であり、西ヨーロッパ諸語で使用されるラテン文字の拡張ASCIIの値を指定しています。コードページは拡張ASCII文字に不可欠なものです。

コードポイント コードポイントはASCIIやUTF-8のテキストシステムで、1つの文字を表す数値です。ASCIIのエンコーディングシステムはで128のコードポイントしか無いため、128文字までした表現することができません。歴史的には、拡張ASCIIで提供される128の追加的なバイトが様々な方法でエンコーディングされてきました。Unicodeでは1,114,112のコードポイントがあり、そのうち、およそ250,000使用されています。StataではUTF-8を利用してUnicodeのエンコーディングをしています。UTF-8でエンコードされたコードポイントは、コードポイントそのものと値が異なっていることに注意してください。

文字、バイト、表示列の違い 文字は単なる文字または記号、文字「a」や区切り記号「.」または表語文字などです。バイトは文字がコンピュータ内に保存される際の保存形式です。表示列はStataの結果ウィンドウとビューワなどの固定された幅のディスプレイで文字を表示するためのスペースです。しかし、1つの表示列では幅が広すぎる文字もあります。それぞれの文字は1つまたは2つの表示列で表示されています。
プレーンテキストでは、常に文字数がバイト数と表示列の数と等しくなります。
UTF-8文字はプレーンテキストとは異なり、1文字が2バイトですが、漢字、ひらがな・カタカナ、ハングルのように3バイトまたは4バイトで構成される文字もあります。これらの文字は1表示列で表すには幅が広いため2つの表示列が使用されます。
一般に、Unicode文字では、文字数とバイト数の関係と、文字数と表示列数の関係はさらに不明瞭になります。Stataでは、すべての文字が4バイト以下で保存され、2表示列以下で表示されます。
[U] 12.4.2.1 Unicode文字列関数と[U] 12.4.2.2 Unicode文字の表示では、コード内で文字、バイト、表示列がどのように扱われているかを解説しています。

display column display cloumnはStataの結果ウィンドウとビューワなどの固定された幅のディスプレイで文字を表示するためのスペースです。文字の中には1つのdisplay columnでは幅が広すぎる文字もあります。それぞれの文字は1つまたは2つの表示列で表示されています。
全てのプレーンテキスト（「M」「9」など）と、プレーンテキストではない(「é」などの)UTF-8文字の多くは固定長フォントでは、同じ大きさのスペースを使用します。これらは、1つのdisplay columnで表示されます。
ラテン文字以外の、漢字、キリル文字、ひらがな・カタカナ、ハングルなどは2つのdisplay columnを必要とします。
詳細は、[U] 12.4.2.2 Unicode文字の表示をご覧ください。

表示フォーマット 変数の表示フォーマットはStataの変数がどのように表示されるかを指定します。数値変数では、小数点以下桁数やコンマを表示するか、指数形式で表示するかなどを表示フォーマットで制御します。数値変数は日付としてもフォーマットできます。つまり、表示フォーマットでは、変数を左・右寄せで表示するか、何桁まで表示するか、を指定します。表示フォーマットはformatコマンドで指定を行います。単独の数値や文字列にも表示フォーマットが使用されます。変数の保存形式と混同しないよう注意してください。

エンコーディング エンコーディングとは、文字をバイトで表示する方法です。ASCIIとUTF-8はエンコーディングシステム

です。Stataはエンコーディングに UTF-8を使用します。
詳細は、[U] 12.4.2.3 エンコーディングをご覧ください。

拡張ASCII 拡張ASCIIは元のASCIIでは指定されていない128から255までの値を持つバイトです。オリジナルのASCIIをサポートするため、これまで様々なコードページが拡張ASCII、発音区別符号を伴うラテン文字（「á」や「Á」など）や漢字、キリル文字、ひらがな・カタカナ、ハングル等の非ラテン文字、および「<」や「±」の複雑な数学記号などを指定するために考案されてきました。
拡張ASCII文字はASCIIエンコーディングでは単一バイトで保存されますが、UTF-8は同じ文字を2から4バイトで保存します。コードページごとに拡張ASCIIの値が異なっているため、拡張ASCIIはフォントやOSによって指定される文字が異なることも特徴の1つです。

フレーム フレームまたはデータフレームはデータセットの解析を行うメモリ領域です。Stataはメモリ上に複数のデータセットを記憶し、それぞれのデータセットはフレームと呼ばれるメモリ領域に展開されます。フレームを管理したり、フレーム内のデータを操作する、様々なコマンドが備えられています。[D] **frames**をご覧ください。

higher ASCII 拡張ASCIIを参照。

直接入力コマンド 直接入力コマンドはメモリ上のデータではなく、入力されたデータを使用します。直接入力コマンドはメモリ上のデータの影響を受けません。概要は[U] **19 直接入力コマンド**をご覧ください。

ロケール ロケールは言語の記述方法を規定を特定するためのコードです。ロケールは、言語全体を指す幅広いもの（英語を示す「en」）や、特定の地域の言語のみを示す（香港の英語を示す「en_HK」）があります。
ロケールは言語の記述方法を指定します。Stataは言語特有の動作を行う際にロケールを使用しています。詳細は、[U] **12.4.2.4 Unicodeのロケール**をご覧ください。

lower ASCII plain ASCIIを参照。

null-terminator binary0を参照。

数値リスト 数値リストはリスト化された数値です。リストは1つ以上の任意の数値または、5,6,7,8,9を意味する5/9のような、範囲を示す省略形を取ります。範囲は昇順または、降順、10.5, 8.5, 6.5, 4.5を示す10.5(-2)4.5のように増分・減分に対応しています。

オプション Stataはオプションを用いて、コマンドの追加設定を行います。例えば、summarizeのdetailオプションは追加的な統計量を指定します。オプションは常に、Stataコマンドの後にコンマによって指定されます。詳細は[U] **11.1.7 オプション**をご覧ください。

プレーンASCII 単にプレーンASCIIと言えば、プログラマーの言うlower ASCIIを指します。これらは、ラテン文字の「a」から「z」または、「A」から「Z」、「0」から「9」までの数字、および「!」などの区切り記号、「+」などの簡単な数学記号と、「 」のような空白文字、タブ文字、改行文字です。
それぞれのプレーンASCIIは0から127の単一バイトで保存されます。もう1つの特徴は、プレーンASCII文字をエンコードするバイトの値はASCIIとUTF-8や異なるOS間であっても共通であるということです。
詳細は、*ASCIIとencodings*をご覧ください。

プリフィックスコマンド プリフィックスコマンドはStataのコマンドに接頭辞として追加するコマンドです。例えば、by 変数リスト:、といったものです。by region: summarize marriage_rate divorce_rateというコマンドは、marrage_rateとdivorce_rateをそれぞれの地域ごとに要約します。詳細は、[U] **11.1.10 プリフィックスコマンド**をご覧ください。

保存形式 保存形式はStataが変数をどのように保存しているかを示します。数値の保存形式は、byte, int, long, float, doubleです。これ以外にもstringという保存形式があります。変数作成時に、変数の前に指定します。詳細は、[U] **12.2.2. 数値の保存形式**、[U] **12.4 文字列**および[D] **Data types**をご覧ください。また、保存形式と表示フォーマットは別々のものです。

str1, str2, ..., str2045 strLを参照。

strL strLは文字列変数の保存形式です。すべての文字列の保存形式は、str1, str2, ..., str2045およびstrLです。
str1, str2, ..., str2045は固定長の保存形式です。変数mystrの保存形式が、str8であれば、mystrのそれぞれの値を保存するために8バイトが利用されます。2,000の観測数があれば、全体で16,000バイトを利用されています。
文字列の長さと保存形式は一致しません。変数mystrの保存形式がstr8であっても、すべての観測値が8文字であるということではありません。最大の長さが8文字ということです。各観測値の文字列の長さはそれぞれ、0,1, ..., 8を取ることができます。しかし、各文字列は8バイトを必要とします。
文字列の保存形式は自動的に決定されるので、注意を払う必要はありません。保存形式がstr8の変数myvarの3つ目の観測値を「Longer than 8」と書き換えると、保存形式は自動的にstr13に変更されます。
変数myvarの3つ目の観測値を2,045文字よりも長い文字列に変更すると、保存形式はstrLとなります。
strL変数は2,045文字より長いものである必要はありません、それ以上または以下であることもあります。strL変数は可変長で保存されます。例えば、変数myothervarがstrLで、3つ目の観測値が「this」であるとします。この観測値で使用されるメモリは64+4+1=69バイトとなります。64バイトは追跡情報です、4バイトは（4文字の）データの内容です、1バイトは文字列の区切りを示しています。5番目の観測値が2,000,000文字を含む文字列であれば、必要なメモリ量は64+2,000,000+1=2,000,069バイトとなります。
str1, str2, ..., str2045とstrLのさらなる違いは、str#保存形式はASCII文字列しか格納できないということです。strLはASCIIまたはバイナリ文字列を格納できます。つまり、strLはワードドキュメントやJPEGファイルなどを格納することが可能です。

タイトルケース、タイトルケース文字列およびUnicodeタイトルケース文字列 タイトルケースは、文中のキーワードの大文字にすることです。Stataでは、(a) 文字列の最初の文字の大文字化、と(b) 文字以外に続く文字をそれぞれ大文字とするの、いずれかです。文字列内の語の重要度や同じ語の文字以外に続く文字を区別しません。例えば、「it’s」は「It’

S」となります。

上記の(a), (b)のいずれかに当てはまれば、それはtitle-cased stringです。strproper()関数で、Zen and the Art of Motorcycle Maintenanceという本のタイトルを変換すると、StataはZe and The Art Of Motorcycle Maintenanceという結果を返します。

Unicode title-scaled stringはUnicode title-scaledが適用されているUnicode文字列です。これは、おおよそUnicodeの最初の文字を大文字に変換したものと言うことができます。Title-scaling文字はロケールに依存するため、ロケールが異なれば同じ文字であっても異なるtitlecaseとなることがあります。例えば、いくつかのロケールでは、語頭の大文字がアクセントではないことがあります。

すべての文字がプレーンASCIIで、かつロケールが英語であれば、結果に違いはありません。例えば、英語ロケールにustrtitle()関数を適用しても、結果はZen And The Art Of Motorcycle Maintenanceを返します。

言語（ロケール）によって適切な大文字化のルールを適用するにはustrtitle()関数を利用してください。

Unicode Unicodeは考えられるすべての言語・死語に対応したエンコーディング規則です。Unicodeは様々な言語の文字に対応するように設計された一連のエンコーディングシステムです。Unicodeは文字とそれらのエンコーディングを定義するだけではなく、任意の言語（ロケール）に対して行われる、大文字変換や並び替えなどの操作のルールも定義しています。Unicodeエンコーディングをm UTF-8, UTF-16, UTF-32があり、StataはUTF-8を使用します。

Unicode文字 技術的には、Unicodeエンコーディングを伴う文字はすべてのUnicode文字です。通常は、プレーンASCII文字と区別して用いられます。

Unicode正規化 Unicode正規化は共通した表現を利用して、いくつかの方法でエンコードされたUnicode文字列を共通の見た目で表示することが可能になります。実際に利用することは稀ですが、Stataは異なる正規化フォーム間で変更を行うustrnormalize()関数を提供します。

例えば、スペイン語のティルダ付きの小文字ｎ「ñ」を検索するとします。この文字は単一のコードポイントU+00F1としてエンコードされている可能性があります。しかし、U+0303（ティルダ）が付属するU+006E（ラテン文字の「ｎ」）はUnicodeではU+00F1と等価です。このような外見上の特徴性は正準等価と呼ばれます。単一コードのフォームはcannonical composited formと、複数コードのものはcanonical decomposed formと呼ばれます。正規化では、内部的に1つ以上の文字列が対応するように正準等価の関係にあるフォームに修正します。文字列がこれらのフォームを混合している場合、文字列やサブストリングを検索する際には、文字列を正規化することが望ましいと思われます。

Unicodeの正規化は同じ意味を持っていながら、外見が異なって表示される文字の存在を許容します。例えば、コードポイントU+FB00（「ff」）を、U+0066 U+0066とエンコードされる、2つのラテン文字「f」と等価としてソートしたいとします。これは互換等価と言います。

Unicode title-cased string titlecase, title-cased string, Unicode title-cased stringを参照。

UTF-8 UTF-8はUniversal character set + Transformation Format - 8 bitの略です。これは、ASCIIとの後方互換性のあるUnicodeエンコーディングシステムであり、Stata14で使用されているものです。

値ラベル 値ラベルは数値データと数値の意味を描写する言葉を結びつけるものです。例えば、病気を示す変数では、1を陽性、0を陰性として結びつけます。詳細は[U] **12.6.3. 値ラベル**をご覧ください。

変数リスト 変数リストは特定の方法で記入された変数のリストです：変数名またはその省略形、アスタリスクを使用して複数の変数を指定することができます。例えば、income*または*1995のように入力して、incomeから始まるすべての変数または、1995で終わるすべての変数を指定します　　。mpg-weightのようにダッシュ (-) で2つの変数を結ぶことで、2つの間のすべての変数を指定します。詳細は、[U] **11.4 変数名と変数リスト**をご覧ください。

索引と著者索引

別途、*Stata Index* 内にあるcombined subject indexとcombined author indexを参照ください。