

Stata 入门

慧航

2020 年 9 月

目录

- ① Stata 入门
- ② 文件与变量
- ③ 描述性统计
- ④ 数据处理
- ⑤ 作图
- ⑥ 标量和宏
- ⑦ 控制语句
- ⑧ Python 接口

关于本视频

- 联系方式: zhiyuezen@126.com, 或者通过知乎、Bilibili 私信
- 数据地址 (两者都可以):
 - https://github.com/sijichun/MathStatsCode/tree/master/code_in_notes/datasets
 - https://gitee.com/sijc/MathStatsCode/tree/master/code_in_notes/datasets
- 讲义地址: <https://gitee.com/sijc/MathStatsCode/raw/master/Stata.pdf>
- PPT 地址: <https://gitee.com/sijc/MathStatsCode/raw/master/StataPPT.pdf>
- 除非本人同意, 禁止商业用途
- 关于软件学习:
 - 最好的学习方法是动手做
 - Stata 本身提供了丰富的学习材料

Stata 简单介绍

- Stata 的版本：
 - ① IC：支持 2048 个变量
 - ② SE：支持 32767 个变量
 - ③ MP：支持 120000 个变量，且并行计算
- 窗口介绍
 - 菜单
 - 结果
 - 变量
 - 命令历史
 - 命令输入框

几个常用的命令

- help: 帮助, 任何有的、没有的命令都可以 help
- set 命令:
 - set more off: 关闭—more—的提示
 - set maxvar : 修改变量个数上限
 - set matsize : 修改矩阵的大小, 偶尔会需要
- clear: 清除内存中的所有数据
- ssc: 安装软件包 (也可以使用 help 安装)

使用 Stata 的方式

我们可以通过两种方式操作 Stata：

- 窗口方式：使用菜单完成操作
 - 优点：简单、直观
 - 缺点：很多时候不能复现结果、效率低
- 命令方式：在命令输入框中输入命令
 - 优点：比鼠标点击更快
 - 缺点：很多时候不能复现结果、需要记忆命令
- do-file 方式：把命令写在 do 文件里面执行
 - 优点：能够复现结果、效率较高、功能强大
 - 缺点：需要记忆命令
 - 是操作 Stata 的最主要方式

do-文件

- 在 Stata 中我们通常将命令都写在 do-文件中，批量运行
- 可以选中某一行，只运行这一行
- 可以使用 quietly 前缀或者 quietly {} 关闭输出
- 注释：
 - 「//」注释，可以用于每一行的末尾，以//开头的都是注释
 - 「*」注释，如果一行的开头是*，该行为注释
 - 「/* */」注释，跨行注释，在/* 以及 */之间的都是注释

数据的打开与导入

- Stata 本身的数据文件后缀名为 .dta，可以直接打开
 - 有时低版本的 Stata 在打开高版本 Stata 的数据文件时会有问题
- 此外 Stata 还支持从其他程序文件中导入数据：
 - csv 文件
 - Excel 文件
 - SPSS 数据
 - SAS 数据（有的格式打不开，需要在 SAS 中导出 Stata 格式）
 - 数据库（推荐使用 ODBC）
 -

Stata 中的命令

Stata 命令的套路:

```
1 [prefix :] command [ varlist ] [=exp] [if] [in]
2 [weight] [using filename]
3 [, options]
```

- prefix: 前缀, 如 by, bysort, bootstrap, simulate 等
- command: 命令, 必须要有
- varlist: 要处理的变量列表
- =exp: 表达式
- if、in: 选择某些观测进行操作
- weight: 权重
- using filename: 需要操作的文件
- , options: 逗号后面跟的是选项, 控制程序执行的方式, 如是否需要标准误调整等等。一条命令一个逗号。

文件的基本操作

- 改变工作目录：cd 命令
- 打开文件：use 命令，加 clear 选项可以先清除内存中的数据再打开

```
1 cd d:\sijichun\  
2 use cfps_adult.dta , clear
```

- 保存文件：save，加 replace 选项可以覆盖已有文件

```
1 save cleaned_data.dta , replace
```

良好的习惯

- 首先 cd 到工作文件夹，方便查找数据、保存数据
- 永远不要覆盖原始数据！

其他文件操作

- 删除文件：rm
- 列出当期工作目录所有文件：ls、dir
- 显示当前工作目录：pwd
- 导入数据：import
 - 建议使用 csv 格式在不同的软件中切换，通用性强
 - 如果数据是.mdb 格式：装 Access，使用 ODBC
- 导出数据：export

变量

- 在 Stata 中，变量 (**variable**) 的概念与统计学相同，通常以列表示变量，以行表示观测 (**observations**)。
- 当我们打开一个数据文件后，在变量列表中可以看到文件中所有的变量
- 如果要查看数据，使用 br 命令：

```

1  br    // 查看所有变量
2  br pid fid14 cfps_gender te4 // 只查看部分变量
3  br te4 if cfps_gender==1 // 只查看男性的教育

```

- 查看变量的窗口：查看和编辑，尽量不要打开编辑。
- 可以使用「order」命令更改变量的显示顺序

生成与修改变量

- 可以使用「generate」命令产生新的变量，该命令基本语法为：

```
1 gen [type] newvar =exp [if] [in]
```

- 可以使用「replace」命令产生新的变量，该命令基本语法为：

```
1 replace var =exp [if] [in]
```

生成与修改变量

产生变量示例

```
1 gen age=2014-cfps_birthy // 减法
2 gen log_income=log(p_income) // 函数
3 // 使用逻辑算式，逻辑为真则等于1
4 gen post90s= cfps_birthy >=1990
5 gen uni_rand=runiform() // 产生0-1的随机数
6 gen normal_rand=rnormal() // 产生标准正态随机数
```

实际上以上生成 post90s 的命令也可以改写为：

```
1 gen post90s= 1 if cfps_birthy >=1990
2 replace post90s=0 if post90==.
```

生成数据可以使用的信息

- 已有的变量的算数运算
- 函数 (help function)
- 在生成数据时还可以使用系统变量，常见的系统变量有：
 - `_n`: 观测的行号
 - `_N`: 观测数
 - `_b[varname]`: 上次回归中变量 `varname` 的回归系数
 - `_se[varname]`: 上次回归中变量 `varname` 的标准误
- 一条命令执行结束之后，还可以使用 `e()` 或者 `r()` 访问上次命令的结果。

变量标签

- 为了更方便的分辨每个变量的含义，可以为每个变量添加标签 (label)
- 使用「label variable」命令为变量添加标签：

```
1 label variable post90s "90后虚拟变量"
```


值标签

- 除了变量标签之外，我们还可以为变量的取值进行标记。
 - 比如，数据中 1 代表“90 后”，0 代表“90 前”，使用「br」命令查看数据时显示 0/1 并不直观
 - 可以使用值标签：

```
1 label def lab_post90 0 "90前"  
2 label def lab_post90 1 "90后", add  
3 label values post_90s lab_post90
```

- 以上代码中首先定义了一个值标签列表“lab_post90”，这个列表直接标明：1 代表“90 后”，0 代表“90 前”，最后使用「label variable」命令将值标签列表“lab_post90”应用在变量 post_90s 上。
- 再次「br」命令查看数据，可以发现变量 post_90s 的显示值变为蓝色，且不再是 0/1，而是“90 前/90 后”。

变量的排序

- 使用「sort」命令对数据进行排序（升序）

```
1 | sort cfps_birthy
```

- 也可以指定多个变量，将首先对前面的变量进行升序排序，当前面变量相同时，按照后面的变量升序排序：

```
1 | sort provcd14 cfps_birthy
```

Stata 中的缺失值

- 在 Stata 中，缺失值使用「.」来表示
- 产生缺失值的情况：
 - 数据本来就是缺失值
 - 不允许的运算，如 $\log(0)$
 - 缺失值参与运算，如「`exp(.)`」
- 在 Stata 的内部实现中，「.」实际上是 Inf ，即正无穷，因而是比任何数都大的数！
 - 缺失值「.」可以参与比较大小！
 - 因而上例中产生 `post90s` 变量的写法是错的，因为所有年龄缺失的观测都被算做了 90 后！
 - 正确写法：

```
1 gen post90s=cfps_birthy >=1990 if cfps_birthy !=.
```

滞后项的生成

- 在 Stata 中，可以使用 `tsset` 命令设定数据为时间序列数据：

```
1 tsset timevar
```

或者使用 `xtset` 命令设定数据为面板数据：

```
1 xtset idvar timevar
```

- 设定了时间结构之后，可以使用滞后算子（L）、向前算子（F）来产生相应的时间变量：

```
1 gen Lx=L.x // x 的一阶滞后  
2 gen L2x=L2.x // x 的二阶滞后  
3 gen F2x=F2.x // x 的一阶滞前
```

删除变量或者观测

- 如果需要删除变量，使用 drop 命令：

```
1 drop x // 删除变量 x
```

- 如果需要删除观测，使用 drop if 命令：

```
1 drop if cfps_birthy > 1990 // 删除 90 后
```

- 如果需要保留变量，其他都删除，使用 keep 命令：

```
1 keep x // 只保留 x，其他变量都删除
```

- 如果需要保留观测，使用 keep if 命令：

```
1 keep if cfps_birthy > 1990 // 保留 90 后
```

数据框

- 在 Stata16 中，首次加入了数据框（frames）的支持
- 借助这一工具，可以避免频繁打开和关闭数据集，或者频繁使用 `preserve/restore` 操作。
- 当我们打开 Stata 时，内存中默认使用「default」frame
- 有条件尽可能使用 frame：
 - 速度快
 - 代码简洁
- 可以使用 `frame dir` 命令查看现在内存中都有哪些数据框，其中 `default` 为打开 Stata 时默认的数据框

数据框的创建

数据框的创建有两种方法：

- 直接创建，比如以下代码创建了一个名称为 sub_data 的数据框：

```
1 frames create sub_data
```

- 创建一个带有变量的数据框，比如以下代码创建了一个名称为 sub_data1 的数据框，同时在这个数据框中创建两个变量 a,b：

```
1 frames create sub_data1 id year
```

之后可以使用 frame post 命令添加数据（数据需要用括号 () 包括），post 后数据的顺序与 create 时的顺序一致：

```
1 frames post sub_data1 (1101) (1999)
```

数据框的使用

为了使用数据框，可以有以下两种方式：

- ① 直接切换数据框，使用「frame change」命令切换到已经创建的数据框中：

```

1 use datasets / cfps_adult
2 frames create family
3 frames change family
4 use datasets / cfps_family_econ
5 frames dir

```

- ② 使用 frames 前缀，接上例：

```

1 frames change default
2 frame family: su fincome1

```

以上两者的区别：frames change 改变当前操作的 frame，而 frame 前缀不改变。

数据框的拷贝

数据框的拷贝：

- 拷贝整个数据集：

```
1 frame copy family new_family
```

- 拷贝一个数据集的几个变量：

```
1 frame put fid14-ft1 , into(family_subvar)
```

- 拷贝部分观测：

```
1 frame put if provcd14==11, into(family_subobs)
```

数据框的删除

数据框的删除包含以下几个操作：

- 删除单个数据框：

```
1 frame drop family
```

- 清除内存中所有的数据框及其数据：

```
1 clear frames
```

或者可以使用 `frames reset` 命令，两者等价。

连续变量的描述性统计

可以使用 summarize 命令进行描述性统计：

- 单独使用 su 命令，汇报样本量、均值、标准差、最小值、最大值

```
1 su qg12 qg1203 p_income  
2 su qg12* // 所有以 qg12 开头的变量的描述性统计  
3 su qg12-p_income // qg12 到 p_income 的所有变量
```

- 加入 detail 选项，额外汇报分位数、偏度、峰度等

```
1 su p_income, de  
2 di r(mean)  
3 di r(sd)  
4 di r(skewness)
```

通配符的使用

Stata 中支持两类通配符：

- * 号，代表任意多个所有字符，比如：
 - qg12* 代表所有以 qg12 开头的变量
 - qea*1 代表所有以 qea 开头、以 1 结束的变量
- -号，代表从某个变量到某个变量，对应于变量列表中的顺序，比如：
 - 在 cfps_adult.dta 中，qp101- qp2032 代表 qp101 qp102 qp2031 qp2032 四个变量

描述性统计的输出

论文中的描述性统计表格可以使用 `outreg2` 输出（需要首先使用 `ssc install outreg2` 进行安装）：

```
1  outreg2 using myfile , [{sum(log)|sum(detail)}
2  replace eqkeep() eqdrop() keep() drop()]
```

其中：

- `sum(log)` 代表最常见的描述性统计，`sum(detail)` 代表详细的描述性统计
- `replace` 代表覆盖已经存在的文件
- `eqkeep()` `eqdrop()` 代表要放弃或保留的统计量
- `keep()` `drop()` 代表要放弃或保留的变量

描述性统计的输出

```

1 use "cfps_adult.dta", clear
2 outreg2 using summary.tex, replace sum(log)
3 keep(qg12-p_income)
    
```

VARIABLES	(1) N	(2) mean	(3) sd	(4) min	(5) max
qg12	37,147	8,415	18,816	-8	800,000
qg1203	37,147	1,649	18,023	-8	3.000e+06
p_income	37,086	8,934	18,819	-9	442,000

离散变量的描述性统计

- 离散变量最简单的描述性统计是数频数（或者频率），使用 `tab` 命令，比如：

```
1 drop if te4 < 0
2 tab te4
```

使用 `tab` 命令时要求不能有负值，因此我们先把负值删掉了。

- 如果有两个离散变量，可以使用双向的 `tab`：

```
1 drop if te4 < 0
2 tab te4 cfps_gender
```

虚拟变量的生成

生成虚拟变量有两种方法：

- 使用 `tab` 命令，加入 `gen` 选项，如：

```

1  drop if te4 < 0
2  tab te4 , gen(educ)
3  su educ*
    
```

- 使用 `i.` 操作符，如：

```

1  drop if te4 < 0
2  su i.te4
    
```


虚拟变量的使用

- i. 操作符还可以「相乘」，比如：

```
1 drop if te4 < 0
2 su i . te4 # i . cfps_gender
```

代表了教育程度和性别的 14 种组合

- i. 操作符也可以与 c. 操作符相乘，比如：

```
1 drop if te4 < 0
2 su i . te4 # c . p_income
```

分组操作

使用 `by` 或者 `bysort` 前缀可以实现分组操作：

```
1 by varlist [, sort rc0]: stata_cmd  
2 bysort varlist [, rc0]: stata_cmd
```

比如如下使用教育程度作为分组，分别进行描述性统计：

```
1 bysort te4: su p_income
```

生成数据的进阶方法

Stata 中除了可以使用 `gen` 命令生成变量，`egen` 命令也可以生成变量，是 `gen` 命令的扩展版：

- `gen` 命令只能同一行进行操作
- `egen` 命令可以进行跨行操作、跨列操作等复杂操作

其语法为：

```
1 egen newvar = fcn(arguments) [if] [in] [, options]
```

其中 `fcn` 为 `egen` 命令支持的函数，可以通过 `help egen` 查看。

egen 命令的函数

egen 命令有很多函数，最常用的有如下三类：

- 生成分组代码，比如：

```
1 egen provid=group(provcd14)
```

- 按行分组类：通常需要在 options 中加入 by(varlist) 选项，在运行时会首先根据 varlist 分组，再进行计算，常用的函数如：均值 (mean)、中位数 (median)、最大值 (max)、最小值 (min)、个数 (count)、求和 (total)、标准差 (sd) 等。比如：

```
1 egen f_num_of_adult=count(pid), by(fid14)
2 egen f_max_income=max(p_income), by(fid14)
3 egen f_min_income=min(p_income), by(fid14)
4 egen f_sum_income=total(p_income), by(fid14)
```

- 按列计算类：这类 fcn 一般 arguments 为变量列表，选项中也要求不能有 by(varlist) 选项，其功能是完成一些按列运算，这些运算通常是 gen 命令需要比较繁琐的操作才能完成的，比如：varlist 中元素的个数 (anycount)、varlist 中是否存在某个元素 (anymatch) 等

合并数据

合并数据有两种：

- 纵向的添加更多的观测：append 命令

```
1 use file1.dta, clear
2 append using file2.dta
3 save file3
```

- 横向的添加更多的变量：merge 命令

```
1 merge 1:1 varlist using filename [, options]
2 merge m:1 varlist using filename [, options]
3 merge 1:m varlist using filename [, options]
4 merge m:m varlist using filename [, options]
5 merge 1:1 _n using filename [, options]
```

merge 命令

- merge 1:1——主文件、using 文件一一对应
- merge m:1——主文件的 m 个观测对于与 using 文件的一个观测，比如主文件是家庭层面，但是有一个变量 provcd 是家庭所处省份，using 文件是省份级变量
- merge 1:m——跟上面反过来
- merge m:m——一般不会用
- merge 1:1 _n——按行匹配

merge 命令

_n	pid	fid	year	age	pincome
1	11	1	2010	29	150000
2	12	1	2010	28	100000
3	21	2	2010	35	200000
4	22	2	2010	33	80000
5	31	3	2010	54	20000

file1.dta

_n	fid	year	hincome
1	1	2010	256000
2	2	2010	300000
3	1	2012	300000
4	2	2012	350000

file2.dta

↓merge

_n	pid	fid	year	age	pincome	hincome	_merge
1	11	1	2010	29	150000	256000	3
2	12	1	2010	28	100000	256000	3
3	21	2	2010	35	200000	300000	3
4	22	2	2010	33	80000	300000	3
5	31	3	2010	54	20000	.	1
6	.	1	2012	.	.	300000	2
7	.	2	2012	.	.	350000	2

file3.dta

使用数据框合并数据

或者我们可以使用数据框进行操作。如果我们有两个 frame，我们可以使用「frlink」命令构建这两个 frame 之间的关系。值得注意的是，frlink 只允许 m:1 和 1:1 两种关系。

```

1 use datasets/cfps_adult
2 frames create family
3 frames change family
4 use datasets/cfps_family_econ
5 frames change default
6 frlink m:1 fid14 , frame(family)

```

之后可以使用「frget」命令从连接的 frame 中获取数据：

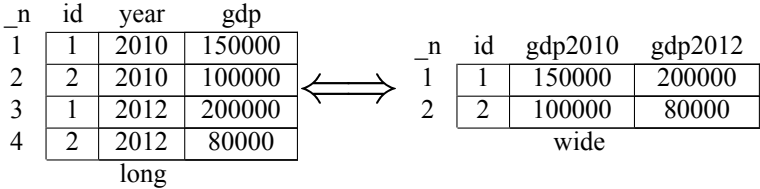
```

1 frget fincome1 , from(family)
2 frget ave_p_income=fincome1_per , from(family)
3 gen delta_p_income=p_income-ave_p_income

```


转置数据

Stata 中有「长」和「宽」两类数据格式：



使用如下命令相互转换：

```

1 reshape wide gdp, i(id) j(year)
2 reshape long gdp, i(id) j(year)

```

标量

标量 (scalar) 可以用来存储数字和字符串，而不仅仅是数字。可以使用 `scalar` 命令声明一个标量，比如：

```
1 scalar question="What is the answer? "  
2 scalar answer=21*2  
3 di question answer  
4 scalar randa=rnormal()  
5 scalar randb=runiform()  
6 scalar randb=randa+randb  
7 di randb
```

标量的问题：名称容易与变量名称混淆！因而除非必须，建议更多的使用局部宏

宏

在 Stata 中区分两种宏：

- 局部宏：即所谓的 local

- 其作用域被限制在一个程序（program）内部或者一个 do 文件内部。
- 使用「local」命令定义，引用时宏名称需要以左引号「'」和右引号「'」包围起来。例如：

```
1 local answer "42"  
2 di 'answer'
```

实际运行 di 命令时，程序会先将 'answer' 替换为 42 再运行。

- 全局宏：即所谓的 global

- 其作用域为全局，一处声明，到处可用。
- 使用「global」命令定义，引用时需要在宏名称前面加上「\$」符号

局部宏

- 需要注意的是，宏仅仅是做了字符串替换的工作，因而以下命令会报错：

```

1 local question "What is the answer?"
2 local answer "42."
3 di `question' `answer'

```

- 正确写法：

```

1 local question "What is the answer?"
2 local answer "42."
3 di "`question'" "`answer'"

```

局部宏

```
1 clear
2 set more off
3 set obs 100
4 scalar p=0.5
5 gen x=runiform()
6 gen y=rnormal()
7 local Knowledge "scatter"
8 local is "y"
9 local power "x"
10 local Francis "scale(scalar(p))"
11 local Bacon "'title("Knowledge is power.")'"
12 'Knowledge' 'is' 'power', 'Francis' 'Bacon'
```

局部宏

当然，局部宏也可以存储数字：

```
1 local a=2+2
2 di 4*`a'+2
```

但是一定要用等号，否则 Stata 不知道这是数字，会出现奇奇怪怪的问题：

```
1 local b 2+2
2 di 4*`b'+2
```

局部宏

最后，宏支持嵌套：

```
1 local s1 "string1"  
2 local s2 "string2"  
3 local n=2  
4 di "`s`n`'"
```

宏与数据框

值得注意的是，宏是与数据框是分割的，宏可以跨越数据框，比如：

```

1 use datasets/cfps_adult
2 frames create family
3 frames change family
4 use datasets/cfps_family_econ
5 su fincome1_per
6 local averincome=r(mean)
7 frames change default
8 gen demean_p_income=p_income-‘averincome’
    
```


条件语句

Stata 中的条件语句为：

```
1 if exp {  
2     commands  
3 }  
4 else if {  
5     commands  
6 }  
7 else {  
8     commands  
9 }
```

- else (包括 else if) 可以不出现
- exp 是一个逻辑语句，如果 exp 为真，那么就会执行大括号里面的语句，否则执行 else 大括号里面的语句。
- 在 Stata 中，左大括号后面不能跟任何字符（注释、空格除外），且右大括号必须单独一行

条件语句

```
1 local a=42
2 if 'a'==42 {
3     di "Yes, it is the answer!"
4 }
5 else if 'a'>42 {
6     di "Too big..."
7 }
8 else {
9     di "Too small..."
10 }
```

循环语句

Stata 中有三种循环语句：

- while 循环：

```
1 while exp {  
2     commands  
3 }
```

- forvalues 循环：

```
1 forvalues lname = range {  
2     commands  
3 }
```

- foreach 循环

while 循环

以下程序计算了： $\sum_{i=10}^{20} i$

```
1 local i=9
2 local sum=0
3 while `++i' <= 20 {
4     di `i'
5     local sum=`sum'+`i'
6 }
7 di `sum'
```

while 循环

在循环中，还可以使用「continue」命令来控制循环继续执行，使用「continue, break」命令来控制循环退出，比如上面的程序可以改写为：

```

1 local i=10
2 local sum=0
3 while 1 {
4     if 'i' <= 20 {
5         local sum='sum'+`i'
6         local i=`i'+1
7         continue
8     }
9     else {
10        continue, break
11    }
12    di `i'
13 }
14 di `sum'
    
```

forvalues 循环

也可以使用 forvalues 循环改写为：

```
1 local sum=0
2 forvalues i = 10/20 {
3     local sum='sum'+ 'i'
4 }
5 di 'sum'
```

foreach in 循环

foreach 循环有以下几种用法：

- foreach i in LIST，其中 LIST 为任意一个 list，比如：

```

1  foreach a in 1 2 3 4 5 {
2     di 'a'^2
3 }
```

或者：

```

1  local strlist "a b c d e f"
2  foreach a in 'strlist' {
3     di "'a'"
4 }
```

foreach of

- `foreach i of varlist LIST`，其中 LIST 为变量列表，支持通配符，比如：

```

1  foreach v of varlist *{
2     cap gen log_`v`=log(`v')
3  }
```

即为所有变量取对数

- `foreach i of local LIST`，其中 LIST 为一个 local，比如：

```

1  use cfps_adult.dta
2  drop if te4 < 0
3  levelsof te4, local(edu)
4  foreach v of local edu{
5     gen edu`v`=te4==`v'
6  }
```

其中 levelsof 用于取出 te4 的所有可能取值，放到 edu 这个 local 中

Stata 的 Python 接口

- Python 是一门通用的脚本语言，随着大数据的兴起，Python 中发展出了大量数据科学有关的包
 - 数值计算：NumPy, Scipy, Pandas
 - 深度学习：PyTorch, Tensorflow
 - 文本学习：NLTK 等
- Stata 作为传统统计，在以上领域比较弱势
- 从 Stata16 版本中，加入了对 Python 的支持，可以结合 Stata 和 Python 一起编程解决问题，优势互补。

Python 的配置

在 Stata 中使用 Python 需要首先配置 Python 解释器的位置。我们可以首先使用

```
1 python query
```

命令查看当前 Python 的配置情况。通常在 Linux 中不需要做额外的配置，如果在 Windows 或者 Mac 中，可能需要额外配置环境变量，Stata 才能做默认的 Python 配置。

或者，可以直接在 Stata 中指定所使用的 Python 解释器地址，比如在我的计算机中安装有 Intel 的 Python 解释器，可以通过如下方式设定：

```
1 set python_exec /opt/intel/intelpython3/bin/python3 , perm
```

其中 perm 是 permanently 的缩写，代表永久性地设置该路径为默认路径。

Python 交互式窗口

在 Stata 中的命令窗口中，输入“python”或者“python:”就可以进入与 Python 一样的交互式窗口，两者区别：

- python 不带冒号：遇到错误不会退出 Python 环境
- python 带冒号：遇到错误就退出 Python 环境

输入 python 后，可以看到结果窗口中有“>>>”的提示符，如果需要退出 Python 环境，输入 end 命令即可。

或者，可以直接使用 python 前缀执行单行的 python 命令，比如：

```
1 python: import numpy
2 python: print(numpy.random.random())
```

反之，在 Python 环境中也可以使用 stata 前缀执行 stata 命令。

在 do-file 中嵌入 Python

我们也可以将 Python 程序嵌入到 do 文件中：

- 以 python 开头以 end 结尾的代码块都属于 Python 代码，错误不退出
- 以 python: 开头以 end 结尾的代码块都属于 Python 代码，错误退出

比如：

```
1 local a=4
2 local b=5
3 python
4 def add(a,b):
5     print(a+b)
6 end
7 python: add('a','b')
```

执行 Python 脚本文件

或者，可以将 Python 写成一个单独的文件，再使用 `python script` 命令执行该脚本。

- 使用 `python script` 命令时，可以使用 `args()` 命令向脚本文件传递参数（用空格隔开）
- 在 Python 脚本中，可以使用 `sys.argv` 接受所传递的参数

比如：

```
1 # filename: print_add.py
2 import sys
3 a,b=(sys.argv[1],sys.argv[2])
4 print(int(a)+int(b))
```

```
1 python script print_add.py, args(2 3)
```

使用 sfi 与 Stata 交互

以上介绍了 Stata 中执行 Python 的方式，然而，很多时候我们需要 Python 和 Stata 之间紧密交互，比如共享数据等。为此，Stata 提供了 sfi 模块供 Python 程序中访问 Stata 的数据。该接口提供了如下模块：

- 数据：Data
- 日期时间：Datetime
- 数据框：Frame
- 矩阵：Matrix
- 宏：Macro
- 缺失值：Missing
- 标量：Scalar
- 标签：ValueLabel
-

等的处理。网址：<https://www.stata.com/python/api16/> 中可以查询到该接口的详细文档。

从 Stata 中读写宏

想要从 Stata 中获取数据，使用每个模块中的 `get*` 函数即可，比如 Macro 模块中的 `getLocal` 可以获得 Stata 中的 local，而 `setLocal` 函数可以将内容写入到 Stata 中的 local 中：

```

1 local a=4
2 local b=5
3 python
4 from sfi import Macro
5 a=int(Macro.getLocal("a"))
6 b=int(Macro.getLocal("b"))
7 Macro.setLocal("c",str(a**b))
8 end
9 di `c'
    
```

以上使用 Macro 模块的 `getLocal` 函数获取 a 和 b 两个 local 的值，注意获得的值为字符串，因而需要使用 `int` 函数转化为数字；最后用 `setLocal` 命令设定 local c 为 a^b ，注意设定的 local 必须为字符串，因而需要用 `str` 函数转化为字符串。

从 Stata 中读写数据

类似地，可以使用 Data 模块中的 get 函数从 Stata 中获取数据，在此过程中需要注意处理缺失值：

```

1 use datasets/cfps_family_econ.dta
2 python
3 from sfi import Data, Macro
4 from sfi import Missing
5 data=Data.get("fincome1")
6 ## 去掉缺失值和 <=0 的值
7 sub_data=[d for d in data \
8     if not Missing.isMissing(d) and d>0]
9
10 inverse_data=[1/d for d in sub_data]
11 harmonic_mean=len(sub_data)/sum(inverse_data)
12 Macro.setLocal("harmonic_mean", str(harmonic_mean))
13 end
14 di 'harmonic_mean'
```


向 Stata 中写入数据

如果需要写入 Stata 的数据集，可以使用 Data 模块中的 add* 命令为 Stata 数据集中增加观测或者重新的变量，使用 store 函数写入数据到 Stata:

```
1 use datasets/cfps_family_econ.dta
2 python
3 from sfi import Data
4 from sfi import Missing
5 import math
6 data=Data.get("fincome1")
7 log_data=[]
8 for d in data:
9     if Missing.isMissing(d) or d<=0:
10         log_data.append(Missing.getValue())
11     else:
12         log_data.append(math.log(d))
13
14 Data.addVarDouble("log_income")
15 Data.store("log_income",None,log_data)
16 end
```