R basics for Stata users

Field Coordinator Training - R Track

Luiza Andrade, Leonardo Viotti & Rob Marty

June 2018

Outline

Introduction

These training sessions will offer a quick introduction to R, its amazing features and why it is so much better than Stata.

R is a powerful and flexible tool with a big and active community of users and developers that constantly posts in blogs and forums.

Introduction

Why choose R over Stata?

- It is way cooler.
- It is less specialized:
  - More flexibility when programming.
  - Many other functionalities.
- Much broader network of users in general:
  - Using google is a lot easier (you'll never want to see Statalist again in your life).
  - Development of new features and bug fixes happens faster.
- It makes prettier graphs.

Introduction

What are the possible disadvantages of R?

- Steeper learning curve (at least in the beginning).
- Stata is more specialized:
  - Certain common tasks are simpler in Stata, especially when you're doing them for the first time in R.
- Stata has wider adoption among micro-econometricians.
  - Network externalities in your work environment.
  - Development of new specialized techniques and tools could happen faster (e.g. *ietoolkit*).

Introduction

Here are some other advantages:

- R is a free and open source software!
- It allows you to have several data sets open simultaneously.
- It can run complex Geographic Information System (GIS) analyses.
- You can use it for web scrapping.
- You can run machine learning algorithms with it.
- You can create complex Markdown documents. This presentation, for example, is entirely done in RStudio.
- You can create dashboards and online applications with the Shiny package.

Introduction

This first session will present the basic concepts you will need to use R.

The next sessions will include:

- **Coding for reproducible research** - Programming basics and best practices.
- **Data processing** - Data processing workflow and main functions.
- **Descriptive analysis** - Exploratory and publication tables.
- **Data visualization** - Exploratory and publication graphs.
- **Spatial analysis** - GIS basics in R.

For the most recent versions of these trainings, visit the R-training GitHub repo at https://github.com/luizaandrade/R-Training

Outline

Getting started

This training requires that you have R installed in your computer:

Installation

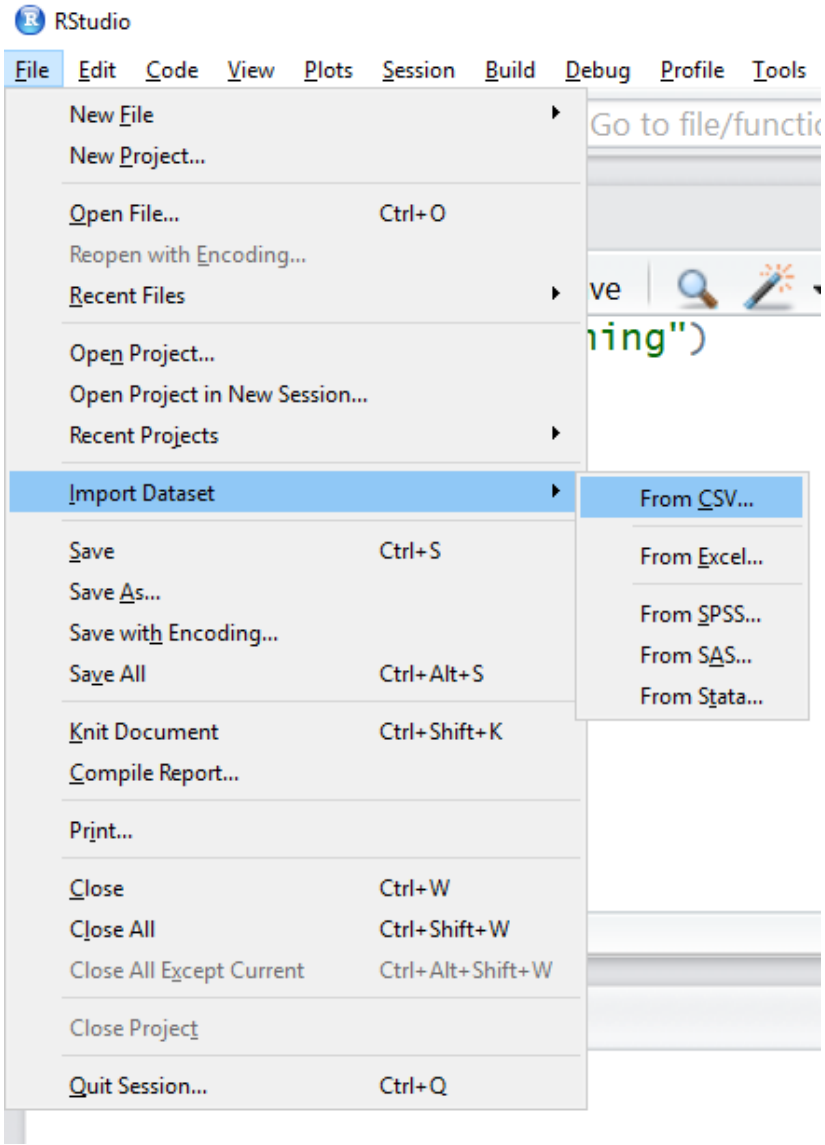
- Please visit (https://cran.r-project.org) and select a Comprehensive R Archive Network (CRAN) mirror close to you.
- If you're in the US, you can directly visit the mirror at Berkley university at (https://cran.cnr.berkeley.edu).
- Although, not necessary, we also strongly suggest installing R studio. You can get it in (https://www.rstudio.com/), but you need to install R first.

Exercise 1: Import data

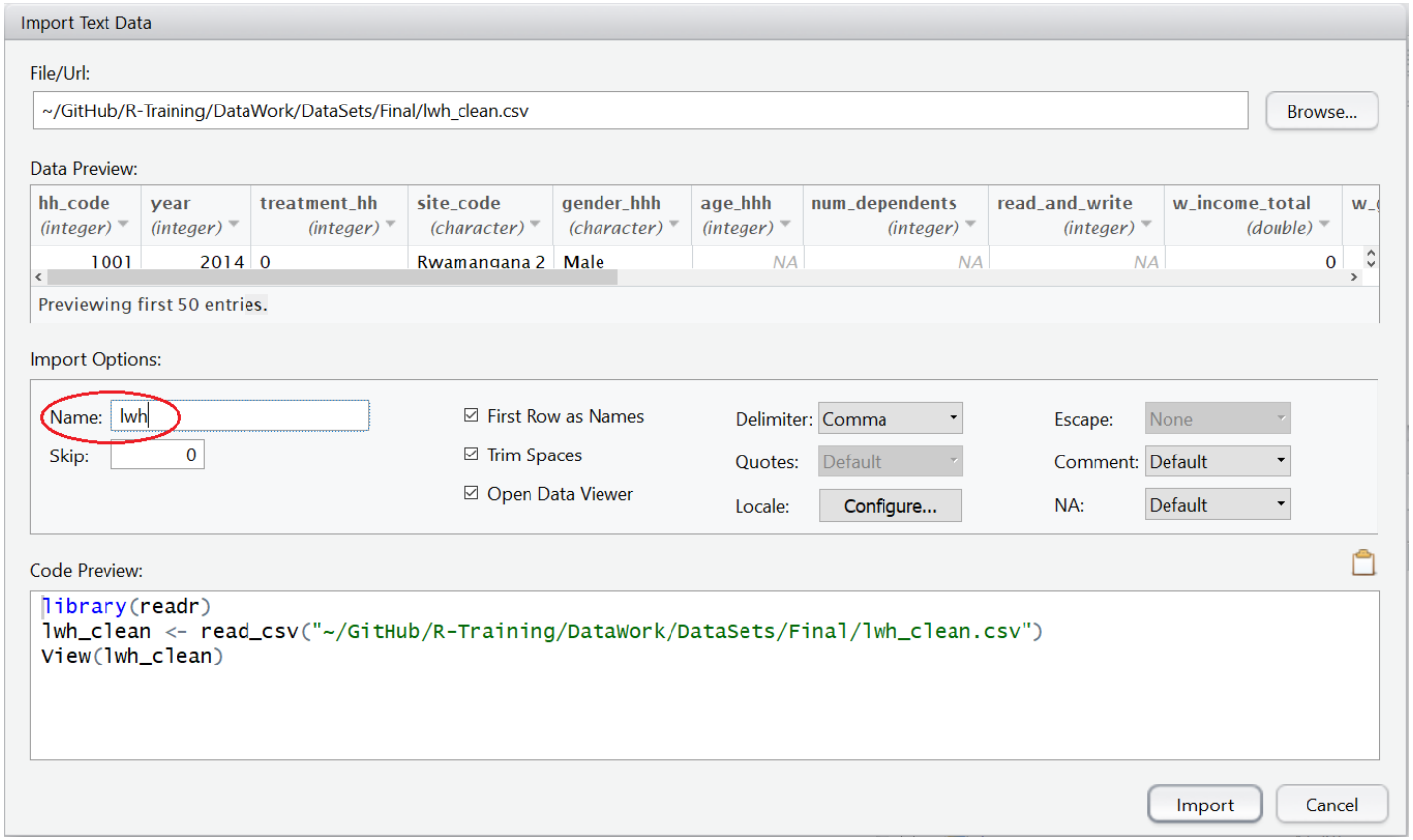
Let's start by loading the data set we'll be using:

1. In RStudio, go to File > Import Dataset > From Text (Base) and open the `lwh_clean.csv` file. Depending on your Rstudio version, it might be From CSV
2. The file should be in Session Materials/R Track/DataWork/DataSets/Final
3. Change the name to 'lwh' on the import window

Getting started

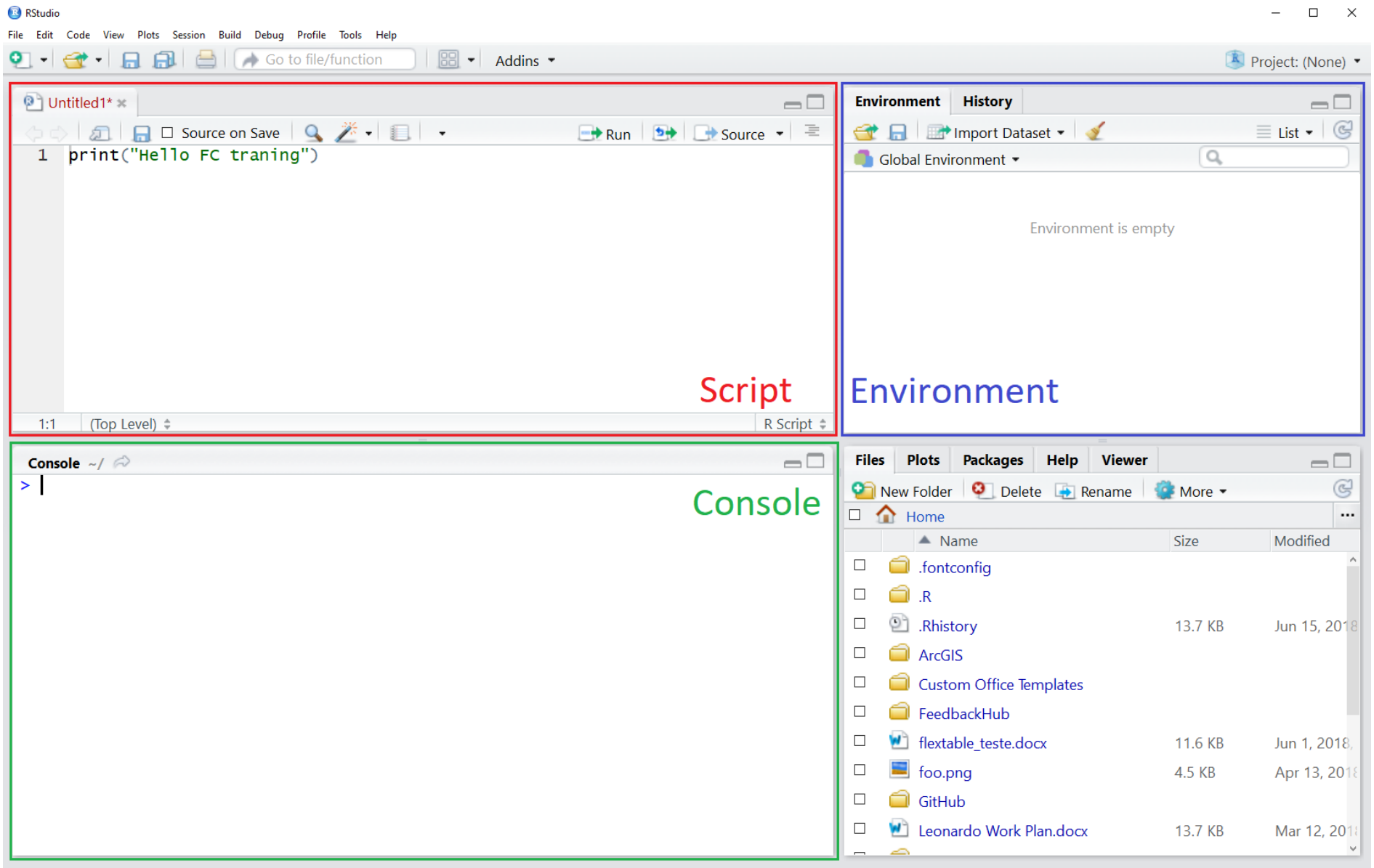


Getting started

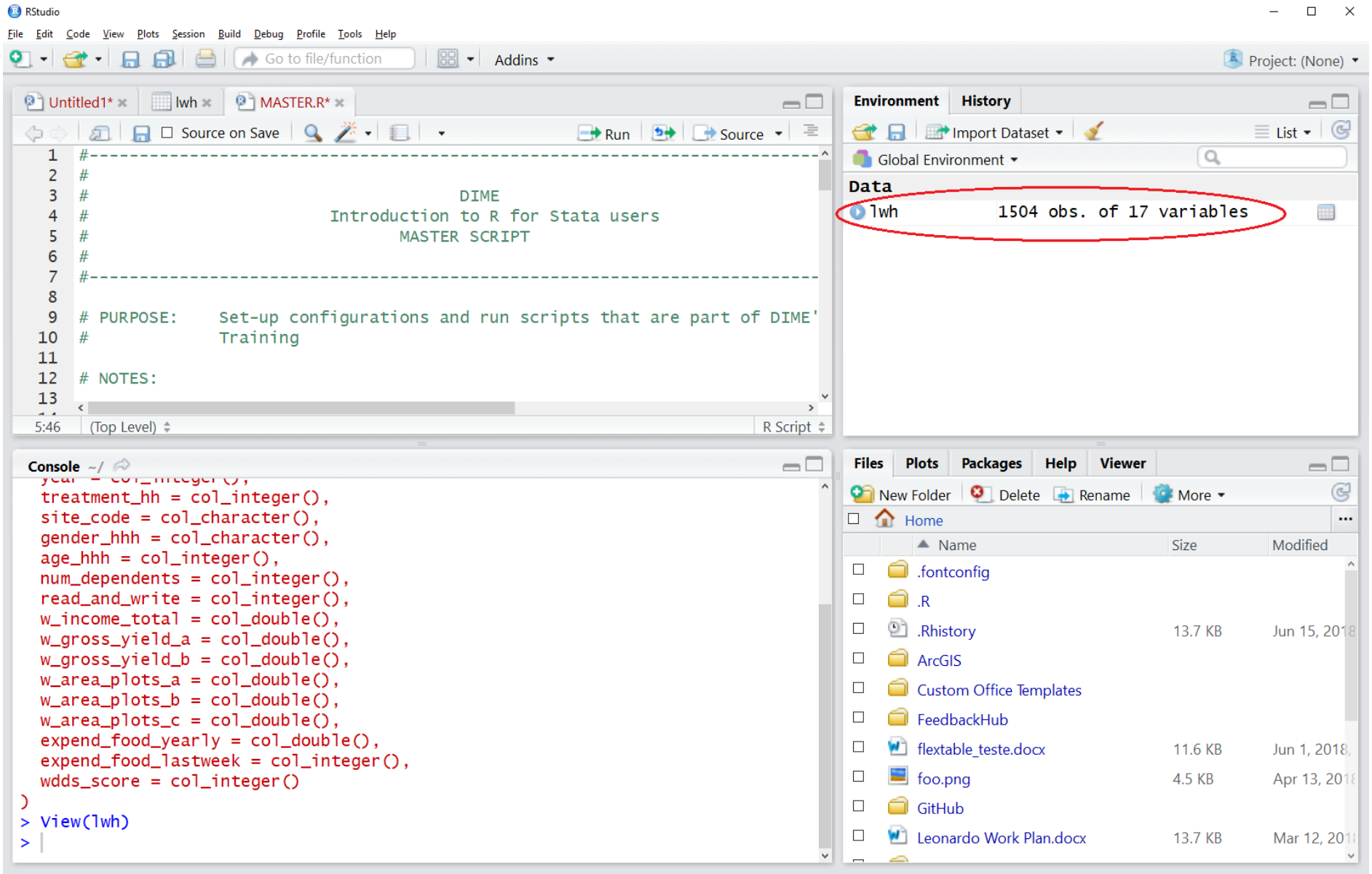


Outline

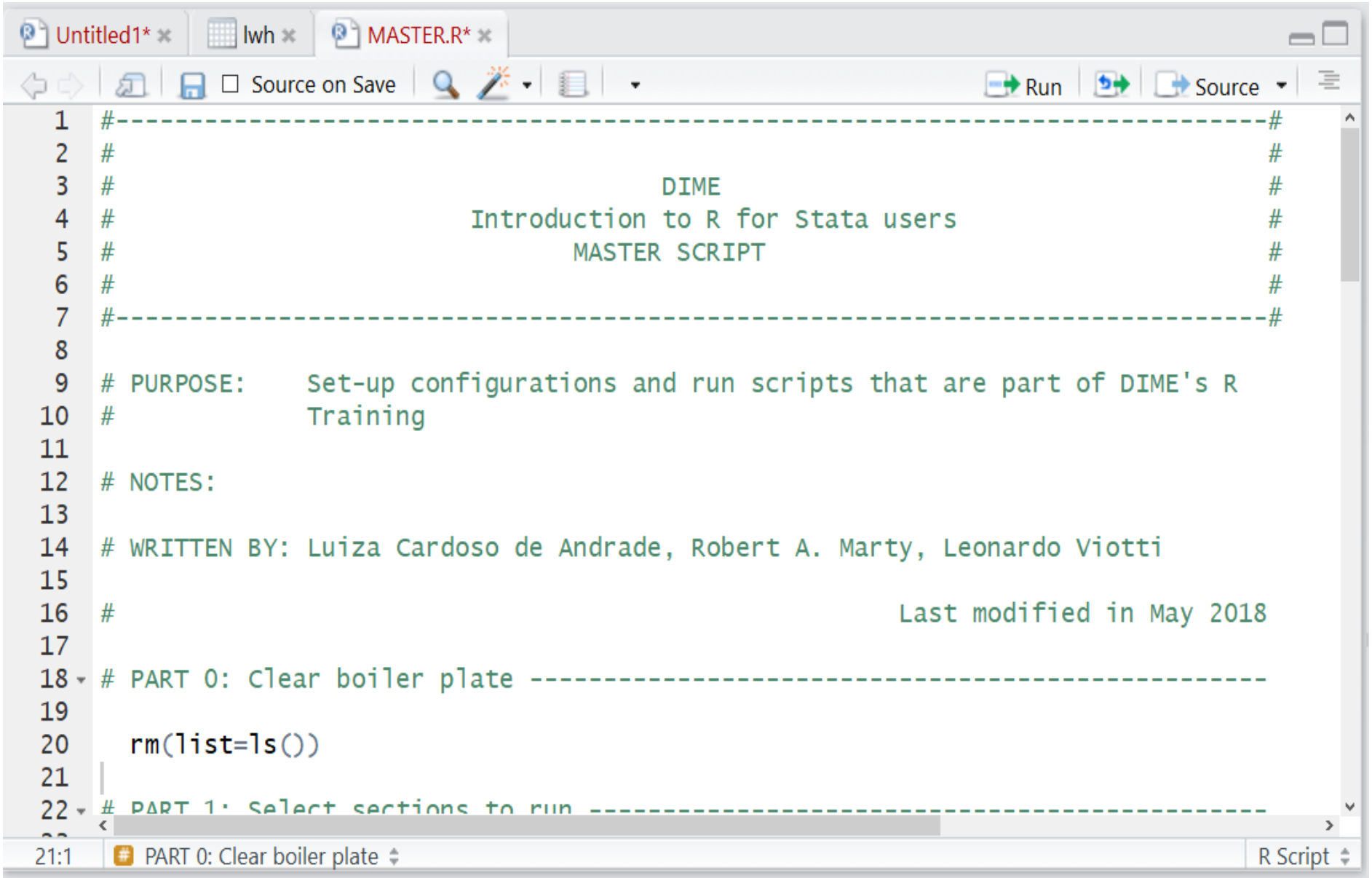
RStudio interface



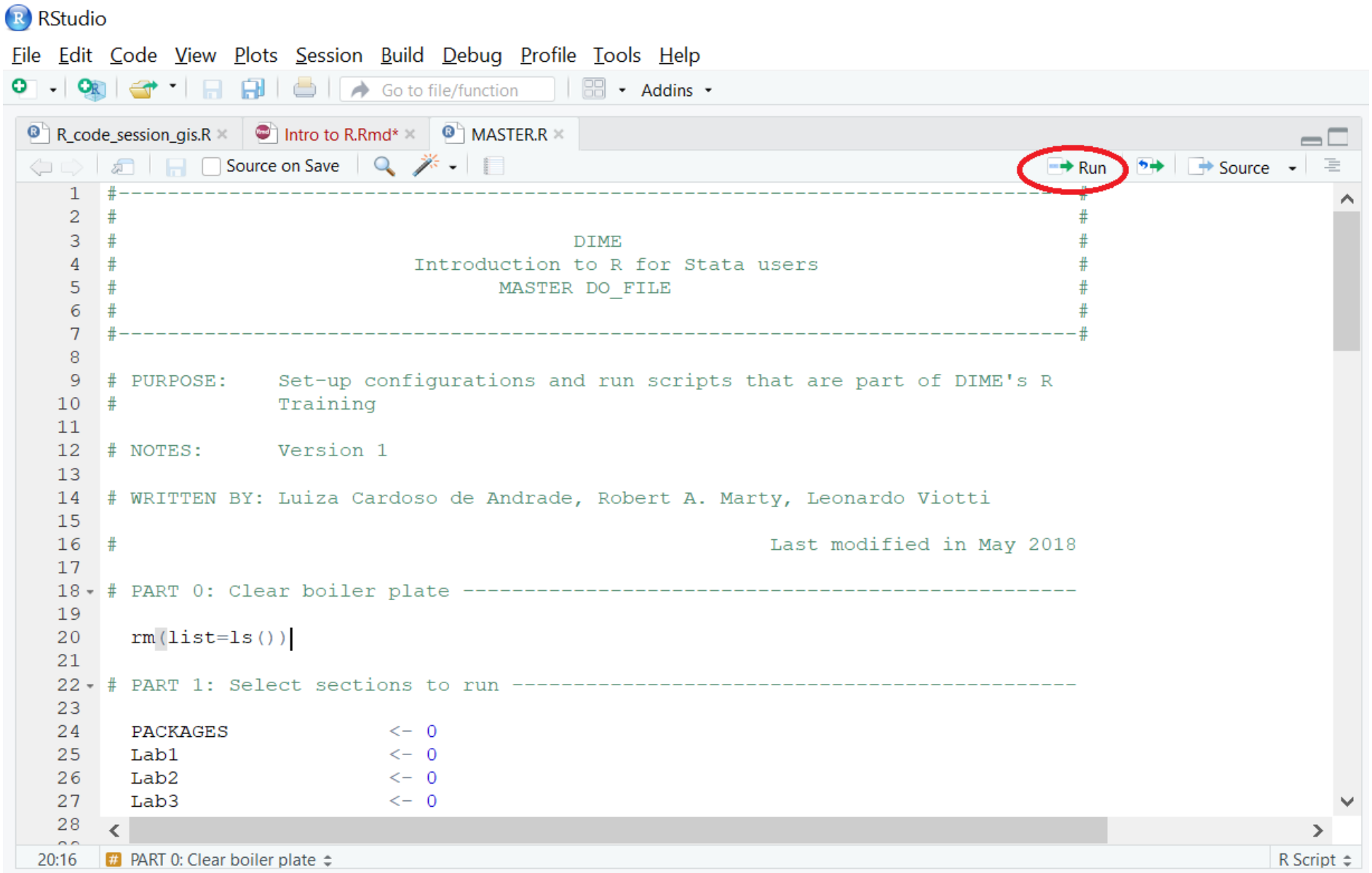
RStudio interface



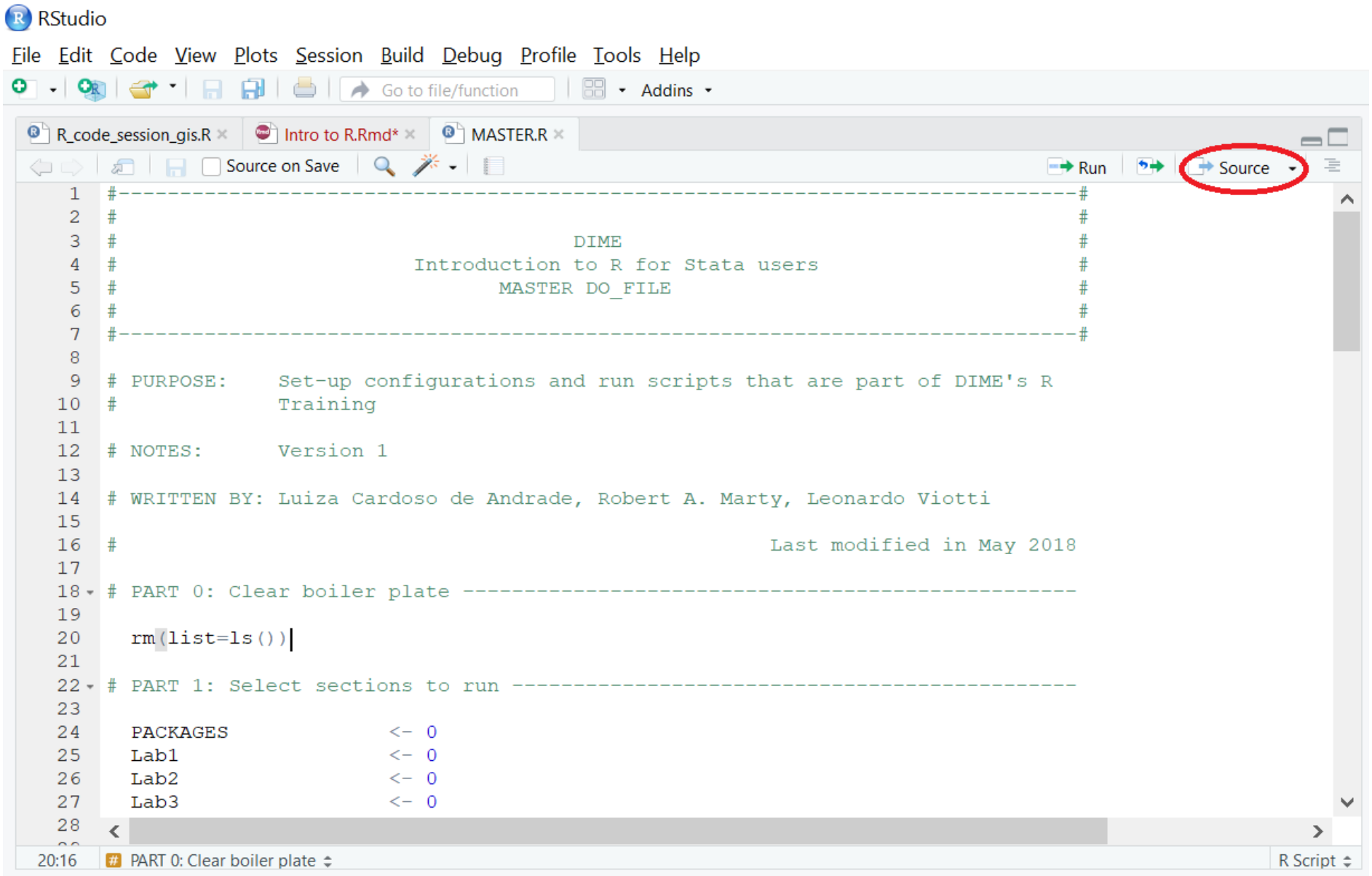
RStudio interface



RStudio interface



RStudio interface



Outline

Object-oriented language

- In Stata, you can open ONE dataset, and perform operations that can change this dataset.
- You can also have other objects, such as matrices, macros and tempfiles, but they are secondary, and most functions only use the main dataset.
- If you wish to do any non-permanent changes to your data, you'll need to preserve the original data to keep it intact.
- R works in a completely different way: you can have as many datasets (objects) as you wish (or your computer's memory allows) and operations will only have lasting effects if you store them.

Object-oriented language

- Everything that exists in R's memory – variables, datasets, functions – is an object.
- An object is a chunk of data stored in the memory that has a name by which you call it (exactly like macros in Stata).
- If you create an object, it is going to be stored in memory until you delete it or quit R.
- Whenever you run anything you intend to use in the future, you need to store it as an object.

Object-oriented language

To better understand the idea, we're going to use the data from the Rwanda LWH project. First, let's take a look at the data.

Type the following code to explore the data:

```
# We can use the function View() to browse the whole data
View(lwh)

# Alternatively we can print the first 5 obs. with head()
head(lwh)
```

Object-oriented language

Now, let's try some simple manipulations. First, assume we're only interested in data of the year 2018.

Exercise 2: Subset the data

1. Subset the `lwh` data set, keeping only observations where variable `year` equals 2018.

```
# To do that we'll use the subset() function
subset(lwh, year == 2018)
```

2. Then, look again at the first 5 observations

```
# Use the head() function again
head(lwh)
```

Object-oriented language

```
##   panel_id hh_code   wave year treatment_hh treatment_site    site_code
## 1       NA      NA   <NA>   NA          <NA>           <NA>         <NA>
## 2   100103    1001   FUP3 2014       Control        Control Rwamangana 2
## 3   100104    1001   FUP4 2016       Control        Control Rwamangana 2
## 4       NA      NA   <NA>   NA          <NA>           <NA>         <NA>
## 5   100202    1002 FUP1&2 2013       Control        Control Rwamangana 2
## 6   100203    1002   FUP3 2014       Control        Control Rwamangana 2
##   gender_hhh age_hhh num_dependents read_and_write w_gross_yield_a
## 1       <NA>      NA             NA             NA              NA
## 2       Male      NA             NA             NA               0
## 3       Male      28             NA             NA           13050
## 4       <NA>      NA             NA             NA              NA
## 5     Female      51              3              0           28000
## 6     Female      NA             NA             NA               0
##   w_gross_yield_b expend_food_yearly expend_food_lastweek
## 1              NA                NA                   NA
## 2       114583.34         156532.5000                 3000
## 3            0.00         250452.0000                 4800
## 4              NA                NA                   NA
## 5       156218.31           260.8875                    5
## 6        61969.74         163576.4700                 3135
```

Object-oriented language

We can see that nothing happened to the original data. This happens because we didn't store the edit we made anywhere.

To store an object, we use the assignment operator (<-):

```
# Assign a value of the Answer to the Ultimate Question of
# Life, the Universe, and Everything
x <- 42
```

From now on, x is associated with the stored value (until you replace it delete it or close R).

Object-oriented language

Exercise 3: Create an object

Create a new data set, called `lwh2018`, that is a subset of the `lwh` data set containing only data from the year 2018.

```
# Using the same function but now assigning it to an object
lwh2018 <- subset(lwh, year == 2018)

# Display the 5 first obs. of the new data
head(lwh2018)

# Notice that we still have the original data set intact
head(lwh)
```

Object-oriented language

```
##    panel_id hh_code    wave year treatment_hh treatment_site    site_code
## 11   100305    1003 Endline 2018      Control         Control Rwamangana 2
## 14   100405    1004 Endline 2018      Control         Control Rwamangana 2
## 19   100605    1006 Endline 2018      Control         Control Rwamangana 2
## 24   100705    1007 Endline 2018      Control         Control Rwamangana 2
## 31   101005    1010 Endline 2018      Control         Control Rwamangana 2
## 33   101105    1011 Endline 2018      Control         Control Rwamangana 2
##    gender_hhh age_hhh num_dependents read_and_write w_gross_yield_a
## 11     Female      53              NA             NA       200238.09
## 14     Female      93              NA             NA       471014.47
## 19       Male      27              NA             NA        51785.71
## 24     Female      58              NA             NA       325555.53
## 31     Female      35              NA             NA       129152.91
## 33     Female      58              NA             NA       129999.99
##    w_gross_yield_b expend_food_yearly expend_food_lastweek wdds_score
## 11       284642.84           46959.75                 900          3
## 14       450310.53          130443.75                2500          4
## 19        38839.29          166968.00                3200          3
## 24       310000.00           52177.50                1000          4
## 31       173316.92          279149.63                5350          5
## 33        98999.99           33915.38                 650          4
```

Object-oriented language

Two an important concepts to take note:

1. In R, if you want to change your data, you need to store it in an object. It is possible to simply replace the original data, but often, it's more practical to create a new dataset.
2. Print (display) is built into R. If you execute an action and don't store it anywhere, R will simply print the results of that action but won't save anything in the memory.

Outline

R objects

Objects can have different structures, that is, different ways to store data. Objects are the building blocks of R programming. You can create and manipulate them to explore your data and construct analytical outputs.

There are several types of objects. Here are the ones we will cover:

- **Vectors:** an unidimensional object that stores a sequence of values
- **Data frames:** a combination of different vectors of the same length (the same as your data set in Stata)
- **Lists:** a multidimensional object that can store several objects of different dimension

R objects

Vectors

A vector is an unidimensional object composed by one or more scalars of the same type.

Use the following code to create vectors in two different ways

```
# Creating a vector with the c() function
v1 <- c(1,1,2,3,5)

# Alternative way to create an evenly spaced vector
v2 <- 1:5
```

You can use brackets for indexing

```
# Print the 4th element of the vector
v2[4]


## [1] 4
```

R objects

Vectors

To R, each of the columns of `lwh` is a vector.

Calling a vector from a `data.frame` column

We use the `$` to call vector (variables) by their names in a `data.frame`

Type the following code:

```
# Create a vector with the values of `age_hhh` variable
age_vec <- lwh$age_hhh

# See the 13th element of the column
lwh$age_hhh[67]
```

```
## [1] 61
```

R objects

Data Frames

The `lwh` and `lwh2018` objects are both data frames. You can also construct a new data.frame from scratch by combining vectors.

Now, type the following code to create a new data frame

```
# Dataframe created by biding vectors
df1 <- data.frame(v1,v2)
df1
```

```
##   v1 v2
## 1  1  1
## 2  1  2
## 3  2  3
## 4  3  4
## 5  5  5
```

Since a data frame has two dimensions, you can use indexing on both:

Numeric indexing

```
# The first column of lwh
lwh[,1]

# The 45th line of lwh
lwh[45,]

# Or the 45th element of the first line
lwh[45,1]
```

Alternatively, you can use the column names for indexing, which is the same as using the $ sign.

Names indexing

```
# Or the 45th element of the hh_code column
lwh[45,"hh_code"]
```

```
## [1] NA
```

R objects

Data Frames

Lists are more complex objects that can contain many objects of different classes and dimensions.

Lists are fancy and can have a lot of functionalities and attributes. They are the output of many functions and are used to construct complex objects.

It would be beyond the scope of this introduction to go deep into them, but here's a quick example

Combine several objects of different types in a list

```
# Use the list() function
lst <- list(v1, df1, 45)
```

Print the list yourself to see how it looks like.

R objects

Lists

```
# Check the contents of lst
print(lst)
```

```
## [[1]]
## [1] 1 1 2 3 5
##
## [[2]]
##   v1 v2
## 1  1  1
## 2  1  2
## 3  2  3
## 4  3  4
## 5  5  5
##
## [[3]]
## [1] 45
```

Outline

Basic types of data

R has different kinds of data that can be recorded inside objects. They are very similar to what you have in Stata, and the main types are string, integer, numeric, factor and boolean.

Let's start with the simpler ones:

Strings

A sequence of characters and are usually represented between double quotes. They can contain single letters, words, phrases or even some longer text.

Integer and numeric

As in Stata, these are two different ways to store numbers. They are different because they use memory differently. As default, R stores numbers in the numeric format (double).

Now we'll use string data to practice some basic object manipulations in R.

Exercise 4: Create a vector of strings

Create two string vector containing the names of commonly used statistical software in order of importance:

```
# Creating string vector
str_vec <- c("R",
             "Python",
             "SAS",
             "Microsoft Excel",
             "Stata")
```

Now print them to check them out.

Exercise 5: Concatenate strings

1. Create a scalar (a vector of one element) containing the phrase "is better than" and cal it `str_scalar`.
2. Use the function `paste()` with 3 arguments separated by commas:
   - The first argument as the 1st element of `str_vec`.
   - The second argument as the `str_scalar`.
   - The third argument as the 5th element of `str_vec`.
3. If you're not sure where to start, type:

```
help(paste)
```

Basic types of data

Strings

```
### Using the paste function to combine strings

# Scalar
str_scalar <- "is better than"

# Using the paste() function
paste(str_vec[1], str_scalar, str_vec[5])
```

```
## [1] "R is better than Stata"
```

Outline

Advanced types of data

R also has other more complex ways of storing data. These are the most used:

Factors

Factors are numeric categorical values with text label, equivalent to labelled variables in Stata. Turning strings into factors makes it easier to run different analyses on them and also uses less space in the memory, which is why data frames do that by default.

Booleans

Booleans are logical binary variables, accepting either `TRUE` or `FALSE` as values. They are automatically generated when performing logical operations

You can see that in `lwh` the `wave`, `treatment_hh`, `treatment_site`, `site_code` and `gender_hhh` are factor variables. You can see in your enviroment panel the type of all your variables, and for factors the number of levels.

We'll learn how to deal with factors in detail on the next session, since they are very important for us. For now, here are two important things to keep in mind when using them:

Warning:

Unlike Stata, in R

1. You use the labels to refer to factors
2. You cannot choose the underlying values

Advanced types of data

Booleans

Boolean data is not normally used directly in data frames, but rather to express the results of a logical condition.

Exercise 6:

Create boolean vector with the condition of annual income below average:

```
# Create vector
bool_vec <- (lwh$income_total_win <
               mean(lwh$income_total_win))

# See the 5 first elements of the vector
head(bool_vec)
```

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

Let's use the boolean vector created to add a dummy variable in the `lwh` data set for the same condition.

Exercise 6:

1. Create a column in `lwh` containing zeros and call it `income_low`. You can do this by typing:

```
lwh$income_low <- 0
```

2. Use `bool_vec` to index the lines of the `income_low` column and replace all observations that meet the condition with the value 1.

```
lwh$income_low[bool_vec] <- 1
```

Advanced types of data

Booleans

```
# Create column with zeros
lwh$income_low <- 0

# Replace with 1 those obs that meet the condition
lwh$income_low[bool_vec] <- 1
```

```
# See the first 5. obs.
head(lwh$income_low)
```

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

Outline

Help, Google and Stackoverflow

Help in R works very much like in Stata: the help files usually start with a brief description of the function, explain its syntax and arguments and list a few examples. There are two ways to access help files:

Exercise 7: Use help

```
# The help() function
help(summary)

# and its abbreviation
?summary
```

Help, Google and Stackoverflow

- The biggest difference, however, is that R has a much wider user community and it has a lot more online resources.
- For instance, in 2014, Stata had 11 dedicated blogs written by users, while R had 550.[1]
- The most powerful problem-solving tool in R, however, is Google. Searching the issue, you have or the error message displayed usually yields tons of results and will probably lead you to a Stack Overflow page where someone asked the same question and several people gave different answers.

[1] Check http://r4stats.com/articles/popularity/ for more.

Outline

Useful resources

Blogs and online courses:

- Surviving graduate econometrics with R: https://thetarzan.wordpress.com/2011/05/24/surviving-graduate-econometrics-with-r-the-basics-1-of-8/
- An Introduction to R at https://cran.r-project.org/
- R programming in Coursera: https://www.coursera.org/learn/r-programming
- Try R in Code School: http://tryr.codeschool.com/
- R programming for dummies: http://www.dummies.com/programming/r/
- R bloggers: https://www.r-bloggers.com/
- R statistics blog: https://www.r-statistics.com/
- The R graph gallery: https://www.r-graph-gallery.com/

Useful resources

Books:

- R for Stata Users - Robert A. Muenchen and Joseph Hilbe
- R Graphics Cookbook - Winston Chang
- R for Data Science - Hadley Wickha and Garrett Grolemund

Thank you!

Outline

Syntax

R's syntax is a bit heavier than Stata's:

- Parentheses to separate function names from its arguments.
- Commas to separate arguments.
- For comments we use the # sign.
- You can have line breaks inside function statements.
- In R, functions can be treated much like any other object Therefore, they can be passed as arguments to other functions.

Similarly to Stata:

- Square brackets are used for indexing.
- Curly braces are used for loops and if statements.
- Largely ignores white spaces.

RStudio interface

Script

Where you write your code. Just like a do file.

Console

Where your results and messages will be displayed. But you can also type commands directly into the console, as in Stata.

Environment

What's in R's memory.

The 4th pane

Can display different things, including plots you create, packages loaded and help files.

Matrices

A matrix a bidimensional object composed by one or more vectors of the same type.

Type the following code to test two different ways of creating matrices

```
# Matrix created by joining two vectors:
m1 <- cbind(v1,v1)

# Matrix using the
m2 <- matrix(c(1,1,2,3,5,8), ncol = 2)
```

Matrices

Now use the following code to check the elements of these matrices by indexing

```
# Matrix indexing: typing matrix[i,j] will give you
# the element in the ith row and jth column of that matrix
#m2[1,2]

# Matrix indexing: typing matrix[i,] will give you the
# ith row of that matrix
m1[1,]

# Matrix indexing: typing matrix[,j] will give you the
# jth column of that matrix (as a vector)
m1[,2]
```

Create a factor verctor using the following code

```
# Basic factor vector
num_vec <- c(1,2,2,3,1,2,3,3,1,2,3,3,1)
fac_vec <- factor(num_vec)

# A bit fancier factor vector
fac_vec <- factor(num_vec,labels=c("A","B","C"))

# Change labels
levels(fac_vec) = c('One','Two','Three')
```

Numbers and integers

Two scalars, one with a round number the other with a fractional part

```
# a numeric scalar with an integer number
int <- 13
num <- 12.99
```

Numbers and integers

Now we can see the objects classes with the *class()* function and test it with the *is.integer()* and *is.numeric()* functions.

```
# you can see the number's format using the class function:
class(int)
```

```
## [1] "numeric"
```

```
class(num)
```

```
## [1] "numeric"
```

```
# you can test the class with the is. method
is.integer(int)
```

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

Numbers and integers

Numbers and integers

We can, however, coerce objects into different classes. We just need to be carefull because the result might not be what we're expecting.

Use the *as.integer()* and *round()* functions on the *num* object to see the difference:

```
as.integer(num)
```

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

```
# and
```

```
round(num)
```

```
## [1] 13
```