Stata Tutorial

相对来说比较清晰易懂且基本无害的 Stata 入门教程

作者：左祥太

Author: Shutter Zor / Xiangtai Zuo

单位：厦门大学管理学院

Affiliation: School of Management, Xiamen University

日期：2023 年 8 月 8 日

版本：1.0

邮箱：Shutter_Z@outlook.com

靡不有初，鲜克有终。

目录

序言

作者简介：左祥太（1999-），男，厦门大学管理学院会计学系博士研究生。开发 Stata 命令两项：`oneclick` 与 `onetext`，可通过 `ssc install oneclick` 与 `ssc install onetext` 进行下载。获软件著作权一项。论文发表于 *Journal of Cleaner Production*、*Technology Analysis & Strategic Management*、科研管理等。

关于本书：本书旨在为 Stata 初学者提供一套较为系统的入门方法。不同于其他主流的 Stata 教程，本书绝对不介绍任何计量内容（因为我也不会，而且打公式好麻烦，即便本书是用 LaTeX 编写），而是专注于实用操作。同时也感谢 ElegantBook 所提供的书籍模板，Overleaf 的在线编辑功能，还有各位粉丝朋友们。

其他内容：可以通过扫描如下的二维码关注我的微信公众号：OneStata。我的公众号主要用于存放一些发布在 Bilibili 上的视频的代码，以及一些其他优质实用的推文。在可预见的未来里，本公众号都将以非盈利的方式运行，坚持内容开源，并接受用户打赏。

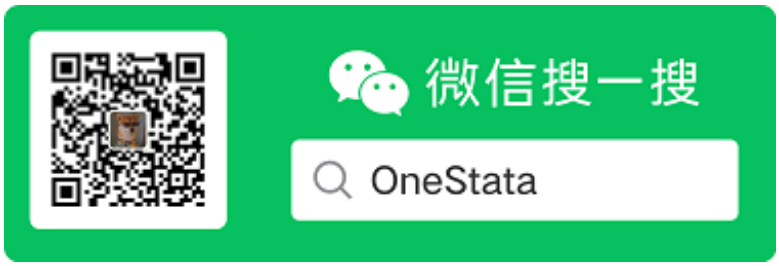

图 1: 我的微信公众号

可以在 Bilibili 搜索：拿铁一定要加冰。我的 Bilibili 视频号主要用于分享一些与 Stata、Python 相关的实用操作。

图 2: 我的 Bilibili

本教程的相关内容收集自本人的 Stata Tutorial 在线网站、Stata 的官方帮助文档等，部分文本内容由 ChatGPT 生成。本教程原始文件及其配套数据（Appendix Data）见我的 GitHub 仓库：StataTutorialAppendixData。**注意**，本教程有些内容来自公开网页，可能存在错误标记来源以及误使用的情况。如读者朋友发现了该问题，烦请通过邮箱与我进行核对校正。

左祥太
Shutter_Z@outlook.com
厦门大学管理学院会计学系
癸卯年六月廿五于湖北潜江

第 1 章 Stata 初印象

Stata（读作 “stay-ta”，通常也可以写作 STATA[7, 8]）是 StataCorp 开发的一款通用统计软件包，用于数据处理、可视化、统计和自动报告。包括生物医学、经济学、流行病学和社会学等许多领域在内的研究人员都在使用它 [1, 4–6, 10]。

Stata 最初由加利福尼亚州的计算资源中心（Computing Resource Center）开发，并于 1985 年发布了第一个版本 [2]。1993 年，该公司迁至德克萨斯州，并更名为 Stata Corporation，即现在的 StataCorp[7]。该司 2003 年发布了一个包括新的图形系统和所有命令的对话框的重要版本 [2]。此后，每两年发布一次新版本 [3]。当前版本为 Stata 18，并于 2023 年 4 月发布 [9]。

1.1 技术概览与相关术语

1.1.1 用户界面

自创建以来，Stata 一直采用集成的命令行界面。从 8.0 版开始，Stata 加入了基于 Qt 框架的图形用户界面，使用菜单和对话框访问许多内置命令。数据集可以电子表格格式查看或编辑。从第 11 版开始，在打开数据浏览器或编辑器的同时，还可以执行其他命令。

1.1.2 数据结构和存储

在第 16 版发布之前，Stata 在同一时间只能打开一个数据集。Stata 允许灵活地为数据指定数据类型。它的压缩命令会自动将数据重新分配为占用内存较少而又不丢失信息的数据类型。Stata 使用的整数存储类型只占用 1 或 2 个字节而不是 4 个字节，浮点数默认使用单精度（4 字节）而不是双精度（8 字节）。

Stata 的数据格式始终是表格格式。Stata 将表格数据的列称为变量。

1.1.3 数据格式的兼容性

Stata 可以导入多种格式的数据。包括 ASCII 数据格式（如 CSV 或数据库格式）和电子表格格式（包括各种 Excel 格式）。

随着时间的推移，Stata 的专有文件格式也在不断变化，但并非每个 Stata 版本都包含新的数据集格式。每个版本的 Stata 都能读取所有旧版本的数据集格式，并能使用 saveold 命令写入当前和最新的旧版本数据集格式。因此，当前的 Stata 版本总是能打开用旧版本创建的数据集，但旧版本无法读取较新格式的数据集。

Stata 可以使用 fdause 和 fdasave 命令读写 SAS XPORT 格式的数据集。

包括 gretl 在内的一些其他计量经济学应用程序可以直接导入 Stata 文件格式。

1.2 发展历史

1.2.1 起源

Stata 的开发始于 1984 年，最初由 William (Bill) Gould 负责，后来由 Sean Becketti 负责。该软件最初的目的是与 SYSTAT 和 MicroTSP 等统计程序竞争 [2]。与现在一样，Stata 最初是用 C 语言编写的，适用于运行 DOS 操作系统的个人电脑。第一个版本于 1985 年发布，共有 44 条命令。

1.2.2 发展

从 1985 年到 2023 年，Stata 共发布了 18 个主要版本，并在各主要版本之间进行了代码和文档更新。早年，Stata 的额外程序集有时作为“工具包”出售，有时以磁盘形式分发。随着 1999 年 Stata 6 的发布，Stata 开始通过网络向用户提供更新。从那时起，Stata 又发布了运行 Unix 变种（如 Linux 发行版、Windows 和 MacOS）的版本。

在 Stata 接近 40 年的发展历程中，社区用户及 Stata 官方为其逐渐添入了数百条命令。事实证明，Stata 的某些发展尤为重要，并一直影响着今天的用户体验，其中包括可扩展性、平台独立性和活跃的用户社区。

Stata 的更新时间线及具体更新内容见附录 A 。

1.2.3 扩展性

程序命令在 Stata 1.2 中得到了应用，用户可以在此基础之上添加自己的独特命令。ado-files 在 Stata 2.1 中得到了跟进，允许将用户编写的程序自动加载到内存中。用户编写的 ado-files 可以提交到由波士顿学院托管的统计软件组件档案（Statistical Software Components Archive，SSC）中。同时，StataCorp 添加了一个 ssc 命令，允许用户在 Stata 中直接加载由社区用户推送的第三方程序。Stata 的最新版本允许用户使用命令调用 Python 和 R 脚本，并允许从一些 Python IDE，如 Jupyter Notebooks 等直接导入并运行 Stata 命令。

1.2.4 用户社区

许多重要的发展都是由 Stata 活跃的用户社区发起的。*Stata Technical Bulletin* 从 1991 年开始发行，通常包含用户创建的命令，每年发行六期。2001 年，它作为同行评审的 *Stata Journal* 重新推出，这是一份季刊，其中包含社区贡献的命令说明和有效使用 Stata 的技巧。1994 年，开始建立列表服务器，作为用户合作解决编码和技术问题的枢纽；2014 年，列表服务器转变为网络论坛。1995 年，StataCorp 开始组织每年一次的用户和开发人员会议。只有每年在美国举行的 Stata 大会由 StataCorp 主办。其他地区用户的交流论坛由当地的 Stata 经销商在当地举办。

1.3 软件产品

Stata 有四个版本：Stata/MP、Stata/SE、Stata/BE 和 Numerics by Stata。Stata/MP 允许对某些命令进行内置并行处理，而 Stata/SE 和 Stata/BE 则存在瓶颈，只能使用一个内核。与 SE 或 BE 版本相比，Stata/MP 在四个 CPU 内核上运行并行处理时，某些命令的运行速度要快 2.4 倍，约为理论最高效率的 60%。

SE 和 BE 版本在数据集可使用的内存量上有所不同。Stata/MP 可以存储 100 到 200 亿个观测值和多达 12 万个变量，而 Stata/SE 和 Stata/BE 则分别可以存储 21.4 亿个观测值，处理 32767 个变量和 2048 个变量。在 Stata/MP 中，一个模型中自变量的最大数量为 65,532 个，在 Stata/SE 中为 10,998 个，在 Stata/BE 中为 798 个。

Stata 的定价和许可取决于其预期用途：商业、政府/非营利、教育或学生。单用户许可证可每年续订或永久续订。其他许可证类型包括供并发用户使用的单个许可证、站点许可证、批量定价的批量单个用户许可证或学生实验室许可证。

第 2 章 Stata 入门

2.1 设定软件外观

2.1.1 为什么要更改外观

设定软件的外观并非必要内容，但好看的外观会减少人在写代码时候的视觉疲惫，总的来说更换软件外观具有如下优点。

1. **提高可视性和易读性：**合理的配色方案可以确保文本和界面元素的对比度，使其更清晰易读。字体与背景之间的明显对比度可以减少阅读时的眼睛疲劳，同时也使用户更容易从界面中获取信息。
2. **改善注意力集中：**通过巧妙选择配色方案，你可以引导用户的视线到特定区域，帮助他们更快地定位和操作目标。这有助于减少界面上的混乱，降低用户迷失的可能性。
3. **情感和氛围影响：**颜色在心理上有很大的影响力，可以触发不同的情绪和感觉。例如，蓝色可能有镇定作用，红色可能唤起活力。通过选择适合任务和使用场景的颜色，可以营造出更适宜的工作氛围。
4. **避免视觉干扰：**白色背景在高亮环境下可能会造成眩光或视觉干扰，尤其是在长时间使用时。更柔和的颜色，如浅灰或淡蓝，可以减轻这种不适，提高界面的视觉舒适度。
5. **降低眼睛疲劳：**一些颜色，特别是暗色调，如深蓝、棕色或深绿，可能对眼睛的疲劳产生较小影响。使用这些颜色作为背景或元素的基调可以使用户在长时间使用软件时感到更舒适。
6. **个性化和创新：**改变配色方案可以为软件增添独特的外观与感觉。你可以根据品牌特点、用户喜好或界面风格，选用一套独特的颜色，从而为软件增加个性化元素和创新性设计。
7. **提升用户体验：**总体来说，更改配色方案的目的是为了提升用户体验。合适的配色可以让用户感到更舒适、愉悦，从而提高工作效率、满意度和忠诚度。

2.1.2 如何更改 Stata 的外观

具体而言，在 Stata 中可以通过如下方式设置软件的外观。依次点击编辑–> 首选项–> 常规首选项，可以设置整体外观与单独设置每一种外观。在这里我推荐将“整体颜色方案”设置为深色。

2.2 Stata 操作方法

总的来说，Stata 主要有以下三种执行命令的方式。

1. **命令窗口执行：**在 Stata 的命令窗口中直接输入命令，然后按下回车键执行。这是最常见和基本的执行命令方式。
2. **Do 文件执行：**创建一个扩展名为 `.do` 的文本文件，其中包含一系列 Stata 命令。你可以选中该文件中的所有内容后，按“Ctrl+D”一次性运行选中的所有命令。`.ado` 是一种特殊的 `.do` 文件，对于不需要自己编写函数的用户来说，不需要掌握。
3. **图形用户界面（GUI）：**Stata 也提供了一个图形用户界面，通过菜单和按钮可以执行各种操作，包括运行命令、数据管理、图表绘制等。这对于不熟悉命令行的用户来说是一个很方便的方式。但是，这种方法的缺点在于，如果记性不好忘记了操作步骤，则很难重复。

2.3 系统命令与帮助文件

2.3.1 几个系统命令

在 Stata 中，“system 命令”是一类特殊的命令，它们允许用户在 Stata 环境中执行操作系统级别的命令，与操作系统进行交互，获取系统信息，进行文件操作等。你可以尝试在 Stata 的命令窗口中依次输入以下命令，并探索其作用。

1. `pwd`，用于显示当前工作路径，与 Stata 窗口左下角的路径一致。
2. `cd`，用于改变路径，可以进入不同的文件夹。
3. `sysdir`，查看所有与 Stata 相关的路径，比如一些函数的存储路径。
4. `ls`，展示当前路径下的所有文件。
5. `dir`，效果与 `ls` 相似。

在展示命令之前，需要说明的是，我的所有文件保存在 F 盘下的一个名为“stata-tutorial”的文件夹中。在这一章，主要使用的文件是这个文件夹（stata-tutorial）下一个名为“P1_Introduction”的文件夹，即我当前的工作路径是 `F:\stata-tutorial\P1_Introduction` 。

```
. pwd                       // 显示当前工作路径
F:\stata-tutorial\P1_Introduction

. cd ..                     // ..表示退回到上级目录，是一种相对路径，这里也可以使用绝对路径
F:\stata-tutorial

. cd P1_Introduction        // 再次返回到最初的工作路径
F:\stata-tutorial\P1_Introduction

. sysdir                    // 展示用于存储Stata相关安装包的路径，包括函数等
   STATA: E:\Stata16\
    BASE: E:\Stata16\ado\base\
    SITE: E:\Stata16\ado\site\
    PLUS: C:\Users\13398\ado\plus\
PERSONAL: C:\Users\13398\ado\personal\
OLDPLACE: c:\ado\

. ls                        // 查看当前文件夹下的所有文件与详细信息
  <dir>  4/24/23 10:44 .
  <dir>  4/24/23 10:44 ..
   0.5k  8/11/22 16:32 case1.do
   2.0k  8/11/22 16:32 case2.do
  11.3k  8/18/22 10:34 P1_Introduction.do
 237.3k  8/06/22 18:48 Stata_Intro.pdf
  28.1k  8/06/22 18:48 usbirth2007.txt
  28.0k  8/06/22 18:48 usbirth2008.txt
  28.1k  8/06/22 18:48 usbirth2009.txt

. dir                        // 查看当前文件夹下的所有文件与详细信息，与ls等价
```

2.3.2 使用帮助文件

开发者为 Stata 的每个命令几乎都适配了帮助文档。这些帮助文档在安装 Stata 软件，或者使用 `ssc install xxx` 的时候就同步完成了下载，使用 `help` 命令即可查看。比如，在命令窗口输入 `help regress` 即可查看 `regress` 命令的帮助文档。

对于 `regress` 命令，可以缩写为 `reg`，由于不存在缩写冲突，所以 `help reg` 与 `help regress` 是等价的。同样地，请在 Stata 的命令窗口输入以下命令，观察运行结果。

```
. help reg
. help regress
. help sum
. help summarize
```

2.4 局部宏与全局宏

在 Stata 中，“宏”（Macro）是一种用于存储和管理文本值、数值或命令的工具。宏允许你在 Stata 中创建变量来保存特定的值或命令，然后在需要的地方引用这些值或命令。宏可以极大地简化和提高代码的可维护性，尤其是当你需要在多个地方使用相同的值或命令时。在命令窗口输入 `help macro` 可以看到 Stata 支持的所有宏。一般来说 `local` 与 `global` 是我们最常使用的宏。前者使用一次便消失，引用方式为 `` `localVar' ``[1]，需要与引用语句同时选中同时运行；后者直至 Stata 关闭都一直存在，引用方式为 `$globalVar`，不需要与引用语句同时选中同时运行。

特别地，类比于 Python，Stata 的 Macro 可以理解为 Python 运行过程中的 Variable，它们都能用于存储一些信息，并进行运算。

2.4.1 局部宏 local 的基本功能

注意，以下内容需要首先写入 `.do` 文件，并在 `.do` 中选中后一起运行，不然无法正确展示结果。在 Stata 中创建 `.do` 文件的方法有两种。

- 打开 Stata 后，依次点击窗口–>do 文件编辑器–> 新 do 文件编辑器。
- 打开 Stata 后，直接按住键盘上的 Ctrl+9。

接下来的操作都将在 `.do` 文件中进行，运行代码的快捷方式为选中代码后按 Ctrl+D。

2.4.1.1 存储数值与数值运算

```
*- do 文件的内容
local num1 5
local num2 7

local plus = `num1' + `num2'
local minus = `num1' - `num2'
local times = `num1' * `num2'
local divide = `num1' / `num2'

display "Sum: `plus'"
display "Difference: `Minus'"
display "Product: `times'"
```

[1]需要注意的是，`` `localVar' ``引用方式中左边的并不是单引号，而是英文输入法下键盘 Esc 下面的符号，这是一个非常容易出错的地方。

```
display "Quotient: `divide'"

*- 将上述do文件的内容复制粘贴至你的do文件，选中后按Ctrl+D，会在Stata的结果窗口显示如下内容
. display "Sum: `plus'"
Sum: 12

. display "Difference: `Minus'"
Difference:

. display "Product: `times'"
Product: 35

. display "Quotient: `divide'"
Quotient: .7142857142857143
```

需要注意的是，在引用内容仅有 Macro 的时候，是否使用双引号会影响 Marco 的显示结果，比如：

```
*- do 文件的内容
local num3 2+2
dis `num3'
dis "`num3'"

*- 将上述do文件的内容复制粘贴至你的do文件，选中后按Ctrl+D，会在Stata的结果窗口显示如下内容
. dis `num3'
4

. dis "`num3'"
2+2
```

2.4.1.2 存储字符串与字符串运算

```
*- do 文件的内容
local str1 Stata Tutorial
dis "`str1'"

local str2 By Shutter Zor
dis "`str2'"

local str3 `str1'`str2'
dis "`str3'"

*- 将上述do文件的内容复制粘贴至你的do文件，选中后按Ctrl+D，会在Stata的结果窗口显示如下内容
. dis "`str1'"
Stata Tutorial

. dis "`str2'"
By Shutter Zor

. dis "`str3'"
```

```
Stata TutorialBy Shutter Zor
```

2.4.1.3 存储变量与变量操作

```
*- do 文件的内容
sysuse auto.dta, clear

local myVar price mpg rep78
sum `myVar'

// 等价于:
sum price mpg rep78
```

将上述 do 文件的内容复制粘贴至你的 do 文件，选中后按 Ctrl+D，会在 Stata 的结果窗口显示如下内容：

```
*************************************************************************************************
. sum `myVar'

    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
       price |         74    6165.257    2949.496       3291      15906
         mpg |         74     21.2973    5.785503         12         41
       rep78 |         69    3.405797    .9899323          1          5

. sum price mpg rep78

    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
       price |         74    6165.257    2949.496       3291      15906
         mpg |         74     21.2973    5.785503         12         41
       rep78 |         69    3.405797    .9899323          1          5
*************************************************************************************************
```

在这个例子当中，我调用了 Stata 自带的 `auto.dta` 数据集[2]，并对其中的三个变量 `price`、`mpg`、`rep78` 进行了描述性统计。

注意，这里加不加双引号也会影响展示结果，比如：

```
*- do 文件的内容
local myVar price mpg rep78
dis `myVar'
dis "`myVar'"

*- 将上述do文件的内容复制粘贴至你的do文件，选中后按Ctrl+D，会在Stata的结果窗口显示如下内容
. dis `myVar'
4099223

```

[2]在 Stata 中，输入 `help dta_examples` 即可查看 Stata 自带的所有数据集的详细信息，通过 `sysuse fileName.dta` 的方式即可调用。**注意**，`clear` 选项的含义是清除当前的数据集，并调用新的数据集。

```
. dis "`myVar'"
price mpg rep78
```

在不加入双引号的时候，展示的是变量 price、mpg、rep78 中的第一个观测值的值拼接起来的内容；在加入双引号的时候，展示的是 `myVar' 的文本内容。

2.4.2 局部宏的进阶功能

在 2.4.1.1 小节中，我们提到了 Stata 的局部 Macro 可以进行一些诸如加减乘除的基本运算。同时，它还支持一些四则运算以外的更高级的运算方法。在 Stata 的命令窗口中输入 help mathematical functions，可以查看在 Macro 中支持的所有数学运算。当然，这些运算也能在除 Macro 外的命令语句中使用。

```
*- do 文件的内容
sysuse auto.dta, clear

local num ceil(sqrt(log(100)))

dis `num'
dis make[3]
dis make[`num']
dis make[`=ceil(sqrt(log(100)))']

*- 将上述do文件的内容复制粘贴至你的do文件，选中后按Ctrl+D，会在Stata的结果窗口显示如下内容
. dis `num'
3

. dis make[3]
AMC Spirit

. dis make[`num']
AMC Spirit

. dis make[`=ceil(sqrt(log(100)))']
AMC Spirit
```

其中，对于 make[x]，可以通过给定 x 的值，来展示当前排序下，变量 make 的第 x 个样本中所存放的值。很显然，make[3] 即表示变量 make 的第 3 个观测样本的值，为 “AMC Spirit”。而在以数学运算 ceil(sqrt(log(100)))[3] 定义局部 Macro `num' 后，直接将其写成 make[`num'] 与 make[`=ceil(sqrt(log(100)))'] 是等价的。**注意**，这里的等号是必须要加的。

2.4.2.1 进阶实例 1-横向求和

```
*- do 文件的内容
// 生成数据
clear
set obs 10    // 设定观测值个数
set seed 12345 // 设定随机数种子，方便复现随机生成结果

```

[3] ceil() 表示向上取整；sqrt() 表示开方；log() 表示以自然数 e 为底的对数，与 ln() 等价。

```
gen var1 = int(10*runiform())
gen var2 = int(10*runiform())
gen var3 = int(10*runiform())
gen var4 = 0

// 利用循环与Macro横向求和
local i = 1
while `i' <= 3 {
  replace var4 = var4 + var`i'
  local i = `i' + 1
}

// 等价于
egen var5 = rowtotal(var1 - var3)

// 展示前10个结果
list var4 var5 in 1/10
```

将上述 do 文件的内容复制粘贴至你的 do 文件，选中后按 Ctrl+D，会在 Stata 的结果窗口显示如下内容：

```
**********************************************************************************************
     +-------------+
     | var4   var5 |
     |-------------|
  1. |    3      3 |
  2. |   13     13 |
  3. |   16     16 |
  4. |   20     20 |
  5. |   17     17 |
     |-------------|
  6. |   14     14 |
  7. |    7      7 |
  8. |    6      6 |
  9. |    9      9 |
 10. |   13     13 |
     +-------------+
**********************************************************************************************
```

2.4.2.2 进阶实例 2-横向求积

这部分内容来自我 Bilibili 视频号：拿铁一定要加冰 的一个视频 【Stata】横向连乘与纵向连乘，暂未了解到有现成的函数可以替代，所以以循环 +Macro 的方式实现横向求积，具体代码如下。

```
*- do 文件的内容
// 生成数据
clear
set obs 10    // 设定观测值个数
set seed 12345 // 设定随机数种子，方便复现随机生成结果

```

```
gen var1 = int(10*runiform())
gen var2 = int(10*runiform())
gen var3 = int(10*runiform())
gen var4 = 1

// 利用循环与Macro横向求积
local i = 1
while `i' <= 3 {
  replace var4 = var4 * var`i'
  local i = `i' + 1
}

// 展示前5个结果
list var1 var2 var3 var4 in 1/5
```

将上述 do 文件的内容复制粘贴至你的 do 文件，选中后按 Ctrl+D，会在 Stata 的结果窗口显示如下内容:

```
*************************************************************************************************
     +---------------------------+
     | var1   var2   var3   var4 |
     |---------------------------|
  1. |    3      0      0      0 |
  2. |    4      0      9      0 |
  3. |    6      4      6    144 |
  4. |    5      6      9    270 |
  5. |    5      8      4    160 |
     +---------------------------+
*************************************************************************************************
```

2.4.3 全局宏 global 的基本功能与进阶功能

Stata 的全局 Macro 与局部 Macro 的使用方法类似，仅需指定 `global` 即可定义全局 Macro，使用起来的引用方式为美元符号 *$* 加 Macro 的名称，比如 *$*`globalVar`。具体使用方法参考 `local`。

2.4.4 宏的管理

该部分内容作用不是很大，了解即可。在 Stata 中，对宏 Macro 进行管理的命令主要有:

1. **macro list**，展示所有的宏。
2. **macro drop globalVar1**，删除名为 `globalVar1` 的全局宏。

2.5 循环语句与条件语句

与大多数编程语言一致，Stata 也具备例如 `while`、`if`、`else` 等在内的循环语句与条件语句，合理地使用这些循环语句能够在一定程度上减少我们的工作量。比如，对于同一个文件夹下的多份类似的数据文件，我们可以使用循环语句将其依次读入 Stata 并完成合并，相比于一份份地读入与合并，使用循环能少写很多的代码。再比如，回到一个大家最关心的问题——变量的显著性上。一键显著的原理就是排列组合控制变量，再将不同的控制变量组合以循环的方式，依次带入回归，最后挑选出来满足显著性要求的组合。

然而，在实际使用过程当中，如果用户不喜欢做一些编程操作，则一般很少会使用 `while` 与 `else`。一般而言，使用频率最高的条件语句是 `if`，使用频率最高的循环语句一般是 `forvalues` 与 `foreach`。

2.5.1 条件语句-if

2.5.1.1 内置 if

在 Stata 中，很多命令都会内置 `if` 选项。举个例子：通过前面的代码，你已经学会了调用 Stata 自带的 `auto.dta` 数据集，并且会做一些描述性统计。那么如果说我现在只需要对进口车辆（`foreign=1`）进行描述性统计应该如何使用 `if` 呢？答案如下。

```
*- do 文件的内容
sysuse auto.dta, clear

sum price if foreign == 1
```

将上述 do 文件的内容复制粘贴至你的 do 文件，选中后按 Ctrl+D，会在 Stata 的结果窗口显示如下内容：

```
********************************************************************************************************
. sum price if foreign == 1

    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
       price |         22    6384.682    2621.915       3748      12990
********************************************************************************************************
```

只需要在变量 `price` 后面加上 `if foreign == 1` 即可实现条件筛选。**注意**，并非所有的命令都支持 `if` 选项，可以通过 `help command` 查看该命令是否支持 `if` 选项。

2.5.1.2 外置 if

（1）外置 if 的一个实例

同样地，也可以使用外置 `if` 来进行条件判断。考虑一个经典的判断，当成绩在 30 分以下时，我们将其定义为 C；在 30 分到 60 分时，我们将其定义为 B；在 60 分以上时，我们将其定义为 A。那么，应该如何写出这样的判断语句呢？答案如下。

```
*- do 文件的内容
// 定义成绩
local score 90

// 条件判断
if (`score'<0)|(`score'>100){
  dis "您输入了错误的成绩，请重新输入"
  exit
}
  if (`score'<=30){
    dis "您的成绩为C"
  }
  if (`score'>30)&(`score'<=60){
    dis "您的成绩为B"
  }
  else {
    dis "您的成绩为A"
  }
```

```
*- 将上述do文件的内容复制粘贴至你的do文件，选中后按Ctrl+D，会在Stata的结果窗口显示如下内容
您的成绩为A
```

（2）外置 if 的一个练习

现在你已经了解了 if 与 else 联用的有趣方法，那么我来考考你，你应该如何与 Stata 进行一场精彩刺激的猜拳游戏呢？该部分内容源自我公众号：OneStata 的一篇推文 Stata-惊险刺激的石头剪刀布游戏。

实现思路如下：

1. 为 Stata 指定一个 0 到 1 的随机数。
2. 将生成的随机数三等分，分别定义为：剪刀、石头和布。
3. 将你出的东西写成条件语句与 Stata 的结果进行逻辑判断。
4. 得出输赢平的结论，并打印在频幕上。

实现代码如下：

```
*- do 文件的内容
local Player "布"
local Random = 10*runiform()

if "`Player'" != "石头" & "`Player'" != "剪刀" & "`Player'" != "布" {
  dis as error "只能输入石头、剪刀和布"
  exit
}

if (`Random'<=3) {
  local Stata "石头"
  if "`Player'" == "石头" {
    dis "Stata本轮出" "`Stata'" "你出的是" "`Player'"
    dis "本次是平局"
  }
  if "`Player'" == "剪刀" {
    dis "Stata本轮出" "`Stata'" "你出的是" "`Player'"
    dis "很遗憾你输了-_-"
  }
  if "`Player'" == "布" {
    dis "Stata本轮出" "`Stata'" "你出的是" "`Player'"
    dis "恭喜你胜利^_^"
  }
}
if (`Random'>3) & (`Random'<=6) {
  local Stata "剪刀"
  if "`Player'" == "石头" {
    dis "Stata本轮出" "`Stata'" "你出的是" "`Player'"
    dis "恭喜你胜利^_^"
  }
  if "`Player'" == "剪刀" {
    dis "Stata本轮出" "`Stata'" "你出的是" "`Player'"
    dis "本次是平局"
  }
  if "`Player'" == "布" {
```

```
    dis "Stata本轮出" "`Stata'" "你出的是" "`Player'"
    dis "很遗憾你输了-_-"
  }
}
if (`Random'>6) {
  local Stata "布"
  if "`Player'" == "石头" {
    dis "Stata本轮出" "`Stata'" "你出的是" "`Player'"
    dis "很遗憾你输了-_-"
  }
  else if "`Player'" == "剪刀" {
    dis "Stata本轮出" "`Stata'" "你出的是" "`Player'"
    dis "恭喜你胜利^_^"
  }
  else if "`Player'" == "布" {
    dis "Stata本轮出" "`Stata'" "你出的是" "`Player'"
    dis "本次是平局"
  }
}

*- 将上述do文件的内容复制粘贴至你的do文件，选中后按Ctrl+D，会在Stata的结果窗口显示如下内容
Stata本轮出石头你出的是布
恭喜你胜利^_^
```

2.5.2 循环语句初体验

2.5.2.1 循环语句-forvalues

`forvalues` 是针对数值进行的循环，通常应用于一系列含有规整排列特点的变量/文件中，实现对变量或者文件的批量处理。比如，可以用以下方法生成从 `var1` ... `var100` 的 100 个满足均匀分布的变量。

```
*- do 文件的内容
clear
set obs 10    // 设定观测值个数
set seed 12345 // 设定随机数种子，方便复现随机生成结果

forvalues i = 1(1)100 {
   gen var`i' = runiform()
}
```

在运行完上述代码之后，在命令窗口输入 `browse`，可以查看生成的 100 个变量。

同样地，可以对有规律的数字构成文件名的文件进行批量处理。从我的 GitHub 仓库：StataTutorialAppendix-Data 找到文件夹 `AppendixData1`，这里面包含了如下代码所使用的三份 `.txt` 文件格式的原始数据，分别叫做 `usbirth2007.txt`、`usbirth2008.txt`、`usbirth2009.txt`。

通过如下循环，我们可以实现将这些有规律（2007、2008，2009）命名的文件批量导入 Stata，并生成 `.dta` 文件。

```
*- do 文件的内容
forvalues i = 2007/2009{
   insheet using usbirth`i'.txt, clear
```

```
    rename v1 region
    rename v2 county
    rename v3 year
    rename v4 nbirth
    rename v5 mage

    label var region "US census region"
    label var county "US countty"
    label var year   "data year"
    label var nbirth "number of births"
    label var mage   "average age of mother"
    save usbirth`i'.dta, replace
}
```

在上面的代码中，我们使用了 forvalues 循环来处理从 2007 年到 2009 年的多个数据文件。下面逐行解释代码的作用：

1. forvalues i = 2007/2009：表示对后面两个正反大括号的包裹起来的部分进行循环操作，是 forvalues 循环的全部内容，循环变量 i 的值从 2007 开始逐步递增到 2009。
2. insheet using usbirth`i'.txt, clear：这行代码使用 insheet 命令依次导入数据文件 usbirth2007.txt、usbirth2008.txt 和 usbirth2009.txt 中的数据，清除当前数据集以便加载新数据。
3. rename v1 region：将变量 v1 重命名为 region，其余同理。
4. label var region "US census region"：这行代码为变量 region 添加了一个标签，标注了其含义，其余同理。
5. save usbirth`i'.dta, replace：这行代码使用 save 命令将处理过的数据保存为名为 usbirth2007.dta、usbirth2008.dta 和 usbirth2009.dta 的数据文件，如果文件已存在则替换。

整个代码块的作用是将多个数据文件导入 Stata，进行变量重命名和添加标签，然后将处理后的数据保存到新的数据文件中。这在数据预处理和整理中是一个常见的步骤，用于确保数据的一致性和易读性。

2.5.2.2 循环语句-foreach

foreach 循环一般用于在指定的内容中进行循环，比如，可以使用如下方法改写上述的循环。

```
*- do 文件的内容
local fileName usbirth2007 usbirth2008 usbirth2009
foreach v in `fileName' {
    insheet using `v'.txt, clear
   rename v1 region
   rename v2 county
   rename v3 year
   rename v4 nbirth
   rename v5 mage

   label var region "US census region"
   label var county "US countty"
   label var year "data year"
   label var nbirth "number of births"
   label var mage "average age of mother"
   save `v'.dta
}
```

最后也能得到一模一样的结果。

2.5.3 循环语句的实用操作

2.5.3.1 批量对数转换

通过如下的循环语句可以对变量进行批量对数转换处理：

```
*- do 文件的内容
sysuse auto.dta, clear

local varName price mpg weight length
foreach v in `varName' {
    gen ln`v' = ln(`v')
    gen log`v' = log(`v')
    label var ln`v' "ln(`v') for `v'"
   label var log`v' "log(`v') for `v'"
}

list lnprice logprice in 1/10
```

将上述 do 文件的内容复制粘贴至你的 do 文件，选中后按 Ctrl+D，会在 Stata 的结果窗口显示如下内容：

```
**************************************************************************************************
     +---------------------+
     |  lnprice   logprice |
     |---------------------|
  1. | 8.318499   8.318499 |
  2. |  8.46569    8.46569 |
  3. | 8.242494   8.242494 |
  4. | 8.479699   8.479699 |
  5. | 8.965335   8.965335 |
     |---------------------|
  6. | 8.663542   8.663542 |
  7. | 8.401333   8.401333 |
  8. | 8.554296   8.554296 |
  9. | 9.246865   9.246865 |
 10. | 8.314342   8.314342 |
     +---------------------+
**************************************************************************************************
```

这段代码的循环表示，依次选择变量 price、mpg、weight，以及 length 进行处理，在对每一个变量的处理过程中，生成对应的含有 ln 前缀或者 log 前缀的新变量，比如生成 lnprice 以及 logprice。

这里有一点需要注意的，也是我们之前提过的，在 Stata 中，log() 与 ln() 是完全等价的[4]。

2.5.3.2 批量缩尾处理

通过如下的循环语句可以对变量进行批量缩尾处理：

[4]如果想生成具有任意底数的对数，则需要实用换底公式。

```
*- do 文件的内容
sysuse auto.dta, clear

local varName price mpg weight length
foreach v in `varName' {
    winsor `v', gen(new_`v') p(0.1)
}

histogram price      // 绘制缩尾前的直方图
histogram new_price  // 绘制缩尾后的直方图
```

缩尾前变量 price 的分布如图 2.1 所示，缩尾后的新变量 new_price 的分布如图 2.2 所示

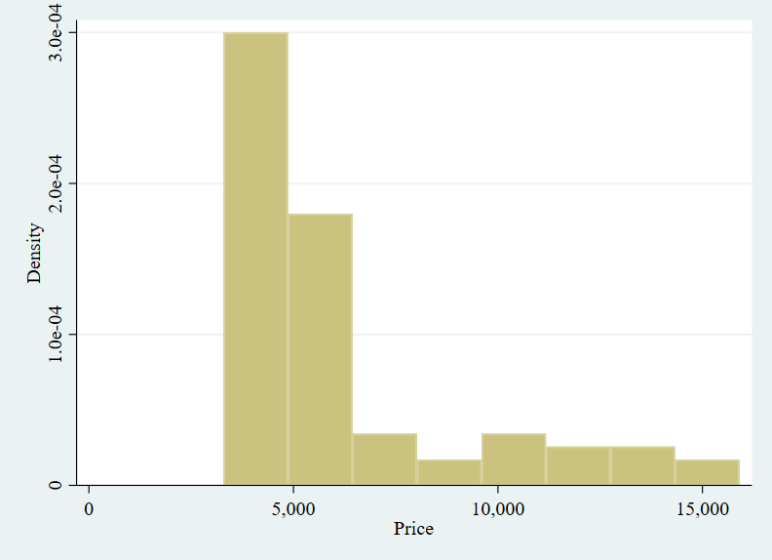

图 2.1: price 缩尾前

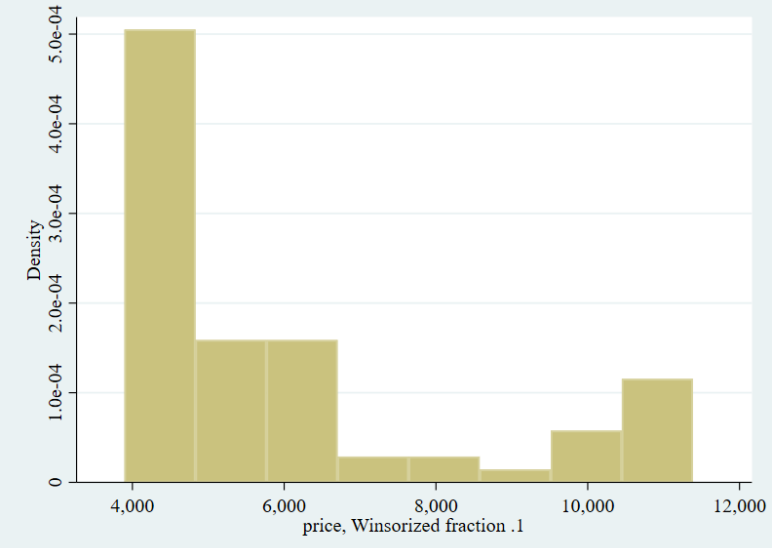

图 2.2: price 缩尾后

2.5.4 循环进阶

除了以上的内容外，Stata 还有一些其他实用的循环命令，比如使用 `levelsof` 对一个变量的所有不重复值进行循环；使用 `local` 配合 `dir` 命令对文件夹中的文件进行循环，类似于 Python 的 `os`。这些内容在我的视频以及公众号推文中都出现过，比如:

1. 利用 Stata 遍历文件夹与批量追加合并：「Stata」遍历文件夹与批量追加合并
2. 配合使用 `levelsof`、`local`，以及 `dir` 遍历并计算供应商、客户关系稳定度。
   (a). Bilibili 视频：「Stata」计算供应商、客户关系稳定度
   (b). 公众号推文：「Stata」计算供应商、客户关系稳定度
3. 使用 `local` 与 `dir` 实现在 Stata 利用词频统计方法计算企业的数字化转型程度。
   (a). Bilibili 视频：「Stata」利用词频统计方法衡量企业年报 MD&A 中的数字化转型
   (b). 公众号推文：「Stata」词频统计下的数字化转型
4. 使用 `local` 与 `dir` 在 Stata 中计算上市公司的 IPO 抑价率。
   (a). Bilibili 视频：「Stata」计算上市公司 IPO 抑价率
   (b). 公众号推文：「Stata」计算 IPO 抑价率

这些有意思的循环能极大地提升我们的效率，尤其是在处理一些有规律的文件的时候。当然，已知 Stata 的 `local` 与 `dir` 的组合使用可以作为 Python 中 `os` 模块的替代品，且 Stata 中的 `copy` 可以作为 Python 中类似于 `requests` 模块的替代品。那么，我们就可以使用 Stata 完成爬虫工作。但是，我并不推荐你使用 Stata 做爬虫，这是一件非常愚蠢的事情（仅代表个人观点），主要原因还是在于 Stata 的正则表达（Regular Expression）能力远不如 Python，且对于文本（网页文件实际是另类的文本文件）的处理能力也不及 Python 的百分之一。

关于在 Stata 中以及在 Python 中进行文本分析的差异化表现，可以参照我放在 arXiv 上的一篇工作论文: Comparing with Python: Text Analysis in Stata。总之，我非常不推荐使用 Stata 做爬虫工具，但是简单的文本分析也可以考虑使用 Stata 完成，比如词频统计类的文本分析工作。最后，关于词频统计，可以下载我的 `onetext` 包。

第 3 章 Stata 数据处理

3.1 导入与导出

3.1.1 导入数据

如果你已经有了一份名为 example_data.dta 文件，那你只需要使用 use example_data.dta, clear 即可导入该文件，输入 browse 命令即可查看文件内容。clear 表示清除内存中已有的数据，并用导入的数据覆盖。

然而，在使用 Stata 的过程中，除了部分调查问卷可能提供 .dta 格式的数据文件外，在大多数情况下，我们的原始数据都会存储在非 .dta 格式的文件当中。在这种情况下，就需要使用到 Stata 的数据导入功能。

一般来说，我们的数据大多以 Excel 的各种格式文件（.xls、.xlsx、.csv 等）为主。再导入这些文件的时候，一般使用 import 命令与 insheet 命令。前者用于导入 .xls、.xlsx，当然也可以导入 .csv 文件；后者则主要用于导入 .csv 文件。

假设你现在需要导入一份名为 example_data.xlsx 的文件，那你可以使用如下命令：

```
import excel using example_data.xlsx, first clear
```

其中，import excel using 用于指定需要导入的文件，选项的含义如下：

- first，指将原文件中的第一行作为变量名。
- clear，指清除内存中的文件，即用新导入的文件替代已导入的文件。如果是刚打开 Stata 第一次导入数据，则不需要这个选项。如果是已经导入了数据，则需要使用该选项覆盖已有的数据。

特别地，对于其他类型的数据，我也记不住命令。在这里我推荐使用 Stata 的图形界面，即依次点击：文件–>导入，选择合适的导入格式，再导入数据。

3.1.2 导出数据

在导入数据并且完成一些操作处理之后，我们需要将数据存储起来。在这种情况下，除非你的数据需要进一步交给其他无法导入 .dta 数据文件的小伙伴进行其他处理，否则一律保存为 .dta 文件。.dta 是 Stata 自有的数据文件格式，当其再次被导入 Stata 时，导入速度会相对快于其他格式的文件。同时，Python 中的 pandas 模块也可以直接导入 Stata 的 .dta 文件，使用 pandas.read_dta()即可。

导出数据的操作命令是 save data.dta, replace，其中，replace 选项的作用是，当工作路径中有同名文件时（已存在了 data.dta ），则使用这份文件（新的 data.dta ）替换原来的文件（旧的 data.dta ）。

3.2 变量生成

3.2.1 系统变量

通过 help _variables，可以看到 Stata 对系统变量（System variables）中的 _variables 的解释：以下划线 “_” 开头的变量被称之为 “下划线变量”，是由 Stata 创建和更新的内置系统变量。

Stata 的下划线变量主要有：

1. _b，用于在回归之后获取指定变量的回归系数值，也可以写作 _coef。
2. _se，用于在回归之后获取指定变量的标准误。
3. _cons，始终等于 1。当以 _b[_cons] 形式调用时，则表示返回回归中常数项的系数值。
4. _n，用于生成当前观测样本从 1 到 n 的顺序序列。
5. _N，用于生成当前观测样本的总数。
6. _pi，机器精度的 π 值。

7. _rc，用于在 capture 命令之后获取运行结果的返回码。

_n 和 _N 变量对生成观测值或生成数字序列非常有用。_n 可以作为分组运行的运行计数器，而 _N 可以计算组内的总数。_rc 对想要利用 Stata 编程的人很有用，特别是当你想测试内部命令的时候到一个程序，可以在忽略报错继续运行。

在其中运用比较多的是 _n 和 _N，所以我将重点介绍它们。

3.2.1.1 _n 与 _N 的区别

我们可以通过如下代码测试 _n 与 _N 的区别。

```
*- do 文件的内容
sysuse auto.dta, clear

gen var1 = _n
gen var2 = _N

list price var1 var2 in 1/5
```

将上述 do 文件的内容复制粘贴至你的 do 文件，选中后按 Ctrl+D，会在 Stata 的结果窗口显示如下内容：

```
. *- do 文件的内容
. sysuse auto.dta, clear
(1978 Automobile Data)

. gen var1 = _n

. gen var2 = _N

. list price var1 var2 in 1/5

     +---------------------+
     | price   var1   var2 |
     |---------------------|
  1. | 4,099      1     74 |
  2. | 4,749      2     74 |
  3. | 3,799      3     74 |
  4. | 4,816      4     74 |
  5. | 7,827      5     74 |
     +---------------------+
```

注意，前面在介绍 _n 的时候，我提到了一个重要的关键词，叫做**生成当前观测样本从 1 到 n 的顺序序列**，也就是说，当你改变了样本的排序方式之后，_n 生成的结果也会随之发生改变，比如。

```
*- do 文件的内容
sort price

gen var3 = _n

list price var1 var3 in 1/5
```

将上述 do 文件的内容复制粘贴至你的 do 文件，选中后按 Ctrl+D，会在 Stata 的结果窗口显示如下内容：

```
. *- do 文件的内容
. sort price

. gen var3 = _n

. list price var1 var3 in 1/5

     +---------------------+
     | price   var1   var3 |
     |---------------------|
  1. | 3,291     34      1 |
  2. | 3,299     14      2 |
```

```
  3. | 3,667     18      3 |
  4. | 3,748     68      4 |
  5. | 3,798     66      5 |
     +---------------------+
```

****

但是，_N 的值是永远不会发生变化的，除非你删除了或者增加了样本。因为 _N 代表的是样本的总数，不随排序发生变化。

3.2.1.2 _n 与 _N 的应用

现在你一定会比较好奇，你或许会想，说了这么多，_n 与 _N 到底有什么作用呢？接下来我就来分别介绍一下这俩非常之有用的变量的非常之有用的作用。

（1）利用 _n 将截面数据扩充为面板数据

当你有一份不重复的股票代码，或者是一些省份、地级市文本数据，可以考虑使用 _n 将这份截面数据变为任意平衡的面板数据。

从我的 GitHub 仓库：StataTutorialAppendixData 找到文件夹 AppendixData2。其中，TextileCode.xls 记录了我国 A 股上市的纺织、纺织服饰行业的 124 家上市公司的股票代码数据。

现在我可以利用 _n 的特性将其扩充为 2011 到 2020 的面板数据，也就是为其增加一个从 2011 到 2020 共计 10 年的年份变量。扩充前的样本数是 124，那么扩充之后就应该是 $124 \times 10 = 1240$。我们可以使用如下代码完成面板扩充。

```
**********************************************************************************************
*- 扩充前的数据
    +--------+
    |   code |
    |--------|
 1. | 000036 |
 2. | 000045 |
 3. | 000158 |
 4. | 000301 |
 5. | 000529 |
    |--------|
 6. | 000681 |
 7. | 000712 |
 8. | 000726 |
 9. | 000779 |
10. | 000803 |
    +--------+
**********************************************************************************************
```

```
*- do 文件的内容
// 调用数据
import excel using TextileCode.xls, first clear

// 扩充10倍
expand 10
```

```
// 生成从2011开始依次增加1的时间序列
bys code: gen year = 2010 + _n

// 展示扩充后的数据
list in 1/10
```

将上述 do 文件的内容复制粘贴至你的 do 文件，选中后按 Ctrl+D，会在 Stata 的结果窗口显示如下内容：

```
**********************************************************************************************
    +---------------+
    |   code   year |
    |---------------|
 1. | 000036   2011 |
 2. | 000036   2012 |
 3. | 000036   2013 |
 4. | 000036   2014 |
 5. | 000036   2015 |
    |---------------|
 6. | 000036   2016 |
 7. | 000036   2017 |
 8. | 000036   2018 |
 9. | 000036   2019 |
10. | 000036   2020 |
    +---------------+
**********************************************************************************************
```

（2）判断行业是否发生变化

同样地，_n 与 _N 的组合使用还能用于判断行业是否发生了变化。比如，还是刚刚的数据，我重新生成一个变量 industry，并用它代表行业，将 2015 以前的行业命名为“纺织”，将 2015 及以后的行业命名为“钢铁”。

```
*- do 文件的内容
// 增加行业
gen industry = "纺织" if year < 2015
replace industry = "钢铁" if year >= 2015

// 判断行业是否发生变化
bys code industry: gen temp = _N
bys code: gen temp1 = _N
gen change = 1 if temp != temp1
```

在运行完上述的代码之后，变量 change 等于 1 所对应的公司就是行业发生了变动的。然后只需要删掉重复值，就能够得到一份观测期内行业发生了变动的企业名单。

3.2.2 时间序列变量

为了方便对一些包含时间序列的变量进行滞后、提前的处理，Stata 也提供了一系列的时间序列处理方法。具体而言，有如下几种：

1. L.，滞后一期，即 x_{t-1}。

2. L2.，滞后两期，即 x_{t-2}。
3. ...
4. F.，提前一期，即 x_{t+1}。
5. F2.，提前两期，即 x_{t+2}。
6. ...
7. D.，下期与本期的差异，即 $x_t - x_{t-1}$。
8. D2.，差异的差异，即 $(x_t - x_{t-1}) - (x_{t-1} - x_{t-2}) = x_t - 2x_{t-1} + x_{t+1}$。
9. ...
10. S.，seasonal difference，不知道怎么翻译，但它等于 $x_t - x_{t-1}$。
11. S2.，等于 $x_t - x_{t-2}$。
12. ...

这些东西看起来非常眼花缭乱，但使用过程当中一般我们用 L. 和 F. 比较多。比如，将变量 GDP 滞后一期，可以使用 gen lagGDP = L.GDP。然而，在回归当中，我们并不需要多此一举，可以直接将 L.GDP 放进回归中进行计算。

3.2.3 虚拟变量/分类变量

3.2.3.1 什么是虚拟变量/分类变量

虚拟变量是一种用来表示分类变量的统计工具。在统计分析中，分类变量是指具有不同类别或水平的变量，例如性别、地区、教育程度等。虚拟变量是为了在数学模型中能够处理这些分类变量而引入的。

个人理解向：虚拟变量与分类变量有一定的联系，也有一定的区别。以性别（gender）这个变量为例，在分类变量中，我们可以用 1 表示男性，用 0 表示女性。在这种情况下，1 是大于 0 的，但是我们能说男性大于女性吗？答案当然是否定的，因为男女是平等的。所以在这种情况下我们就可以引入虚拟变量，虚拟变量是一组变量所构成的向量。比如说我生成两个新的变量 gender1 和 gender2，让它们分别只能取 0 和 1，这样就可以用向量 $[0,1]$ 和向量 $[1,0]$ 分别表示男性跟女性，并且不会出现数值上的大小比较问题。

在这里你肯定会想，能不能都取 0 或者都取 1？当然不行，因为这玩意儿合在一起得是一个满秩的矩阵，有几个分类就会生成一个几维的单位矩阵。比如，性别是二分类（不考虑沃尔玛购物袋这种畸形的阿迈里肯式分类），所以只能构成如下的虚拟变量矩阵。

$$Gender = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

通过引入虚拟变量，我们可以将分类变量纳入回归分析、方差分析等统计模型中，使模型能够更好地解释不同类别之间的差异对结果的影响。这有助于处理分类变量对于统计分析和建模的挑战，使得我们可以更准确地理解和预测数据。

3.2.3.2 虚拟变量/分类变量回归

需要注意的是，在上述的性别二分类矩阵中，向量 $\overrightarrow{gender1}$ 与向量 $\overrightarrow{gender2}$ 是共线的，所以将其同时放入回归的时候，Stata 会自动 omitted 一个。但是，在实际操作过程当中，我们并不需要如此麻烦的方法，我们只需要在变量前面加上一个 i. 前缀，声明其用作因子变量（Factor variables）进行回归即可。关于因子变量的解释见 3.2.4。

同样地，我们可以利用以下示例，来帮助我们理解先生成虚拟变量再进行回归，与直接使用具有 i. 前缀的因子变量进行回归的差异。

```
*- do 文件的内容
sysuse auto.dta, clear
```

```
3
4  tab foreign, gen(temp)  // 对变量foreign生成虚拟变量
5
6  reg price temp1 temp2
7
8  reg price i.foreign
9
10 reg price i0.foreign
```

将上述 do 文件的内容复制粘贴至你的 do 文件，选中后按 Ctrl+D，会在 Stata 的结果窗口显示如下内容：

```
**************************************************************************************************
. reg price temp1 temp2
note: temp2 omitted because of collinearity

      Source |       SS           df       MS      Number of obs   =        74
-------------+----------------------------------   F(1, 72)        =      0.17
       Model |  1507382.66         1  1507382.66   Prob > F        =    0.6802
    Residual |   633558013        72  8799416.85   R-squared       =    0.0024
-------------+----------------------------------   Adj R-squared   =   -0.0115
       Total |   635065396        73  8699525.97   Root MSE        =    2966.4

------------------------------------------------------------------------------
       price |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
       temp1 |  -312.2587   754.4488    -0.41   0.680    -1816.225    1191.708
       temp2 |          0  (omitted)
       _cons |   6384.682   632.4346    10.10   0.000     5123.947    7645.417
------------------------------------------------------------------------------

. reg price i.foreign

      Source |       SS           df       MS      Number of obs   =        74
-------------+----------------------------------   F(1, 72)        =      0.17
       Model |  1507382.66         1  1507382.66   Prob > F        =    0.6802
    Residual |   633558013        72  8799416.85   R-squared       =    0.0024
-------------+----------------------------------   Adj R-squared   =   -0.0115
       Total |   635065396        73  8699525.97   Root MSE        =    2966.4

------------------------------------------------------------------------------
       price |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
     foreign |
    Foreign  |   312.2587   754.4488     0.41   0.680    -1191.708    1816.225
       _cons |   6072.423    411.363    14.76   0.000     5252.386     6892.46
------------------------------------------------------------------------------
```

```
. reg price i0.foreign

      Source |       SS           df       MS      Number of obs   =        74
-------------+----------------------------------   F(1, 72)        =      0.17
       Model |  1507382.66         1  1507382.66   Prob > F        =    0.6802
    Residual |   633558013        72  8799416.85   R-squared       =    0.0024
-------------+----------------------------------   Adj R-squared   =   -0.0115
       Total |   635065396        73  8699525.97   Root MSE        =    2966.4

------------------------------------------------------------------------------
       price |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
     foreign |
   Domestic  |  -312.2587   754.4488    -0.41   0.680    -1816.225    1191.708
       _cons |   6384.682   632.4346    10.10   0.000     5123.947    7645.417
------------------------------------------------------------------------------
*******************************************************************************
```

我在这里展示了三种回归结果，可以发现第一个回归与第三个回归是等价的。其中，新生成的变量 `temp1` 表示变量 `foreign` 中的国产（Domestic）；新生成的变量 `temp2` 表示变量 `foreign` 中的进口（Foreign）。在第二个回归与第三个回归当中：第二个回归中我直接将 `i.foreign` 放进回归命令中，可以发现 Stata 默认以进口（Foreign）作为基准组（base level）；在第三个回归当中，我指定国产（Domestic）为基准组（base level），就可以直接看到国产车与价格之间的回归关系。而这两者（`temp1` 与 `temp2`）是共线的，所以都带入的时候（如第一个回归所示），Stata 会自动忽略掉一个，被忽略掉的变量中即原分类变量中类别数更少的一个。在本例中，Domestic 有 52 个，Foreign 有 22 个，所以忽略 Foreign，即变量 `foreign` 取 1 的情况，也就是 `temp2` 。当类别数都一样多时，Stata 会忽略掉放在后面的变量。

3.2.3.3 分类变量的标签赋值

在上述的例子中，我使用了 `auto.dta` 这份数据集，你可以在命令窗口输入 `browse` 或者直接点击 Stata 左上角的浏览数据（有放大镜标识的那个按钮）。可以发现，它的变量 `foreign` 是蓝色的，这是 Stata 中的一种特殊数据形式，即以文本形式展示的数值变量。同样的文本会对应同样的数值。比如，你点击 Domestic，会发现正上方的框框中显示的值为 0；点击 Foreign，会发现正上方的框框中显示的值为 1。

那么，如何个任意数值赋予文本标签值呢？比如我现在有一份新的数据，其中有一个变量叫做 `score`，它记录了一个班级里所有学生的成绩。当成绩是 85 以上时，我们将这个学生的评级定为 A；成绩在 60-85 时，评级为 B；成绩在 60 以下时，评级为 C。**注意**，这样的操作没有太大的实际意义，只是为了看起来好看一点，以及避免遗忘。

有两种方法都可以实现将文本与数值对应起来的操作：

1. 可以建立一个与成绩对应的文本变量 `rank`，具体操作如下：

```
gen rank = A if score >= 85
replace rank = B if (score < 85) & (score >= 60)
replace rank = C if score < 60
```

2. 使用 Stata 的“文本-数值”对应型变量，具体操作如下：

```
gen rank = 2 if score >= 85
```

```
    replace rank = 1 if (score < 85) & (score >= 60)
    replace rank = 0 if score < 60

    label define rank_label 2 "A" 1 "B" 0 "C"
    label values rank rank_label
```

不难发现，两者的区别并不是很大，并且都生成了新的变量，所以我一直觉得这种“文本-数值”型变量的存在比较鸡肋。当然，如果数据需要在多个合作者之间进行流转，这样的标签能更好地表现分类变量不同类别的含义。

3.2.4 因子变量

通过 help fvvarlist 可以发现，Stata 对因子变量的解释是：因子变量是现有变量 varlists 的扩展。因子变量能够从分类变量中产生指标变量、构建分类变量间的交互作用、分类变量与连续变量的交互作用以及连续变量间的交互作用。大多数估计和后估计命令以及少数其他命令都允许在估计当中使用因子变量。

主要的因子变量操作符或者说指示符有如下几种：

1. i.，用来声明分类变量，比如：i.foreign。**特别地**，可以通过 in. 的方式指定该分类变量中的第 n 种类别单独参与回归。比如在虚拟变量的章节中我指定了 i0.foreign。
2. c.，用来声明连续变量，但在回归中 c.var 直接等价于 var。
3. o.，用来声明需要忽略的变量或者指标？几乎没用过，我也不太了解。
4. #，用来表示两个变量的交互项，比如将 reg y x1#x2 等价于将两个变量相乘后得到的第三个变量 x3 带入回归，即 reg y x3。
5. ##，用来表示两个变量的交互项及其单独项，比如 reg y x1##x2 等价于 reg y x1#x2 x1 x2

3.2.5 巧用 egen

通过 help egen，我们可以发现，Stata 对 egen 的介绍是：对命令 generate，即 gen 的扩展。相比于命令 gen，egen 提供了更丰富的功能。

3.2.5.1 与 gen 的一个差异

对于命令 egen 与命令 gen，它们都能使用 sum() 函数，但是两个命令在使用的工程当中存在一定的差异，具体见下。

```
*- do 文件的内容
sysuse auto.dta, clear

gen sumprice1 = sum(price)
egen sumprice2 = sum(price)

list price sumprice* in 1/10
list price sumprice* in -1     // 展示最后一个观测样本
```

将上述 do 文件的内容复制粘贴至你的 do 文件，选中后按 Ctrl+D，会在 Stata 的结果窗口显示如下内容：

```
*******************************************************************************************
. list price sumprice* in 1/10

     +-----------------------------+
```

```
     |  price   sumpri~1   sumpri~2 |
     |------------------------------|
  1. |  4,099       4099     456229 |
  2. |  4,749       8848     456229 |
  3. |  3,799      12647     456229 |
  4. |  4,816      17463     456229 |
  5. |  7,827      25290     456229 |
     |------------------------------|
  6. |  5,788      31078     456229 |
  7. |  4,453      35531     456229 |
  8. |  5,189      40720     456229 |
  9. | 10,372      51092     456229 |
 10. |  4,082      55174     456229 |
     +------------------------------+

. list price sumprice* in -1

     +------------------------------+
     |  price   sumpri~1   sumpri~2 |
     |------------------------------|
 74. | 11,995     456229     456229 |
     +------------------------------+
```

****

在上述代码中，我先调用了 auto.dta 数据集，然后分别使用 gen 与 egen 命令的 sum() 函数对变量 price 进行纵向的加总求和处理。不难发现，对于命令 gen，其 sum() 函数是累加的求和方法，即第 3 项的值是 price 中前 3 个观测值的和。而对于命令 egen，其 sum() 函数是总和的求和方法，即对该变量的所有观测值进行求和。所以，当我们直接展示最后一个观测值的时候，它俩的取值完全一样。

3.2.5.2 其他实用函数

同样地，利用 egen 的其他函数功能，我们可以快速的求出很多描述性统计指标[1]。我们依然调用 auto.dta 数据集，并对其中的变量 price 做如下计算：

1. mean()，计算均值，具体使用方法为：egen Average = mean(price)。联合 by() 函数用可以计算分组的均值，比如：egen Average = mean(price), by(foreign)。
2. median()，计算中位数，具体使用方法为：egen Median = median(price)，同样可以通过 egen Median = median(price), by(foreign) 计算分组中位数。
3. sd()，计算标准差，具体使用方法为：egen StdDev = sd(price)，可联合 by() 函数，具体写法同上。
4. min()/max()，计算最小值/最大值，具体使用方法为：egen Min = min(price)，最大值的方法类似，同样可以联合 by() 函数。
5. diff()，横向比对同意样本的两个变量的值是否相等，具体使用方法为：egen Diff = diff(var1 var2)。
6. rowmean()/rowmedian()/rowmin()/rowmax()，横向计算同一个样本在这些变量间的均值、中位数、最小值，以及最大值，使用方法为：egen xxx = rowmean(var1 var2 var3)。

此外，egen 命令的函数功能是非常丰富的，还可以计算移动平均（ma() 函数），可以自行通过 help egen 查看其支持的所有功能。使用代码与上述有序列表中提及的内容类似。

[1] **当然**，这些内容也可以通过 summarize 命令实现

3.3 重复值、缺失值与离群值

3.3.1 重复值

对于一份数据，有一些 Bug 是我们无法预测的，比如一些重复的样本，变量的一些缺失值以及离群值等，而在回归当中，我们往往考虑的是一种平均的效应，所以我们需要处理掉这些不合常理的数值。

3.3.1.1 识别重复值

在 Stata 中，我们可以通过 `isid` 来判断变量是否重复，我们仍然调用 `auto.dta` 数据据，具体的使用方法如下：

```
*- do 文件的内容
sysuse auto.dta, clear

isid make

isid foreign
```

将上述 do 文件的内容复制粘贴至你的 do 文件，选中后按 Ctrl+D，会在 Stata 的结果窗口显示如下内容：

```
********************************************************************************
. isid make

. isid foreign
variable foreign does not uniquely identify the observations
r(459);
********************************************************************************
```

可以发现，对不不存在重复值的变量 `make`，运行该命令后没有任何反应；对于存在重复值的变量 `foreign`，运行之后显示了一串信息，用于提示我们改变量存在重复值。

3.3.1.2 剔除重复值

对于已知存在重复值的样本，比如在上市公司的面板数据中，我们一般会要求保持每一个公司一年只存在一组观测样本，对应到省级/市级面板就是每一个省份、地级市在每一年的观测值都应只存在一组。当然，这里的面板指的是最小面板单位，如果你的面板数据中既有省份又有地级市，就应当保证每一个地级市只存在一年的观测样本。

删除重复值的命令是 `duplicates drop stkcd year, force`，同理可以扩展到省级面板数据:`duplicates drop province year, force`，以及市级面板数据：`duplicates drop city year, force`。

这里的关键点是，你需要保证 `drop` 后面跟随的两个变量能够组成最小的面板单位，也就是你使用 `xtset` 时，后面跟随的两个变量。

3.3.2 缺失值

通过 `help missing`，我们可以查看 Stata 中所有缺失值的种类[2]。虽然数值型缺失值的种类繁多，但好消息是我们并不需要具体地知道这些数值型的缺失值都是哪些以及怎么写。

[2] **冷知识**：Stata 中一共有 27 中数值型的缺失值

3.3.2.1 缺失值的影响

有一些命令在使用的过程中会自动忽略掉缺失值，而有一些则不会忽略。如果不处理缺失值，则有可能会导致一些命令的使用失效，尤其是在处理数据的初级阶段。同样地，我还是提供一个例子以供大家更好地理解不处理缺失值的后果。老规矩，还是使用 auto.dta 数据集。

```
*- do 文件的内容
sysuse auto.dta, clear

replace price = . if price > 4000     // 手动将价格大于4000的替换为缺失值

sum price if price > 3000             // 对价格大于3000的样本做描述性统计

count if price > 3000                 // 统计大于3000的样本数
```

将上述 do 文件的内容复制粘贴至你的 do 文件，选中后按 Ctrl+D，会在 Stata 的结果窗口显示如下内容：

```
************************************************************************************************
. sum price if price > 3000

    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
       price |         11    3750.909    246.9297       3291       3995

. count if price > 3000
  74
************************************************************************************************
```

你可以发现，在描述性统计当中，该命令自动的忽略了缺失值，告诉我们价格大于 3000 且不是缺失值的有效样本一共有 11 个。而在 count 命令中（该命令用于统计样本个数），它告诉我们价格大于 3000 的样本个数一共有 74 个。这样的结果意味着两个方面：

1. Stata 默认缺失值的数值是大于所有非缺失数值的。
2. 有一些命令会将缺失值也统计进来。

所以我们非常有必要在处理完重复样本问题之后，观察各个变量的缺失值情况。那么如何查找数据当中是否存在缺失值呢？请看如下方法。

3.3.2.2 识别缺失值

首先，我提供一种方法观察变量中的缺失值情况。在之前的章节当中，我们学到了 _N 可以用来表示样本的总数，而 sum 命令是可以统计变量的有效观测值个数的，所以结合这两个命令的结果，我们就可以判断某一变量当中是否存在缺失值，具体操作如下：

```
*- do 文件的内容
sysuse auto.dta, clear

sum

dis _N        // 观测值不等于样本数，变量有缺失
```

将上述 do 文件的内容复制粘贴至你的 do 文件，选中后按 Ctrl+D，会在 Stata 的结果窗口显示如下内容：

```
*******************************************************************************************
. sum

    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
        make |          0
       price |         74    6165.257    2949.496       3291      15906
         mpg |         74     21.2973    5.785503         12         41
       rep78 |         69    3.405797    .9899323          1          5
    headroom |         74    2.993243    .8459948        1.5          5
-------------+---------------------------------------------------------
       trunk |         74    13.75676    4.277404          5         23
      weight |         74    3019.459    777.1936       1760       4840
      length |         74    187.9324    22.26634        142        233
        turn |         74    39.64865    4.399354         31         51
displacement |         74    197.2973    91.83722         79        425
-------------+---------------------------------------------------------
  gear_ratio |         74    3.014865    .4562871       2.19       3.89
     foreign |         74    .2972973    .4601885          0          1

. dis _N
74
*******************************************************************************************
```

在命令 sum，即 summarize 的简写中，Obs 为 0 所对应的变量一般为文本型变量，即以文本形式记载的变量，无法参与统计。除无法统计的部份外，其余的变量所对应的 Obs 的值意味着该变量非缺失的有效观测值个数。而 _N 显示，对于这样一份观测数据来说，一共有 74 个观测样本。所以我们可以推出，当观测样本总数不等于 sum 中的 Obs 时，则该变量就存在缺失值。

3.3.2.3 确定缺失值比例

要想处理缺失值，首先就需要确定该变量的缺失比例。如果缺失比例比较低，对连续变量我们可以采用均值填充，对分类变量可以采用众数填充；如果缺失比例比较高，我们可以考虑直接删除这个变量。比例多高算高，多低算低，我想没有绝对的定义。菜菜 TsaiTsai在她的机器学习教程中提到了一个比例是 30%，我认为是可取的。

所以，我给出一段自己写的，并且经常用的代码，来帮大家处理变量中的缺失比例计算问题，具体如下：

```
*- do 文件的内容
sysuse auto.dta, clear

local Variables price mpg rep78 headroom trunk weight length
foreach v in `Variables' {
  quietly summarize `v'
  dis _skip(10) "`v'的缺失比例为 = " %4.2f (_N-r(N))/_N*100 "%"
}
```

将上述 do 文件的内容复制粘贴至你的 do 文件，选中后按 Ctrl+D，会在 Stata 的结果窗口显示如下内容：

```
**********************************************************************************************
          price的缺失比例为 = 0.00%
          mpg的缺失比例为 = 0.00%
          rep78的缺失比例为 = 6.76%
          headroom的缺失比例为 = 0.00%
          trunk的缺失比例为 = 0.00%
          weight的缺失比例为 = 0.00%
          length的缺失比例为 = 0.00%
**********************************************************************************************
```

不难发现，我的这串循环很好地展示了每个数值型变量的缺失比例。如果你需要在你的数据当中使用它，你只需要将你的数据导入后，把我的这个循环复制粘贴到你的 .do 文件中，然后将局部宏 Variables 后面的变量 price mpg rep78 headroom trunk weight length 全部替换为你自己的数值型变量，然后选中这个 local 与下面的循环，并一起运行，就可以自动计算每一个变量的缺失比例了。

3.3.2.4 处理缺失值

刚刚我们提到，当变量当中存在缺失值的时候，我们需要先确定缺失值占样本总数的比例。当比例过高时（我们的阈值是 30%）我们可以直接删除该变量，处理代码是：drop var。当比例小于等于 30% 时，我们需要根据该变量的类型进行均值或者众数的填充。值得注意的是，对于一些上市公司，如果行业差别很大，也可以考虑进一步按行业均值/众数填充来填充变量。

结合我们之前章节提到的循环的实用操作，我们可以写出对连续变量的均值填充方法，代码如下：

```
*- do 文件的内容
// 填充单个变量
egen rep78mean = mean(rep78)
replace rep78 = rep78mean if rep78 == .
drop rep78mean

// 批量填充
local Variables price mpg rep78
foreach v in `Variables' {
    egen `v'mean = mean(`v')
    replace `v' = `v'mean if `v' == .
   drop `v'mean
}
```

结合我们之前章节提到的循环的实用操作，我们可以写出对分类的众数填充方法，代码如下：

```
*- do 文件的内容
sysuse auto.dta, clear

// 手动生成缺失值
replace foreign = . in 3
replace foreign = . in 70

// 查看分类变量的类别分布情况，即观察众数
tab foreign

// 利用众数进行填充
```

```
replace foreign = 0 if foreign == .
```

需要注意的是，对于缺失值的处理方法有很多种，比如还有什么多重插补、线性填充等。考虑到我们在后面还有很多复杂的实证检验方法，所以这部分我建议大家从简，只考虑均值填充跟众数填充两种较为简单的方法，除非你的研究主题是不同的缺失值填充方法对 xxx 的影响。

3.3.3 离群值

离群值是脱离数据群体的值，极大极小值不一定为离群值。数据群体指的是处于 (p25-1.5iqr, p75+1.5iqr) 之间的值。iqr 即四分位间距（interquartile range），等于 $p75-p25$，p25、p75 分别指处于第 1 个四分位（第 25 个百分位）的值，p25、p50、p75 分别叫第 1、2、3 个四分位，p50 即中位数。

3.3.3.1 离群值的影响

我举一个例子，可以帮助大家更快地了解到离群值对回归结果的影响，具体如下：

```
*- do 文件的内容
sysuse auto.dta, clear

// 查看第25分位与第75分位的值
_pctile price, p(25 75)
dis "p25=" r(r1)
dis "p75=" r(r2)

// 标记离群值（tempVar），超过上限与低于下限的都标记为1
local Lower r(r1)-1.5*(r(r2)-r(r1))
local Upper r(r2)+1.5*(r(r2)-r(r1))

gen tempVar = 1 if price>`Upper' | price<`Lower'
sum tempVar

// 回归
reg price mpg rep78
est store Model1
reg price mpg rep78 if tempVar != 1
est store Model2
esttab Model1 Model2, nogap compress mtitle("Yes" "No")
```

将上述 do 文件的内容复制粘贴至你的 do 文件，选中后按 Ctrl+D，会在 Stata 的结果窗口显示如下内容：

```
*******************************************************************************************************
// 变量price的第25分位（第一个4分位）的值
. dis "p25=" r(r1)
p25=4195

// 变量price的第75分位（第三个4分位）的值
. dis "p75=" r(r2)
p75=6342
```

```
// 比较模型之间的差异
. esttab Model1 Model2, nogap compress mtitle("Yes" "No")

------------------------------------
                  (1)          (2)
                  Yes           No
------------------------------------
mpg           -271.6***    -103.6***
             (-4.70)      (-3.87)
rep78          667.0        496.1**
              (1.95)       (3.29)
_cons         9657.8***    5639.0***
              (7.17)       (9.23)
------------------------------------
N                 69           58
------------------------------------
t statistics in parentheses
* p<0.05, ** p<0.01, *** p<0.001
********************************************************************************************
```

在上述的代码中，我引入了一些新的命令。具体而言：

1. `est store Model1`，放在回归之后，意思是将回归的结果存储在名为 `Model1` 的变量中，这里你可以起任何你喜欢的名字。
2. `esttab Model1 Model2`，展示回归模型 `Model1` 和 `Model2` 的内容。其中，`nogap` 表示不要留空隙，`compress` 表示压缩一下表格，主要是用来紧凑一下展示结果。`mtitle("Yes" "No")` 表示这两个回归展现出来的名字一个叫 Yes，另一个叫 No。

在模型（1）中，我没有做任何操作，即包含离群值在内进行回归；在模型（2）中，我使用 `if` 条件来限制离群值，即仅使用离群值以外的变量进行回归。可以发现，在模型（1）与模型（2）的结果当中，变量 `rep78` 的显著性发生了变化，从不显著变为了显著。**因此**，我们可以判定，离群值的存在可能会改变我们对模型的解释。

3.3.3.2 查找离群值

那么，我们应该如何查找一个变量是否存在离群值呢？同样地，我也提供一个我自己编写的非常实用的小循环，具体如下：

```
*- do 文件的内容
sysuse auto.dta, clear

local Variables price mpg rep78 headroom trunk
foreach v in `Variables' {

    _pctile `v', p(25 75)
    scalar Lower = r(r1)-1.5*(r(r2)-r(r1))
    scalar Upper = r(r2)+1.5*(r(r2)-r(r1))

    quietly sum `v'
    if (r(max)>Upper) | (r(min)<Lower) {
        dis _skip(15) "变量`v'存在离群值"
```

```
    }
    quietly gen Outlier`v'=1 if `v'>Upper | `v'<Lower
    quietly replace Outlier`v'=0 if Outlier`v'==.
}
```

将上述 do 文件的内容复制粘贴至你的 do 文件，选中后按 Ctrl+D，会在 Stata 的结果窗口显示如下内容：

```
**************************************************************************************************
                变量price存在离群值
                变量mpg存在离群值
                变量rep78存在离群值
**************************************************************************************************
```

可以看到，运行完这串代码之后，Stata 显示这三个变量存在离群值。同样地，你也可以将这些代码放在你的 .do 文件中，只需要将局部宏 Variables 后面的变量 price mpg rep78 headroom trunk 改成你自己的，然后再选中这个 local 与下面的循环，并一起运行，就可以用于提示存在离群值的变量了。

此外，也还有一种更加直观地做法，就是画箱型图。绘图代码为：graph box price，画出来的图如 3.1 所示。

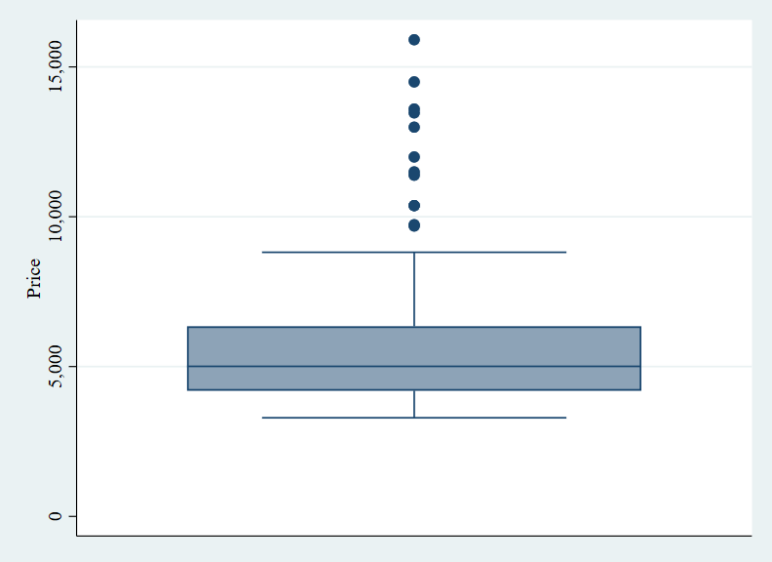

图 3.1: 变量 price 的离群值

3.3.3.3 处理离群值

与处理缺失值一样，处理离群值的方法有也很多种，不过最常见的还是截尾、缩尾以及对数转换。

1. 对数转换，将存在离群值的变量取对数，常见于处理上市公司年龄、专利数量等，具体操作是：gen newVar = ln(Var)
2. 截尾，即直接删除所有的离群值，具体操作如下：

```
*- do 文件的内容
// 调用数据
sysuse auto.dta, clear

// 定位第1和第3四分位数
_pctile price, p(25 75)
```

```
7
8     // 删除该变量的离群值
9     local Lower r(r1)-1.5*(r(r2)-r(r1))
10    local Upper r(r2)+1.5*(r(r2)-r(r1))
11    drop if price > `Upper' | price < `Lower'
```

3. 缩尾，即将右侧离群值减少至右侧边界，将左侧离群值增加至左侧边界，不改变样本数，具体操作如下：

```
1     *- do 文件的内容
2     // 下载缩尾处理的命令
3     ssc install winsor
4
5     // 调用数据
6     sysuse auto.dta, clear
7
8     // 双侧各1%的缩尾处理
9     winsor price, gen(W_price) p(0.01)
10
11    // 仅左侧1%的缩尾处理
12    winsor price, gen(W_price) p(0.01) low
13
14    // 仅右侧1%的缩尾处理
15    winsor price, gen(W_price) p(0.01) high
```

注意，上述的这些命令都可以配合 `foreach` 循环实现多变量的批量处理，请读者自行探索。

3.4 数据的合并于追加

在实际的操作过程当中，我们几乎往往都需要将多份 `.dta` 文件进行合并或者追加。这里的合并指的是横向合并，即给当前样本增加更多的变量，比如将固定资产与经营现金流的相关数据进行合并；这里的追加实际上指代的是纵向合并，即给当前的样本增加更多的观测值，比如把 2020 年的观测数据与 2021 年的观测数据进行合并。

3.4.1 横向合并

横向合并相当于是增加变量，与 Excel 中的 `VLOOKUP` 函数一致，与 Python 中的 `pandas.concat(axis=1)` 一致。合并的要点在于找到用于合并的关键变量，以及选择合并的模式。

3.4.1.1 一对一合并

在两份数据当中，具有相同的关键变量，且每一个表中的该变量下的每一个值都对应一条唯一的样本。

从我的 GitHub 仓库：StataTutorialAppendixData 找到文件夹 `AppendixData2`。其中，`MergeData1.dta` 与 `MergeData2.dta` 是我用来演示一对一合并的主要数据文件。

分别点击打开这两份数据文件，可以发现变量 `make` 所对应的样本是唯一的，所以我们以该变量作为合并的关键变量（Key）进行合并，合并代码如下：

```
1  *- do 文件的内容
2  use MergeData1.dta, clear          // 调用MergeData1.dta
3
4  merge 1:1 make using MergeData2.dta // 以make为关键变量合并MergeData2.dta
```

将上述 do 文件的内容复制粘贴至你的 do 文件，选中后按 Ctrl+D，会在 Stata 的结果窗口显示如下内容：

```
***********************************************************************************************
. merge 1:1 make using MergeData2.dta

    Result                           # of obs.
    -----------------------------------------
    not matched                             0
    matched                                74  (_merge==3)
    -----------------------------------------
***********************************************************************************************
```

现在我们对合并的结果进行以下解读，可以发现，结果很清晰的展示了合并上的样本数与没有合并上的样本数。在这里我们可以发现有一个东西叫 _merge，同时，在你的数据当中也会新生成一个名为 _merge 的变量。这个变量的取值详细的说明了合并的情况，具体如下：

1. _merge==1，变量仅在主表中出现，主表即我们的合并表，在本例中是 MergeData1.dta。举个更直观的例子就是，make 变量中的某一个制造商在 MergeData1.dta 中出现了，但是在 MergeData2.dta 中没有出现，则合并的时候就会在 not matched 中详细显示 from master 的个数，即 _merge==1 的个数。
2. _merge==1，变量仅在使用表中出现，使用表即我们的被合并表，在本例中是 MergeData2.dta。举个更直观的例子就是，make 变量中的某一个制造商在 MergeData2.dta 中出现了，但是在 MergeData1.dta 中没有出现，则合并的时候就会在 not matched 中详细显示 from using 的个数，即 _merge==2 的个数。
3. _merge==3，皆大欢喜，表示合并表中的关键变量在被合并表中都出现了。以上市公司的面板数据为例，也就是说我们的资产负债表面板数据能够完整地跟利润表、所有者权益表对上号，不存在说有哪个公司有资产负债表的相关数据，但是没有利润表、所有者权益表数据的情况。

3.4.1.2 多对一情况

在两份数据当中，具有相同的关键变量，且在合并表中的关键变量出现了多次，但在被合并表中仅出现了一次。举一个实际的例子，比如说我们已经有了企业的资产负债表面板数据，但是我们希望跟企业的地址进行合并。企业的地址是不随时间发生变化的，所以存储企业地址信息的数据当中只有企业的股票代码跟详细地址信息。在这种情况下，我们就需要选择关键变量为企业的股票代码，并使用多对一的合并情况。

从我的 GitHub 仓库：StataTutorialAppendixData 找到文件夹 AppendixData2。其中，MergeData3.dta 与 MergeData4.dta 是我用来演示多对一合并的主要数据文件。

分别点击打开这两份数据文件，可以发现变量 Num 所对应的样本在 MergeData3.dta 中有多个，但是在 MergeData4.dta 中只有一个。那么在这种情况下，我们就可以选择以 Num 作为关键变量，进行多对一的合并，具体操作如下：

```
*- do 文件的内容
use MergeData3.dta, clear          // 调用MergeData3.dta

merge m:1 Num using MergeData2.dta // 以Num为关键变量合并MergeData4.dta
```

将上述 do 文件的内容复制粘贴至你的 do 文件，选中后按 Ctrl+D，会在 Stata 的结果窗口显示如下内容：

```
***********************************************************************************************
. merge m:1 Num using MergeData4.dta

    Result                           # of obs.
```

```
    ----------------------------------------
    not matched                             0
    matched                               100  (_merge==3)
    ----------------------------------------
*******************************************************************************************************
```

对合并结果的解读可以参照一对一合并中所提到的内容。

除此之外，Stata 还有一对多的合并 `1:m`，以及多对多的合并 `m:m`。其中，一对多的合并是多对一合并的逆过程，只是交换了合并表与被合并表的合并顺序，如果掌握了多对一的合并，那么完全不需要再理会一对多的合并；多对多的合并是非常混乱的，如果没有注意好样本的排序，那么大概率会出现合并紊乱的问题。

3.4.1.3 合并操作练习

如果你读完了这些内容还是有点糊，不妨使用我提供的练习数据自行练习。从我的 GitHub 仓库：StataTutorialAppendixData 找到文件夹 `AppendixData2`。其中，文件夹 `Test1` 包含了这次练习的所有内容。在该文件夹中：

1. `FirmProvince.dta`，为企业所在的省市情况数据文件。
2. `RD.dta`，为企业的研发支出情况数据文件。
3. `Patent.dta`，为企业的绿色专利数据文件。

请你使用我前面提到的一对一合并与多对一合并的方法，将这三份文件合并成一份完整的面板数据文件。

3.4.2 纵向合并

相比于较为复杂的横向合并，Stata 中的纵向合并要简单很多，无非就是追加样本。相当于在 Excel 中往表格末端复制粘贴增加数据，也相当于 Python 的 `pandas.concat()`。

从我的 GitHub 仓库：StataTutorialAppendixData 找到文件夹 `AppendixData2`。其中，`MergeData1.dta` 与 `MergeData2.dta` 是我用来演示纵向合并的主要数据文件。

在纵向合并当中，我们需要保证：1、保证变量名完全一致，包括大小写。Stata 区分大小写，Stata 与 stata 是不同的名称；2、需要保证两个数据文件中相同变量格式相同。比如对于两张表中的同一个变量 `make`，不能在第一张表中是文本型，而在第二张表中是数值型。对于第 2 点，有一种情况是允许的，比如两张表中同一变量都为数值型或者文本型，但他们的数据格式不一样，这种情况只需要在 `append` 命令后面加上 `force` 选项即可。

```
*- do 文件的内容
use MergeData1.dta, clear                  // 调用MergeData1.dta

append using MergeData2.dta, gen(_append) // 以纵向合并MergeData2.dta
```

这里面不会有任何反馈，但是生成了一个叫 `_append` 的变量。在这个变量里面，0 对应的是最初调用的表，即 `MergeData1.dta`；1 对应的是后来追加合并的表，即 `MergeData2.dta`。

3.4.3 究极缝合怪之合并练习

看到这里，我想你已经掌握了循环的使用、横向合并以及纵向合并。所以，我要给你来一个非常非常有意思的合并大练习。

从我的 GitHub 仓库：StataTutorialAppendixData 找到文件夹 `AppendixData2`。其中，文件夹 `Test2` 包含了这次练习的所有内容。在该文件夹中：

1. `usbirthn.txt`，包含了美国各个州第 n 年的出生率情况。
2. `uscountyA.txt` 与 `uscountyB.txt`，为美国各个州的一些其他信息。

请你使用我前面提到的一对一合并与多对一合并的方法，将这三份文件合并成一份完整的面板数据文件。**注意**，这是比较有规律的文件名，所以你可以考虑使用循环，非常有意思。

3.5 长宽数据转换

好像很多教程都会提到长宽数据的转换，但是这个东西在实际的使用过程，能接触到的频率还是比较低的。而且受限于变量命名特点，也没有比较万金油的解决方法。所以，如果你正在为这个问题烦恼，不如直接请教身边大佬（skr）。但是鉴于这个问题确实也会遇到，所以我象征性地给出一些代码供大家参考。

```
*- do 文件的内容
* 宽转长生成数据

clear
input id sex inc80 inc81 inc82 xx80 xx81 xx82
    1 0 5000 5500 6000 1 2 3
    2 1 2000 2200 3300 2 3 4
    3 0 3000 2000 1000 6 4 8
end

reshape long inc xx, i(id) j(year)

// 修正年份
replace year = real("18"+string(year))

* 长转宽生成数据

clear
input id year sex inc xx
    1 1880 0 5000 1
    1 1881 0 5500 2
    1 1882 0 6000 3
    2 1880 1 2000 2
    2 1881 1 2200 3
    2 1882 1 3300 4
    3 1880 0 3000 6
    3 1881 0 2000 4
    3 1882 0 1000 8
end

reshape wide inc xx, i(id) j(year)
```

虽然 Stata 提供了支持长款数据转换的 `reshape long` 跟 `reshape wide`，但是 Family who knows 啊，这玩意儿真的巨难用。所以我还是建议你去参考一下连享会的推文，学习一下第三方的 `gather` 与 `spread` 命令。再者，实在不会的话直接稳身边大佬！

3.6 文字变量处理

文字变量的处理是我比较喜欢的环节，因为我们经常会与这个东西打交道，所以我强烈建议你把这个当重点中的重点进行学习。我向来喜欢把文字变量分为两类：一类是本身是数值型变量，但被识别成了文字类型；一类是天生的就该以文本形式记录的变量。前者多出现在导入 CSMAR 数据的过程中，后者多见于一些类似于地址、姓名等信息的记载。

所以，在后面的内容当中，我也将分两个小节分别介绍这样中我认为的文字变量。对第一类文字变量，主要讲述如何处理与转换它；对第二类文字变量，主要讲述如何从中提取某些特征。

3.6.1 第一类文字变量

3.6.1.1 转换数值文本

从我的 GitHub 仓库：StataTutorialAppendixData 找到文件夹 `AppendixData2`。其中，`CharacterData.xls` 是我用来演示对第一类文字变量进行处理的主要数据文件。导入 Stata 后（使用 `import excel`），可以发现，该数据如图 3.2 所示。

	var1	var2	var3
1	1	.	
2	0	1	0
3	2	3	.6666666666666666
4	3	4	.75
5	4	0	
6	5	6	N/A

图 3.2: Stata 中的文本型变量

注意，我这里修改过暗色模式，所以跟你显示出来的样式可能有点不一样。对于默认模式的 Stata 而言，文本型变量，即 `var2` 与 `var3` 应该是红色的（如果我没记错的话）。你可以发现，在我的这份数据中有两种缺失模式，一种是含有缺失值 `.` 的缺失；另一种是含有空值或者 `N/A` 的缺失。

在你处理该变量的过程中，你需要思考两个问题：第一，是否需要保留原有文本变量；第二，空值是否有意义。当你需要保留原有文本变量，即在已有变量的基础之上重新生成一个新的数值型变量，需要使用的代码是：`destring var2, gen(var2N)`，其意思是，对已有变量 `var2` 进行数值化处理，并将完成数值化处理后的结果记录在新变量 `var2N` 中。当你不需要保留原有文本变量的时候，可以直接将 `gen(var2N)` 替换为 `replace` 选项，这样就会以新生成的数值型结果覆盖原有的文本。当你觉得空值没有意义的时候，可以使用 `force` 选项，来强制要求将例如 `N/A` 在内的文本内容直接转换为缺失值 `.`。

3.6.1.2 转换百分数文本

对于一些以文本类型存储的百分数，也可以使用 `destring` 命令将其直接转换为百分数，比如：

```
*- do 文件的内容
clear
input str6 Percent
    "10%"
    "20%"
    "30%"
    "40%"
    "50%"
end
```

```
browse

destring Percent, gen(Num) percent
```

将上述 do 文件的内容复制粘贴至你的 do 文件，选中后按 Ctrl+D，会在 Stata 的结果窗口显示如下内容：

```
. list

     +---------------+
     | Percent   Num |
     |---------------|
  1. |     10%    .1 |
  2. |     20%    .2 |
  3. |     30%    .3 |
  4. |     40%    .4 |
  5. |     50%    .5 |
     +---------------+
```

你也可以使用 `gen Num1 = substr(Percent,1,2)` 从变量 Percent 中提取文本的前两位。然后使用 `destring Num1, replace` 将提取出来的文本转换为数值变量。最后使用 `replace Num1 = Num1/100`。同样可以得到一模一样的结果。

3.6.2 第二类文字变量

第二类文字变量指的是对一些天然就应当以文字形式存在的变量，比如地址等。接下来我将介绍一些在 Stata 中常用的，用来处理文字变量的命令。

3.6.2.1 文字变量拆分

首先我生成一份地址数据。

```
*- do 文件的内容
clear
input str18 City
    台湾省台北市
    台湾省高雄市
    湖北省潜江市
    湖北省武汉市
    湖南省长沙市
    四川省成都市
end
```

将上述 do 文件的内容复制粘贴至你的 do 文件，选中后按 Ctrl+D，会在 Stata 的中生成一些有效的数据。通过观察我们可以发现，变量 City 中都是以省份 + 城市的方式构成的，并且有明确的以“省”结尾的省份和以“市”结尾的地级市。那在这种情况下，我们就可以使用 split 命令拆分省份与地级市。

```
*- do 文件的内容
split City, parse("省")
```

```
replace City1 = City1 + "省"

rename City1 province
rename City2 city
```

将上述 do 文件的内容复制粘贴至你的 do 文件，选中后按 Ctrl+D，会在 Stata 的结果窗口显示如下内容：

```
**********************************************************************************************
. list,separator(10)

     +----------------------------------+
     |         City   province     city |
     |----------------------------------|
  1. | 台湾省台北市      台湾省    台北市 |
  2. | 台湾省高雄市      台湾省    高雄市 |
  3. | 湖北省潜江市      湖北省    潜江市 |
  4. | 湖北省武汉市      湖北省    武汉市 |
  5. | 湖南省长沙市      湖南省    长沙市 |
  6. | 四川省成都市      四川省    成都市 |
     +----------------------------------+
**********************************************************************************************
```

在上述命令中：

1. `split City, parse("省")`，对变量 City 以“省”为关键字进行分割。以“台湾省台北市”为例，分割之后就变成了“台湾”跟“台北市”，其对应的变量名分别是 City1 跟 City2。
2. `replace City1 = City1 + "省"`，对分离后的变量 City1 我们要给他补上“省”的行政区后缀。比如把分离出来的“台湾”二字补全成“台湾省”。

同理，我们也可以通过截取的方式提取省与市。首先可以 `gen province1 = substr(City,1,9)`，再 `gen city1 = substr(City,10,9)`。同样也可以得到上面的结果。需要注意的是，在 `substr()` 函数下，1 个汉字对应 3 个占位。

当然，你可能已经想到了一个比较关键的点：由于我国幅员辽阔，地大物博，省级行政单位并不完全是以省结尾，市级行政单位也非都以市结尾。省级单位特殊情况诸如新疆维吾尔自治区；市级单位特殊情况诸如内蒙古自治区锡林郭勒盟；此外还应当考虑四个直辖市。所以通过 `split` 以及 `substr()` 进行分割的情况仅仅适用于整齐规律的文本变量，而无法适用所有情况。能够这个想法是非常棒的，但是**我预判了你的预判**，我将会在后面介绍使用正则表达的方法从长文本中提取 CEO 籍贯信息的方法。

3.6.2.2 文字变量的其他处理

通过 `help string functions`，我们可以查看 Stata 所支持的所有能够用于处理文字变量的函数。我在这里介绍几个可能有用的。

（1）更改大小写

```
*- do 文件的内容
dis lower("ASejksjdlwASD")

dis upper("sadhASDkSss")
```

将上述 do 文件的内容复制粘贴至你的 do 文件，选中后按 Ctrl+D，会在 Stata 的结果窗口显示如下内容：

```
**************************************************************************************************
. dis lower("ASejksjdlwASD")
asejksjdlwasd

. dis upper("sadhASDkSss")
SADHASDKSSS
**************************************************************************************************
```

（2）测量文本长度

```
*- do 文件的内容
dis length("汉字")

dis length("English ")
```

将上述 do 文件的内容复制粘贴至你的 do 文件，选中后按 Ctrl+D，会在 Stata 的结果窗口显示如下内容：

```
**************************************************************************************************
. dis length("汉字")                    // 一个汉字长度为 3
6

. dis length("English ")                // 一个字母长度为 1，且空格为 1
8
**************************************************************************************************
```

（3）测量文本个数

```
*- do 文件的内容
dis wordcount("汉字 English ")
```

将上述 do 文件的内容复制粘贴至你的 do 文件，选中后按 Ctrl+D，会在 Stata 的结果窗口显示如下内容：

```
**************************************************************************************************
. dis wordcount("汉字 English ")        // 按空格分割
2
**************************************************************************************************
```

（4）匹配文本是否出现

```
*- do 文件的内容
dis strmatch("xxx出生在中国湖北省潜江市", "潜江")
dis strmatch("xxx出生在中国湖北省潜江市", "*潜江*")

dis strmatch("xxx出生在中国台湾省台北市", "台北市*")
dis strmatch("xxx出生在中国台湾省台北市", "*台北市*")

dis strmatch("Stata", "s")
dis strmatch("Stata", "s")
dis strmatch("Stata", "S")
dis strmatch("Stata", "S*")
```

将上述 do 文件的内容复制粘贴至你的 do 文件，选中后按 Ctrl+D，会在 Stata 的结果窗口显示如下内容：

```
**********************************************************************************************
. dis strmatch("xxx出生在中国湖北省潜江市", "潜江")
0

. dis strmatch("xxx出生在中国湖北省潜江市", "*潜江*")
1

. dis strmatch("xxx出生在中国台湾省台北市", "台北市*")
0

. dis strmatch("xxx出生在中国台湾省台北市", "*台北市")
1

. dis strmatch("Stata", "s")
0

. dis strmatch("Stata", "s")
0

. dis strmatch("Stata", "S")
0

. dis strmatch("Stata", "S*")
1
**********************************************************************************************
```

不是很推荐使用这个方法匹配文本是否出现。你可以发现 Stata 的文本匹配机制非常笨，比如我直接用“潜江”两个字在“xxx 出生在中国湖北省潜江市”中进行匹配，Stata 的返回结果是 0，因为不匹配。但是当你在“潜江”两个字的前后分别加上一个通配符 * 时，就能匹配上。也就是说，Stata 的 `strmatch()` 函数在匹配一段字符是否在一串长文本中出现时，需要加上通配符才行，不然无法匹配。

（5）去除文本中的空格

```
*- do 文件的内容
dis " 我还 在漂泊  你是错 过的烟火 "

// 去除两端空格
dis strtrim(" 我还 在漂泊  你是错 过的烟火 ")

// 去除左边空格
dis strltrim(" 我还 在漂泊  你是错 过的烟火 ")

// 去除右边空格
dis strrtrim(" 我还 在漂泊  你是错 过的烟火 ")

// 去掉中间空格
dis stritrim(" 我   还 在  漂泊  你  是错 过的烟火 ")
help stritrim()  // 将中间若干空格压缩成一个空格

```

```
// 若要去掉所有空格
dis subinstr(" 我   还 在  漂泊  你  是错 过的烟火 "," ","",.)
help subinstr()  // 点表示替换所有空格

dis subinstr(" 我   还 在  漂泊  你  是错 过的烟火 "," ","",2)

dis subinstr(" 我   还 在  漂泊  你  是错 过的烟火 "," ","",4)
```

将上述 do 文件的内容复制粘贴至你的 do 文件，选中后按 Ctrl+D，会在 Stata 的结果窗口显示如下内容：

```
**********************************************************************************************
. dis " 我还 在漂泊  你是错 过的烟火 "
 我还 在漂泊  你是错 过的烟火

. // 去除两端空格
. dis  strtrim(" 我还 在漂泊  你是错 过的烟火 ")
我还 在漂泊  你是错 过的烟火

. // 去除左边空格
. dis  strltrim(" 我还 在漂泊  你是错 过的烟火 ")
我还 在漂泊  你是错 过的烟火

. // 去除右边空格
. dis  strrtrim(" 我还 在漂泊  你是错 过的烟火 ")
 我还 在漂泊  你是错 过的烟火

. // 去掉中间空格
. dis  stritrim(" 我   还 在  漂泊  你  是错 过的烟火 ")
 我 还 在 漂泊 你 是错 过的烟火

. // 若要去掉所有空格
. dis subinstr(" 我   还 在  漂泊  你  是错 过的烟火 "," ","",.)
我还在漂泊你是错过的烟火

. dis subinstr(" 我   还 在  漂泊  你  是错 过的烟火 "," ","",2)
我  还 在  漂泊  你  是错 过的烟火

. dis subinstr(" 我   还 在  漂泊  你  是错 过的烟火 "," ","",4)
我还 在  漂泊  你  是错 过的烟火
**********************************************************************************************
```

注意，以上 dis 后面的部分均可以用于配合 gen 生成新的变量。

3.6.3 正则表达

3.6.3.1 什么是正则表达

来看看 ChatGPT 对正则表达的描述：正则表达式（Regular Expression，简称为 Regex 或 RegExp）是一种用来匹配、搜索、替换文本模式的工具。它是一种强大而灵活的字符串处理技术，常用于文本处理、搜索引擎、数据清洗、编程语言中的字符串操作等领域。

正则表达式由一系列字符和特殊字符组成，用来描述字符串的模式。它可以帮助您执行诸如以下任务的操作：

- 匹配：正则表达式可以用来检查一个字符串是否与特定模式相匹配。
- 搜索：正则表达式可以在文本中搜索符合特定模式的子字符串。
- 替换：正则表达式可以用来在文本中查找并替换特定模式的内容。
- 提取：正则表达式可以从文本中提取出满足特定模式的部分。
- 验证：正则表达式可以用来验证用户输入是否符合特定格式要求。
- 分割：正则表达式可以根据特定模式将文本分割成不同的部分。

正则表达式的语法相对复杂，但一旦掌握，可以极大地提高文本处理的效率。不同编程语言和工具对正则表达式的支持有所不同，但基本的正则表达式语法通常是类似的。如果您想要学习和应用正则表达式，可以查阅相关的教程和资料，逐步了解其基本概念和语法规则。

好的，感谢 ChatGPT 的精彩解答。正如大家所看到的，正则表达式一个非常牛 X 的工具。在 Stata 中也有一些支持正则表达工具，但我个人觉得不如 Python 的 `re` 模块好用。

3.6.3.2 正则表达初印象

输入以下内容，你可以观察一下所有运行出来的结果。其中不理解的符号可以参考：菜鸟教程-正则表达元字符。

```
*- do 文件的内容
clear
input str10 String
    "abc"
    "ab"
    "aa"
    "abcd"
    "aad"
    "aab123"
    "cdf12345"
    "123"
    "Abc"
end

gen Number1 = ustrregexs(0) if ustrregexm(String,"[0-9]")
gen Number2 = ustrregexs(0) if ustrregexm(String,"[0-9]+")
gen Number3 = ustrregexs(0) if ustrregexm(String,"[0-9]*")
gen Number4 = ustrregexs(0) if ustrregexm(String,"[0-9]?")

gen Number5 = ustrregexs(0) if ustrregexm(String,"[0-9]$")
gen Number6 = ustrregexs(0) if ustrregexm(String,"[0-9]+$")
gen Number7 = ustrregexs(0) if ustrregexm(String,"[0-9]*$")
gen Number8 = ustrregexs(0) if ustrregexm(String,"[0-9]?$")
```

```
24
25 gen Number9 = ustrregexs(0) if ustrregexm(String,"^[0-9]")
26 gen Number10 = ustrregexs(0) if ustrregexm(String,"^[0-9]+")
27 gen Number11 = ustrregexs(0) if ustrregexm(String,"^[0-9]*")
28 gen Number12 = ustrregexs(0) if ustrregexm(String,"^[0-9]?")
```

3.6.3.3 正则表达常用操作

```
1  *- do 文件的内容
2  clear
3  input str16 String
4      "ab"
5      "AA"
6      "ABcd"
7      "aab123"
8      "AA133"
9      "Cdf12345"
10     "123"
11     "错过的烟火"
12     "红颜如霜123"
13 end
14
15 // 提出所有数字
16 gen Number = ustrregexs(0) if ustrregexm(String,"\d+")
17
18 // 提出所有小写字母
19 gen LetterL = ustrregexs(0) if ustrregexm(String,"[a-z]+")
20
21 // 提出所有大写字母
22 gen LetterU = ustrregexs(0) if ustrregexm(String,"[A-Z]+")
23
24 // 提出所有字母
25 gen Letter = ustrregexs(0) if ustrregexm(String,"[a-zA-Z]+")
26
27 // 提出所有汉字
28 gen Character = ustrregexs(0) if ustrregexm(String,"[\u4e00-\u9fa5]+")
```

其余的，诸如匹配提取身份证、邮政编码、网址，以及邮箱等操作所需要的正则表达式，可以参考：菜鸟工具-正则表达式在线测试。

3.6.3.4 正则表达实战示例

从 CSMAR 的相关数据中利用正则表达式提取上市公司 CEO 的籍贯信息。从我的 GitHub 仓库：StataTutorialAppendixData 找到文件夹 AppendixData2。其中，CEONativePlace.dta 是我在以下示例中使用的数据。

```
1 *- do 文件的内容
2 use CEONativePlace.dta, clear
3 compress
4
```

```
// 提取省份
gen Province=ustrregexs(0) if ustrregexm(NativePlace,".*省|.*自治区|.*市|.*特别行政区")

// 提取所在地级市
gen City = ustrregexs(2) if ustrregexm(NativePlace,"(.*省|.*自治区)?(.*市|.*自治州|.*地区|.*盟)")
```

将上述 do 文件的内容复制粘贴至你的 do 文件，选中后按 Ctrl+D，会在 Stata 的结果窗口显示如下内容：

```
*************************************************************************************************
. list in 1/5

     +-------------------------------------------------------------------+
     |  Stkcd   Year     Name        NativePlace   Province       City |
     |-------------------------------------------------------------------|
  1. | 000001   2020   胡跃飞       江西省吉安市     江西省      吉安市 |
  2. | 000005   2020   郑列列             广东省     广东省             |
  3. | 000009   2020   陈政立       广东省深圳市     广东省      深圳市 |
  4. | 000016   2020   刘凤喜   黑龙江省七台河市   黑龙江省   七台河市 |
  5. | 000016   2020    周彬             海南省     海南省             |
     +-------------------------------------------------------------------+
*************************************************************************************************
```

其中，变量 NativePlace 是数据中的原始字段，变量 Province 为我提取出来的省份信息，变量 City 为我提取出来的城市信息。**当然**，关于这里更详细的说明，可以参考我公众号的一篇推文：Stata-正则表达（regular expression）。以及我的 B 站对该点的详细解读：【Stata】正则表达提取 CEO 籍贯信息。

第 4 章 Stata 数据统计

4.1 类别变量统计

类别变量即包含多种分类的变量，比如性别就是一个典型的类别变量，有男性跟女性。在有些实证论文中，我们会使用到微观调查数据。在这些数据中，由于大多数调查问卷都是以选择题的方式进行填写，所以大多数的变量都是类别变量。在这种情况下我们就非常有必要了解有关类别变量的统计方法。

4.1.1 类别数目统计

我们可以使用 tab 命令来初步了解类别变量的分布情况。

```
*- do 文件的内容
sysuse auto.dta, clear
tab foreign
```

将上述 do 文件的内容复制粘贴至你的 do 文件，选中后按 Ctrl+D，会在 Stata 的结果窗口显示如下内容：

```
********************************************************************************************************
. tab foreign

   Car type |      Freq.     Percent        Cum.
------------+-----------------------------------
   Domestic |         52       70.27       70.27
    Foreign |         22       29.73      100.00
------------+-----------------------------------
      Total |         74      100.00
********************************************************************************************************
```

可以发现，在我们的数据中，我们使用 tab 命令对类别变量 foreign 进行了统计。其中，该变量的总样本数（不包含缺失值）为 74；类别为 Domestic 的样本有 52 个，占比 70.27%；类别为 Foreign 的样本有 22 个，占比 29.73%。

注意，对文本变量而言，也可以使用类别变量进行统计，这样统计出来的结果就是各种文本的数量与占比情况；对于数值型变量，如果把它当作类别变量进行统计，统计出来的结果就是各个数字的数量与占比情况。在本例中，变量 foreign 是一种“文本-数值”型变量，所以虽然在数据中其背后真实的值是数字，但使用 tab 命令时仍以文本进行列示。

4.1.2 分组统计量

进一步地，我们可能还需要用一些其他的方法来分组统计类别变量中，不同组别的一些特征，比如分组描述性统计：

```
*- do 文件的内容
sysuse auto.dta, clear
by foreign: sum price

// 等价于
sum price if foreign == 0
```

```
sum price if foreign == 1
```

将上述 do 文件的内容复制粘贴至你的 do 文件，选中后按 Ctrl+D，会在 Stata 的结果窗口显示如下内容：

```
*******************************************************************************************************
. by foreign: sum price

-------------------------------------------------------------------------------------------------------
-> foreign = Domestic

    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
       price |         52    6072.423    3097.104       3291      15906

-------------------------------------------------------------------------------------------------------
-> foreign = Foreign

    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
       price |         22    6384.682    2621.915       3748      12990
*******************************************************************************************************
```

当然也还有很多其他的命令，但是我一般比较喜欢用自由度更高的，比如配合 egen 计算各种其他指标之类的，然后再手动赋值粘贴到论文中。

4.2 连续型数值变量统计

对连续型数值变量的统计主要就是做描述性统计跟导出描述性统计的结果，具体的例子。

```
*- do 文件的内容
// 首先下载sum2docx命令，这个命令对中文比较友好，不会乱码
sysuse auto.dta, clear

rename price    价格
rename headroom  头顶空间
rename length    车长
rename weight   车重

local Variables 价格 头顶空间 车长 车重
sum2docx `Variables' using 描述性统计表.docx, ///
    replace stats(N mean sd median p25 p75 min max) ///
    title("描述性统计表1") font("宋体",12,"black") ///
    pagesize(A4)

// 设置横向页面
sum2docx `Variables' using 描述性统计表2.docx, ///
   replace stats(N mean sd median p25 p75 min max) ///
   landscape title("描述性统计表2") font("宋体",12,"black") ///
```

```
20    pagesize(A4)
```

运行完后会在你的当前工作路径下生成两个 .docx 文件，记得选中后一起运行，/// 表示换行符，只选一半是运行不出结果的。其实在实际的使用过程中，我们不太需要设置表格的大小以及字体的内容，完全可以导出表格之后自己再手动调整，唯一需要设置的就是选择导出什么样的统计量。设定统计量的关键选项是 stats，通过 help sum2docx 可以查看 stats 选项所支持的全部统计量，自由设定即可。

4.3 文本型变量统计

关于对文本型变量的统计，在这里需要介绍我的一篇放置于 arXiv 的工作论文，以及一个 Stata 包。放置在 arXiv 的工作论文是：Comparing with Python: Text Analysis in Stata [13]，使用的 Stata 包是：ONETEXT: Stata module to perform simple Chinese text analysis [12]，可以通过 ssc install onetext 进行下载。

与 Stata 有关的文本分析可以参见我公众号的一篇推文：「Stata」词频统计下的数字化转型，对应的视频讲解是：「Stata」利用词频统计方法衡量企业年报 MD&A 中的数字化转型。

接下来我介绍一下用 Stata 进行文本分析的一些操作要点。这部分的内容在我之前的一个文本分析讲座中的“Word Frequency”部分有提及到，对应的 B 站视频是：「公开讲座」文本分析实录：原理、应用与操作，对应我的一个 GitHub 仓库地址是：TA-Presentation 中的“Part I. Word frequency”中的内容。

4.3.1 读入文本文件

假设，你现在有一个文件夹叫做 myFolder，其中 Code.do 存放了你的处理代码，resources 文件夹存放了其他文本文件。与 Code.do 同级的一个名为 resources 的文件夹中的 files 文件夹，以 .txt 的文件格式保存了一些上市公司年报的 MD&A（管理层讨论与分析）的文本文件。resources 中的一个名为 PosDict.txt 的文件存储了所有的积极情绪词语，每个词语占一行；resources 中的一个名为 NegDict.txt 的文件存储了所有的积极情绪词语，每个词语占一行。总的来说文件排布如下：

```
*****************************************************************************************
resources文件夹
    files文件夹
        MD&A1.txt
        MD&A2.txt
        ...
    PosDict.txt
    NegDict.txt
Code.do文件
*****************************************************************************************
```

那么，在这种情况下，结合我前面讲的 local 以及循环的方式，你可以参考以下代码循环写入文件。

```
1  *- do 文件的内容
2  local txtFiles : dir "resources/files" files "*.txt"
3
4  local N = 1
5  foreach singleFile in `txtFiles' {
6
7    import delimited "resources/files/`singleFile'", delimiter("shutterzor", asstring) varnames(nonames)
       encoding(UTF-8) clear
8
```

```
  gen stkcd = ustrregexs(0) if ustrregexm("`singleFile'", "\d+")
  gen year = ustrregexs(0) if ustrregexm("`singleFile'", "_\d+-")
  replace year = substr(year, 2, 4)

  tempfile file`N'
  save "`file`N''"
  local N = `N' + 1
}

use "`file1'", clear
forvalues fileNum = 2/5 {
  append using "`file`fileNum''"
}
rename v1 content
save "MDAText.dta", replace
```

4.3.2 统计词频

可以通过如下方式分别统计积极情绪与消极情绪的词语频率：

```
*- do 文件的内容
use MDAText.dta, clear

 *- 积极词汇
 preserve
   import delimited "resources/PosDict.txt", encoding(UTF-8) clear
   levelsof v1, local(posWords)
   local posNum = _N
 restore

 local tempCount = 1
 foreach posWord of local posWords{
   quietly onetext content, k("`posWord'") m(count) g(posWord`tempCount')
    local tempCount = `tempCount' + 1
   dis %4.2f (`tempCount'-1)/`posNum'*100
 }

   *- 计算积极情绪，并删除无用变量
   egen posSentiment = rowtotal(pos*)
   keep stkcd year content posSentiment

 *- 消极词汇
 preserve
   import delimited "resources/negDict.txt", encoding(UTF-8) clear
   levelsof v1, local(negWords)
   local negNum = _N
 restore

 local tempCount = 1
```

```
foreach negWord of local negWords{
  quietly onetext content, k("`negWord'") m(count) g(negWord`tempCount')
   local tempCount = `tempCount' + 1
  dis %4.2f (`tempCount'-1)/`negNum'*100
}

  *- 计算积极情绪，并删除无用变量
  egen negSentiment = rowtotal(neg*)
  keep stkcd year posSentiment negSentiment

*- 保存结果
save "sentimentResult.dta", replace
```

最后，虽然我是用的情绪词语的词频统计举的例子，但是这种方法是可以拓展的，你可以用来统计任意文本的任意词语的词频，比如数字化转型、环境规制等。

第 5 章 Stata 图形绘制

在 Stata 中，绘图的代码一般是 `graph xxx` 或者 `graph twoway xxx`，其中 `xxx` 一般用于指代图形的类别，对单一变量进行画图操作时一般使用 `graph xxx`，并且可以简写为 `gr xxx`；`graph twoway xxx` 一般可以简写为 `tw xxx`。

通过 `help graph` 我们可以查看 Stata 支持的所有图像。需要注意的是，我在后面的教程中将只会写明有用的点，关于图片如何保存，我的建议是直接点击图片导出。考虑到在 Stata 中，大部分图片的基本命令是一样的，所以我将以散点图的绘制为例，解释利用 Stata 绘图的一些基本逻辑。

5.1 什么是散点图

有请 ChatGPT 同学解释一下什么是散点图

散点图（Scatter Plot）是一种用于展示两个变量之间关系的数据可视化方法。它在二维平面上以点的形式表示数据集中的每个数据点，其中每个点的位置由两个变量的数值决定。其中一个变量通常被称为横轴或 X 轴，另一个变量被称为纵轴或 Y 轴。

散点图的主要作用是帮助我们观察和理解两个变量之间的关系，从而揭示出可能的趋势、模式或异常情况。以下是散点图的一些常见作用：

1. 关联分析：散点图可以用来观察两个变量之间的关联程度。如果数据点呈现出明显的趋势，可能存在线性或非线性的关系。
2. 趋势检测：通过观察散点图中的趋势，我们可以判断是否存在某种趋势，如上升趋势、下降趋势或周期性变化。
3. 异常检测：散点图可以帮助我们发现与主要趋势不一致的异常点，这些点可能代表数据中的异常情况或离群值。
4. 聚类观察：在散点图中，如果存在不同的簇或群集，可能表明数据可以分成几个不同的类别。
5. 模式识别：散点图可以帮助我们发现数据中的重复模式，从而揭示出数据集的一些特征。
6. 相关性分析：散点图可以用来观察两个变量之间的相关性，即它们在何种程度上随着彼此的变化而变化。

综上所述，散点图是一种简单而有效的数据可视化方法，可以帮助我们从数据中提取有关两个变量之间关系的有用信息，进而支持数据分析、决策和预测。

5.2 绘制一副散点图

在 Stata 中，绘制两个变量之间散点图的基本命令是：`tw scatter y x`，举个例子：

```
*- do 文件的内容
sysuse auto.dta, clear

tw scatter price weight
```

绘图结果如 5.1 所示。

类似地，通过将 `tw` 后面的 `scatter` 选项指定为其他选项，你可以画出不同的二维图。常见的图形包括：折线图（`line`）、连线图（`connected`）、棒状图（`bar`）。请读者朋友自行探索。

进一步地，通过 `gr matrix` 命令，可以绘制一幅多变量散点图，如 5.2 所示。

```
*- do 文件的内容
sysuse auto.dta, clear

```

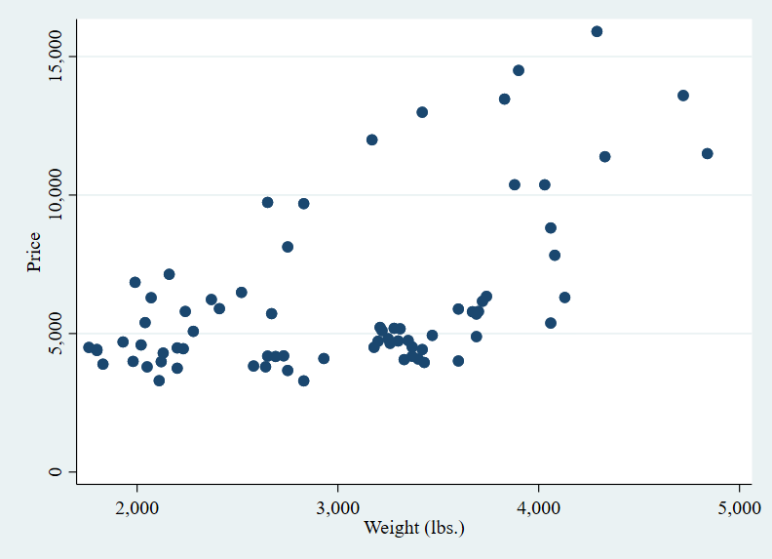

图 5.1: 散点图

```
4 graph matrix price-trunk
```

在这个命令当中，我们绘制了从变量 price 一直到变量 trunk 的两两组合之间的散点图，显然这个矩阵散点图是沿主对角线对称的。

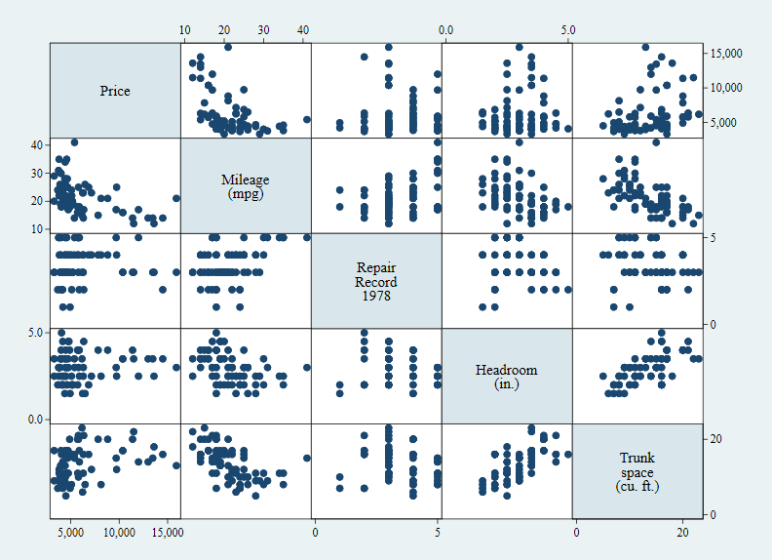

图 5.2: 矩阵散点图

5.3 散点图进阶

5.3.1 更换散点样式

在 Stata 中，其绘图命令具有很多进阶的选项。在这些选项当中又分为通用选项以及专有选项：通用选项是对其他画图方法也使用的选项，比如调整坐标轴、添加坐标轴标题，以及添加图片标题等；专有选项是仅对当前图像生效的选项。通过 help graph，我们可以依次点击进入某一绘图命令的帮助文件，通过查阅帮助文件，我们就可以完成对图片的细节修改。Stata 的帮助文件写的比较清晰，一般来说自己阅读可以找到答案。

以散点图为例，我们输入 `help graph`，点击进入 `graph twoway`，再点击 `scatter`，就可以看到散点图的帮助文件。这里面有着让人非常眼花缭乱的选项，说实话，我也没有把握搞清楚每一个选项，但是可以根据描述大致猜一猜，并不断地调整图像。比如，假设你现在需要改变散点图中点的外观，比如把点改成三角形这种。那么你就到选项的描述里面一个个找，可以发现在 `marker_options` 的描述里提到了：change look of markers (color, size, etc.)，那这个大概率就是我们需要的内容。点击一下蓝色的 `marker_options`，它的详细信息就自动跳转到了我们帮助文件浏览窗口的顶端。在这个文件里面可以发现有一个 `msymbol` 选项，能够改变 `marker` 的形状，我们再点进去，就发现了一些形状的样式命名。可以发现，`triangle`，可以简写为 `T`，也就是代表着三角形。

接下来我们可以在图形中验证一下，将绘图代码更改为：`tw scatter price weight, msymbol(T)`，绘制出来的图片如 5.3 所示。

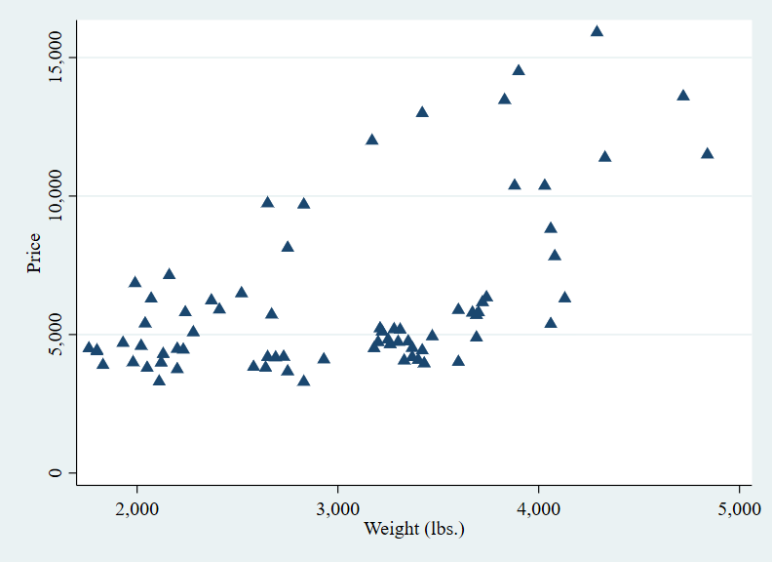

图 5.3: 散点图：更换散点样式

5.3.2 更换字体

你或许也注意到了，如果你使用的是默认的 Stata，绘制出来的图片的字体跟我的可能有不一样。所以本小节内容主要是说明如何在 Stata 中修改字体。更换字体的前提是你电脑上有这样的字体，Mac 跟 Windows 可能有一些区别，我使用的是 Windows 电脑。如判断电脑上有没有某种字体呢？很简单，打开 Word，Word 里面有的字体你就都能使用。

输入以下命令，可以查看不同字体的区别，绘图结果如 5.4 所示。**注意**，将 `tw scatter` 简写为 `scatter` 也一样可行。

```
*- do 文件的内容
sysuse auto.dta, clear

scatter mpg weight, title(`"The {stSerif:relationship between price} {stSans:and weight}"')subtitle(`"
    汽车价格{fontface "华文楷体":与重量}{fontface "宋体":之间的关系}"')
```

我把这个图片放大了一点，以便于你更仔细地观察。结合命令不难发现，对于标题 `title` 中的“The relationship between price”这结果单词，我使用的字体是“Times New Roman”，而对于“and weight”我使用的字体是“宋体”。对于副标题 `subtitle` 中的“汽车价格”四个字，我使用的是默认字体，对于“与重量”三个字我使用的是华文楷体，对于“之间的关系”五个字我使用的是“宋体”。你或许会问为什么对英文跟中文之间使用的宋体不一样？讲道理我只能提供一个看似合理的解释，就是 `stSans` 对中文有时候可能会失效。**同样地**，你可以将双引号之间的中文字体换成任意你电脑上有的，都能跑出来结果（应该）。

图 5.4: 散点图：更换图形内字体

5.3.3 其他修改

在上面我介绍了更改标题字体的方法。同样地，对于图形中的任意的文本，你都可以使用类似的方法进行更改。比如更改坐标轴的标题，你可以分别在 `xtitle` 和 `ytitle` 中做相应的设定。此外，Stata 也支持加粗或者斜体，比如在上述的代码中，`fontface "华文楷体":与重量` 用于将“与重量”三个字设置为华文楷体，你可以使用如下代码为“与重量”三个字增加斜体形式，以及把“之间的关系”这五个字加粗。

```
*- do 文件的内容
scatter mpg weight, title(`"The {stSerif:relationship between price} {stSans:and weight}"') ///
        subtitle(`"汽车价格{fontface "华文楷体":{it:与重量}}{fontface "宋体":{bf:之间的关系}}"')
```

此外，还可以自定义坐标轴范围等，不过一般用不着，Stata 的默认范围就看起来挺不错了。

5.3.4 更改绘图模板

你或许也发现了，Stata 的默认绘图模板不大好看，虽然有一些期刊不要求图形优美，但是你把图画好了，作为审稿人，我会更倾向于接收你的论文。通过 `help schemes`，你可以找到 Stata 所有支持的默认模板，在所有的绘图命令后面加上 scheme 选项，并指定一个模板，比如：scheme(s2mono)，你就可以将当前的模板更换为 `s2mono` 格式。你可以利用循环同时绘制多个模板下的图像，观察哪个好看，然后再决定用哪个。

对于中文期刊，默认黑白图片就好了。对于外文期刊，我也不建议你使用 Stata 自带的绘图模板，太朴素了。我建议你参考我很早以前在连享会发布的一篇推文：Stata 绘图：用 Stata 绘制一打精美图片-schemes，使用其中的 `white_tableau` 模板绘图，会非常好看。利用该模板绘制出来的成品图片参考我发表于 *Journal of Cleaner Production* 的论文：Conservation or revolution? The sustainable transition of textile and apparel firms under the environmental regulation: Evidence from China [11]，或者我的公众号推文：Stata-我的一些绘图代码。

5.4 绘制中国地图

除了一般的图形外，Stata 也可以用来绘制中国地图。用 Stata 绘制中国地图的详细方法可以参考：

1. 我的 B 站视频：【Stata】绘制一份完完整整的中国地图！。
2. 连享会推文：Stata：空间计量之用 spmap 绘制地图 。

5.5 手动绘制调节效应图

结合 `tw function` 的函数绘图功能，同样也可以在回归之后手动绘制调节效应的图片，具体可以参考

1. 我的 B 站视频：【Stata】手动绘制调节效应图。
2. 我的公众号推文：Stata-手动绘制调节效应图。

第 6 章 Stata 实证写作

本章节将重点介绍实证写作中常见的中介效应以及调节效应。进一步地，也会给出一份标准化的代码，以供后续使用时进行参考。

6.1 中介效应

6.1.1 什么是中介效应

有请 ChatGPT 同学回答这个问题：中介效应（Mediation Effect）是统计学和社会科学研究中的一个重要概念，用于解释一个变量与另一个变量之间关系的机制。中介效应发生在一个变量（中介变量）在解释或传递一个自变量（独立变量）对因变量（依赖变量）的影响时起到中介作用的情况。

中介效应通常通过以下步骤来解释：

1. 自变量（X）与中介变量（M）之间的关系：首先，研究者研究自变量（独立变量）与中介变量之间的关系，确定是否存在一个关联。这是确定中介效应是否可能存在的第一步。
2. 中介变量（M）与因变量（Y）之间的关系：然后，研究者研究中介变量（中介效应）与因变量（依赖变量）之间的关系，确定中介变量是否与因变量存在关联。
3. 自变量（X）与因变量（Y）之间的关系：接着，研究者研究自变量与因变量之间的关系，确定是否存在一个直接的关联。

如果以上步骤确认了中介变量在自变量与因变量之间的关系中发挥了作用，那么就可以得出存在中介效应的结论。中介效应表明，自变量通过影响中介变量，进而影响了因变量。它帮助我们理解背后的机制和途径，为研究结果提供更深刻的解释。

中介效应的分析通常涉及统计方法，如回归分析和路径分析。通过这些方法，研究者可以量化中介效应的大小，并确定其在整个关系中的作用程度。中介效应在很多领域都有应用，包括心理学、医学、社会科学、经济学等，帮助研究者深入理解变量之间的关系和作用机制。

6.1.2 中介效应三步法

6.1.2.1 三步法简介

检验中介效应最经典的方法就是三步法。我知道你想说江艇老师在《中国工业经济的》上发表的“因果推断经验研究中的中介效应与调节效应”一文指出了中介效应 xxxx，但是你听我说你先别急，先看看你所在领域的顶级、次顶级期刊是否全盘否定了中介效应，如果大家还在大量地使用中介效应，那你最好不要过于激进地否定这种方法。同样地，当你听闻了某一种因果推断的新方法，也不要过于激进地否定你现在所使用的固定效应模型，虽然两者可能有一定的交叉。**不要看不起这种方法看不惯那种模型，要结合自己专业领域的实际情况自己判断该方法是否合适**。

回到正题上，中介效应的三步检验法如下：

$$Y = cX + e_1 \tag{6.1}$$

$$M = aX + e_2 \tag{6.2}$$

$$Y = c'X + bM + e_3 \tag{6.3}$$

在上述公式中，方便起见我只展示了自变量（X）、因变量（Y）以及中介变量（M），省略了所有的控制变量以及固定效应。如公式所示，你可以看到，中介效应的三步检验法其实就是三个回归模型，也就是说你需要在 Stata 中分别对这三个模型进行回归，并判断系数及其显著性。

对于以上三个方程，当且仅当系数 a，b，c 都显著时，中介效应才成立。并且，在满足中介效应成立的前提下，系数 c' 也显著的话，则将其称为完全中介效应。

6.1.2.2 中介效应示例

举一个简单的例子来观察中介效应，如下：

```
*- do 文件的内容
sysuse auto.dta, clear

reg price weight
est store m1

reg mpg weight
est store m2

reg price weight mpg
est store m3

esttab m1 m2 m3, compress nogap
```

将上述 do 文件的内容复制粘贴至你的 do 文件，选中后按 Ctrl+D，会在 Stata 的结果窗口显示如下内容：

```
**************************************************************************************************
. esttab m1 m2 m3, compress nogap

------------------------------------------------
                (1)          (2)          (3)
              price          mpg        price
------------------------------------------------
weight         2.044***  -0.00601***     1.747**
              (5.42)     (-11.60)       (2.72)
mpg                                     -49.51
                                       (-0.57)
_cons         -6.707        39.44***    1946.1
              (-0.01)      (24.44)       (0.54)
------------------------------------------------
N                74           74           74
------------------------------------------------
t statistics in parentheses
* p<0.05, ** p<0.01, *** p<0.001
**************************************************************************************************
```

在上面这个例子中，我们的因变量是 $price$，自变量是 $weight$，中介变量是 mpg。根据之前章节对中介效应的介绍，c 对应的是模型（1）中 $weight$ 的系数，是显著的；a 对应的是模型（2）中 $weight$ 的系数，是显著的；c' 对应的是模型（3）中 $weight$ 的系数，是显著的；b 对应的是模型（3）中 mpg 的系数，是不显著的。所以在该模型中，中介效应是不成立的。

现在假设它们都是显著的，那么我们如何解释这种结果呢？答案是根据模型（1）而言，$weight$ 促进了 $price$；根据模型（2）而言，$weight$ 抑制了 mpg；根据模型（3）而言，mpg 抑制了 $price$。也就是说，在以 mpg 为中

介变量时，X 抑制了 M，同时 M 也进一步抑制了 Y，但是总的来说 X 是促进 Y 的，所以负负得正？这个就得结合你的故事去解释为什么会出现这种现象了。

当然，这个例子是比较荒诞的。如果你要是用中介效应，你起码得保证从理论上来说这个“X–>M–>Y”的传导是能够解释通的。

6.1.2.3 多中介变量导出

结合上面的例子你可能发现了，中介变量带入之后有概率检验不出来中介效应，所以你可能希望有一个简单的方法，能同时检验多个中介变量的中介效应，然后选一个中介效应显著的作为你故事中的机制。当然是没问题的，我在这里提供一份我自己使用的多中介变量导出中介效应结果的循环代码，如下。

```
*- do 文件的内容
********************* 导入数据 *******************
sysuse auto.dta, clear

******************* 设置关键信息 ******************
// 定义因变量
global Y price
// 定义自变量
global X length
// 定义中介变量
global ME weight turn headroom
// 定义控制变量
global CV rep78

************ 批量导出多个调节效应结果 **************
reg $Y $X $CV
est store main

foreach v in $ME {

  reg `v' $X $CV
   est store `v'2

  reg $Y $X `v' $CV
   est store `v'3

}

foreach v in $ME {

  reg2docx main `v'2 `v'3 using ME`v'.docx, ///
    scalars(N r2(%9.3f) r2_a(%9.2f)) ///
    star(* 0.1  ** 0.05  *** 0.01) ///
    b(%9.3f) t(%7.2f) replace ///
    order($X $MO $CV)
}
```

在上述例子中，你只需要将你自己的各种变量填入“设置关键信息”环节的全局 Macro 之后，再全部选中后同时运行，就能一次性导出多个中介变量的中介效应回归结果。**需要注意的是**，如果你需要自定义模型，比

如不想使用 `regress` 回归命令，而是想使用 `xtreg` 或者 `reghdfe`，那么你可以将对应的 `reg` 所在的行，进行相应地修改，以达到完全自定义的目的。

6.1.3 中介效应的其他检验方法

关于中介效应的其他检验方法可以依次参考以下推文：

1. 连享会推文：Stata+R: 一文读懂中介效应分析。
2. 我的 B 站视频 1 号：【Stata】中介效应及其进阶操作！。
3. 我的 B 站视频 2 号：「Stata」中介效应检验 - 带固定效应 Sobel 与 Bootstrap

6.2 调节效应

6.2.1 什么是调节效应

有请 ChatGPT 同学回答这个问题：调节效应（Moderation/Interaction Effect）是统计学和社会科学研究中的一个重要概念，用于描述在某种情况下，一个变量对另一个变量之间关系的影响程度是否取决于第三个变量的不同水平。换句话说，调节效应发生在某个变量（调节变量）在两个其他变量（自变量和因变量）之间的关系上产生影响的情况。

调节效应通常涉及三个主要变量：

1. 自变量（X）：这是独立变量，研究者感兴趣的因素之一。
2. 因变量（Y）：这是依赖变量，受自变量影响的结果。
3. 调节变量（M）：这是一个第三个变量，研究者想要了解其如何影响自变量与因变量之间关系的因素。

在分析调节效应时，主要考虑的是在不同调节变量水平上，自变量对因变量的影响是否有所不同。如果在调节变量的不同水平上，自变量对因变量的影响呈现出不同的模式，那么就可以认为存在调节效应。

调节效应的分析通常使用交互项（interaction term）来量化。交互项是指将自变量和调节变量相乘而得到的新变量，用于观察调节变量对自变量与因变量之间关系的影响。

调节效应的理解可以帮助研究者更深入地理解变量之间的复杂关系。它有助于揭示在什么条件下关系会改变，以及不同因素如何相互作用来影响研究结果。调节效应在许多领域都有应用，如心理学、医学、社会科学等，帮助研究者更准确地解释变量之间的关系。

6.2.2 调节效应示例

举一个简单的例子来观察调节效应，如下：

```
*- do 文件的内容
sysuse auto.dta, clear

reg price weight
est store m1

reg price c.weight##c.length
est store m2

esttab m1 m2, compress nogap
```

将上述 do 文件的内容复制粘贴至你的 do 文件，选中后按 Ctrl+D，会在 Stata 的结果窗口显示如下内容：

```
**********************************************************************************************
```

```
. esttab m1 m2, compress nogap

------------------------------------
                  (1)          (2)
                price        price
------------------------------------
weight          2.044***     -0.967
               (5.42)       (-0.28)
length                      -174.6**
                            (-2.98)
c.weight~h                   0.0286
                             (1.74)
_cons          -6.707       25227.9**
              (-0.01)        (2.65)
------------------------------------
N                  74           74
------------------------------------
t statistics in parentheses
* p<0.05, ** p<0.01, *** p<0.001
********************************************************************************************
```

在上面的例子中，对于模型（1）而言，我们的自变量 *weight* 对因变量 *price* 的影响是显著的，说明汽车重量越大价格越高；对于模型（2）而言，我们的交互项 $weight \times length$[1] 的系数是 0.0286，是不显著的。所以我们不能说 *length* 在 *weight* 与 *price* 之间存在调节效应。

那么，如果这些系数都是显著的，我们应该如何解释这种调节效应呢？可以看到，假设系数都是显著的，那么我们的 *weight* 促进了 *price*，同时交互项 $weight \times length$ 的系数也是正（与 *weight* 对 *price* 的影响同号），所以 *length* 的存在进一步强化了 *weight* 对 *price* 的促进作用。如果交互项的系数与主（自）变量的系数相反，则削弱了自变量与因变量之间的作用。

6.2.3 多调节变量导出

与多中介变量的导出相同，调节效应也不总是显著的，往往需要验证多种变量的调节效应，想好多种机制，然后挑一种显著的结果进行解释，所以我也提供一份可以同时导出多个变量调节效应结果的循环模板供大家使用，如下。

```
*- do 文件的内容
********************* 导入数据 *******************
sysuse auto.dta, clear

****************** 设置关键信息 ******************
// 定义因变量
global Y price
// 定义自变量
global X length
// 定义中介变量
global MO weight turn headroom
```

[1]在 Stata 数据处理章节的因子变量小节中有提到：`##`，用来表示两个变量的交互项及其单独项，比如 `x1##x2` 等价于 `x1*x2 x1 x2`

```
// 定义控制变量
global CV rep78

************ 批量导出多个调节效应结果 **************
reg $Y $X $CV
est store main

foreach v in $MO {

  reg $Y c.($X )##(c.`v') $CV
   est store `v'

}

reg2docx main $MO using MO.docx, ///
  scalars(N r2(%9.3f) r2_a(%9.2f)) ///
  star(* 0.1 ** 0.05 *** 0.01) ///
  b(%9.3f) t(%7.2f) replace ///
  order($X $MO $CV)
```

同样地，你只需要将你自己的各种变量填入“设置关键信息”环节的全局 Macro 之后，再全部选中后同时运行，就能一次性导出多个调节变量的调节效应回归结果。**需要注意的是**，如果你需要自定义模型，比如不想使用 `regress` 回归命令，而是想使用 `xtreg` 或者 `reghdfe`，那么你可以将对应的 `reg` 所在的行，进行相应地修改，以达到完全自定义的目的。

6.2.4 调节效应绘图

绘制调节效应图可以参考以下推文：

1. 连享会推文：Stata - 图示调节效应。
2. 我的 B 站视频 1 号：【Stata】调节效应及其进阶操作——纳入调节项后主效应系数相反怎么办？。
3. 我的 B 站视频 2 号：【Stata】手动绘制调节效应图。
4. 我的公众号推文：Stata-手动绘制调节效应图。

第 7 章 结语

“感谢信任，越来越诚惶诚恐，诚实地感到惶恐，恐怕自己名不符实，说名不符实并不是谦虚，是真的有这种感觉，运气呢并非成就，命运之手把我托举到所不配有的高度，让人飘然，让人晕眩，最终让人诚惶诚恐，一直以来我都觉得自己不过像一颗渺小的尘埃，风把我带向我从未向往的高处，相信有一天它也会把我轻放在神秘莫测的他处，我们不过在借来的时间中生活，你所暂时保管的精彩，并不真正属于你，有一天，你必须交给下一位接棒者，并希望他能做得更加精彩，再次感谢。”

——**罗翔**

2021 百大 UP 主颁奖获奖感言

花费了小半个月的时间，终于写完了我的《Stata Tutorial》（相对来说比较清晰易懂且基本无害的 Stata 入门教程）一书的第一个版本。它一定是不完美的，但可以持续地更新。这本教程很轻，轻到 15 克的，容量为 16G 的 U 盘足以装载其近万份；同时这本教程也很重，它的背后是我近三年对 Stata 的不完美学习，也承载了许多来自小伙伴们的支持。最后，如果本教程对各位小伙伴们起到了帮助，也欢迎各位通过以下方式对我进行打赏。赞助名单将会以附录形式放置于附录 B 中，并随本书版本更新而发行。

图 7.1: One cup of coffee.

参考文献

[1] Adrian Colin Cameron, Pravin K Trivedi, et al. *Microeconometrics using stata*. Vol. 2. Stata press College Station, TX, 2010.

[2] Nicholas J Cox. "A brief history of Stata on its 20th anniversary". In: *The Stata Journal* 5.1 (2005), pp. 2–18.

[3] William W. Gould and Nicholas J. Cox. *When was Stata first released? When were later versions released?* https://www.stata.com/support/faqs/resources/history-of-stata/, Last accessed on 2023-8-12. 2021.

[4] Roberto G Gutierrez. "Stata". In: *Wiley Interdisciplinary Reviews: Computational Statistics* 2.6 (2010), pp. 728–733.

[5] Ulrich Kohler and Frauke Kreuter. *Data analysis using Stata*. Stata press, 2005.

[6] Michael N Mitchell. *A visual guide to Stata graphics*. Stata Press, 2008.

[7] H Joseph Newton. "A conversation with William Gould". In: *The Stata Journal* 5.1 (2005), pp. 19–31.

[8] LLC StataCorp. "Stata: software for statistics and data science". In: *Stata longitudinal-data/panel-data reference manual release* 16 (2019).

[9] LLC StataCorp. *When was Stata first released? When were later versions released?* https://www.stata.com/new-in-stata/, Last accessed on 2023-8-12. 2023.

[10] Jonathan AC Sterne. *Meta-analysis in Stata: an updated collection from the Stata Journal*. StataCorp LP, 2009.

[11] Shutter Zor. "Conservation or revolution? The sustainable transition of textile and apparel firms under the environmental regulation: Evidence from China". In: *Journal of Cleaner Production* 382 (2023), p. 135339.

[12] Shutter Zor. "ONETEXT: Stata module to perform simple Chinese text analysis". In: (2022).

[13] Xiangtai Zuo. "Comparing with Python: Text Analysis in Stata". In: *arXiv preprint arXiv:2307.10480* (2023).

附录 A Stata 发展时间线

本附录包含了 Stata 从创立之初至今的版本发布情况。本部分内容翻译自 Wikipedia，受编者能力限制，不一定完全准确。

1985/01 更新：版本 1.0 正式发布
① 首次发布
② 44 条命令

1985/02 更新：版本 1.1 正式发布
① 修复已知 bug

1985/05 更新：版本 1.2 正式发布
① 新的菜单系统
② 更好的在线帮助
③ keep 命令发布

1985/08 更新：版本 1.3 正式发布
① Stata/图形
② program 命令

1986/08 更新：版本 1.4 正式发布
① 新的文档
② 新的格式支持

1987/02 更新：版本 1.5 正式发布
① anova 命令
② logit，probit 命令

1988/06 更新：版本 2.0 正式发布
① 新的图形
② 字符型变量
③ 生存分析：Cox and Kaplan-Meier
④ 逐步回归

1990/09 更新：版本 2.1 正式发布
① 字节型变量
② 因子分析
③ ado-files
④ reshape 命令

1992/03 更新：版本 3.0 正式发布
① logistic、ologit、oprobit、mlogit 命令
② tobit、cnreg、rreg、qreg、weibull、ereg 命令

③ epitab 命令
④ pweights 命令

1993/08 更新：版本 3.1 正式发布
① mvreg、sureg、heckman、nlreg、areg、canon 命令
② nbreg 命令
③ 约束线性回归 constrained linear regression
④ ml 命令
⑤ codebook 命令

1995/01 更新：版本 4.0 正式发布
① xtreg 命令
② glm 命令

1996/10 更新：版本 5.0 正式发布
① xtgee、xtptobit 命令
② prais、newey、intreg 命令
③ 生存分析命令
④ fracpoly 命令
⑤ st 扩展

1999/01 更新：版本 6.0 正式发布
① 网络资源
② 新的 ml 命令
③ 时间序列操作
④ arima、arch 命令
⑤ 重述 st 命令

2000/12 更新：版本 7.0 正式发布
① frailty 命令
② xtabond 命令
③ 聚类分析
④ nlogit 命令
⑤ roc 命令
⑥ SMCL

2003/01 更新：版本 8.0 正式发布
① 图形
② 扩展图形用户界面，所有命令都有对话框
③ manova 命令
④ 更多的调查
⑤ 更多的时间序列（VARs、SVARs）
⑥ 更多 GLLAMM internalization

2003/07 更新：版本 8.1 正式发布

① 更新 ml 命令

2003/10 更新：版本 8.2 正式发布
① 图形界面变更

2005/04 更新：版本 9.0 正式发布
① 加入 mata 矩阵编程语言
② survey features
③ linear mixed models
④ multinominal probit models

2005/09 更新：版本 9.1 正式发布
①

2006/04 更新：版本 9.2 正式发布
①

2007/06 更新：版本 10.0 正式发布
① 图形编辑器
② 具有复杂嵌套误差成分的 logistic 和 Poisson 模型

2008/08 更新：版本 10.1 正式发布
①

2009/07 更新：版本 11.0 正式发布
① 因子变量
② margins 命令
③ multiple imputation

2010/06 更新：版本 11.1 正式发布
①

2011/03 更新：版本 11.2 正式发布
①

2011/07 更新：版本 12.0 正式发布
① 自动化的内存管理
② 结构方程模型

2012/01 更新：版本 12.1 正式发布
①

2013/06 更新：版本 13.0 正式发布
① 长文本
② 处理效应

2013/10 更新：版本 13.1 正式发布

①

2015/04 更新：版本 14.0 正式发布

① 支持 unicode 编码

② Bayesian 统计分析

2015/10 更新：版本 14.1 正式发布

①

2016/09 更新：版本 14.2 正式发布

①

2017/06 更新：版本 15.0 正式发布

① latent class analysis

② PDF 与 Word 文件

③ 图形的透明度、不透明度调整

2017/11 更新：版本 15.1 正式发布

①

2019/06 更新：版本 16.0 正式发布

① 数据帧（内存中的多个数据集）

② lasso 回归

③ 自动报告

④ 更新选择模型

2020/02 更新：版本 16.1 正式发布

①

2021/04 更新：版本 17.0 正式发布

① 更新 tables 命令

② Bayesian 计量经济学

2023/04 更新：版本 18.0 正式发布

① Bayesian 模型平均

② 因果中介效应

③ 异质性 DID 分析

附录 B 赞助名单