~サルでもわかる Stata 入門~

澤田ゼミ 2017 年度 version

著者：金丸、八下田

目次

赤文字はコードの基本形

青文字は実際にデータを用いたコード例

1 章　scalar、演算子

この章では、基礎的な概念である scalar を学びます。

基本的な四則演算をできるようになり、論理演算子の扱いがわかるようになるのが目標です。

使用ファイルは特にありません。

Stata で四則演算をしたいとき、次回以降扱う変数では不便です。これは、変数は列ベクトルであるからです。

そこで、scalar を導入します。

1. scalar の定義と四則演算、論理演算子

1.1 scalar の定義と表示

[scalar スカラー名=数式 or"文字列"]　で、scalar を定義することができる。

[display スカラー名]で、スカラーを表示する。

使い道はないが、一応文字列は""で囲う。

```
scalar sc="hello world"
display sc
```

1.2 四則演算

計算したいときは、まず scalar を定義し、それを表示する形をとる。

四則は+-*/

① オーソドックスな足し算。

```
scalar s1=3+2
display s1
```

②（）の使い方や、足し算割り算は通常の表記と同じ。

```
scalar s2=2*3/(5-1)
display s2
```

③指数とルート(ルートはもちろん 1/2 乗でも OK)。

```
scalar s3=sqrt(3^4)
display s3
```

④既に定義した scalar は式中にも入れられる。ln()は自然対数。

```
scalar s4=ln(10*28/7-s1)
display s4
```

1.3 関数の利用

関数も利用可能

⑤max, min

```
scalar s5=max(s1,s2,s3,s4)
scalar s6=min(s1,s2,s3,s4)
display s5 s6
```

1.4 floor, round ceil

⑥floor(N)は N を越えない最大の整数。ceil(N)は N を越える最小の整数。round(N)は四捨五入した結果の整数

```
scalar s7=floor(3.3)
scalar s8=ceil(3.3)
display s7,s8
```

1.5 論理演算子

いずれ使うので覚えましょう。=は代入という意味でのイコール、==が一般的なイコールの意味なのに注意。!=は≠

<, >, <=, >=, ==, !=

&は and、|は or

2 章　変数の管理

この章では、Stata で最も頻繁に使う、変数の基本的な管理の仕方を学びます。

変数の作成や整理ができるようになるのが目標です。

使用ファイルとして、example.dta を用います。これは、一応生徒何人かのテストの点数や身長などをまとめたものですが、内容に意味はありません。

Stata では、列ベクトルである変数を利用して統計分析を行うのが基本です。

2. 変数の基本的管理

2.1 既にある変数の消去

頻出の drop,keep

[drop 変数名] は変数自体を落とす。[drop_all]で全変数、[drop val*]で、val から始まる全変数を落とす。

```
drop error
```

[drop if 条件]は条件に合ったサンプルを落とす。keep は逆に残す。

```
drop if height>=200
```

2.2 変数の並び替え

sort は並び替えを行う。複数の場合は、まず sex、次に height で並び替え。

```
sort sex height
```

2.3　変数の作成

2.3.1　変数の作成①gen

①全てのサンプルで２をとる val という変数を作る。

```
gen val=2
```

②sex が male となるときだけ 1 をとる変数を作る。

```
gen male=1 if sex=="male"
```

③height が 175 以上のとき 1 をとる変数を作る。

```
gen high=1 if height>=175
```

④国語と算数の合計値をとる変数を作る。

```
gen stotal=lang+math
```

⑤身長の 2 乗項となる変数を作る。

```
gen height2=height^2
```

2.3.2 変数の作成②egen

変数の作成に関数を使う場合は基本的に egen を使う。

①国語の平均点を、全サンプルに入れる変数を作る。

```
egen ave=mean(lang)
```

2.3.3 特殊な例（sum 関数）

①[egen 新変数名=sum(変数名)]で、変数の合計値を全サンプルに入れる変数を作る。 ,by(変数名)はオプションで、変数ごとの合計値にできる（今回だと性別ごとの合計点）。

```
egen sum1=sum(math), by(sex)
```

②[gen 新変数名=sum(変数名)]だと上から順に足していった値が入る。（by オプションは使えない）

```
gen sum2=sum(math)
```

補足：sum や mean は scalar の中では使えない。例えば mean を scalar 値として利用したい場合、[gen 新変数名=mean(変数)　scalar スカラー名=変数名[1]]のようにする。(後で別の方法も示す)

```
scalar avescalar=ave[1]
display avescalar
```

2.4 string 形式の解消（group 関数）、ダミー変数の作成

2.4.1 group 関数

group 関数は、変数内で同じ観測値ごとに数字を割り当てる。ID を振るようなもの。

使用データだと、student が string(文字列)形式で使いづらいので、group 関数を使う。

```
egen StudentID=group(student)
```

2.4.2 ダミー変数の作成（tab+gen）

tab は次回詳述するが、[tab 変数, gen(新変数名)]でダミー変数を一気に作成することができる。カテゴリーの数（今回のデータの sex だと male と female の２つ）分、その値をとるときに１、その他の場合は０をとる変数が作成される。

手動で条件分を使って作るより楽。

```
tab sex, gen(sex)
```

2.5 名前や観測値の変更

2.5.1 観測値の変更（recode）

[recode 変数名　（○＝△）（△＝□）]で、観測値の○を△に、△を□に変更できる。

Stata では、欠損値は.で表すことに注意。

```
recode male(1=2)(.=1)
```

2.5.2 変数名の変更（rename）

[rename 変数名　新変数名]で、変数名を変えられる。

```
rename student seito
```

2.6 変数のラベリング

2.6.1 ラベリングの確認（describe）

[describe]で変数のラベリングの状況を確認する。(次回の範囲)

```
describe
```

2.6.2 変数に新しくラベルを付ける。(label var)

[label var 変数名　"ラベル内容"]で、変数にラベルを付けられる。一般に変数の定義を書くことが多い。

label var math "scores of the math test"

2.6.3 変数に新しく値ラベルを付け、変数に付与する。（label define, label val）

[label define 値ラベル名　値１　観測値１　値２　観測値２…]で、観測値１に値１を、観測値２に値２を付与する値ラベルを定義できる。

例えば、０、１からなるカテゴリ変数があるとき、０に"yes", １に"no"を付与できる。

これを[label val 変数名　値ラベル名]で、変数に貼り付ける。このとき変数は category 変数でないといけない。

```
egen sex3=group(sex)
label define sexlabel 2 "maleeeee" 1 "female"
label val sex3 sexlabel
```

1 行目…string 形式の変数には値ラベルが付与できないので、まず group 関数で category 変数にしている。

2 行目…２に"male"、１に"female"を対応させる、sexlabel という value label を作る。

3 行目…sex3 という変数に、上で作った sexlabel を対応させる。

ポイントは、value label が変数とは独立にあって、それを変数にくっつけるイメージだということ。

2.6.4 ラベルを更新する、ラベルを消す

既に値ラベルが付いていて、それを変えたいときは、modify、付け加えるときは add を使う。

```
label define sexlabel 2 "male", modify
```

ラベルを落とすには label drop を用いる。

```
label drop sexlabel
```

3 章　変数の基本統計情報を見る

この章では、変数の基本的な統計情報を見る方法を学びます。これは、回帰分析などを行う前によく行います。

変数の統計情報を自由に見られるようになるのが目標です。

使用ファイルとして、**example.dta** を用います。

3. 変数の統計情報

3.1 変数の Stata 上の格納情報を見る(des)

describe(des)で各変数の格納形式やラベリング状態、書式などがわかる。

```
des
```

3.2 変数の平均や標準偏差を見る(sum)

3.2.1 基本統計量の参照(sum)

[sum(変数名)]でサンプル数、平均、標準偏差などの基本統計量がわかる。

```
sum(math)
```

3.2.2 基本統計量の scalar としての利用

sum の後、[return list]で、求めた値をスカラー値で保存する方法がわかる。下の例は、観測数、平均、合計。

```
return list
scalar e1=r(N)
scalar e2=r(mean)
scalar e3=r(sum)
display e1 e2 e3
```

3.3 変数の観測値の密度や度数表を見る。（tab）

[tab 変数名]で観測値ごとの密度がわかる。

```
tab height
```

[tab 変数名　変数名]で、2 変数の間の度数表がわかる。

tab sex breakfast

3.4 １つの変数のヒストグラムを見る。（hist）

[hist　変数名]で、ヒストグラム表示が可能。bin の長さなどのオプションもあるので hist の help 参照。

hist height, wid(15)

3.5 ２変数の分布図を書く。（scatter）

[scatter 変数名　変数名]で、分布図が表示できる。

scatter lang math

3.6 複数の変数の間の相関係数を表示する。（corr）

[corr(変数名　変数名　変数名…)]で、各変数間の相関係数が表示される。（pwcorr でも同様）

corr(lang math height)

3.7 条件を満たす数を数える。（count）

数を数える count 関数。[by 変数名: count if 条件]　で、条件を満たすサンプルの数を返す。

by は、ある変数の観測値ごとにというオプションなのでなくてもいい。by を使う場合、by 以下の変数での sort が必要。

sort sex

by sex: count if math>=80

4 章　回帰分析と t 検定

この章では、最も基本的な統計分析である回帰分析と t 検定を行う方法を学びます。

回帰分析をできるようになるのが目標です。

使用ファイルとして、**example.dta** を用います。

4. 回帰分析、t 検定

4.1 回帰分析

4.1.1 回帰分析の基本(reg)

回帰分析の基本は、[reg 被説明変数　説明変数　説明変数…]

下の例は国語の点数を被説明変数、数学の点数と身長を説明変数にしての重回帰分析。

```
reg lang math height
```

4.1.2 回帰結果の利用①推定値の保存、表示

[est store 推定結果名]で、推定結果を保存できる。

[est table 推定結果名(複数も可)]で、推定された係数を表にする。

推定結果名カンマ以降で、載せる推定結果を選べる。（b＝係数、t＝ t 値、p＝p 値など、est table の help 参照）

他にも有効数字の桁数を変えたり、決定係数を加えたりなどが可能。(stats 以下は reg の help 参照)

今回は係数、t 値、p 値、観測数、修正済み決定係数を表示。

```
reg lang math height
est store e1
est table e1
est table e1, b(%7.4f) t(%7.4f) p(%7.4f) stats(N, r2_a)
```

4.1.3 回帰結果の利用②推定値のスカラー利用

reg の後システム変数を用いれば、推定値をスカラーとして利用できる。

_b[_cons]で、定数項、_b[説明変数名]で、その説明変数の推定された係数をスカラーとして利用可能。

```
reg lang math height
gen fittedvalue=_b[_cons]+_b[math]*math+_b[height]*height
```

上の例では、（定数項＋係数×説明変数＋係数×説明変数）を計算して手動で fitted value を作成している。

4.1.4 回帰結果の利用③fitted value

reg の後[predict 新変数名]で、fitted value を作成できる。[predict 新変数名,residual]なら残差を保存。

```
reg lang math height
predict bhat
```

一個上で手動で作った fittedvalue と一致するか確認しよう。

4.1.5 回帰分析で用いるオプション

[quietly]は結果の省略、[,robust]は頑健標準誤差、[if]は条件の付与。

```
quietly reg lang math
reg lang math, robust
reg lang math if height>=160
```

4.1.6 交差項やカテゴリカル変数の利用

i.変数を使えば、カテゴリカル変数を簡単に（カテゴリー数ー１）個入れられる。（値ごとにダミー変数を入れなくてもいい）

```
reg lang i.breakfast
```

#を用いると簡単に交差項を作れる。[reg 被説明変数　ダミー変数#（or##）ダミー変数]で、各ダミーとその交互作用が一気に入る。#と##で少し違うが、##の方が便利と思う。

```
reg lang male##i.breakfast
```

4.2 t 検定

[ttest 変数名＝変数名]で、2 変数に統計的に有意な差があるかの t 検定ができる。

```
ttest math=lang
```

4.3 F 検定

reg のあとに[test 説明変数…]とすることで説明変数の係数が全て 0 という帰無仮説を検証する F 検定ができる。(通常 reg の結果表の中に含まれる。)

```
reg height lang math
```

```
test lang math
```

5 章　ファイルの結合、グラフ表示

この章では、ファイル結合の方法と、分析結果のグラフ表示の方法を学びます。

ファイル結合とグラフ表示の一番基本となるコマンドを覚えるのが目標です。

ファイル結合のところは使用ファイルはありません。グラフ表示のところでは **example.dta** を用います。

5. ファイルの結合、グラフ表示

5.1 ファイルの結合

5.1.1 ファイル結合のファイル結合の前提

Stata を使っていると、１つの統計分析を行うために２つのファイルを結合したいということが頻繁にある。

２つのファイルを結合するコマンドは merge と append。基本的なクロスセクションでは merge を使うが、パネルデータでは

形式によっては append を用いる。

基本的に２つのファイルで同じ名前のキー変数を指定して、それを基に結合する。

5.1.2 ファイル結合の基本形（merge）

[merge 1:1 キー変数 using "ファイル名"]が基本形。

1:1 のところ、1:m, m:1, m:m にもなる。ここの：は、master 側:using 側を表す。これは、キー変数に指定された

変数について、観測値にかぶりがある方を m(many)で表す。

例えば、master 側（現在開いている方）では student というキー変数について、2 回以上登場する生徒が存在し、

using 側（結合する方）では全ての生徒が１度だけ登場するとする。その場合は、m:1 を用いるということになる。

m:m を用いるのが一番無難ではある。詳しくは help merge 参照。

5.1.3 結合状況の確認

merge を行うと、_merge という変数が自動生成される。これは、キー変数のある観測値が、_merge==1 のときは master ファイルのみ、_merge==2 のときは using ファイルのみに存在し、_merge==3 のときはいずれにも存在するということである。

5.2 グラフ表示

5.2.1 ２変数のグラフ作成（twoway）

２変数のグラフを書きたいときは、以前少し紹介した scatter などでもいいが、twoway のコードが非常に有用。

[twoway]を使えば複数のグラフを同一平面上に表示できる。

下では、分布図と、fitted value を表示している。他のグラフ形式については help twoway 参照。

```
twoway(scatter lang math)(lfit lang math)
```

5.2.2 グラフ作成のオプション

条件を指定したり、大きさや色を変えたりなど色々できる。詳しくは help twoway 参照。

```
twoway(scatter lang math, mc(red) ms(huge))(lfit lang math if height>160)
```

5.3 求めた結果の外部への出力(outreg2)

[outreg2]コマンドを使えば、回帰結果などをワードファイルにして出力できる。

ここではコードは載せないので、詳しくは help outreg2。

keep コマンドで説明変数の内載せたいもののみを選んだりもできるので、論文を書くときには使う。

6 章　プログラミング

この章では、簡単なプログラミング方法を学びます。

こうした方法によって処理を簡略化できることを理解することが目標です。

使用ファイルとして、example.dta を用います。

6. プログラミング

6.1 プログラミングの基本

Stata では、プログラミングに近いコードを利用することで、処理を大幅に簡略化できる。

その方法は program や foreach などいくつかあるが、基本的に同じものを何度も入れるといった繰り返し処理はプログラミングを用いて簡略化できる可能性が高い。

6.2　program コマンド

program を用いると、同じ処理を複数回したい場合などに、その処理の一部を変数にして、値を簡単に代入できる。

下は、数学の点数を被説明変数、身長を説明変数にして回帰したもの。身長ごとに条件付けしているので、program を組んで大幅に簡略化した。

```
capture program drop program1
program program1
quietly reg `1' height if height>=`2'
est store k`2'
end

program1 math 175
program1 math 170
program1 math 165
program1 math 160
program1 math 155
program1 math 150
```

```
est table k*
```

1 行目…[capture program drop program 名]で、同名のプログラムを落とす。これはそのまま覚えてください。

2~5 行目…[program program 名~（処理）~end]で、プログラムを作成。処理内で、`1',`2'などの形で、変数や文字を置き換える。

7~12 行目…[program 名　代入する値　代入する値　…]で、program の`1'`2'...の部分に順に値が入って実行される。この場合には、{`1'に math、`2'に 175}、{`1’ に math、`2'に 170}…と入っていく。

14 行目…k から始まる推定結果を一括表示。今回は k150~k175 が一気に表示されるはず。

6.3 foreach コマンド

foreach を用いると、1 変数の場合 program より簡単に値を代入できる。

基本形は、[foreach i in 入れる値… {処理}]。

代入するところにとる引数は`i'。下の例は、１から７まで足すプログラム。

```
scalar sum=0
foreach i in 1 2 3 4 5 6 7{
scalar sum=sum+`i'
}
display sum
```

下のようにすれば、１から 100 までなどと指定することもできる。例は、1 から 100 まで足すプログラム。

```
scalar sum=0
foreach i of num 1/100{
scalar sum=sum+`i'
}
display sum
```

6.4 while コマンド

あまり使わないが、while 構文なども存在する。

[while 条件 {処理}]が基本形。条件が満たされる間処理を繰り返す。

下の例では、8*1*2*3*4*5*6*7*8*9*を計算している。

```
scalar aa=1

scalar bb=8


while aa<10{

scalar bb=bb*aa

scalar aa=aa+1

}

display bb
```

7 章　時系列分析、パネルデータ分析

この章では、時系列分析、パネルデータ分析の初歩を導入します。

時系列、パネルを扱うときにまず何をするべきかを理解するのが目標です。

使用ファイルとして、時系列では example2.dta、パネルでは example3.dta を用います。

7. 時系列、パネル

7.1 時系列データの分析

7.1.1 date 関数

まずは Stata の time-variable の格納形式を知るために date 関数を導入する。

date 関数は、日次や月次データで、time-variable が string 形式の場合に用いる。

文字形式の日付などを Stata が扱える形に変換する。

```
gen date=date(datestring, "YMD")
```

[gen 新変数名=date(変数名, "YMD")]のような形が基本形。"YMD"のところは、年月日の順ならこう書く。日月年なら"DMY"

Stata では、日時データは 1960 年の何日かを基準にして、そこからの総日数で表す。

一般に論文では年次データを使うことが多いので、見る機会は多くはない。

7.1.2 time-variable の設定（tset）

時系列データ分析は time-variable の設定を行うところから始まる。コマンドは[tset]。

このとき time-variable が Stata の時間形式になっていることが必要。それを行うのが上の date 関数。西暦の場合、特に date 関数など使わなくても普通にできる。

time-variable に被りがあるとエラーになるので注意。

```
tset date
```

7.1.3 時系列データの視覚化（tsline）

時系列データでは、まずデータを視覚化する[tsline]をよく用いる。

```
tsline(quant aveprice)
```

twoway(tsline quant, yaxis(1))(tsline aveprice, yaxis(2))

下のコードは、ｙ軸を 2 変数で変えるためのものだが、非常に有用。

7.1.4 時系列分析のキーとなる単位根検定（dfuller）

時系列分析でまず行うのは単位根検定。

単位根検定は拡張 DF 検定を[dfuller 変数名]で行うことができる。p-value が 0.05 以下なら有意水準 5%で単位根は存在しないということになる。trend などのオプションは help dfuller 参照。

dfuller quant

より発展的な時系列分析のやり方は自分で学んでください。

7.2 パネルデータの導入

7.2.1 パネルの x と t の設定(xtset)

パネルデータの導入は、まず xtset から行う。[xtset x 側の変数　t 側の変数]が基本。

（１行目は、x 側が string 形式だと使えないので直しているだけ）

egen itemID=group(itemname)

xtset itemID date

7.2.2 パネルデータ分析（POLS,FE,RE）

パネルデータ分析には、POLS、FE、RE がある。

reg quant aveprice

xtreg quant aveprice

xtreg quant aveprice, fe

xtreg quant aveprice i.date, fe

一つ目が POLS、2 つ目が RE、3 つ目・4 つ目が FE の書き方。4 つ目は、i.date によって時間ダミーをすべて入れた形。

どれがいいかの判別とかは自分で学んでください笑。

7.3 パネルデータや時系列データで利用可能なシステム変数

Stata にはいくつか有用なシステム変数（システム変数を変数の前後につければ自動的に変換された変数となる）がある。

①ある観測値の上からの位置（ID）を表す[_n]

```
display height[2]
```

height の上から 2 番目の観測値を表示できる。

```
gen height2=height[_n-1]
```

height に対して、すべて一つ上の観測値をとる height2 を作成。これは、観測値の被りを消したい場合などに利用できる。これはクロスセクションでも利用可能。

②ある変数の一期前の値を表す l.変数(lag オペレータ)

[l.変数名]で、1 期前の値、

[l5.変数名]で、5 期前の値を表示する。

③ある変数の一期先の値を表す f.変数(lead オペレータ)

[f.変数名]で、1 期先の値、

[f5.変数名]で、5 期先の値を表示する。

④差分を表す差分演算子 d.変数(差分オペレータ)

[d.変数名]で、{変数[_n]-変数[_n-1]}を表示。

8 章　発展的な分析

この章では、発展的な分析を学びます。

実際に分析を行うときに何で **help** すればいいかをわかるようになるのが目標です。

使用ファイルはありません。また、今回のみは **dofile** を回してもエラーとなります。

8. 発展的な分析

8.1　操作変数法（ivreg)

内生変数と操作変数を＝で繋ぐという点だけ覚えておけば、後はその外にコントロール変数などを書くだけ。

[ivreg 2sls 被説明変数 外生説明変数…　（内生変数 = 操作変数　コントロール変数…)]

例えば、双子ダミーを IV に使って、子供の数の増加が子供の教育年数に与える影響を見る場合、

ivreg 2sls schoolyear parentincome race.. (familysize=twindummy birthage..)

（＊ここに関してはデータが存在しないので、Stata に打ち込んでも意味はありません）

などとなる。familysize が schoolyear の内生変数なので、schoolyear の外生変数である twindummy で置き換えているというイメージ。

2sls のところは、推定方法の指定になる。他にも liml(最尤法)などがあるが、基本 2sls と書いておけば間違いないと思われる。

8.2 probit モデル(probit)

probit モデルは、reg を prob に変えるだけ。

ただし、probit や logit では、出てきた係数≠限界効果。そのため、margins コマンドを用いて限界効果を出す必要がある。その方法は 2 通りあるが、詳しくは probit や logit をやる際に自分で学んでください。

[prob 被説明変数　説明変数…]

[margins, dydx(*) atmeans]

[margins, dydx(*)]

8.3 logit モデル(logit)

logit モデルは probit モデルとほぼ同じ。

[logit 被説明変数　説明変数…]

[margins, dydx(*) atmeans]

[margins, dydx(*)

8.4 tobit モデル(tobit)

tobit は、普通の回帰にカットオフラインを追加するという形をとる。

reg に、lower-limit(ll)か upper-limit(ul)のカットオフのラインを指定するオプションを付け、ll(0)のような形で、実際に数字を指定するというイメージ。

[tobit 被説明変数　説明変数…, ll(数字)]

8.5　heckman の 2 段階推定(heckman)

heckman は、1 段階目のダミー変数の推定に用いる説明変数を select の中に入れて、twostep をつけるだけ。

heckman はいくつかタイプがあるが、それは heckman のところで学んでください。

[heckman 被説明変数　説明変数… ,select(1 段階目の説明変数…) twostep]