

R basics

R for Stata Users

Luiza Andrade, Leonardo Viotti & Rob Marty

November-December 2018



Outline

- 1 Introduction
- 2 Getting started
- 3 RStudio interface
- 4 Object-oriented language
- 5 R objects
- 6 Basic types of data
- 7 Advanced types of data
- 8 Help, Google and Stackoverflow
- 9 Useful resources
- 10 Appendix

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

Introduction

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

The next sessions will include:

- **Exporting summary statistics and regression tables** (Nov 20)
- **Creating awesome-looking plots** (Nov 29)
- **Writing reproducible and readable code** (Dec 6)
- **Creating awesome-looking maps** (Dec 13)
- **Cleaning data** (Dec 20)

For the most recent versions of these trainings, visit the R-training GitHub repo at <https://github.com/worldbank/dime-r-training>

(currently under the 'development' branch)

Some advantages of R over Stata:

- It is less specialized:
 - More flexibility when programming.
 - Many more functionalities.
- Much broader network of users:
 - More resources online, which makes using Google a lot easier. You'll never want to see Statalist again in your life.
 - Development of new features and bug fixes happens faster.
- It is way cooler.

Some possible disadvantages of R:

- Higher cost of entry than Stata.
 - That doesn't mean that the learning curve is steeper all the way up!
- Stata is more specialized:
 - Certain common tasks are simpler in Stata.
- 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*).

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 interactive dashboards and online applications with the Shiny package.

What about Python?

- Python is even more flexible and has more users than R. So, why should I bother to learn R?
- Despite being super popular for data science, Python has fewer libraries developed for econometrics.
- Python still cannot do everything Stata does without some trouble, R can.
- R and Python are very similar, specially if your background is in Stata.

Outline

- 1 Introduction
- 2 Getting started
- 3 RStudio interface
- 4 Object-oriented language
- 5 R objects
- 6 Basic types of data
- 7 Advanced types of data
- 8 Help, Google and Stackoverflow
- 9 Useful resources
- 10 Appendix

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

Installation

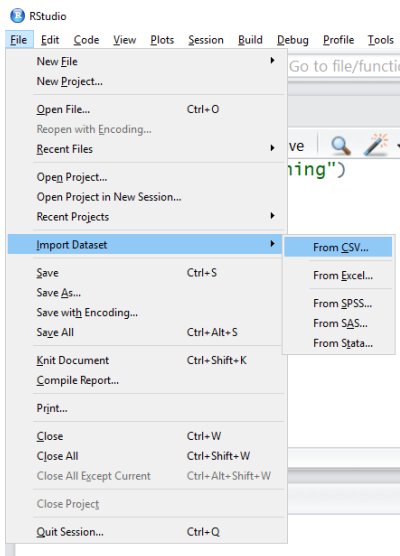
- Please visit (<https://cran.r-project.org>) and select a Comprehensive R Archive Network (CRAN) mirror close to you.
- If you're in the US, you can directly visit the mirror at Berkley university at (<https://cran.cnr.berkeley.edu>).
- we also strongly suggest installing R studio. You can get it in (<https://www.rstudio.com/>), but you need to install R first.

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

Exercise 1: Import data

- 1 In RStudio, go to File > Import Dataset > From Text (Base) and open the `whr_panel.csv` file.
 - Depending on your Rstudio version, it might be File > Import Dataset > From CSV
- 2 The file should be in `GitHub/dime-r-training/DataWork/DataSets/Final`
- 3 Change the name to `whr` on the import window

Getting started



Getting started

Import Dataset

Name lwh

Encoding Automatic

Heading ☒ Yes ☐ No

Row names Automatic

Separator Comma

Decimal Period

Quote Double quote (")

Comment None

na.strings NA

☒ Strings as factors

Input File

```
"hh_code","year","treatment_hh","site_code","gender_hhh"  
1001,2014,0,"Rwamangana 2","Male",NA,NA,NA,0,0,114583.33  
1001,2016,0,"Rwamangana 2","Male",28,NA,NA,9000,13050,0,  
1002,2013,0,"Rwamangana 2","Female",51,3,0,46000,28000,1,  
1002,2014,0,"Rwamangana 2","Female",NA,NA,NA,0,0,61969.7  
1002,2016,0,"Rwamangana 2","Male",54,NA,NA,0,0,88000,0.5,  
1003,2014,0,"Rwamangana 2","Female",NA,NA,NA,140000,1565  
1003,2016,0,"Rwamangana 2","Female",53,NA,NA,22500,0,0,0  
1003,2018,0,"Rwamangana 2","Female",53,NA,NA,1e+05,20023  
1004,2014,0,"Rwamangana 2","Female",NA,NA,NA,79000,39862  
1004,2018,0,"Rwamangana 2","Female",93,NA,NA,60000,47101  
1005,2016,0,"Rwamangana 2","Male",74,NA,NA,177000,345238  
1006,2013,0,"Rwamangana 2","Male",24,2,1,1e+05,8700,4861  
1006,2014,0,"Rwamangana 2","Male",NA,NA,NA,0,64444.44531  
1006,2016,0,"Rwamangana 2","Male",NA,NA,NA,0,0,0,0,0,0
```

Data Frame

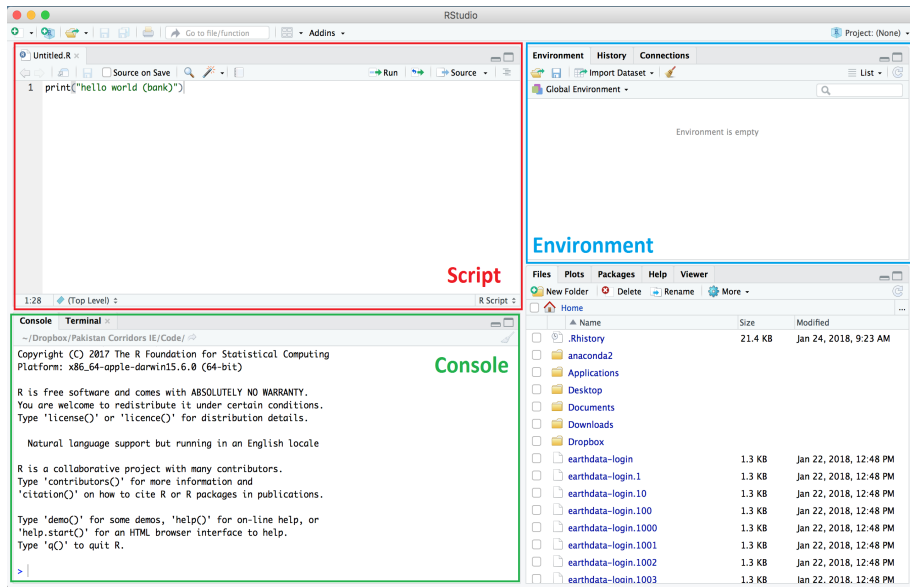
hh_code	year	treatment_hh	site_code	gender_hhh
1001	2014	0	Rwamangana 2	Male
1001	2016	0	Rwamangana 2	Male
1002	2013	0	Rwamangana 2	Female
1002	2014	0	Rwamangana 2	Female
1002	2016	0	Rwamangana 2	Male
1003	2014	0	Rwamangana 2	Female
1003	2016	0	Rwamangana 2	Female
1003	2018	0	Rwamangana 2	Female
1004	2014	0	Rwamangana 2	Female
1004	2018	0	Rwamangana 2	Female
1005	2016	0	Rwamangana 2	Male
1006	2013	0	Rwamangana 2	Male
1006	2014	0	Rwamangana 2	Male

Import Cancel

Outline

- 1 Introduction
- 2 Getting started
- 3 RStudio interface**
- 4 Object-oriented language
- 5 R objects
- 6 Basic types of data
- 7 Advanced types of data
- 8 Help, Google and Stackoverflow
- 9 Useful resources
- 10 Appendix

RStudio interface



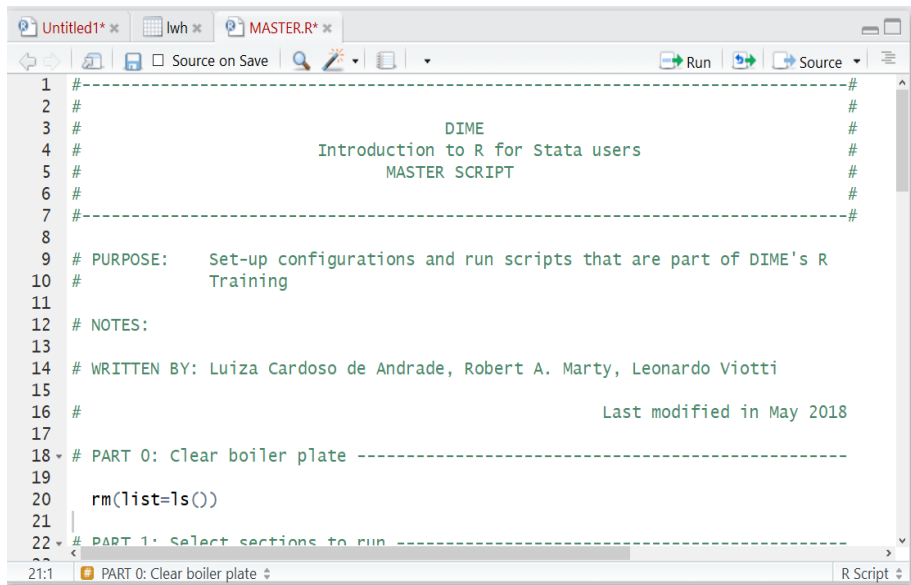
RStudio interface

The screenshot displays the RStudio interface with the following components:

- Menu Bar:** File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, Help.
- Toolbar:** Includes icons for opening files, saving, running code, and other standard IDE functions.
- Source Editor:** Contains an R script file named 'Intro to R.Rmd'. The script includes a header section for 'DIME' (Introduction to R for Stata users) and a 'PURPOSE' section describing the setup for training.
- Environment Pane:** Shows the 'Global Environment' which is currently empty.
- Console:** Displays the current working directory as '~/.GitHub/R-Training/Presentations/Spatial Data/Data/shapefiles/'. The prompt is '>'. There is also a 'Terminal' tab.
- File Explorer:** Shows a file tree with the following items:

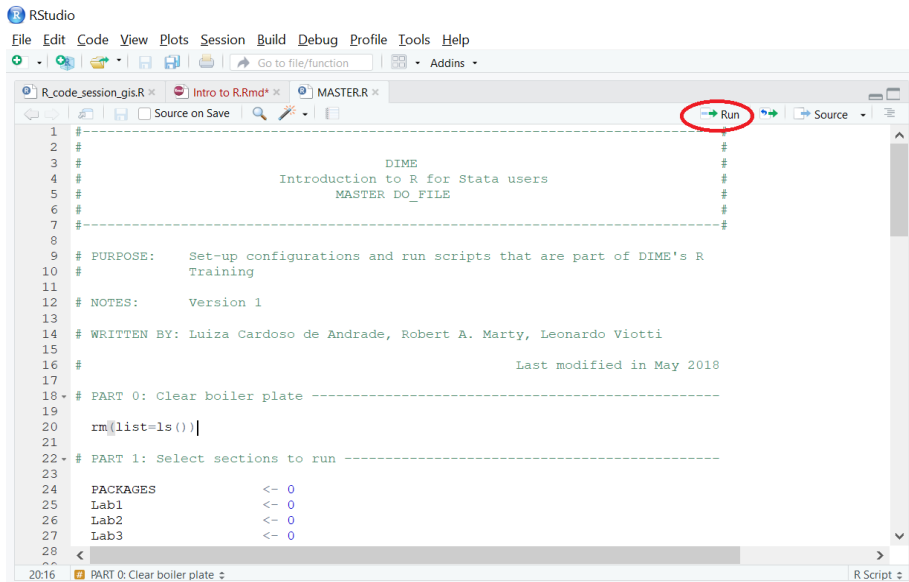
Name	Size	Modified
.rhistory	8.9 KB	Jun 13, 2018, 10:10
GIS DataBase		
GitHub		
R		
temp3.log	48.3 KB	May 26, 2018, 7:50
temp3.pdf	152.8 KB	May 26, 2018, 7:50

RStudio interface



```
1 #-----#
2 #                                           #
3 #                               DIME          #
4 #       Introduction to R for Stata users    #
5 #               MASTER SCRIPT              #
6 #                                           #
7 #-----#
8
9 # PURPOSE:   Set-up configurations and run scripts that are part of DIME's R
10 #           Training
11
12 # NOTES:
13
14 # WRITTEN BY: Luiza Cardoso de Andrade, Robert A. Marty, Leonardo Viotti
15
16 #                                           Last modified in May 2018
17
18 # PART 0: Clear boiler plate -----
19
20 rm(list=ls())
21
22 # PART 1: Select sections to run -----
23
24 # PART 0: Clear boiler plate
```

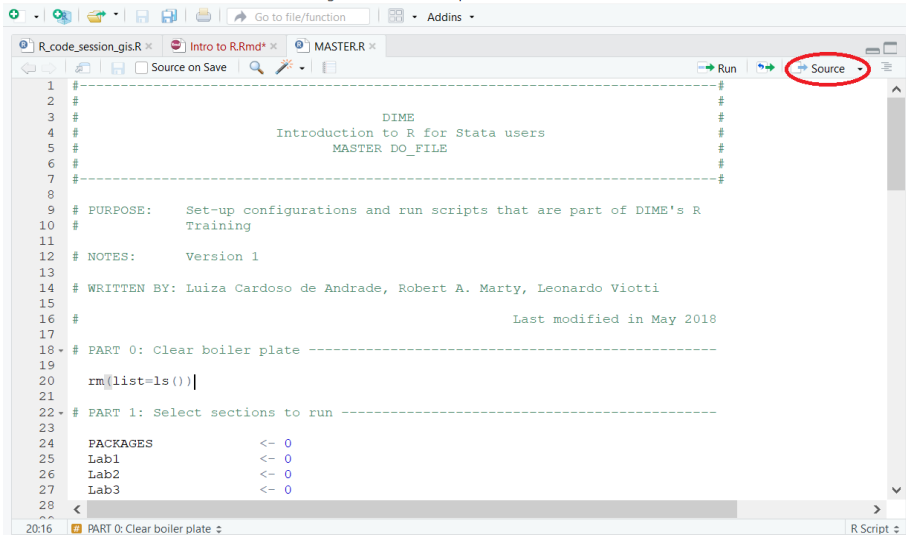
RStudio interface



RStudio interface

RStudio

File Edit Code View Plots Session Build Debug Profile Tools Help

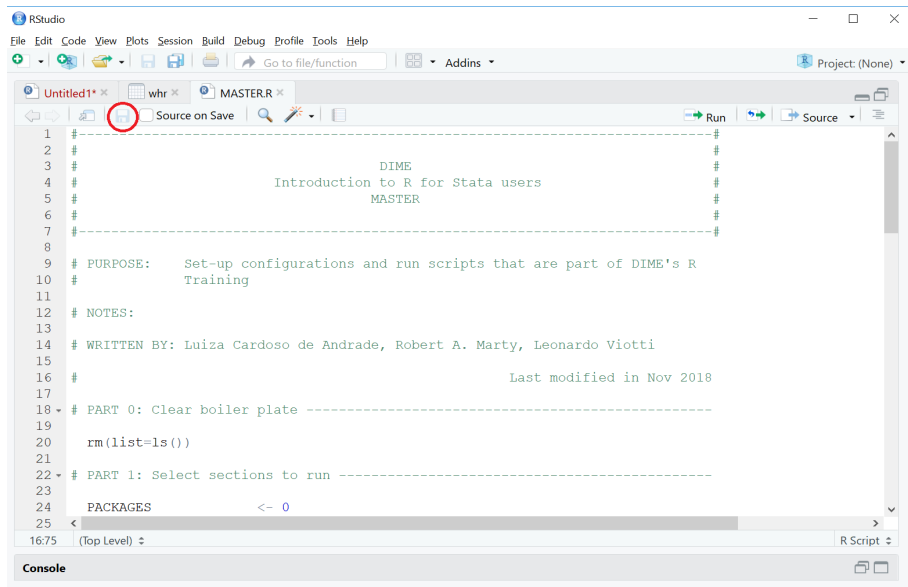


The screenshot shows the RStudio interface with the Source editor open. The top toolbar contains buttons for Run, Source (circled in red), and other functions. The Source editor displays R code for a script named 'MASTER.R'. The code includes comments about the purpose and notes, followed by a section for clearing the boiler plate.

```
1 # -----#
2 # #
3 # #
4 # DIME #
5 # Introduction to R for Stata users #
6 # MASTER DO_FILE #
7 # -----#
8
9 # PURPOSE: Set-up configurations and run scripts that are part of DIME's R
10 # Training
11
12 # NOTES: Version 1
13
14 # WRITTEN BY: Luiza Cardoso de Andrade, Robert A. Marty, Leonardo Viotti
15
16 # Last modified in May 2018
17
18 # PART 0: Clear boiler plate -----
19
20 rm(list=ls())
21
22 # PART 1: Select sections to run -----
23
24 PACKAGES <- 0
25 Lab1 <- 0
26 Lab2 <- 0
27 Lab3 <- 0
28
29 <
30
31 >
```

20:16 PART 0: Clear boiler plate R Script

RStudio interface



Outline

- 1 Introduction
- 2 Getting started
- 3 RStudio interface
- 4 Object-oriented language**
- 5 R objects
- 6 Basic types of data
- 7 Advanced types of data
- 8 Help, Google and Stackoverflow
- 9 Useful resources
- 10 Appendix

- 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.

- 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.

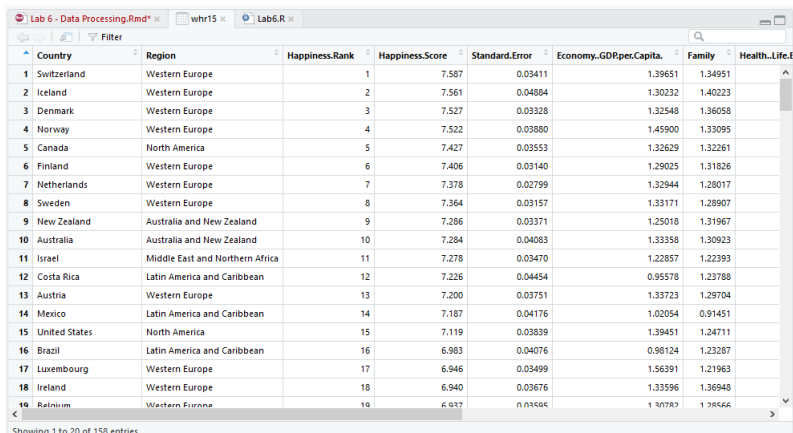
Object-oriented language

To better understand the idea, we're going to use the data from the United Nations' World Happiness Report. 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(whr)  
  
# Alternatively we can print the first 6 obs. with head()  
head(whr)
```


Object-oriented language



Country	Region	Happiness.Rank	Happiness.Score	Standard.Error	Economy..GDP.per.Capita.	Family	Health..Life.E
1 Switzerland	Western Europe	1	7.587	0.03411	1.39651	1.34951	
2 Iceland	Western Europe	2	7.561	0.04884	1.30232	1.40223	
3 Denmark	Western Europe	3	7.527	0.03328	1.32548	1.36058	
4 Norway	Western Europe	4	7.522	0.03880	1.45900	1.33095	
5 Canada	North America	5	7.427	0.03553	1.32629	1.32261	
6 Finland	Western Europe	6	7.406	0.03140	1.29025	1.31826	
7 Netherlands	Western Europe	7	7.378	0.02799	1.32944	1.28017	
8 Sweden	Western Europe	8	7.364	0.03157	1.33171	1.28907	
9 New Zealand	Australia and New Zealand	9	7.286	0.03371	1.25018	1.31967	
10 Australia	Australia and New Zealand	10	7.284	0.04083	1.33358	1.30923	
11 Israel	Middle East and Northern Africa	11	7.278	0.03470	1.22857	1.22393	
12 Costa Rica	Latin America and Caribbean	12	7.226	0.04454	0.95578	1.23788	
13 Austria	Western Europe	13	7.200	0.03751	1.33723	1.29704	
14 Mexico	Latin America and Caribbean	14	7.187	0.04176	1.02054	0.91451	
15 United States	North America	15	7.119	0.03839	1.39451	1.24711	
16 Brazil	Latin America and Caribbean	16	6.983	0.04076	0.98124	1.23287	
17 Luxembourg	Western Europe	17	6.946	0.03499	1.56391	1.21963	
18 Ireland	Western Europe	18	6.940	0.03676	1.33596	1.36948	
19 Belgium	Western Europe	19	6.937	0.03595	1.30782	1.28565	

Showing 1 to 20 of 158 entries

Object-oriented language

```
head(whr)
```

```
##      country      region year happy_rank happy_score  gdp_pc  family
## 1      Norway Western Europe 2017         1      7.537 1.616463 1.533524
## 2      Denmark Western Europe 2017         2      7.522 1.482383 1.551122
## 3      Iceland Western Europe 2017         3      7.504 1.480633 1.610574
## 4 Switzerland Western Europe 2017         4      7.494 1.564980 1.516912
## 5      Finland Western Europe 2017         5      7.469 1.443572 1.540247
## 6 Netherlands Western Europe 2017         6      7.377 1.503945 1.428939
##      health  freedom trust_gov_corr generosity dystopia_res
## 1 0.7966665 0.6354226      0.3159638  0.3620122      2.277027
## 2 0.7925655 0.6260067      0.4007701  0.3552805      2.313707
## 3 0.8335521 0.6271626      0.1535266  0.4755402      2.322715
## 4 0.8581313 0.6200706      0.3670073  0.2905493      2.276716
## 5 0.8091577 0.6179509      0.3826115  0.2454828      2.430182
## 6 0.8106961 0.5853845      0.2826618  0.4704898      2.294804
```

Object-oriented language

Now, let's try some simple manipulations. First, assume we're only interested in data of the year 2015.

Exercise 2: Subset the data

- 1 Subset the data set, keeping only observations where variable year equals 2015.

To do that we'll use the subset() function
`subset(whr, year == 2015)`

- 2 Then, look again at the first 6 observations

Use the head() function again
`head(whr)`

Object-oriented language

```
head(whr)
```

```
##      country      region year happy_rank happy_score  gdp_pc  family
## 1      Norway Western Europe 2017         1      7.537 1.616463 1.533524
## 2      Denmark Western Europe 2017         2      7.522 1.482383 1.551122
## 3      Iceland Western Europe 2017         3      7.504 1.480633 1.610574
## 4 Switzerland Western Europe 2017         4      7.494 1.564980 1.516912
## 5      Finland Western Europe 2017         5      7.469 1.443572 1.540247
## 6 Netherlands Western Europe 2017         6      7.377 1.503945 1.428939
##      health  freedom trust_gov_corr generosity dystopia_res
## 1 0.7966665 0.6354226      0.3159638  0.3620122      2.277027
## 2 0.7925655 0.6260067      0.4007701  0.3552805      2.313707
## 3 0.8335521 0.6271626      0.1535266  0.4755402      2.322715
## 4 0.8581313 0.6200706      0.3670073  0.2905493      2.276716
## 5 0.8091577 0.6179509      0.3826115  0.2454828      2.430182
## 6 0.8106961 0.5853845      0.2826618  0.4704898      2.294804
```

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 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).

Exercise 3: Create an object

Create a new data set, called `whr2015`, that is a subset of the `whr` data set containing only data from the year 2015.

Using the same function but now assigning it to an object

```
whr2015 <- subset(whr, year == 2015)
```

Display the 5 first obs. of the new data

```
head(whr2015)
```

Notice that we still have the original data set intact

```
head(whr)
```

Object-oriented language

```
head(whr2015)
```

```
##          country      region year happy_rank happy_score  gdp_pc  family
## 313 Switzerland Western Europe 2015         1      7.587 1.39651 1.34951
## 314      Iceland Western Europe 2015         2      7.561 1.30232 1.40223
## 315      Denmark Western Europe 2015         3      7.527 1.32548 1.36058
## 316       Norway Western Europe 2015         4      7.522 1.45900 1.33095
## 317       Canada North America 2015         5      7.427 1.32629 1.32261
## 318       Finland Western Europe 2015         6      7.406 1.29025 1.31826
##          health freedom trust_gov_corr generosity dystopia_res
## 313 0.94143 0.66557      0.41978      0.29678      2.51738
## 314 0.94784 0.62877      0.14145      0.43630      2.70201
## 315 0.87464 0.64938      0.48357      0.34139      2.49204
## 316 0.88521 0.66973      0.36503      0.34699      2.46531
## 317 0.90563 0.63297      0.32957      0.45811      2.45176
## 318 0.88911 0.64169      0.41372      0.23351      2.61955
```

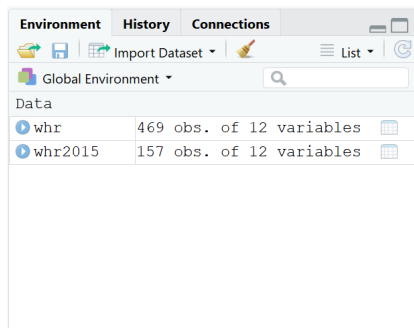
Object-oriented language

```
head(whr)
```

```
##      country      region year happy_rank happy_score  gdp_pc  family
## 1      Norway Western Europe 2017         1      7.537 1.616463 1.533524
## 2      Denmark Western Europe 2017         2      7.522 1.482383 1.551122
## 3      Iceland Western Europe 2017         3      7.504 1.480633 1.610574
## 4 Switzerland Western Europe 2017         4      7.494 1.564980 1.516912
## 5      Finland Western Europe 2017         5      7.469 1.443572 1.540247
## 6 Netherlands Western Europe 2017         6      7.377 1.503945 1.428939
##      health  freedom trust_gov_corr generosity dystopia_res
## 1 0.7966665 0.6354226      0.3159638 0.3620122      2.277027
## 2 0.7925655 0.6260067      0.4007701 0.3552805      2.313707
## 3 0.8335521 0.6271626      0.1535266 0.4755402      2.322715
## 4 0.8581313 0.6200706      0.3670073 0.2905493      2.276716
## 5 0.8091577 0.6179509      0.3826115 0.2454828      2.430182
## 6 0.8106961 0.5853845      0.2826618 0.4704898      2.294804
```


Object-oriented language

You can also see that your environment pane now has two objects:



Two important concepts to take note:

- ❶ In R, if you want to change your data, you need to store it in an object.
- ❷ It is possible to simply replace the original data. This happens if you assign the new object to the same name as the original.
- ❸ Print (display) is built into R. If you execute any action without storing it, R will simply print the results of that action but won't save anything in the memory.

Outline

- 1 Introduction
- 2 Getting started
- 3 RStudio interface
- 4 Object-oriented language
- 5 R objects**
- 6 Basic types of data
- 7 Advanced types of data
- 8 Help, Google and Stackoverflow
- 9 Useful resources
- 10 Appendix

Objects are the building blocks of R programming. This section will explore some of the most common.

This will give you the foundation to explore your data and construct analytical outputs.

Here are the types of object we will cover today:

- **Vectors:** an uni-dimensional 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 uni-dimensional 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 `whr` is a vector.

Calling a vector from a `data.frame` column

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

Type the following code:

```
# Create a vector with the values of the `year` variable  
year_vec <- whr$year  
  
# See the 3 first elements of the year column  
whr$year[1:3]  
  
## [1] 2017 2017 2017
```

R objects

Data Frames

The `whr` and `whr2015` objects are both data frames. You can also construct a new data frame from scratch by combining vectors with the same number of elements .

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 whr

```
whr[,1]
```

The 45th line of whr

```
whr[45,]
```

Or the 45th element of the first line

```
whr[45,1]
```

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

Names indexing

```
# Or the 22th element of the country column  
whr[22, "country"] # The same as whr$country[22]
```

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

```
## 164 Levels: Afghanistan Albania Algeria Angola Argentina ..
```

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

- 1 Introduction
- 2 Getting started
- 3 RStudio interface
- 4 Object-oriented language
- 5 R objects
- 6 Basic types of data**
- 7 Advanced types of data
- 8 Help, Google and Stackoverflow
- 9 Useful resources
- 10 Appendix

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).

Basic types of data

Strings

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",  
             "Excel",  
             "Stata")
```

Now print them to check them out.

Basic types of data

Strings

Exercise 5: Concatenate strings

- 1 Create a scalar (a vector of one element) containing the phrase “is better than” and call 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

- 1 Introduction
- 2 Getting started
- 3 RStudio interface
- 4 Object-oriented language
- 5 R objects
- 6 Basic types of data
- 7 Advanced types of data**
- 8 Help, Google and Stackoverflow
- 9 Useful resources
- 10 Appendix

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 your memory.

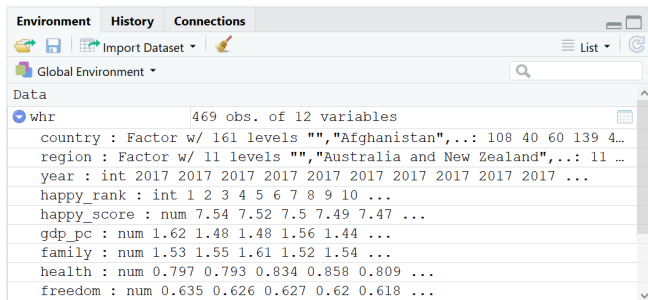
Booleans

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

Advanced types of data

Factors

In `whr`, we can see that `country` and `region` are factor variables. In your environment panel, there is the information of the type of all variables, and for factors the number of levels.



The screenshot shows the RStudio Environment panel with the 'Global Environment' selected. The 'Data' section lists the 'whr' dataset, which contains 469 observations of 12 variables. The variables and their types are as follows:

Variable	Type	Levels / Values
country	Factor	w/ 161 levels: "", "Afghanistan", ...
region	Factor	w/ 11 levels: "", "Australia and New Zealand", ...
year	int	2017
happy_rank	int	1 2 3 4 5 6 7 8 9 10 ...
happy_score	num	7.54 7.52 7.5 7.49 7.47 ...
gdp_pc	num	1.62 1.48 1.48 1.56 1.44 ...
family	num	1.53 1.55 1.61 1.52 1.54 ...
health	num	0.797 0.793 0.834 0.858 0.809 ...
freedom	num	0.635 0.626 0.627 0.62 0.618 ...

Advanced types of data

Factors

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 the result of logical conditions

- Whenever you're using an `if` statement in Stata, you're implicitly using boolean data.
- The difference is that in R, this can be done in 2 steps.

Advanced types of data

Booleans

Exercise 6:

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

```
# Create vector
bool_vec <- (whr$happy_rank <
             mean(whr$happy_rank))

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

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

Advanced types of data

Booleans

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

Exercise 6:

- 1 Create a column in `whr` containing zeros and call it `rank_low`. You can do this by typing:

```
whr$rank_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.

```
whr$rank_low[bool_vec] <- 1
```


Advanced types of data

Booleans

```
# Replace with 1 those obs that meet the condition  
whr$rank_low [bool_vec] <- 1  
# is the same as  
whr$rank_low[whr$happy_rank < mean(whr$happy_rank)] <- 1  
# This in stata would be  
# replace rank_low = 1 if (...)
```

```
# We can use the summary function to get descriptives  
summary(whr$rank_low)
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.  
## 0.0000  0.0000  0.0000  0.4979  1.0000  1.0000
```

Outline

- 1 Introduction
- 2 Getting started
- 3 RStudio interface
- 4 Object-oriented language
- 5 R objects
- 6 Basic types of data
- 7 Advanced types of data
- 8 Help, Google and Stackoverflow**
- 9 Useful resources
- 10 Appendix

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
```

- 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 something yields tons of results.
- Often that means a Stack Overflow page where someone asked the same question and several people gave different answers. Here's a typical example: <https://stackoverflow.com/questions/1660124/how-to-sum-a-variable-by-group>

¹Check <http://r4stats.com/articles/popularity/> for more.

Outline

- 1 Introduction
- 2 Getting started
- 3 RStudio interface
- 4 Object-oriented language
- 5 R objects
- 6 Basic types of data
- 7 Advanced types of data
- 8 Help, Google and Stackoverflow
- 9 Useful resources**
- 10 Appendix

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/>

Books:

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

Thank you!

Outline

- 1 Introduction
- 2 Getting started
- 3 RStudio interface
- 4 Object-oriented language
- 5 R objects
- 6 Basic types of data
- 7 Advanced types of data
- 8 Help, Google and Stackoverflow
- 9 Useful resources
- 10 Appendix

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.

Appendix - 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.

Appendix - Matrices

A matrix is a bi-dimensional 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)
```

Appendix - 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]
```

Appendix - Advanced types of data - Factors

Factors

Create a factor vector 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')
```

Appendix - 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
```

Appendix - 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
```


Appendix - Numbers and integers

Numbers and integers

We can, however, coerce objects into different classes. We just need to be careful 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
```