Data Processing

Field Coordinator Training

Luiza Andrade, Leonardo Viotti & Rob Marty

June 2018

i2i DIME
TRANSFORM DEVELOPMENT

WORLD BANK GROUP

1. Introduction
2. Loading a data set
3. Exploring a data set
4. ID variables
5. Subsetting
6. Factor variables
7. Aggregating variables
8. Treating outliers

Outline

1. Introduction
2. Loading a data set
3. Exploring a data set
4. ID variables
5. Subsetting
6. Factor variables
7. Aggregating variables
8. Treating outliers
9. Saving a data set
10. Running a script from the master

Introduction

- In this session, you'll be introduced to some basic concepts of data cleaning in R. The contents covered are:
  - Importing data
  - Exploring a data set
  - Uniquely and fully indentifiable ID variables
  - Creating new variables
  - Dropping variables
  - Subsetting a data set
  - Dealing with factor variables
  - Treating outliers
- There are many other tasks that we usually perform as part of data cleaning that are beyond the scope of this session

Introduction

Before we start, let's make sure we're all set:

1. Make sure the packages `readstata13` and `dplyr` are listed in the `packages` vector of your Master script. If they are not installed (they should be!), install them.
2. If you haven't yet in this session, run the Master script to load all the necessary packages and set file paths.
3. Remeber to disable the `PACKAGES` switch in the Master if you already installed them. This will save you a lot of time.
4. Open the script called `Lab 3` in DataWork > Code. We've provided you with some code to build upon that will save you some time during this session.

Introduction

Here's a shortcut if you missed the last session:

```r
# Install packages
install.packages(c("readstata13","dplyr"),
                 dependencies = TRUE)

# Load packages
library(dplyr)
library(readstata13)

#  Set folder paths
projectFolder  <- "YOUR/FOLDER/PATH"
finalData <- file.path(projectFolder, "DataWork", "DataSets", "Final")
rawlData <- file.path(projectFolder, "DataWork", "DataSets", "Raw")
```

Outline

1. Introduction
2. Loading a data set
3. Exploring a data set
4. ID variables
5. Subsetting
6. Factor variables
7. Aggregating variables
8. Treating outliers
9. Saving a data set
10. Running a script from the master

Loading a data set from CSV

- In R, we usually save data in CSV format
- CSV files can be a lot lighter than binary files
- You can do version control of CSV files in .git
- On the other hand, the data they store is in a much simpler format, and we'll see some shortcomings of that soon
- To load a data set, we use the `read.csv()` function

Loading a data set from CSV

`read.csv(file)`

- **file**: is the path to the file you want to open, including it's name and format (`.csv`)

Exercise 1

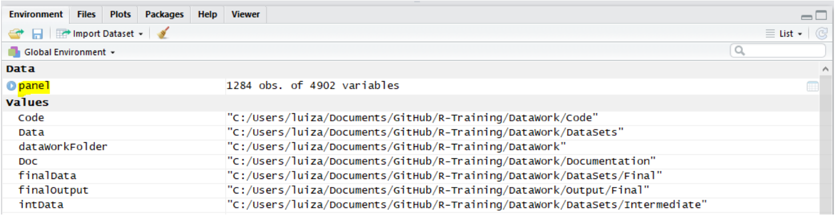
Load the `lwh_panel.csv` data set from DataWork > DataSets > Raw. Create an object called `panel` with this data set.

- TIP: use the `file.path()` function and the `rawData` object created in the master to simplify the folder path.

Loading a data set from CSV

```r
# Load a CSV data set
panel <- read.csv(file.path(rawData, "lwh_panel.csv"))
```

If you look at the Environment pane, you'll be able to see this data set. Click on the blue arrow to see the list of variables.



```
Data
panel            1284 obs. of 4902 variables
Values
  Code           "C:/Users/luiza/Documents/GitHub/R-Training/DataWork/Code"
  Data           "C:/Users/luiza/Documents/GitHub/R-Training/DataWork/DataSets"
  dataWorkFolder "C:/Users/luiza/Documents/GitHub/R-Training/DataWork"
  Doc            "C:/Users/luiza/Documents/GitHub/R-Training/DataWork/Documentation"
  finalData      "C:/Users/luiza/Documents/GitHub/R-Training/DataWork/DataSets/Final"
  finalOutput    "C:/Users/luiza/Documents/GitHub/R-Training/DataWork/Output/Final"
  intData        "C:/Users/luiza/Documents/GitHub/R-Training/DataWork/DataSets/Intermediate"
```

Loading a data set from CSV

- Note that R reads string variables as factors as default
- This format saves memory, but can be tricky if you actually want to use the variables as strings (which is rarely the case)
- You can specify the option `stringsAsFactors = FALSE` to prevent R from turning strings into factors
- You can also simply recast any variables you want to be strings (as they'll probably be just a few) using the `as.character()` function

Loading a Stata data set

- You can also load a .dta file, i.e., a Stata data set, using the read.dta13() function.
- This function takes exactly the same argument as the read.csv function

Exercise 2

Use the function read.dta13() to load the endline_data_raw.dta data set from DataWork > DataSets > Raw. Create an object called endline with this data set.

Loading a Stata data set

```
 # Load the raw data from LWH endline
endline <- read.dta13(file.path(rawData,
                                "endline_data_raw.dta"))
```

```
## Warning in read.dta13(file.path(rawData, "endline_data_raw.dta")):
##   hhh_change:
##   Missing factor labels - no labels assigned.
##   Set option generate.factors=T to generate labels.

## Warning in read.dta13(file.path(rawData, "endline_data_raw.dta")):
##   decisionmaker:
##   Duplicated factor levels detected - generating unique labels.

## Warning in read.dta13(file.path(rawData, "endline_data_raw.dta")):
##   respondent_main:
##   Duplicated factor levels detected - generating unique labels.

## Warning in read.dta13(file.path(rawData, "endline_data_raw.dta")):
##   respondent_finance:
##   Duplicated factor levels detected - generating unique labels.

## Warning in read.dta13(file.path(rawData, "endline_data_raw.dta")):
##   ag10_16_1_1:
```

Loading a Stata data set

- In Stata, you can have different levels of a labelled variable with the same value label. That's not true for R factors, which is why all those warnings were created
- If you go to the `Environment` pane and click on the blue arrow, you'll see that string variables were imported as strings, and only labelled values were imported as factors
- If you scroll down on the variables list, you can also find some metadata such as variable labels, value labels and Stata formats saved as variables' attributes

Outline

1. Introduction
2. Loading a data set
3. Exploring a data set
4. ID variables
5. Subsetting
6. Factor variables
7. Aggregating variables
8. Treating outliers
9. Saving a data set
10. Running a script from the master

Exploring a data set

Some useful functions:

- **`class()`:** reports object type or type of data stored
- **`dim()`:** reports the size of each one of an object's dimension
- **`names()`:** returns the variable names of a data set
- **`str()`:** general information on a R object, similar to `codebook` in Stata

Exploring a data set

Exercise 3

Use some of the functions listed in the previous slides to explor the `endline` and `panel` objects.

- TIP: all functions take a single argument, which is the object to be described.

Exploring a data set

```r
# See objects' formats
class(endline)
```

```
## [1] "data.frame"
```

```r
class(panel)
```

```
## [1] "data.frame"
```

```r
# How many observations and variables?
dim(endline)
```

```
## [1] 1067 9198
```

```r
dim(panel)
```

```
## [1] 1284 4902
```

Exploring a data set

```r
# Variable names
names(endline)
```

```
##  [1] "submissiondate" "starttime"      "endtime"        "deviceid"
##  [5] "subscriberid"   "duration"       "text_audit"     "note_gps_on"
##  [9] "enumerator_ID"  "supervisor_ID"  "id_05"          "pl_sample"
```

```r
str(panel)
```

```
## 'data.frame':    1284 obs. of  5 variables:
##  $ hh_code       : int  1001 1001 1002 1002 1002 1003 1003 1003 1004 1004 ...
##  $ wave          : Factor w/ 5 levels "Baseline","Endline",..: 4 5 3 4 5 4 5 2 4
##  $ year          : int  2014 2016 2013 2014 2016 2014 2016 2018 2014 2018 ...
##  $ treatment_hh  : Factor w/ 2 levels "Control","Treatment": 1 1 1 1 1 1 1 1 1 1
##  $ treatment_site: Factor w/ 2 levels "Control","Treatment": 1 1 1 1 1 1 1 1 1 1
```

```
##  Factor w/ 2 levels "Control","Treatment": 1 1 1 1 1 1 1 1 1 1 ...
```

Outline

1. Introduction
2. Loading a data set
3. Exploring a data set
4. **ID variables**
5. Subsetting
6. Factor variables
7. Aggregating variables
8. Treating outliers
9. Saving a data set
10. Running a script from the master

ID variables

Desired properties of an ID variable: *uniquely and fully identifying*

- An ID variable cannot have duplicates
- An ID variable may never be missing
- The ID variable must be constant across a project
- The ID variable must be anonymous

ID variables

Some useful functions to identify an ID variable in R

- **unique():** displays unique occurrences in an object
- **duplicated():** returns a boolean vector showing which observations of the object are duplicated
- **length():** length of an object
- **is.na():** returns a boolean vector showing which observations of the object have missing values

ID variables

Exercise 4

Use one of the functions in the previous slide to tell if `hh_code` is a uniquely and fully identifying variables for the `panel` data set.

- TIP: All these functions take a single argument, which is the object we're testing.

ID variables

- Uniquely identifiying?

```
# See all unique values of hh_code
duplicated(panel$hh_code)
```

```
##  [1] FALSE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE FALSE
## [12] FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE FALSE
```

```
# Count duplicates
sum(duplicated(panel$hh_code))
```

```
## [1] 755
```

ID variables

- Fully identifiying?

```
# See all unique values of hh_code
is.na(panel$hh_code)
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
# Count NAs
sum(is.na(panel$hh_code))
```

```
## [1] 0
```

ID variables

Ok, that was a tricky question, as we already knew this is a panel data set. So let's try with what is a more credible ID variable

Exercise 5

Count the number of missing observations and duplicates values in the variables `hh_code` and `year` in the `panel` data set.

- TIP: you can use `panel[,c("hh_code","year")]` as the function's argument to test both variables at the same time. Note that this is still a single argument, but this time it's a data frame with two columns.

ID variables

```r
# Uniquely identifying?
sum(duplicated(panel[,c("hh_code","year")]))
```

```
## [1] 0
```

```r
# Fully identifiying?
sum(is.na(panel$year))
```

```
## [1] 0
```

```r
sum(is.na(panel[,c("hh_code","year")]))
```

```
## [1] 0
```

ID variables

Exercise 6

Create new variable called `id` in the `panel` data set containing a uniquely and fully identifiable ID.

- TIP: To create a new variable in a data set, we simply assign a value to a column that doesn't year exist. You can make the ID unique by concatenating the year and the household ID.

ID variables

```
# Create panel ID
panel$id <- (panel$hh_code * 10000) + panel$year

# Check properties
sum(duplicated(panel$id))
```

```
## [1] 0
```

```
sum(is.na(panel$id))
```

```
## [1] 0
```

```
# Here's a shortcut similar to the isid variable in Stata
length(unique(panel$id, na.rm = TRUE)) == length(panel$id)
```

```
## [1] TRUE
```

Outline

1. Introduction
2. Loading a data set
3. Exploring a data set
4. ID variables
5. Subsetting
6. Factor variables
7. Aggregating variables
8. Treating outliers
9. Saving a data set
10. Running a script from the master

Subsetting

- The `panel` has a lot more variables than we'll actually use for analysis
- Let's construct a new data set with only the variables we'll you
- For starters, we'll subset the `panel` data set and keep only some variables of interest
- These variables are already listed in the script we gave you
- But first let's take a look at how to find variables to select in a data set

Subsetting

The grep function is similar to the wildcard (∗) in Stata: it looks for all elements of an object containing a certain pattern of strings.

grep(pattern, x): finds the elements of x that contain the pattern in pattern and returns a vector of indexes with their position

- **pattern:** the string expression you want to look for
- **x:** a vector of strings

Subsetting

Exercise 7

Create a vector called `plot_area_vars` containing the names of the variables with winsorized plot areas in seasons A, B and C:

1. Use the `grep()` function to display the position index of all the variables in the `panel` data set whose name contains the expression `area`.
2. Use the resulting vector from the `grep()` function to subset the vector of all variable names in the `panel` data frame.
3. Create the `plot_area_vars` vector containing only the names of the winsorized plot area variables

- TIP: these variables' names start with "w_".

Subsetting

```r
# Get plot area variables indexes
grep("area", names(panel))

# Get plot area variables names
names(panel)[grep("area", names(panel))]

# These are the ones we want:
plot_area_vars <- c("w_area_plots_a",
                    "w_area_plots_b",
                    "w_area_plots_c")
```

Subsetting

- In the previous sessions, we talked about indexing and how we can use it to select specific rows and columns of objects
- Here's a quick recall:

```
## Select a column (variable) in a data frame:
# First column
panel[, 1]

# Select by name
panel[, "hh_code"]
panel$hh_code

## Select elements in a vector

# Select one element in a vector
panel$hh_code[2]

# Select multiple elements in a vector
panel$hh_code[c(3,6,89)]

# Which is the same as
panel[c(3,6,89), "hh_code"]
```

Subsetting

- Now, let's use this information to subset our `panel` data frame and keep only the variables we have already listed

Exercise 8

Create a new object, called `lwh`, containing only the variables we selected from the `panel` data set.

Subsetting

```
 # Select variables
 id_vars          <- c("hh_code", "wave", "year", "treatment_hh",
                       "treatment_site", "site_code")
 demographic_vars <- c("gender_hhh", "age_hhh", "num_dependents", "read_and_write")
 yield_vars       <- c("w_gross_yield_a", "w_gross_yield_b")
 food_vars        <- c("expend_food_yearly", "expend_food_lastweek", "wdds_score")
 income_vars      <- names(panel)[28:37]
 plot_area_vars   <- c("w_area_plots_a", "w_area_plots_b", "w_area_plots_c")

# Subset data
lwh <- panel[, c(id_vars,
                 demographic_vars,
                 yield_vars,
                 food_vars,
                 income_vars,
                 plot_area_vars)]

 # Check result
 names(lwh)
```

```
##  [1] "hh_code"            "wave"                 "year"
##  [4] "treatment_hh"       "treatment_site"       "site_code"
##  [7] "gender_hhh"         "age_hhh"              "num_dependents"
## [10] "read_and_write"     "w_gross_yield_a"      "w_gross_yield_b"
## [13] "expend_food_yearly" "expend_food_lastweek" "wdds_score"
## [16] "inc_livestock"      "inc_livestockprod"    "inc_on_farm"
## [19] "inc_non_farm"       "inc_rent_land"        "inc_sale_land"
## [22] "inc_remittances"    "inc_int_div"          "inc_pension"
## [25] "inc_other"          "w_area_plots_a"       "w_area_plots_b"
## [28] "w_area_plots_c"
```

Outline

1. Introduction
2. Loading a data set
3. Exploring a data set
4. ID variables
5. Subsetting
6. **Factor variables**
7. Aggregating variables
8. Treating outliers
9. Saving a data set
10. Running a script from the master

Creating a factor

Let's take a look at the variable gender_hhh in the lwh data set:

```
# Tabulate gender of household head
table(panel$gender_hhh)
```

```
##
##   0   1
## 413 784
```

This is supposed to be categorical variable, but for now it's just numeric.

Creating a factor

To create a factor variable, we use the `factor` function:

`factor(x, levels, labels)` : turns numeric or string vector `x` into a factor vector * **`x:`** the vector you want to turn into a factor * **`levels:`** a vector containing the possible values of `x` * **`labels:`** a vector of strings containing the labels you want to apply to your factor variable

Exercise 9

Replace the variable `gender_hhh` in the `lwh` data set by a factor. Label the factor so that 0 is "Female" and 1 is "Male".

- TIP: the first string in the `labels` argument will be applied to the first value in the `levels` argument, the second to the second and so on.

Creating a factor

```
# The numeric variable
table(lwh$gender_hhh)
```

```
##
##   0   1
## 413 784
```

```
# Turn numeric variable into factor
lwh$gender_hhh <- factor(lwh$gender_hhh,
                         levels = c(0, 1),
                         labels = c("Female", "Male"))

# The factor variable
table(lwh$gender_hhh)
```

```
##
## Female   Male
##    413    784
```

Exploring factors

Here are a few useful functions to explore factors:

- **`str()`:** we've already seen this function. When using it with factors is that is shows the number and some variable labels
- **`levels()`:** displays the different values of a factor variable
- **`table()`:** displays a frequency table per level of a factor variable

Exercise 10

1. Use the functions listed above to explore the `wave` variable in the `lwh` data set.
2. (challenge) Use the `table()` function to see the year of each wave. Use the help if necessary.

Exploring factors

```
# First look at the variable
str(lwh$save)
```

```
##  NULL
```

```
# List all levels
levels(lwh$wave)
```

```
## [1] "Baseline" "Endline"  "FUP1&2"   "FUP3"     "FUP4"
```

```
# Observations per level
table(lwh$wave)
```

```
##
## Baseline  Endline   FUP1&2     FUP3     FUP4
##      213      307      126      345      293
```

```
# Year of each wave
table(lwh$wave, lwh$year)
```

```
##
##             2012 2013 2014 2016 2018
##   Baseline   213    0    0    0    0
##   Endline      0    0    0    0  307
##   FUP1&2       0  126    0    0    0
##   FUP3         0    0  345    0    0
##   FUP4         0    0    0  293    0
```

Ordering factors

- The wave variable's levels are not ordered correctly
- That's because R creates the number underlying a factor in alphabetical order
- However, that can be misleading, specially if we're using this variable for tables and graphs
- If we want the levels of a factor to be in an specific order, we need to set the argument ordered of the factor() function to TRUE
- This will tell R that the order of the levels is exactly the same as the one specified in the levels argument

Ordering factors

Exercise 11

Make the `wave` variable in the `lwh` dataset an ordered factor, ordering them by the year in which they happened

Ordering factors

```r
# Order variable
panel$wave <- factor(panel$wave,
                     levels = c("Baseline", "FUP1&2", "FUP3", "FUP4", "Endline"),
                     ordered = T)

# Check new order
levels(panel$wave)
```

```
## [1] "Baseline" "FUP1&2"   "FUP3"     "FUP4"     "Endline"
```

```r
table(panel$wave, panel$year)
```

```
##
##             2012 2013 2014 2016 2018
##   Baseline  213    0    0    0    0
##   FUP1&2      0  126    0    0    0
##   FUP3        0    0  345    0    0
##   FUP4        0    0    0  293    0
##   Endline     0    0    0    0  307
```

Outline

1. Introduction
2. Loading a data set
3. Exploring a data set
4. ID variables
5. Subsetting
6. Factor variables
7. Aggregating variables
8. Treating outliers
9. Saving a data set
10. Running a script from the master

Aggregate variables

- For our analysis in the next sessions, we will not use the disaggregated income variables
- We'll rather be interested in the total income of a household
- To calculate the total income, we'll use the `rowSums()` function, equivalent to `rowtotal` in Stata
- `rowSums()` is part of a function family that allows you to aggregate row and columns of a data set
- They all have the same syntax

Aggregate variables

`rowSums(x, na.rm = FALSE)`: sums selected variables in all rows of a data set

- **`x`:** the two-dimensional object whose columns you want to add
- **`na.rm`:** by default, when any of the values you're trying to add are missing, the result will be a missing value. Make this argument `TRUE` for R to treat missing values as zero in the sum

Aggregate variables

Exercise 12

Create a variable called `income_total` in the `lwh` data set containing the sum of all the income variables you listed in the `income_vars` vector.

Aggregate variables

Exercise 13

Create a variable called income_total in the lwh data set containing the sum of all the income variables you listed in the income_vars vector.

```
# List variables
income_vars
```

```
##  [1] "inc_livestock"     "inc_livestockprod" "inc_on_farm"
##  [4] "inc_non_farm"      "inc_rent_land"     "inc_sale_land"
##  [7] "inc_remittances"   "inc_int_div"       "inc_pension"
## [10] "inc_other"
```

```
# Add columns
lwh$income_total <-
  rowSums(lwh[, income_vars],
          na.rm = TRUE)
```

Dropping variables

Now, let's remove the disaggregated income variables from data set, since we're using them anymore.

The easiest way to remove variables from a data set is to make those variables `NULL`. Like this:

```r
dataset$variable <- NULL
```

To do this for a series of variables at the same time, you can select them by their names. Like this:

```r
dataset[, c("var1", "var2", "var3")] <- NULL
```

Dropping variables

Exercise 14

Remove all of the raw income variables listed in the `income_vars` vector from the `lwh` data data set.

Dropping variables

Exercise 15

Remove all of the raw income variables listed in the `income_vars` vector from the `lwh` data data set.

```
# Remove income variables
lwh[, income_vars] <- NULL
```

Outline

1. Introduction
2. Loading a data set
3. Exploring a data set
4. ID variables
5. Subsetting
6. Factor variables
7. Aggregating variables
8. Treating outliers
9. Saving a data set
10. Running a script from the master

Treating outliers

- We all know outliers can bias analysis results
- But most importantly, they make our plots ugly and out of scale
- There are different ways to treat outliers and there's not a single best method for doing it
- In the end, this is a research question and you should always discuss it with your PI before deciding what to do
- The one thing you should never do is drop a whole observation from your data set because of an outlier in a single variable
- There are a lot of

Winsorizing

To winsorize a variable means to replace all observations with value above (or below) a given percentile by the value of that percentile, therefore capping the distribution of the resulting variable.

As we've seen in session 2, you can create functions in R using a function called `function()`. In the next slide, we'll give you some code to create a function that

1. Calculates the value of the 90th percentile of its argument
2. Replaces all values above this percentile in its argument by the 90th percentile
3. Returns the resulting variable

Winsorizing

```r
# Create a function to winsorize at the 90th percentile
winsor <- function(x) {
   x[x > quantile(x, 0.9, na.rm = T)] <-
     quantile(x, 0.9, na.rm = T)
   return(x)
 }

 # Create winsorized income
 lwh$income_total_win <- winsor(lwh$income_total)
```

Trimming

To trim a variable means to replace all observation above a certain value (usually a given percentile) with missing values. It's particularly useful to create non-distorted graphs.

Exercise 16

Edit the `winsor()` function created earlier to create a new function that trims observations:

1. Instead of replacing the values with the 90th quantile, replace it with "NA"
2. Call this function `trim`
3. Use this function to trim the `income_total` variable in the `lwh` data set and create a new variable called `income_total_trim`

Trimming

```r
# Create the trim function
trim <- function(x) {
  x[x > quantile(x, 0.9, na.rm = T)] <- NA
  return(x)
}

# Trim income
lwh$income_total_trim <- trim(lwh$income_total)
```

Treating outliers

```
# Compare variables
summary(lwh$income_total)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       0   15240   52800   88549  130740  441000
```

```
summary(lwh$income_total_win)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       0   15240   52800   81070  130740  240000
```

```
summary(lwh$income_total_trim)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##       0   12480   47040   63777   96000  240000     126
```

Outline

1 Introduction

2 Loading a data set

3 Exploring a data set

4 ID variables

5 Subsetting

6 Factor variables

7 Aggregating variables

8 Treating outliers

9 Saving a data set

10 Running a script from the master

Saving a data set

- As mentioned before, R data sets are usually save as CSVs
- To save a data set, we use the `write.csv()` function:

```
write.csv(x, file, row.names = TRUE)
```

- **`x`:** the object (usually a data frame) you want to export to CSV
- **`file`:** the file path to where you want to save it, including the file name and the format (".csv")
- **`row.names`**: by default, R adds a column to the CSV file with the names (or numbers) of the rows in the data frame. Set it to `FALSE` if you don't want that column to be exported

Saving a data set

Exercise 17

Save the `lwh` data set to DataWork > DataSets > Final.

- TIP: Use the `file.path()` function and the object `finalData` created in the master to simplify the folder path.

Saving a data set

```r
# Save the lwh data set
write.csv(lwh,
          file.path(finalData,"lwh_clean.csv"),
          row.names = F)
```

Outline

1. Introduction
2. Loading a data set
3. Exploring a data set
4. ID variables
5. Subsetting
6. Factor variables
7. Aggregating variables
8. Treating outliers
9. Saving a data set
10. Running a script from the master

Running a script from the master

Exercise 18

1. Save this script
2. Add a switch for Lab3 in your master
3. Create an if statement that runs this script if the switch is on. To do this, use the `source()` function, which is equivalent to the `do` function in Stata.

- TIP: just like the `do` function in Stata, the only argumento of the `source()` function is the complete file path to your script. Don't forget to you explicit and dynamic file paths!