Econometrics I - R summary

Maite Cabeza-Gutes

February,2021

Contents

1 Introduction to *RStudio* **3**
- 1.1 Installing *RStudio* . . . 3
- 1.2 *RStudio* layout . . . 3
- 1.3 R as a calculator . . . 7
- 1.4 *R* scripts . . . 8
- 1.5 RMarkdown documents . . . 10

2 Data objects: variables and dataframes **13**
- 2.1 Creating a variable by entering its values . . . 13
- 2.2 Creating a dataframe . . . 17
- 2.3 Importing data files . . . 20
- 2.4 Saving objects in the workspace as R data objects . . . 25
- 2.5 Basic data manipulation . . . 26

3 Simulating random variables **28**
- 3.1 Simulating a discrete random variable . . . 28
- 3.2 Simulating a continuos random variable . . . 28

4 Statistical tables **30**
- 4.1 Statistical tables: t-distribution . . . 30
- 4.2 Statistical tables: F-distribution . . . 31

5 Basic descriptive statistics **32**
- 5.1 Measures of location . . . 32
- 5.2 Measures of dispersion . . . 33
- 5.3 Measures of shape . . . 33
- 5.4 Measures of linear association . . . 34

6 Graphs using ggplot2 package **35**
- 6.1 Basic structure . . . 35
- 6.2 One variable plots: Histograms . . . 36
- 6.3 2 variable plots: Scatters . . . 39

7 Vectors and matrices in R **44**
- 7.1 Creating vectors and matrices using function **matrix()** . . . 44
- 7.2 Matrix Operations . . . 47

8 Simple regression model: *OLS* estimation **51**
- 8.1 *OLS* estimation . . . 51
- 8.2 *OLS* estimation using matrix algebra . . . 52
- 8.3 Coefficient of determination . . . 53
- 8.4 *OLS* estimation: standard errors . . . 53
- 8.5 *OLS* estimation using function **lm()** . . . 54
- 8.6 Plotting observations and fitted line . . . 57

9 Simple regression model: inference with t statistic 58
9.1 Significance test of each regressor (t-values, p-values, $*$) . . . 58
9.2 Confidence intervals . . . 59

10 Simple regression model: prediction 61
10.1 Point prediction . . . 61
10.2 Interval prediction . . . 61

11 Multiple regression model: estimation 62
11.1 *OLS* estimation . . . 62
11.2 Googness of fit . . . 63
11.3 *OLS* estimation using function **lm()** . . . 65
11.4 Calculating variance inflating factors . . . 69
11.5 Multiple regression estimation: logs, polynomial forms, interaction terms . . . 69

12 Simulating behavior of *OLS* estimator 71
12.1 Generating a sample from a given data generating process *dgp* . . . 71
12.2 Monte Carlo experiments . . . 71

13 Multiple regression model: inference 73
13.1 Statistics for significance test of each regressor (t-values, p-values) . . . 73
13.2 Confidence intervals . . . 74
13.3 Inference with F statistic using function **linearHypothesis()** . . . 75

14 Multiple regression model: prediction 76
14.1 Point prediction . . . 76
14.2 Interval prediction . . . 77

1 Introduction to *RStudio*

1.1 Installing *RStudio*

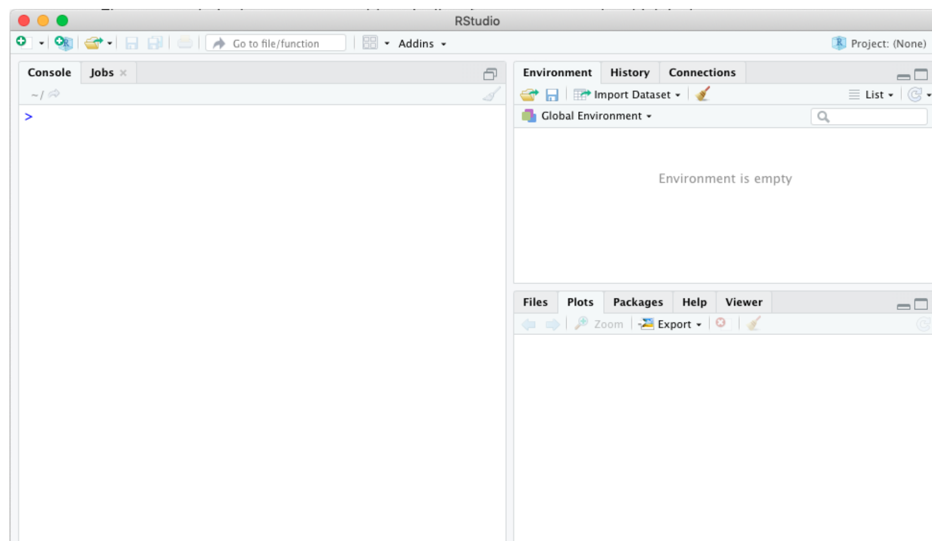
R is an open source language and environment for statistical computing and graphics. *RStudio* is an IDE (Integrated Development Environment) that makes working with *R* much easier. To use *RStudio* we need to first install *R* and then, install *RStudio*.

- Step 1: Use https://cran.r-project.org/ to install *R* in your computer.
- Step 2: Use https://www.rstudio.com/products/rstudio/download/#download to install *RStudio*.

Once they are both installed, only *RStudio* needs to be launched to start working.

1.2 *RStudio* layout

1.2.1 Initial layout: 3 windows



- **Left window**: console or command window.
- **Top right window**: environment, history tabs.
- **Bottom right window**: files, plots, packages, help, viewer tabs.

As we shall see, once we introduce *R* scripts, a fourth window, the **Script window**, will appear above the console.

1.2.2 Setting your working directory

- To set your working directory: Use top menu option: *Session/Set Working Directory/Choose Directory* to select which folder will act as your working directory in a given session.
- To get information of which folder is set to be your current directory use command **getwd()**. You can also use the **File** tab (bottom-right window) to which folder is set as your working directory.
- For a quick access to the files included in your working directory use **Files** tab (bottom-right window). If we click on a file in the working directory it will automatically open if it is a .R file (R script) or .Rmd (R Markdown). If we click on a R data file (.Rda or RData), we can directly loaded into our Environment. If click on another type of data file (.xls, .xlsx, .csv,..) we can be easily import it. Finally, if we click on a .pdf file, it will automatically open in our pre-set pdf reader.

1.2.3 Commands

- Once we open *RStudio*, *R* is waiting for your commands. A *command* is an instruction given to your computer in *R* programming language.
- Commands can be given by entering them in the console using *R* language. Some particular commands can also be entered using *RStudio* menu.
- The prompt sign (‘>’) in the console indicates that *R* is waiting for a command.
- After typing a command, we hit the *Return* key to run it. (*Sometimes, information or warning messages appear in red, even if no error is made. Don't worry!*)
- Use command **help()** (or, **?**) to get help on any command. Example **help(remove)** (or, **?remove**). An explanation about the command in question appears in the **Help** tab (bottom-right window). Alternatively, you can also use the **Help** tab directly and type the name of the command you need help with, like ‘remove’ (e.g.), in the corresponding search box.
- Important: *R* is case-sensitive!
- To clear the console use ctrl+L keys (or the corresponding broom-icon).
- Comments can be added by using hashtag symbol ‘#’. *R* will ignore anything to the right of a #. The ‘#’ can be written at the beginning of a line of next to a command. Example:

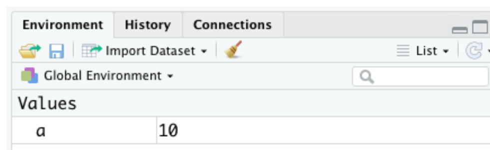
```
# Example 1
log(10) # this command calculates natural log of 10
```

```
[1] 2.302585
```

1.2.4 Objects

- *R* stores information in so called *objects*. Everything we generate using commands (variables, new data sets, statistics, plots,...) is stored in R as an object. By now, we will focus on data objects.
- To access a defined object at any point, or use it to define new objects, we need to assign it a name. This is done using the *assignment operator*:**<-** (or =). *Careful!* The name given can only contain letters and numbers, but it cannot start with a number. Once the name of a data object is assigned, the object with its given name appears listed in the **Environment** tab (top-right window) window. Example:

```
#Assign name "a" to scalar 3+7
a<-3+7
```



- If we want to see the object displayed in the console, we need to call it by typing the assigned name.

```
a<-3+7
a
```

```
[1] 10
```

- Careful: If we define an object that contains 1000 different values, it will not be a good idea to have it displayed in the console!
- Once a name is assigned to an object, it can be used in other commands. Example:

```
a<-3+7
b<-2*a
b
```

```
[1] 20
```

- Notice that if we assign the same name to a different object, its original value will be replaced by the latest expression. Example:

```
#Now, assign name "a" to 3+6
a<-3+7
a
```

```
[1] 10
```

```
a<-3+9
a
```

```
[1] 12
```

- A specific, say object A, can be removed from the **Environment** tab (top-right window), by using command **remove(A)**, or **rm(A)**, for short. To clear all listed objects use command **rm(list=ls())** or the tab broom-icon.
- Not assigning a name to an object: If an object is generated, but we do not assign it a name, the object will be listed in the console. This is ok, if we are generating object that contain very few values. But, it is a bad idea, if the object includes a large number of values, as they would be displayed in the console. Also, if an object is generated without a name, we will not be able to retrieve unless we generate it again.
- Plots are also stored as R objects. When a plot is generated without assign it a name, it will automatically be displayed in the **Plots** tab (bottom-right window). As for data objects, we can also assign a name to a plot using the *assignment operator*:**<-** (or =). Plots can be exported or copied using the corresponding menu. The corresponding broom icon can be used to clear an undesired plot.

1.2.5 Functions

- A function, in a programming environment, is a set of instructions, or operations, we want to perform to an object.
- A function is followed by a parentheses. It may or may not include arguments.
- Some initial examples:

```
# Square root
sqrt(9)
```

```
[1] 3
```

```r
# Generate a sequence of integers from 1 to 10
seq(1:10)
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

```r
# Calculate the mean
Z<-seq(1:10)
mean(Z)
```

```
[1] 5.5
```

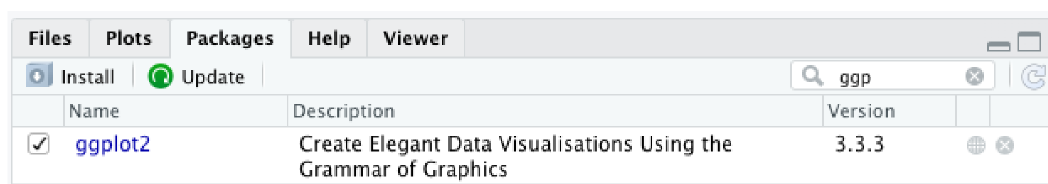
R has many functions ready to be used (*built-in* functions), but any user can create their own. In our course we will only use predefined functions.

1.2.6 Packages

- There are many built-in functions ready to be used after installing *R* and *RStudio*. Many more functions are available by installing extensions, also known as *packages* or *libraries*.
- After installing R/RStudio, you can check the list of packages installed by default by going to **Packages** tab (bottom-right window).
- To use the functions included in a given package:
  - (1) the package needs to have been installed (only once)
  - (2) the package needs to be activated (loaded) during a given *RStudio* session (i.e., every session).
- To install a package use function **install.packages()**. Example: to install package **ggplot2** (package we will use in this course for data visualization):

```r
install.packages("ggplot2")
```

- After installation, the package will be listed in the **Packages** tab (bottom-right window).
- You will have to install the necessary packages for the course in your personal computer, but not in the computers on campus. Hence, do not run the **install.packages()** command in any of the computers in classroom 24, 25 or 26! All the packages we need have already been installed by authorized personnel.
- If you are unsure whether a package has already been installed and hence, avoid re-installation, use command:
- To activate a given package, two options available:
  - Option 1: Use command **library()**. Example, to use **ggplot2** package in a given session:
  - Option 2: locate the package needed using the search box in the **Packages** tab (bottom-right window) and tick the corresponding box.



- List of packages you will need to install in your personal computer for this course:
  - **readr**
  - **readxl**
  - **stargazer**
  - **ggplot2**
  - **gridExtra**
  - **car**
  - **rmarkdown**
  - **knitr**
  - **AER**
  - **MASS**
  - **datasets**
  - **utils**

1.3 R as a calculator

1.3.1 Basic operators

- Addition : +
- Subtraction: -
- Multiplication: *
- Division: /
- Exponentiation: ^

1.3.2 Mathematical functions we will use frequently:

- **srqt()** (square root)
- **log()** (natural log)
- **exp()** (exponential)

1.3.3 Some examples:

- Square root:

```
a<-121
b<-sqrt(a)
b
```

```
[1] 11
```

- Exponential function:

```
e<-exp(1)
e
```

```
[1] 2.718282
```

- Logs:

```
log10(100)
```

```
[1] 2
```

```
e<-exp(1)
log(e)
```

```
[1] 1
```

```
log(1)
```

```
[1] 0
```

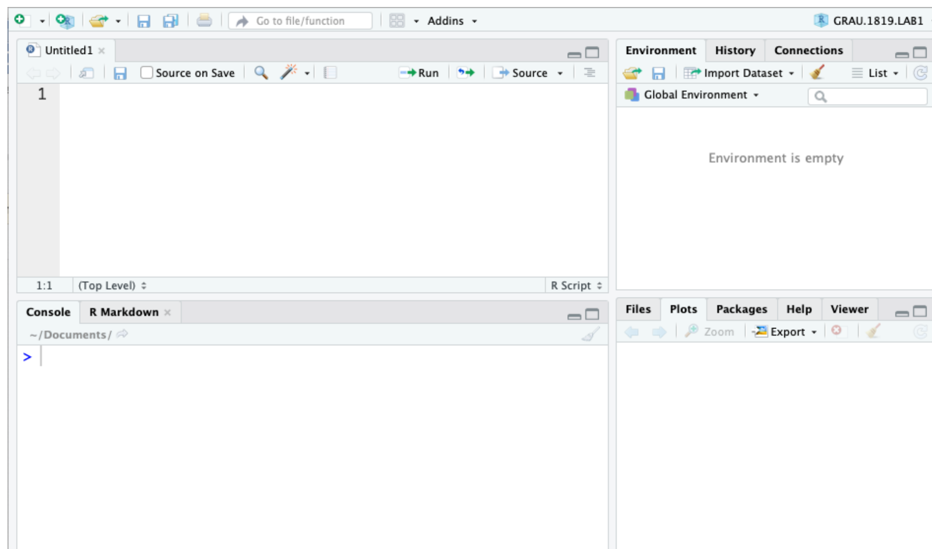
```
log(0)
```

```
[1] -Inf
```

1.4 *R* scripts

1.4.1 Using scripts

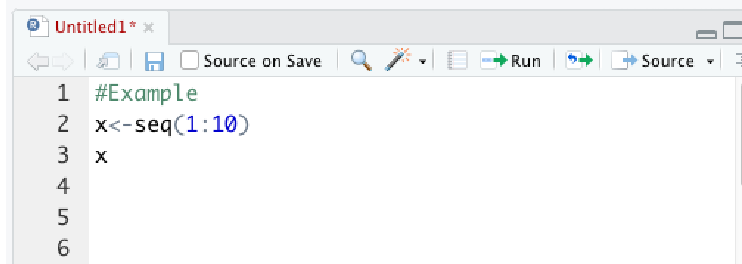
- A *R* script is a plain text files that contains a set of command lines written in *R* code. These scripts are typically files with extension **.R**.
- We create *R* scripts to type a set of commands we wish to run, save it, and use it later on or easily edit it. We can also open *R* scripts created by other users.
- The command lines included in a given script can be run one after the other, or all at once.
- After we select to create a new script, or we open an existing one, *RStudio* will display four windows, instead of three:



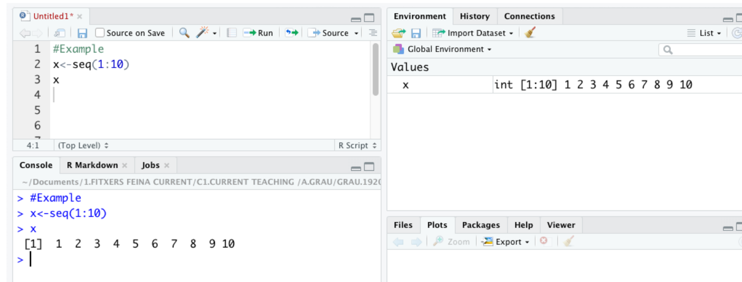
1.4.2 Creating a new *R* script

- To create a new *R* script we can use the menu: *File/New File/R Script*. Alternatively, you can use the icon with the green + sign (top-left icon in the RStudio window), that displays a drop down menu, and select *R Script*.
- Once a new script is opened, a blank file is ready for us to enter the desired commands. Commands lines are numbered and entered one after the other.

- As for the case of the console, comments can be added into a script by using hashtag symbol #. As explained before, *R* will ignore anything written to the right of #.
- To run a single command line: place cursor in the desired command line (or highlight the line) and click at the run icon, located on the top right of the **script** window. Example:



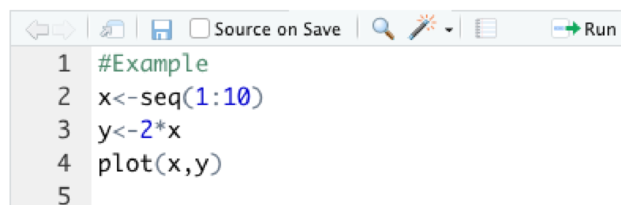
- To run a group of command lines at once: select the desired command lines (highlight all of them) and click on the **Run** icon.
- The command(s) you run in a script will be automatically transferred to the console and executed. Example:



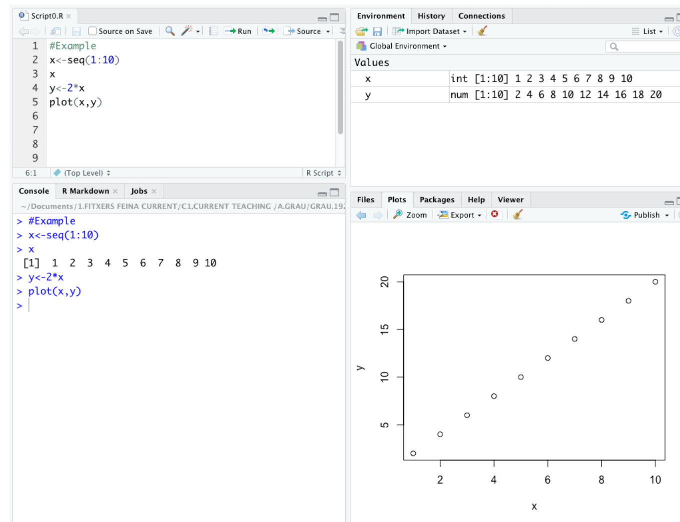
- To save the script use menu option *File/Save*. After a script is saved, a file with the given name and *.R* extension will be added to your working directory.

1.4.3 Openning an existing *R* script

- To open an existing script use *File/Open file....*, which will take you directly to the folder you established as your working directory. If the file is not located in your working directory, you can browse to locate it.
- If your file is in the set working directory, it can also be openned by going to the **Files** tab in the bottom-right window, and clicking on it.
- Example: Open saved script *Script0.R* and edit it:



- After running the script, variables generated appear in the **Environment** tab (top-right window) and the plot in the **PLots** tab (bottom-right window).



1.5 RMarkdown documents

1.5.1 About R Markdown documents

- *R Markdown* documents are dynamic documents that combine text, math expressions, *R* code (i.e., a set of commands) to run, and the associated output(s).
- *Rstudio* allow us to create, edit and run *R Markdown* documents. Packages **rmarkdown** and **knitr** are needed.
- *R* code is inserted in a *R Markdown* document inside a so called *code chunks*. We can include a single code chunk, or several code chunks distributed along the document.
- Each code chunk can be run individually, using the corresponding run icon (green triangle on the top-right of the chunk). In this case, the generated output (tables, plots,..) appear just below the code chunk.
- You can learn more on your own about markdown language, but just as a brief note about text format: ** Use hashtag (#) to indicate beginning of a section, two hashtags (##) to indicate beginning of a sub-section,. . . . ** Place a word between asterik () *to get word in* italics* **Place a word between two asterisk ()** to get word in **bold Begin a line with a single asterisk to create a list (itemize)** Place a math expression between 1 ()$or 2 dollar signs$($) to include a math expression.

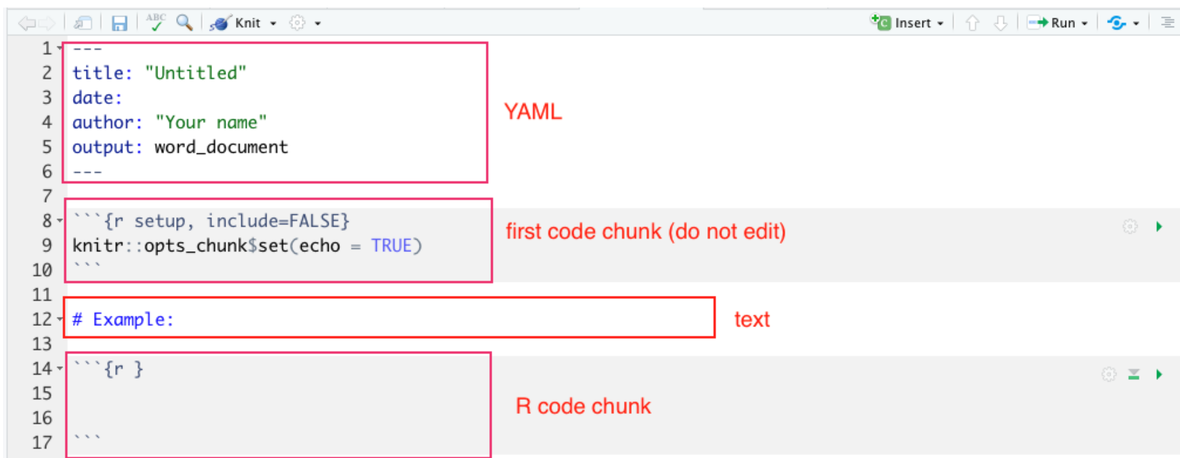
1.5.2 Creating and saving a R Markdown file

- To create a new *R Markdown* document use menu option *File/New File/ R markdown*. Alternatively, you can use the icon with the green + sign (top-left icon in the RStudio window), that displays a drop down menu, and select *R Markdown*. In this course, the basic *R Markdown* document will always be provided.
- To save the *R Markdown* use menu option File/Save. After a *R Markdown* is saved, a file with **.Rmd** extension will be added in your working directory.
- After verifying all code chunks run properly, we can produce a single document, including all the text and all the outputs. The document generated from a *R Markdown* file can be saved in different formats: *.doc*, *.html* or *.pdf*. Important: For this course we will only work with *.doc*'s. *Do not use the pdf option!*

1.5.3 Working with *R Markdown* documents: An Example

- Initial document:

- Main parts of an *R Markdown* file:



Document begins with a header (called the *YAML* header). This header is placed between 3 dash symbols (—).

Important: You should only edit text between "" under *title*, *date* and *author*. Do not edit anything else in the header. That is do not add or remove spaces, neither edit the output specification with in *YAML* header.

Below the *YAML* header you will find a first code chunk label as *setup*. Please ignore this code. Do not edit it, neither run it. Leave it as is. ** Rest of the document can be edited as you please.

In this case the document includes a section named *Example*, a math expression $y = 2x$, and an R code chunk ready for R code to be entered.

- We edit this basic document by entering R code in the code-chunk. Example:

```
---
title: "Hello World"
date:
author: "Your name"
output: word_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

Example

Generating two variables x and y, where:

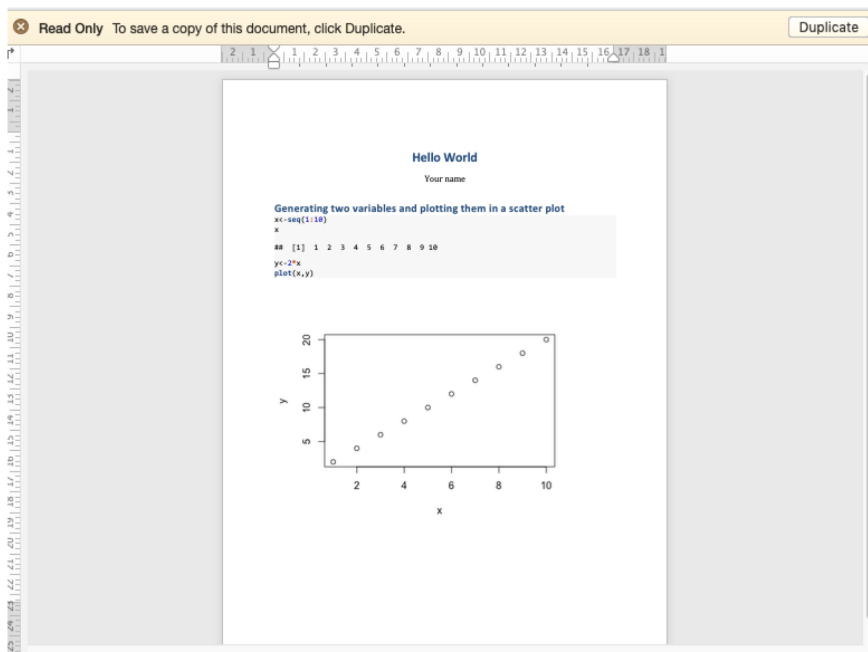
$$
y=2x
$$

and then, plotting them in a scatter plot.

```{r }
x<-seq(1:10)
x
y<-2*x
plot(x,y)

```
```

- Using menu option *File/Save* we can save the *RMarkdown*.
- We can use the *Knit* icon to generate the *Word* version of our dynamic document.
- After knitting, the *Word* document opens into *Word* as a 'read only' document in your working directory.



- To further edit the *.doc* document in Word, you need to close it, and open it again, and proceed as usual when working with *.doc* documents. After you are happy with the document, you can save it as a *.pdf* copy, if needed.
- File *Myfirst.Rmd* includes this basic example of what an *Rmarkdown* document looks like. You should practice editing and knitting this document in your computer.

2 Data objects: variables and dataframes

In this section we will consider two types of data objects: variables (*vectors*) and collections of variables organized in two-entry tables (*dataframes*).

The data to be analyzed needs to be available in our **Environment** tab (top-right window).

We will always assign a name to the variables and dataframes we create.

2.1 Creating a variable by entering its values

We can create a variable and enter its values using **c()** function. This function combines the elements we enter inside the parenthesis to form an object (*vector*) saved under the name indicated by the assign operator (**<-**).

Elements inside can be *numeric* (integer or not integer), *characters* , *factors* or *dates*. To get information of the type of variable we can use function **class()**.

Created variables with a name assigned will appear listed in the **Environment tab** (top-right window). To view its elements we can just type the name assigned, and the values will be displayed in the console. *Careful!* Do not use this option if the variable contains many elements.

Created data objects without a name assigned will not be listed in the Environment tab, but its elements will automatically be displayed in the console. Again, *careful!*

2.1.1 Creating a numeric variable:

- Example

```
Z<-c(2, 4.5, 6, 8, 10.1, 12)
Z
```

```
[1]  2.0  4.5  6.0  8.0 10.1 12.0
```

```
class(Z)
```

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

- Shortcut: to create a variable that includes a sequence of values we can use function **seq()**. Examples:

```
a<-seq(1:10)
a
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

```
c<-seq(2,5,by=0.5)
c
```

```
[1] 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

- Shortcut: to create a variable that is a constant (i.e., all its values are the same) we can use function **rep()**. Example:

```
d<-rep(1, 4)
d
```

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

2.1.2 Creating a variable including characters: Example

```
W<-c("Student A", "Student B", "Student C", "Student C")
W
```

```
[1] "Student A" "Student B" "Student C" "Student C"
```

```
class(W)
```

```
[1] "character"
```

2.1.3 Creating a categorical variable (factor)

- To identify a variable as a categorical variable we use the **factor()** function. The categorical variable can include two or more categories. Once a variable has been identified as a factor, we can use function **levels()** to see the different categories included in that variable. Example:

```
a<-c("male", "female","female", "male","male","male")
af<-factor(a)
class(af)
```

```
[1] "factor"
```

```
levels(af) # (levels are organized in alphabetic order!)
```

```
[1] "female" "male"
```

- The labels of a categorical variable can be changed to the usual 0,1 categories. Following the example above:

```
#
a<-c("male", "female","female", "male","male","male")
af<-factor(a, labels=c("1","0"))
af
```

```
[1] 0 1 1 0 0 0
Levels: 1 0
```

```
levels(af)
```

```
[1] "1" "0"
```

```
class(af)
```

```
[1] "factor"
```

- Notice that the categorical variable created defined as a binary variable includes categories "0" and "1". These are not treated as numerical (notice they are quoted inside ""). If we want the categorical variable to be treated as numeric we need to use **as.numeric()** function. In this case, the categories/levels of the variable are no longer recognized. Example:

```
a<-c("male", "female","female", "male","male","male")
af<-factor(a, labels=c("1","0"))
af<-as.numeric(as.character(af))
af
```

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

```
levels(af)
```

```
NULL
```

```
class(af)
```

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

2.1.4 Creating a time variable

- To create a variable that includes a sequence of dates we use **seq(as.Date())** function. For example, to create a variable named *Time* including monthly data with first 5 months of 2021:

```
Time<-seq(as.Date("2021/1/1"), by="month", length=5)
Time
```

```
[1] "2021-01-01" "2021-02-01" "2021-03-01" "2021-04-01" "2021-05-01"
```

```
class(Time)
```

```
[1] "Date"
```

2.1.5 Information about a variable

- To know the class of the variable use function **class()** as seen above.
- To know the length (i.e., number of elements) of a variable use function **length()**. Example:

```
Z<-c(2, 4.5, 6, 8, 10.1, 12)
length(Z)
```

```
[1] 6
```

- To list all the elements of a given variable we can type its name, as seen in the examples above. Do not use this option if the length of the variable is large.
- To list the first elements of a given variable use function **head()**. To view the last elements, use function **tail()**. Examples:

```
Z<-seq(1:100)
# List the first 6 elements (default)
head(Z)
```

```
[1] 1 2 3 4 5 6
```

```
# List the first 5 elements
head(Z, 5)
```

```
[1] 1 2 3 4 5
```

```
# List the last 6 elements (default)
tail(Z)
```

```
[1]  95  96  97  98  99 100
```

```
# List the first 5 elements
tail(Z, 5)
```

```
[1]  96  97  98  99 100
```

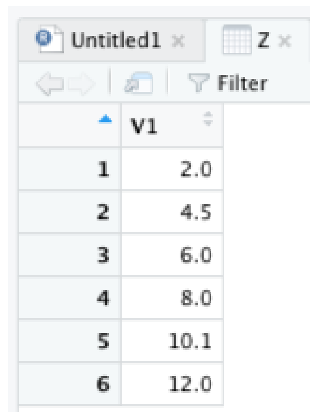
- To list specific elements of a given variable use the selection brackets [ ]. Example:

```
Z<-seq(1,100)
# to list elements from 10 to 15
Z[10:15]
```

```
[1] 10 11 12 13 14 15
```

- To browse the elements of a variable use function **View()**. The contents of the variable will be opened in a new tab in the top-left window. You can scroll up and down to see its elements. Example:

```
Z<-c(2, 4.5, 6, 8, 10.1, 12)
View(Z)
```



2.2 Creating a dataframe

Most of the data we will analyze will be in the form of a *dataframe*. A dataframe is a two dimensional data structure, arranged in a table layout containing values (rows for observations and columns for variables).

Variables included in a dataframe need not be of the same type. Some can be numeric (12.5, 2.3, 5,..), others characters ("Algeria", "Argentina", . . .), for example. Each variable in the dataframe is identified by a name. During a given R session, we can work with several dataframes simultaneously. This is a nice feature as it will allow us to work with different data sets at the same time.

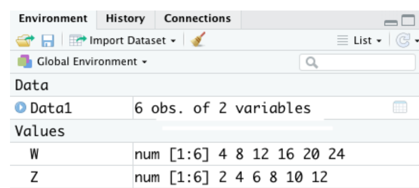
2.2.1 Creating a dataframe

- Variables that have been generated in a session can be combined into a dataframe using function **data.frame()** .
- We will always assign a name to a created dataframe using the assign operator ("**<-**"). Example:

```
Z<-c(2,4,6, 8, 10, 12)
W<-2*Z
Data1<-data.frame(Z,W)
Data1
```

```
   Z  W
1  2  4
2  4  8
3  6 12
4  8 16
5 10 20
6 12 24
```

- After a dataframe is created, it appears listed in our **Environment** tab. Example:



- New dataframes can be created combining data objects (variables, other dataframes) listed in our **Environment** tab (top-right window). Example:

```
Z<-c(2,4,6, 8, 10, 12)
W<-2*Z
Data1<-data.frame(Z,W)
Data1
```

```
   Z  W
1  2  4
2  4  8
3  6 12
4  8 16
5 10 20
6 12 24
```

```
Y<-Z-2
Data2<-data.frame(Data1,Y)
Data2
```

```
   Z  W  Y
1  2  4  0
2  4  8  2
3  6 12  4
4  8 16  6
5 10 20  8
6 12 24 10
```

Global Environment

Data	
Data1	6 obs. of 2 variables
Data2	6 obs. of 3 variables

2.2.2 Information about a dataframe

- To know the length (i.e., number of variables/columns) of a dataframe use function **length()**. Important, this function, when applied to a variable, gives the number of values it contains. If applied to a dataframe, it gives us the number of variables/columns included. Example:

```
Z<-seq(0,50, by=0.25)
W<-2*Z
Data1<-data.frame(Z,W)
length(Data1)
```

```
[1] 2
```

- To know the number of rows of values included for the variables in a given dataframe (i.e., number of rows) use function **nrow()**. Example:

```
Z<-seq(1, 50, by=0.25)
W<-2*Z
Data1<-data.frame(Z,W)
nrow(Data1)
```

```
[1] 197
```

- To know the dimension (i.e., number of rows and columns) included in a given dataframe (i.e., number of rows) use function **dim()**. Example:

```
Z<-seq(1,50, by=0.25)
W<-2*Z
Data1<-data.frame(Z,W)
dim(Data1)
```

```
[1] 197   2
```

- To list the names of the variables in a given dataframe (i.e., number of rows) use function **names()**. Example:

```
Z<-seq(1,50, by=0.25)
W<-2*Z
Data1<-data.frame(Z,W)
names(Data1)
```

```
[1] "Z" "W"
```

- To list all the elements of the dataframe we can type its name. Once more, do not use this option if the dataframe includes many elements.
- To list the first elements of a given dataframe use function **head()**. Example:

```
Z1<-seq(1,100)
Z2<-2*Z1
Data1<-data.frame(Z1,Z2)
# List the first 6 elements (default)
head(Data1)
```

```
  Z1 Z2
1  1  2
2  2  4
3  3  6
4  4  8
5  5 10
6  6 12
```

- To view the last elements, use function **tail()**. Example:

```
Z1<-seq(1,100)
Z2<-2*Z1
Data1<-data.frame(Z1,Z2)
# List the last 6 elements (default)
tail(Data1)
```

```
     Z1  Z2
95   95 190
96   96 192
97   97 194
98   98 196
99   99 198
100 100 200
```

- To list selected observations of all variables in given dataframe use selection brackets [ ]. Example:

```
Z1<-seq(1:100)
Z2<-2*Z1
Data1<-data.frame(Z1,Z2)
# List observations from 10 to 12 of Z1 and Z2  (default)
```

```
Data1[10:12,]
```

```
   Z1 Z2
10 10 20
11 11 22
12 12 24
```

- To browse the elements of a variable use function **View()**. Example: **View(Data1)**. The contents of dataframe *Data1* will be opened as a new tab in the top-left window as a spreadsheet. You can scroll up and down to see its elements.

2.3 Importing data files

A dataframe can also be created from a given data file that includes variables and observations already organized as a two-entry table: rows for observations and columns for variables.

Most of the data we will analyze in the course will in *.csv* format (*comma separated values*) or excel type (*.xls*, *.xlsx*). To work with a given data file, we need to import it.

Important: All files we would like to work with should be included in our working directory, to avoid having to give the whole path to locate the file.

2.3.1 Importing a *.csv* data file (i.e., text type data file)

- To create a data frame from a *.csv* file we will use function **read_csv** from the **readr** package. This function allows us to import a *.csv* data file, create a dataframe and name it in a single command.
- Example: import data file *Example.csv* included in our working directory, create dataframe with its variables and observations, and name the data frame *Wage_data*.

```
library(readr)
Wage_data<-read_csv(file="Example.csv")
```

```
Parsed with column specification:
cols(
  wage = col_double(),
  educ = col_double(),
  exper = col_double()
)
```

- Notice that running this command creates the dataframe and provides us information (in red) of the types of variables included. That is if the variable is numeric (integer or double), or a character (among other types).

2.3.2 Importing an excel (*.xls*, *.xlsx*) data file

- Function **read_excel()** from the **readxl** package allows us to import a *.xls* or *.xlsx* data files and create a dataframe in one single command. Example: open a file *.csv* and create dataframe labeled as *Wage_data*,

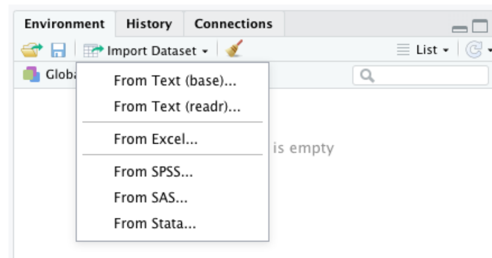
```
library(readxl)
Wage_data2<-read_excel("Example.xlsx")
```

2.3.3 Alternative 2: Creating dataframes from a file included in our working directory

- We can import a data file that is in the working directory. In this case we need to go to the **File** tab (bottom-right window), click on the file name and select *'Import Dataset'* option. The commands to import the file and name the dataframe are already written for you. You can always edit the name of the dataframe given by default.

2.3.4 Alternative 3: Creating dataframes from local files using Import Dataset icon

- We can import any data file into *RStudio* using the *Import Dataset* icon from the **Environment** tab (top-right window) and browsing to locate the file.



- In this case, you need to use the dialog window that opens to choose the name for the dataframe.

2.3.5 Information about the dataframe created from imported file

- To get the names of all the variables included in the dataframe created from imported file use function **names()**, as explained above. Example:

```
Wage_data<-read_csv(file="Example.csv")
names(Wage_data)
```

```
[1] "wage"  "educ"  "exper"
```

- To get information about the number of variables included in the dataframe created from imported file use function **length()**, as explained above. Example:

```
Wage_data<-read_csv(file="Example.csv")
length(Wage_data)
```

```
[1] 3
```

- To get information about the number of observations included in the dataframe created from imported file use function **nrow()**, as explained above. Example:

```
Wage_data<-read_csv(file="Example.csv")
nrow(Wage_data)
```

```
[1] 1472
```

- To get information about the dimension the dataframe created from imported file use function **dim()**, as explained above. Example:

```
Wage_data<-read_csv(file="Example.csv")
dim(Wage_data)
```

```
[1] 1472    3
```

2.3.6 Using variables belonging to a dataframe: Variable accessor

- Variables to be used in our analysis during a session need to be available in our **Environment** (top-right window) tab. Variables can be part of a dataframe, or we might have generated them during the session.
- If variables are part of a dataframe created from an imported datafile, using them requires to identify both, the dataframe they belong to and the name of the variable. It is important to know the exact spelling of the variables, and also, to remember that *R* is *case-sensitive*.
- To work with variables belonging to a dataframe we need to use the accessor operator **$**. Example, if we want to display the first 6 observations of variable *wage* included in dataframe *Wage_data*:

```
Wage_data<-read_csv(file="Example.csv")
Wage_data$wage[1:6]
```

```
[1]  7.780208  4.818505 10.563644  7.042430  7.887521  8.200057
```

- If a variable needs to be used repeatedly, then we can define it for the session. Example:

```
W<-Wage_data$wage
head(W)
```

```
[1]  7.780208  4.818505 10.563644  7.042430  7.887521  8.200057
```

2.3.7 Using variables from a dataframe without using operator *$*: attach() function

- An alternative to use the accessor operator (*$*) is to use function **attach()** after creating the dataframe. This function *attaches* the data set included in a dataframe to the *R* search path. This basically means that variables in a given dataframe can be used simply by typing their names. Example:

```
Wage_data<-read_csv(file="Example.csv")
names(Wage_data)
```

```
[1] "wage"  "educ"  "exper"
```

```
attach(Wage_data)
head(wage)
```

```
[1]  7.780208  4.818505 10.563644  7.042430  7.887521  8.200057
```

- Important: Before closing the session we need to detach the dataframe using the **detach()** function, to avoid using the wrong object in future sessions.

```
detach(Wage_data)
```

- Important: Always include the name of the dataframe inside the detach function. I.e., do not run it empty!

2.3.8 Selecting a subset of observations from dataframe

- We can define a dataframe with a subset of observations.
- Here we consider how to select observations under a given criterion using brackets [ ].
- Example 1: Creating a new dataframe by selecting observations from dataframe *Wage_data* where variable $edu = 1$. Notice we need to use double equal sign (==).

```
n<-nrow(Wage_data)
n
```

```
[1] 1472
```

```
Subset1<-Wage_data[Wage_data$educ==1,]
n1<-nrow(Subset1)
n1
```

```
[1] 99
```

- Example 2: Creating a new dataframe by selecting observations from dataframe *Wage_data* where variable $educ \neq 1$. Notice we need to use != to indicate $\neq$.

```
n<-nrow(Wage_data)
n
```

```
[1] 1472
```

```
Subset2<-Wage_data[Wage_data$educ!=1,]
nn1<-nrow(Subset2)
nn1
```

```
[1] 1373
```

- Example 3: Creating a new dataframe by selecting observations from dataframe *Wage_data* where variable $educ > 1$. Notice we need to use >= to indicate greater or equal.

```
n<-nrow(Wage_data)
n
```

```
[1] 1472
```

```
Subset3<-Wage_data[Wage_data$educ>=3,]
nn1<-nrow(Subset3)
nn1
```

```
[1] 1108
```

- Example 4: Creating a new dataframe by selecting observations from dataframe *Wage_data* where variable *educ* == 1 and variable *exper* $\neq$ 10. Notice we need to use **&** to indicate that both conditions need to hold.

```
n<-nrow(Wage_data)
n
```

```
[1] 1472
```

```
Subset3<-Wage_data[Wage_data$educ==1 & Wage_data$exper>=10, ]
nn1<-nrow(Subset3)
nn1
```

```
[1] 95
```

- Example 5: Creating a new dataframe by selecting observations from dataframe *Wage_data* where variable *educ* == 1 or *educ* == 10. Notice we need to use **|** to indicate that either condition need to hold.

```
n<-nrow(Wage_data)
n
```

```
[1] 1472
```

```
Subset4<-Wage_data[Wage_data$educ==1 | Wage_data$exper>=10, ]
nn1<-nrow(Subset4)
nn1
```

```
[1] 1084
```

2.3.9 Missing observations

- To get information of whether the data file imported includes missing observations for any of the variables use function **sum(is.na())**. This function will count the number of missing observations. Example:

```
Wage_data2<-read_csv(file="Example2.csv")
sum(is.na(Wage_data2))
```

```
[1] 2
```

- To get information, variable by variable, of whether the data file imported includes missing observations use function **colSums(is.na())**. This function will count the number of missing observations. Example:

```
Wage_data2<-read_csv(file="Example2.csv")
colSums(is.na(Wage_data2))
```

```
 wage  educ exper 
    0     1     1 
```

- To get information whether a given variable from the data file imported includes missing observations use function **colSums(is.na())**. This function will count the number of missing observations. Example:

```
Wage_data2<-read_csv(file="Example2.csv")
names(Wage_data2)
```

```
[1] "wage"  "educ"  "exper"
```

```
sum(is.na(Wage_data2$wage))
```

```
[1] 0
```

```
sum(is.na(Wage_data2$educ))
```

```
[1] 1
```

```
sum(is.na(Wage_data2$exper))
```

```
[1] 1
```

- To create a new dataframe that includes only complete observations, use function **na.omit()**. Example:

```
Wage_data2<-read_csv(file="Example2.csv")
Wage_data2c<-na.omit(Wage_data2)
```

2.4 Saving objects in the workspace as R data objects

We can save selected *R* objects in our workspace, to use them in another session. Objects in the workspace are saved with extension **Rda** (or, equivalently, **RData**).

2.4.1 Saving a single object from the workspace

- To save a single object from the workspace we use the **save()** function. Example: saving dataframe *Wage_data* as R data file *MyData1.Rda*

```
save(Wage_data, file="MyData1.Rda")
```

- File *MyData1.Rda* will be added to our working directory. To see it listed in the **Files** tab (bottom-right window), use the corresponding refresh icon (grey circle arrow on the right).
- The R data file can be loaded in another session using function **load()**.

```
load("MyData1.Rda")
```

2.4.2 Saving several objects from the workspace

- Several objects from the workspace can be saved using also the **save()** function. Example: saving dataframe *Wage_data* and also variable *Z* created during the session:

```
save(Wage_data, Z, file="MyData2.Rda")
```

2.4.3 Saving all objects from the workspace

- To save all objects from the workspace we use the **save.image()** function. Example:

```
save.image(file="AllMyData.Rda")
```

2.5 Basic data manipulation

2.5.1 Variable to variable functions

- New variables can be created by transforming existing ones. If variables we wish to transform belong to a given dataframe, the dataframe needs to be properly identified, unless function **attach()** has been used. If the variable has already been identified in the session, then, no need.
- Natural log transformation. Example:

```
Wage_data<-read_csv(file="Example.csv")
names(Wage_data)
```

```
[1] "wage"  "educ"  "exper"
```

```
# Define natural log of variable wage
lnwage<-log(Wage_data$wage)
```

- Polynomial transformations. Example:

```
Wage_data<-read_csv(file="Example.csv")
names(Wage_data)
```

```
[1] "wage"  "educ"  "exper"
```

```
# Define square of variable exper
exper2<-Wage_data$exper^2
```

2.5.2 Variable to scalar functions

- Scalars can be created from existing variables.
- Adding all the observations of a given variable. Example:

```
Wage_data<-read_csv(file="Example.csv")
names(Wage_data)
```

```
[1] "wage"  "educ"  "exper"
```

```
# Sum all the values of variable wage
sum(Wage_data$wage)
```

```
[1] 16266.51
```

```
# Sum  of the square of each of the values of variable wage
sum(Wage_data$wage^2)
```

```
[1] 208891.1
```

2.5.3 Sorting data

- Function **sort()** allows us to sort data in a dataframe. Example, if we want to create a new dataframe that includes observations of dataframe *Wage_data*, sorted from smallest to largest (default) w.r.t. *educ*

```
Wage_data1<-sort(Wage_data$educ)
```

- Sorting *Wage_data* from largest to smallest w.r.t. *educ*

```
Wage_data1<-sort(Wage_data$educ, decreasing = TRUE)
```

3 Simulating random variables

In this section we will consider generating (pseudo) observations for a variable with known statistical distribution.

In simulating the behavior of random variables we will always select the *seed* for replicability, using **set.seed()** function. That means that for a given seed, the set of observations generated from a given distribution will always be the same.

Please find below some examples. In class we will simulate a lot, as it is a fantastic learning tool. We will see many examples!

3.1 Simulating a discrete random variable

3.1.1 Example 1: Bernoulli distribution (E.g: Flipping a coin)

- Flipping a fair coin 5 times

```
set.seed(1234)
n<-5
Z<-rbinom(n,1,0.5)
Z
```

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

- Notice that if we do not set the seed, then, everytime you run the command, we would get a different realization of the random sample.

3.1.2 Example 2: Discrete uniform distribution (E.g.:rolling a die)

- Rolling a die 10 times

```
#
Z<-sample(1:6,10, rep=TRUE)
Z
```

```
 [1] 4 1 5 6 4 2 6 2 6 6
```

- Notice that function **sample()** is used. This function includes 2 arguments: discrete values to choose from (1,2,3,4,5,6 in our die) and the number of observations needed (1 in this example).

3.2 Simulating a continuos random variable

3.2.1 Example 1: $W \sim N(10, 4)$ (Normal distribution: $E(W) = 10, var(W) = 4$)

```
# Generating 5 observations
set.seed(101)
n<-5
W<-rnorm(n,0,2)
W
```

```
[1] -0.6520730  1.1049237 -1.3498877  0.4287189  0.6215384
```

- Notice that function **rnorm()** is used. This function includes 3 arguments: number of observations needed (5 in this example), expected value (10 in this case) and standard deviation (2).

3.2.2 Example 2: $Y \sim N(0, 1)$ (Standard Normal)

```
# Generating 5 observations
set.seed(101)
n<-5
Y<-rnorm(n,0,1)
Y
```

```
[1] -0.3260365  0.5524619 -0.6749438  0.2143595  0.3107692
```

- Notice that function **rnorm()** is used. This function includes 3 arguments: number of observations needed (5 in this example), expected value (0 for the standard normal) and standard deviation (1 for the standard normal).

3.2.3 Example 3: $U \sim U(0, 10)$ (Continuous Uniform between 0 and 10)

```
# Generating 5 observations
set.seed(101)
n<-5
U<-runif(n,0,10)
U
```

```
[1] 3.7219838 0.4382482 7.0968402 6.5769040 2.4985572
```

- Remember that a $U(0, 10)$ can take any value in the interval from 0 to 10.

4 Statistical tables

4.1 Statistical tables: t-distribution

4.1.1 t-distribution : Finding a value associated with a given probability

- Example 1: Given $t \sim t(10)$, find the value 'c' such that $Prob(t < c) = 0.05$:

```
c<-qt(0.05,10)
c
```

```
[1] -1.812461
```

- Example 2: Given $t \sim t(10)$, find the value 'c' such that $Prob(t > c) = 0.05$:

```
#Option 1:
c<-qt(0.95,10)
c
```

```
[1] 1.812461
```

```
#Option 2:using symmetry of t distribution
c<- 1-qt(0.05,10)
c
```

```
[1] 2.812461
```

4.1.2 t-distribution : Finding a probability associated with a given value (*p-value*)

- Example 1: Given $t \sim t(10)$, find the value 'p' such that $Prob(t < -2.5) = p$:

```
p<-pt(-2.5,10)
p
```

```
[1] 0.01572342
```

- Example 2: Given $t \sim t(10)$, find the value 'p' such that $Prob(t > 2.5) = p$:

```
#Option 1:
p<-1-pt(2.5,10)
p
```

```
[1] 0.01572342
```

```
#Option 2: using symmetry of t distribution
p<- pt(-2.5,10)
p
```

```
[1] 0.01572342
```

4.2 Statistical tables: F-distribution

4.2.1 F-distribution: Finding a value associated with a given probability

- Example 1: Given $F \sim F(2, 50)$, find the value 'c' such that

$$Prob(F < c) = 0.8$$

```
c<-qf(0.8,2,50)
c
```

```
[1] 1.662374
```

- Example 2: Given $F \sim F(2, 50)$, find the value 'c' such that

$$Prob(F > c) = 0.05$$

```
#Option 1:
c<-qf(0.95,2,50)
c
```

```
[1] 3.18261
```

```
#Option 2:
c<-qf(1-0.05,2,50)
c
```

```
[1] 3.18261
```

4.2.2 F-distribution: Finding a probability associated with a given value

- Example 1: Given $F \sim F(2, 50)$, find the value 'p' such that

$$Prob(F < 4.2) = p$$

```
p<-pf(4.2,2,50)
p
```

```
[1] 0.9793971
```

- Example 2: Given $F \sim F(2, 50)$, find the value 'p' such that

$$Prob(F > 4.2) = p$$

```
p<-1-pf(4.2,2,50)
p
```

```
[1] 0.02060293
```

5 Basic descriptive statistics

In this section we review some basic descriptive statistics for variables in data set *Example.csv*. First we will import the data file into *RStudio* using function **read_csv()** from **readr** package:

```
library(readr)
Wage_data<-read_csv(file="Example.csv")
```

Variables included in this dataframe are:

```
names(Wage_data)
```

```
[1] "wage"  "educ"  "exper"
```

5.1 Measures of location

- Mean

```
#Sample mean
mean(Wage_data$wage)
```

```
[1] 11.05062
```

- Conditional mean: If we want to calculate the sample mean for people with $educ = 1$

```
mean(Wage_data$wage[Wage_data$educ==1])
```

```
[1] 8.429049
```

- Median

```
#Sample median
median(Wage_data$wage)
```

```
[1] 10.12665
```

- Set of measures (mean, median and quartiles)

```
summary(Wage_data)
```

```
      wage             educ           exper      
 Min.   : 2.191   Min.   :1.000   Min.   : 0.00  
 1st Qu.: 8.113   1st Qu.:3.000   1st Qu.: 9.00  
 Median :10.127   Median :3.000   Median :16.50  
 Mean   :11.051   Mean   :3.378   Mean   :17.22  
 3rd Qu.:12.755   3rd Qu.:4.000   3rd Qu.:24.00  
 Max.   :47.576   Max.   :5.000   Max.   :47.00  
```

5.2 Measures of dispersion

- Smallest and largest values

```
max(Wage_data$wage)
```

```
[1] 47.57552
```

```
min(Wage_data$wage)
```

```
[1] 2.190978
```

- Variance

```
#Sample variance
var(Wage_data$wage)
```

```
[1] 19.80707
```

- Standard deviation

```
#Sample standard deviation
sd(Wage_data$wage)
```

```
[1] 4.450513
```

5.3 Measures of shape

- Coefficient of skewness

```
library(moments)
skewness(Wage_data$wage)
```

```
[1] 1.953399
```

- Coefficient of kurtosis

```
library(moments)
kurtosis(Wage_data$wage)
```

```
[1] 10.31803
```

5.4 Measures of linear association

- Covariance

```
#Sample covariance
cov(Wage_data$wage,Wage_data$educ)
```

```
[1] 2.089511
```

- (Pearson) correlation coefficient

```
cor(Wage_data$wage,Wage_data$educ)
```

```
[1] 0.3897805
```

6 Graphs using ggplot2 package

This section includes plotting data using **ggplot()** function from **ggplot2** package. We will only consider 2D graphs.

All plots generated, regardless of whether they have been given a name or not, are going to be displayed in the **Plots** tab (bottom-right window). If we have generated more than one plot, we can navigate from one to the other using the corresponding arrows available in the **Plots** tab menu.

A given plot can be saved, or copied into the clipboard, using the *Export* option available in the **Plots** tab menu. The saved files will be placed in your working directory.

To use package **ggplot2** we need to have the package already installed and then load it either using **library()** command (or clicking on the box next to the package name in the **packages** tab).

6.1 Basic structure

After loading the package, to first step in creating a **ggplot** is to define a *ggplot* object. This is done using the **ggplot()** function that initializes the plot. This creates a blank slate.

The first argument inside the **ggplot()** function is information about the dataframe where the variables to be included in the plot belong to. This is done by using **ggplot(data=)** and writing inside name of the dataframe. Example, to use variable(s) from dataframe *Wage_data*, we use **ggplot(data=Wage_data,)**.

If we wish to generate a plot using variables generated during the session we can use **ggplot(data=NULL,)** or just leave the first argument blank **ggplot(,)**.

The second argument inside the **ggplot()** function it to identify the which variable(s) we wish to include in the plot. As we shall see, this will be done by adding argument **aes()** inside the **ggplot()** function.

A *ggplot2* plot is created by adding layers to the basic **ggplot()** function, using the **+** sign. That is:

$$ggplot() + LAYER1 + LAYER2 + LAYER3 + ...$$

- LAYER 1: The first layer usually defines the geometry (type of graph) we want to create. Examples:
  - ggplot() + **geom_histogram()** (to generate a histogram)
  - ggplot() + **geom_point()** (to generate a scatter)
  - ggplot() + **geom_line()** (to generate a line plot)
- LAYER 2,3,4…: Choice to editing axis labels, axis scale, adding lines, … Order does not matter.

As example we will use variables in data set *Example.csv*. First we will import the data file into *RStudio* using function **read_csv()** from **readr** package:

```
library(readr)
Wage_data<-read_csv(file="Example.csv")
```

```
names(Wage_data)
```

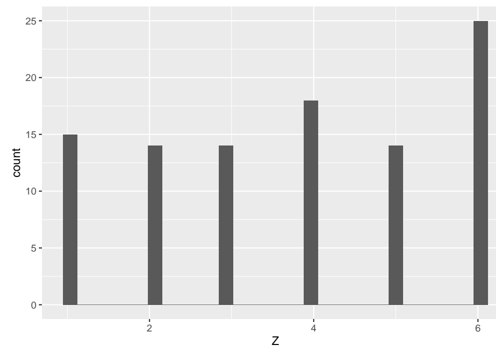
```
[1] "wage"  "educ"  "exper"
```

6.2 One variable plots: Histograms

6.2.1 Absolute frequency histogram (discrete variable)

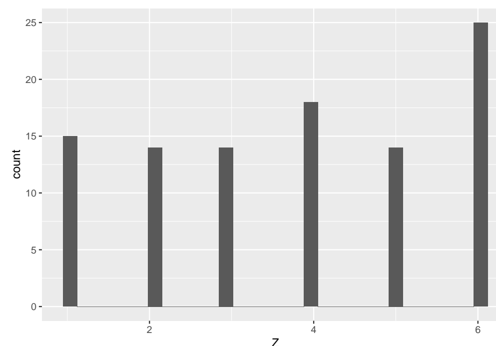
- Example: Absolute frequency histogram for a discrete variable Z, we have generated.

```
n<-100
set.seed(1234)
Z<-sample(1:6,n, rep=TRUE)
ggplot(data=NULL , aes(x=Z))+
      geom_histogram(aes(y=..count..))
```



Or, for short:

```
n<-100
set.seed(1234)
Z<-sample(1:6,n, rep=TRUE)
ggplot(, aes(x=Z))+
      geom_histogram(aes(y=..count..))
```



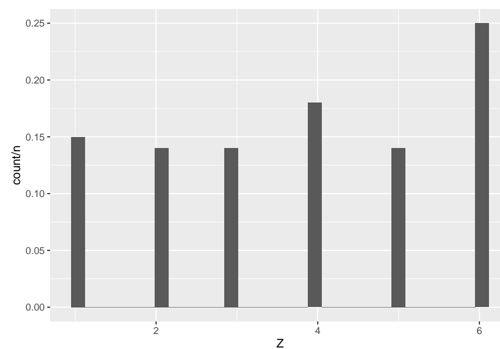
- Notice the variable to be placed in the vertical axis (absolute frequencies in this case) are directly calculated by *ggplot*.

6.2.2 Relative frequency histogram (discrete variable)

- Example: Relative frequency histogram for a discrete variable Z, we have generated.

```
n<-100
set.seed(1234)
```

```
Z<-sample(1:6,n, rep=TRUE)
ggplot(data=NULL, aes(x=Z))+
      geom_histogram(aes(y=..count../n))
```

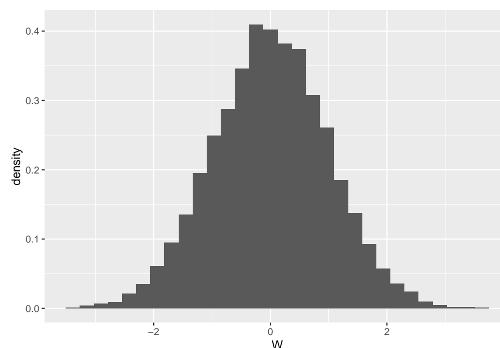


- Notice the variable to be placed in the vertical axis (relative frequencies in this case) are directly calculated by *ggplot* .

6.2.3 Density histogram (continuous variable)

- Example: Density histogram for a discrete variable Z, we have generated.

```
n<-10000
set.seed(1234)
W<-rnorm(n,0,1)
ggplot(data=NULL , aes(x=W))+
      geom_histogram(aes(y=..density..))
```



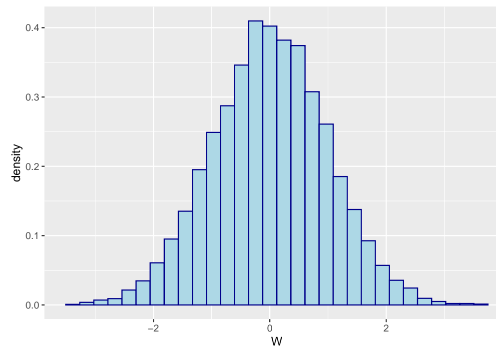
- Notice the variable to be placed in the vertical axis (densities in this case) are directly calculated by *ggplot*.

6.2.4 Editing

- We can edit the color of the histogram (filling and lines). Example:

```
#
n<-10000
set.seed(1234)
W<-rnorm(n,0,1)
ggplot(data=NULL  , aes(x=W))+
```

```
    geom_histogram(aes(y=..density..), fill="lightblue", color="darkblue")
```

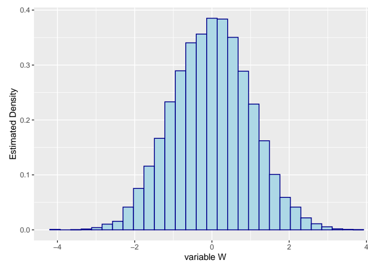


- To see color palette you can check (e.g.): http://sape.inf.usi.ch/quick-reference/ggplot2/colour

6.2.5 Additional layers: editing axes labels, range, plot title, ...

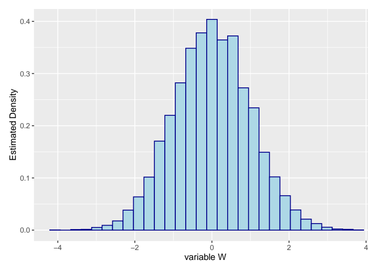
- Editing axis labels

```
n<-10000
W<-rnorm(n,0,1)
ggplot(data=NULL , aes(x=W))+
      geom_histogram(aes(y=..density..),fill="lightblue", color="darkblue") +
      xlab("variable W ") + ylab("Estimated Density")
```



- Notice layers can be added at the same time of generating the plot, or, they can be added, from an already defined and labels plot. Example:

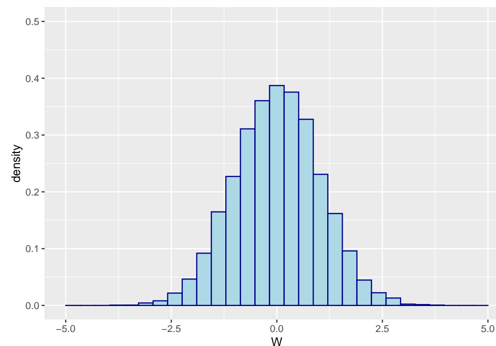
```
n<-10000
W<-rnorm(n,0,1)
p<- ggplot(data=NULL , aes(x=W))+
      geom_histogram(aes(y=..density..),fill="lightblue", color="darkblue")
p+xlab("variable W ") + ylab("Estimated Density")
```



- Editing axes range

```
n<-10000
W<-rnorm(n,0,1)
p<- ggplot(data=NULL , aes(x=W))+
      geom_histogram(aes(y=..density..),fill="lightblue", color="darkblue")

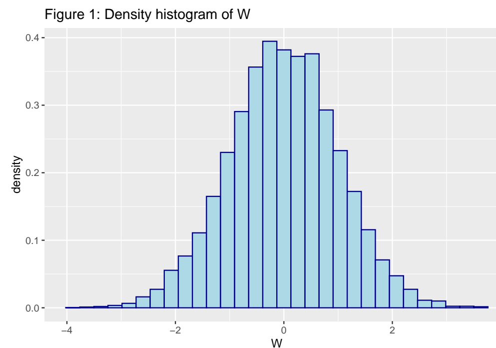
p+ xlim(-5,5) + ylim(0,0.5)
```



- Adding plot title

```
n<-10000
W<-rnorm(n,0,1)
p<- ggplot(data=NULL , aes(x=W))+
      geom_histogram(aes(y=..density..),fill="lightblue", color="darkblue")

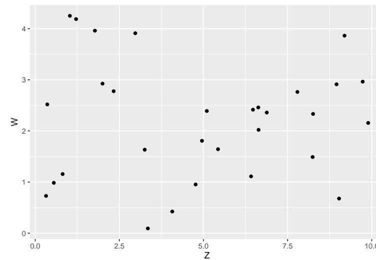
p+ ggtitle("Figure 1: Density histogram of W")
```



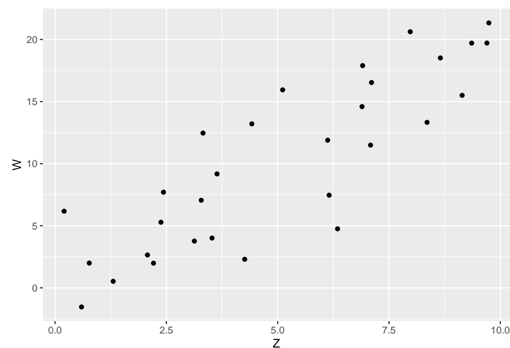
6.3 2 variable plots: Scatters

- To select a histogram we will use **geom_point()** geometry.
- First we will generate observations of two variables, *Z1* and *Z2*, create a dataframe including these two variables and then, indicate we want a scatter plot. To create a 2d-scatter plot, we need to identify the two variables in the **ggplot()** function, indicating which variable acts as x (horizontal axis) and which one acts as y (vertical axis)

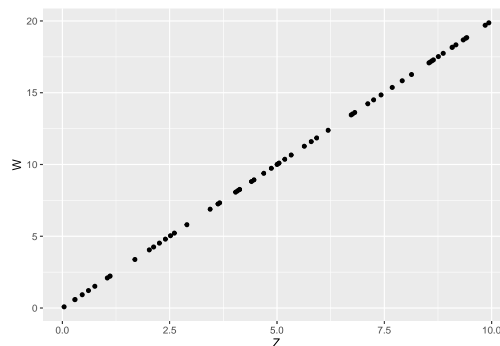
```
# Example 1: Z, W unrelated
n<-30
Z<-runif(n,0,10)
W<-runif(n,0,5)
ggplot( data=NULL , aes(x=Z, y=W)) +
      geom_point()
```




```
# Example 2: Z, W linearly related
n<-30
Z<-runif(n,0,10)
W<-2*Z+rnorm(n,0,3)
ggplot(data=NULL , aes(x=Z, y=W)) +
      geom_point()
```

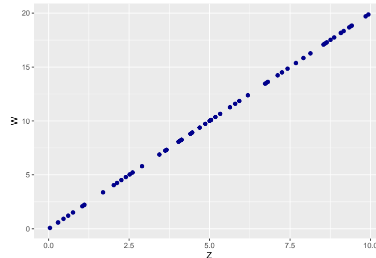



```
# Example 3: Z, W perfectly correlated
n<-60
Z<-runif(n,0,10)
W<-2*Z
ggplot(data=NULL, aes(x=Z, y=W)) +
      geom_point()
```



6.3.1 Editing shape, color and size of markers

```
ggplot(data = NULL, aes(x=Z, y=W)) +
      geom_point(shape=20, color="darkblue", size=3)
```

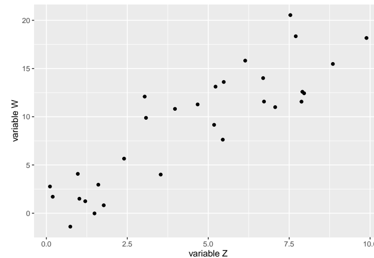


- To see options for marker shape: http://sape.inf.usi.ch/quick-reference/ggplot2/shape

6.3.2 Additional layers: editing axes labels, scale, range, ...

- Editing axes labels

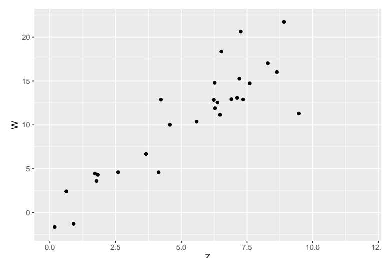
```
n<-30
Z<-runif(n,0,10)
W<-2*Z+rnorm(n,0,3)
p<-ggplot(data=NULL , aes(x=Z, y=W)) + geom_point()
p+ xlab("variable Z") + ylab("variable W")
```



- Editing axes range

```
n<-30
Z<-runif(n,0,10)
W<-2*Z+rnorm(n,0,3)

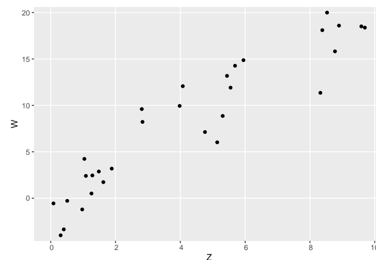
p<- ggplot(data=NULL , aes(x=Z, y=W)) + geom_point()
p+ xlim(0,12) + ylim(-2,22)
```



- Editing axes range and brakes

```
n<-30
Z<-runif(n,0,10)
W<-2*Z+rnorm(n,0,3)

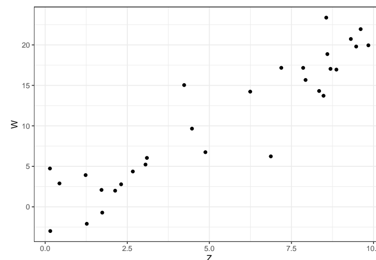
p<-ggplot(data=NULL , aes(x=Z, y=W)) + geom_point()
p+ scale_x_discrete(limits=seq(0,10,by=2 )) +
   scale_y_discrete(limits=seq(0,20,by=5 ))
```



- Editing: removing plot background

```
n<-30
Z<-runif(n,0,10)
W<-2*Z+rnorm(n,0,3)

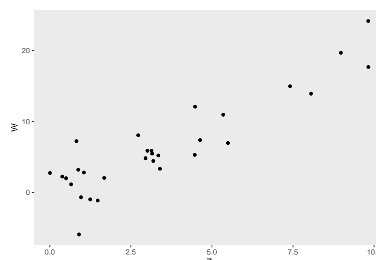
p<-ggplot(data=NULL , aes(x=Z, y=W)) + geom_point()
p+theme_bw()
```



- Editing: removing plot grid lines

```
n<-30
Z<-runif(n,0,10)
W<-2*Z+rnorm(n,0,3)

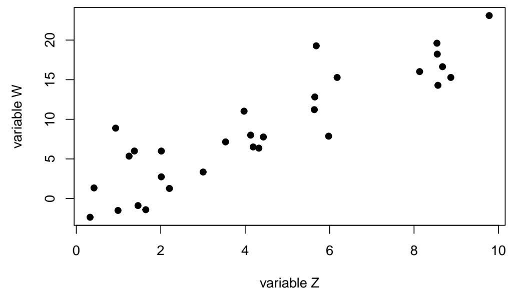
p<-ggplot(data=NULL , aes(x=Z, y=W)) + geom_point()
p+theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank())
```



6.3.3 Scatter plots without using package ggplot2()

Scatter plots can be also easily produced without using **ggplot2** package if you wish. In this case, you can use function **plot()**. Example:

```
n<-30
Z<-runif(n,0,10)
W<-2*Z+rnorm(n,0,3)
plot(Z,W, xlab="variable Z", ylab="variable W", pch=19)
```



7 Vectors and matrices in R

In this section we will learn to create vectors and matrices by entering its elements one-by-one, by using variables created in a session, or variables that belong to a dataframe listed in our **environment window**.

7.1 Creating vectors and matrices using function matrix()

- We will learn to create vectors or matrices using function **matrix()**. Function **matrix()** creates a matrix object from the given set of values (only numbers!) inside the argument, according to the specified number of columns (or rows).

7.1.1 Creating a vector or matrix by entering its elements

- Creating row vector. Example a creating row vector v:

$$v = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

```
v<-matrix(c(1,2,3),ncol=3)
v
```

```
     [,1] [,2] [,3]
[1,]    1    2    3
```

- Creating a column vector. Example a creating column vector w:

$$w = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

```
w<-matrix(c(1,2,3),ncol=1)
w
```

```
     [,1]
[1,]    1
[2,]    2
[3,]    3
```

- Creating a matrix. Example of creating matrix A:

$$A = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

```
A1<-c(1,2,3)
A2<-c(4,5,6)
```

```r
A<-matrix(c(A1,A2), ncol=2)
A
```

```
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

7.1.2 Creating a vector or a matrix: shortcuts

- Creating a constant vector: Examples

```r
#Row vector of 1s
u <-matrix(rep(1,4),ncol=4)
u
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    1    1    1
```

```r
#Column vector of 1s
u <-matrix(rep(1,4),ncol=1)
u
```

```
     [,1]
[1,]    1
[2,]    1
[3,]    1
[4,]    1
```

- Creating a diagonal matrix. Examples:

```r
d<-c(1,2,3)
A <-diag(d)
A
```

```
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    2    0
[3,]    0    0    3
```

Or, in the case we want to create an identity matrix 4x4:

```r
d<-rep(1,4)
I <-diag(d)
I
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]    0    1    0    0
[3,]    0    0    1    0
[4,]    0    0    0    1
```

7.1.3 Creating a matrix using variables from a data set

- We can create a vector, or a matrix, from variables included in a given dataframe.
- Example:

```
library(readr)
Wage_data<-read_csv(file="Example.csv")
A<-matrix(c(Wage_data$educ, Wage_data$exper), ncol=2)
```

In this example, a matrix A including all observations of variables *educ* and *exper* from dataframe *Wage_data* will be created.

7.1.4 Information about a matrix

- Number of rows of a matrix. Example:

```
A1<-c(1,2,3)
A2<-c(4,5,6)
A<-matrix(c(A1,A2), ncol=2)
nrow(A)
```

```
[1] 3
```

- Number of columns of a matrix. Example:

```
A1<-c(1,2,3)
A2<-c(4,5,6)
A<-matrix(c(A1,A2), ncol=2)
ncol(A)
```

```
[1] 2
```

- Dimension of a matrix. Example:

```
A1<-c(1,2,3)
A2<-c(4,5,6)
A<-matrix(c(A1,A2), ncol=2)
dim(A)
```

```
[1] 3 2
```

```
\medskip
```

7.2 Matrix Operations

7.2.1 Operations involving one matrix

- Operation involving a matrix, A or B. Example:

$$A = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 4 \\ 2 & 6 \end{bmatrix}$$

```
A1<-c(1,2,3)
A2<-c(4,5,6)
A<-matrix(c(A1,A2), ncol=2)
A
```

```
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

```
B1<-c(1,2)
B2<-c(4,6)
B<-matrix(c(B1,B2), ncol=2)
B
```

```
     [,1] [,2]
[1,]    1    4
[2,]    2    6
```

- Multiplying matrix A by a scalar:

```
C<-2*A
C
```

```
     [,1] [,2]
[1,]    2    8
[2,]    4   10
[3,]    6   12
```

- Transposing matrix A

```
tA<-t(A)
tA
```

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

- Rank of matrix A:

```
rankA<-qr(A)$rank
rankA
```

```
[1] 2
```

- Inverse of matrix B

```
iB<-solve(B)
iB
```

```
     [,1] [,2]
[1,]   -3  2.0
[2,]    1 -0.5
```

- Selecting elements of a matrix

```
# Select second row
A[2,]
```

```
[1] 2 5
```

```
# Select first column
c1<-A[,1]
c1
```

```
[1] 1 2 3
```

```
# Select element (3,2) (position: third row, second column)
p23<-A[3,2]
p23
```

```
[1] 6
```

7.2.2 Operations involving two matrices

- Operation involving matrices. Example using matrix A, B, C:

$$A = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & -1 \\ 1 & -1 \\ 1 & -1 \end{bmatrix} \qquad C = \begin{bmatrix} 1 & 1 & 2 \\ 1 & -1 & 0 \end{bmatrix}$$

First let us define these matrices:

```
A1<-c(1,2,3)
A2<-c(4,5,6)
A<-matrix(c(A1,A2), ncol=2)
```

```
A
```

```
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

```
B1<-c(1,1,1)
B2<- c(-1,-1,-1)
B<-matrix(c(B1,B2), ncol=2)
B
```

```
     [,1] [,2]
[1,]    1   -1
[2,]    1   -1
[3,]    1   -1
```

```
C1<-c(1,1)
C2<- c(-1,-1)
C3<-c(2,0)
C<-matrix(c(C1,C2,C3), ncol=3)
C
```

```
     [,1] [,2] [,3]
[1,]    1   -1    2
[2,]    1   -1    0
```

- Matrix addition or subtraction

  Matrices are add/subtract by adding/subtracting *element by element*. To add (or subtract) two matrices, they have to be conformable. What does conformable imply here? they need to have the same dimension. Hence to calculate $A + B$ or $A - B$, we require $dim(A) = dim(B)$.

- Matrix multiplication (only type of matrix multiplication we are going to need!)

  To multiply two matrices A and B, as seen in your matrix algebra math course in your first year, they have to be conformable. What does conformable imply here? Number of columns first matrix=number of rows second matrix. That is, to multiply $A \cdot B$ we require: $ncol(A) = nrow(B)$.

  To multiply two matrices we need to use operator $\% * \%$ Please be careful and do not use $*$ by itself!

```
E<-A%*%C
E
```

```
     [,1] [,2] [,3]
[1,]    5   -5    2
[2,]    7   -7    4
[3,]    9   -9    6
```

Summary of matrix operations:

Operation	Description	Condition for operation
ncol(A)	number of columns	-
nrow(A)	number of rows	-
dim(A)	Matrix dimension	-
diag(A)	Selecting elements of diagonal	A square
A[i,j]	Select matrix element (i, j)	-
t(A)	Matrix transpose	-
det(A)	Matrix determinant	A square
qr(A)$rank	Matrix rank	-
solve(A)	Matrix inverse	A square and non singular
A+B	Element-wise summation	dim(A)= dim$B$
A-B	Element-wise subtraction	dim(A)= dimen(B)
A*B	Element-wise multiplication	dim(A)= dim(B)
A%*%	Matrix multiplication	ncol(A)=nrow(B)

8 Simple regression model: OLS estimation

This section includes some basic commands for OLS estimation of a simple regression model, using as example data file *Example.csv* and regression model:

$$wage = \beta_0 + \beta_1 educ + u$$

To begin with we need to read the data file, located in the working directory, and create dataframe named *Wage_data* with it. We use function **readr_csv()** from package **readr**.

```
library(readr)
Wage_data<-read_csv(file="Example.csv")
```

```
Parsed with column specification:
cols(
  wage = col_double(),
  educ = col_double(),
  exper = col_double()
)
```

```
names(Wage_data)
```

```
[1] "wage"  "educ"  "exper"
```

As explained, to use the variables in this dataframe without having to identify the name of the dataframe, we will use accessor operator *$* (unless we have used the **attach()** function, as explained in section 2.3.7.). In this case we will use the accessor operator to rename the variables.

```
W<-Wage_data$wage
ED<-Wage_data$educ
EX<-Wage_data$exper
```

8.1 OLS estimation

8.1.1 Parameter estimates

OLS estimator in summation form for this example is given by:

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(W_i - \overline{W})(ED_i - \overline{ED})}{\sum_{i=1}^{n}(ED_i - \overline{ED})^2} \qquad \hat{\beta}_0 = \overline{W} - \hat{\beta}_1\overline{ED}$$

Then, OLS estimates in summation form:

```
dED<-ED-mean(ED)
dW<-W-mean(W)
b1hat<-sum(dED*dW)/sum(dED^2)
b0hat<-mean(W)-b1hat*mean(ED)
b0hat
```

```
[1] 6.18513
```

```
b1hat
```

```
[1] 1.440176
```

8.1.2 Fitted values and residuals

$$\widehat{W}_i \equiv \widehat{\beta}_0 + \widehat{\beta}_1 ED_i \qquad \widehat{u}_i \equiv W_i - \widehat{W}_i$$

```
What<-b0hat+b1hat*ED
uhat<-W-What
```

We can create a dataframe with original variables, fitted values and residuals:

```
W<-Wage_data$wage
ED<-Wage_data$educ
Wage_dataB<-data.frame(ED,W,What, uhat)
```

To view these values you can use the **View(Wage_dataB)** command seen before, that will open this dataframe as a spreadsheet in a new tab in the **top left window**.

8.2 *OLS* estimation using matrix algebra

8.2.1 Parameter estimates using matrix algebra

OLS estimator in matrix algebra:

$$\hat{\beta} = (X'X)^{-1}X'y$$

Matrix X: First column will include the *constant* regressor, second column will include the 13 observations of variable ED in dataframe **Wage_data**. Vector y will be a column vector including the 13 observations of variable W, belonging to the same dataframe. Given that variables ED and W have already been defined using the accessor operator $, we can use them directly to construct X and y.

```
n<-nrow(Wage_data)
C<-rep(1,n)
ED<-Wage_data$educ
X<-matrix(c(C, ED),ncol=2)
W<-Wage_data$wage
y<-matrix(W,ncol=1)
```

```
bhat<-solve(t(X)%*%X)%*%t(X)%*%y
bhat
```

```
         [,1]
[1,] 6.185130
[2,] 1.440176
```

8.2.2 Vector of fitted values and residuals using matrix algebra

$$\widehat{y} = X\widehat{\beta} \qquad \widehat{u} = y - \widehat{y}$$

```
What<- X %*% bhat
uhat<- W-What
```

8.3 Coefficient of determination

$$R^2 = \frac{SSE}{SST} \qquad or \qquad R^2 = 1 - \frac{SSR}{SST}$$

where:

$$SST = \sum_i W_i - \overline{W}^2 \qquad SSE = \sum_i W_i - \widehat{W}^2 \qquad SSR = \sum_i \widehat{u}_i^2$$

Using fitted values ($What$) and residuals ($uhat$) are already defined above:

```
SST<-sum((W-mean(W))^2)
SSE<-sum((What-mean(What))^2)
SSR<-sum(uhat^2)
R2<-SSE/SST
R2
```

```
[1] 0.1519288
```

```
R2<-1-SSR/SST
R2
```

```
[1] 0.1519288
```

8.4 OLS estimation: standard errors

Recall:

$$se(\widehat{\beta}_0) = \sqrt{\widehat{\sigma}^2 (X'X)^{-1}_{1,1}} \qquad se(\widehat{\beta}_1) = \sqrt{\widehat{\sigma}^2 (X'X)^{-1}_{2,2}}$$

where:

$$\widehat{\sigma}^2 = \frac{SSR}{n-2}$$

Then:

```
#Estimate sigma2
n<-13
s2hat<-SSR/(n-2)
s2hat
```

```
[1] 2246.324
```

```
#
#Estimate variance of OLS estimator
#
vbhat<-s2hat*solve(t(X)%*%X)
vbhat
```

```
          [,1]      [,2]
[1,] 13.539046 -3.555832
[2,] -3.555832  1.052521
```

```
#
# Verify standard errors
#
seb0hat<-sqrt(vbhat[1,1])
seb1hat<-sqrt(vbhat[2,2])
seb0hat
```

```
[1] 3.679544
```

```
seb1hat
```

```
[1] 1.025924
```

8.5 *OLS* estimation using function lm()

R provides function **lm()** to directly calculate *OLS* estimates plus some additional statistics associated with the *OLS* estimation. Example:

```
lm(wage~educ, data=Wage_data)
```

```
Call:
lm(formula = wage ~ educ, data = Wage_data)

Coefficients:
(Intercept)         educ
      6.185        1.440
```

If variables to be used in the regression have already been defined in the session, or, if we have used the **attach()** function, then no need to identify the name of the dataframe where the variables belong to. That is:

```
attach(Wage_data)
lm(wage~educ)
```

```
Call:
lm(formula = wage ~ educ)

Coefficients:
(Intercept)         educ
      6.185        1.440
```

8.5.1 Options after using function lm()

- Getting parameter estimates: Option 1

```
Model1<-lm(wage~educ, data=Wage_data)
Model1
```

```
Call:
lm(formula = wage ~ educ, data = Wage_data)

Coefficients:
(Intercept)         educ
      6.185        1.440
```

- Getting parameter estimates: Option 2

```
Model1<-lm(wage~educ, data=Wage_data)

bhat <- Model1$coefficients
b0hat<-bhat[1]
b0hat
```

```
(Intercept)
    6.18513
```

```
b1hat <- bhat[2]
b1hat
```

```
    educ
1.440176
```

Vector of OLS estimates and also individual coefficient estimates will be added to the workspace.

- Getting fitted values and residuals:

```
Model1<-lm(wage~educ, data=Wage_data)

What<-Model1$fitted
uhat<-Model1$residuals
```

Fitted values $What$ and residuals $uhat$ will be added to the workspace (**Environment** tab in top-right window).

- Getting coefficient of determination

```
Model1<-lm(wage~educ, data=Wage_data)
R2<-summary(Model1)$r.squared
R2
```

```
[1] 0.1519288
```

- Getting overall summary statistics from the estimation

```
Model1<-lm(wage~educ, data=Wage_data)
summary(Model1)
```

```
Call:
lm(formula = wage ~ educ, data = Wage_data)

Residuals:
    Min      1Q  Median      3Q     Max 
-10.569  -2.731  -0.615   1.907  34.190 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  6.18513    0.31830   19.43   <2e-16 ***
educ         1.44018    0.08875   16.23   <2e-16 ***
---
Residual standard error: 4.1 on 1470 degrees of freedom
Multiple R-squared:  0.1519,    Adjusted R-squared:  0.1514 
F-statistic: 263.3 on 1 and 1470 DF,  p-value: < 2.2e-16
```

- Getting overall summary statistics from the estimation using package **stargazer**

  We can use built-in function **lm()** for *OLS* estimation of our model, but then use function **stargazer()** from package **stargazer** to present output in a nicer table. To use function **stargazer()** we need to load stargazer package (once per session), using:

```
library(stargazer)
```

```
Model1<-lm(wage~educ, data=Wage_data)
stargazer(Model1, type = "text")
```

```
===============================================
                        Dependent variable:    
                    ---------------------------
                               wage            
-----------------------------------------------
educ                         1.440***          
                              (0.089)          
                                               
Constant                     6.185***          
                              (0.318)          
                                               
-----------------------------------------------
Observations                   1,472           
R2                             0.152           
Adjusted R2                    0.151           
Residual Std. Error      4.100 (df = 1470)     
F Statistic          263.345*** (df = 1; 1470) 
===============================================
Note:               *p<0.1; **p<0.05; ***p<0.01
```

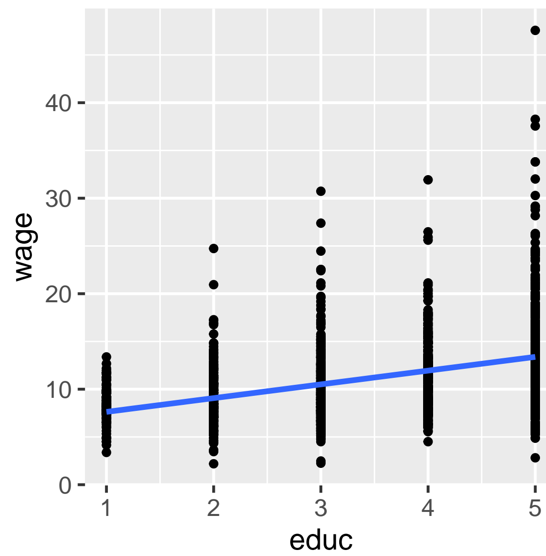
In this course, we will use the **stargazer()** function to get the complete output of our OLS estimation.

8.6 Plotting observations and fitted line

We can plot observations and fitted line using **ggplot2** package. Example

```
library(ggplot2)
ggplot(data=Wage_data, aes(x=educ, y=wage)) + geom_point( size = 1) +
      geom_smooth(method='lm', se = FALSE )
```

```
`geom_smooth()` using formula 'y ~ x'
```



9 Simple regression model: inference with t statistic

This section includes basic commands for inference based on *OLS* estimator using as example data file *Example.csv* and regression model:

$$wage = \beta_0 + \beta_1 educ + u$$

To begin with we need to read the data file, located in the working directory, and create dataframe named *Wage_data* with it.

```
library(readr)
Wage_data<-read_csv(file="Example.csv")
names(Wage_data)
```

```
[1] "wage"  "educ"  "exper"
```

For inference we will need the statistical tables. In the case of the simple regression model, the tables of the t-distribution will be needed. Please see *Sub-section 4.1.*

9.1 Significance test of each regressor (t-values, p-values, $*$)

9.1.1 Using lm() and stargazer() functions

After running a regression using built-in function **lm()**, **stargazer()** function will print, by default, among others: name of the regressor (v), *OLS* estimates (c), corresponding stardard error in parentheses (s) below each estimate, and the astericks notation to indicate at what level (1%,5%, 10%) a regressor is significant ($*$). Example:

```
Model1<-lm(wage~educ, data=Wage_data)
library(stargazer)
stargazer(Model1 ,  type = "text")
```

```
===============================================
                        Dependent variable:
                    ---------------------------
                               wage
-----------------------------------------------
educ                         1.440***
                              (0.089)

Constant                     6.185***
                              (0.318)
-----------------------------------------------
Observations                   1,472
R2                             0.152
Adjusted R2                    0.151
Residual Std. Error      4.100 (df = 1470)
F Statistic          263.345*** (df = 1; 1470)
===============================================
Note:               *p<0.1; **p<0.05; ***p<0.01
```

Function **stargazer()** also gives you the option to select which statistics related to the significance test of each regressor we wish to include in the output, using function **report**.

Options available include: name of the regressor (v), OLS estimates (c), standard errors (s), t-values (t), p-value (p) and to report or not report the $*$'s. Example:

```
Model1<-lm(wage~educ, data=Wage_data)
library(stargazer)
stargazer(Model1 ,  type = "text" , single.row=TRUE, report="vcstp*")
```

```
===============================================
                        Dependent variable:
                    ---------------------------
                               wage
-----------------------------------------------
educ                       1.440 (0.089)
                            t = 16.228
                           p = 0.000***
Constant                   6.185 (0.318)
                            t = 19.432
                           p = 0.000***
-----------------------------------------------
Observations                   1,472
R2                             0.152
Adjusted R2                    0.151
Residual Std. Error      4.100 (df = 1470)
F Statistic          263.345*** (df = 1; 1470)
===============================================
Note:               *p<0.1; **p<0.05; ***p<0.01
```

Important: You should know how $t-values$ and $p-values$ are calculated without using the **report** option!

9.2 Confidence intervals

9.2.1 Confidence intervals using function confint

- Calculating the confidence interval (default=95% confidence level) for our regression parameters can be done using function **confint()**. This function needs to be used after estimating a model using built-in function **lm()**. Example:

```
Model1<-lm(wage~educ, data=Wage_data)
confint(Model1)
```

```
               2.5 %   97.5 %
(Intercept) 5.560766 6.809493
educ        1.266092 1.614260
```

- Calculating the confidence interval changing the default 95% confidence level to, say, 99% confidence level, can be done using option **level**. Example:

```
Model1<-lm(wage~educ, data=Wage_data)
confint(Model1, level=.99)
```

```
              0.5 %   99.5 %
(Intercept) 5.364187 7.006073
educ        1.211282 1.669070
```

9.2.2 Confidence intervals using function lm() and stargazer() function

Function **stargazer()** also allows us to calculate confidence intervals by including option **ci.custom**. Example:

```
Model1<-lm(wage~educ, data=Wage_data)
stargazer(Model1, ci.custom = list(confint(Model1)), type = "text")
```

```

===============================================
                        Dependent variable:
                    ---------------------------
                               wage
-----------------------------------------------
educ                         1.440***
                          (1.266, 1.614)

Constant                     6.185***
                          (5.561, 6.809)

-----------------------------------------------
Observations                   1,472
R2                             0.152
Adjusted R2                    0.151
Residual Std. Error      4.100 (df = 1470)
F Statistic          263.345*** (df = 1; 1470)
===============================================
Note:               *p<0.1; **p<0.05; ***p<0.01
```

10 Simple regression model: prediction

10.1 Point prediction

```
model1 <- lm(wage~educ, data=Wage_data)
predict(model1, newdata=data.frame(educ=c(20)))
```

```
       1
34.98865
```

10.2 Interval prediction

```
model1 <- lm(wage~educ, data=Wage_data)
predict(model1, newdata=data.frame(educ=c(20)), interval="prediction")
```

```
       fit      lwr      upr
1 34.98865 26.43909 43.53821
```

11 Multiple regression model: estimation

This section includes some basic commands for inference based on *OLS* estimator for a multiple regression model, using as example data file *Example.csv* and regression model:

$$wage = \beta_0 + \beta_1 educ + \beta_2 exper + u$$

To begin with we need to read the data file, located in the working directory, and create dataframe named *Wage_data* with it.

```
library(readr)
Wage_data<-read_csv(file="Example.csv")
```

```
names(Wage_data)
```

```
[1] "wage"  "educ"  "exper"
```

11.1 *OLS* estimation

11.1.1 Parameter estimates using matrix algebra

- *OLS* estimator in matrix algebra:

$$\hat{\beta} = (X'X)^{-1}X'y$$

Defining necessary matrices:

```
n<-nrow(Wage_data)
C<-rep(1,n)
ED<-Wage_data$educ
EX<-Wage_data$exper

X<-matrix(c(C,ED,EX),ncol=3)
W<-Wage_data$wage
y<-matrix(c(W),ncol=1)
```

OLS estimates:

```
bhat<-solve(t(X)%*%X)%*%t(X)%*%y
bhat
```

```
         [,1]
[1,] 1.073736
[2,] 1.930375
[3,] 0.200687
```

11.1.2 Fitted values and residuals

Recall:

$$\widehat{y} = X\widehat{\beta} \qquad \widehat{u} = y - \widehat{y}$$

Vector of *OLS* Fitted and *OLS* residuals:

```
yhat<- X %*% bhat
uhat<- y-yhat
```

11.2 Googness of fit

Coefficient of determination and adjusted coefficient of determination

$$R^2 = \frac{SSE}{SST} \qquad or \qquad R^2 = 1 - \frac{SSR}{SST} \qquad\qquad \overline{R}^2 = 1 - \frac{SSR/(n-(K+1))}{SST/(n-1)}$$

where:

$$SST = \sum_i \left(testscr_i - \overline{testscr}\right)^2 \qquad SSE = \sum_i \left(\widehat{testscr}_i - \overline{testscr}\right)^2 \qquad SSR = \sum_i \widehat{u}_i^2$$

11.2.1 Coefficient of determination:

```
SST<-sum((W-mean(W))^2)
SSE<-sum((yhat-mean(yhat))^2)
SSR<-sum(uhat^2)
R2<-SSE/SST
R2
```

```
[1] 0.3444999
```

```
R2<-1-SSR/SST
R2
```

```
[1] 0.3444999
```

11.2.2 Adjusted coefficient of determination:

```
SST<-sum((W-mean(W))^2)
SSE<-sum((yhat-mean(yhat))^2)
SSR<-sum(uhat^2)
df1<-(n-3)
df2<-n-1
adjR2<- 1 - (SSR/df1)/(SST/df2)
adjR2
```

```
[1] 0.3436074
```

11.2.3 *OLS* estimation: standard errors

Recall:

$$se(\widehat{\beta}_0) = \sqrt{\widehat{\sigma}^2 (X'X)^{-1}_{1,1}} \qquad se(\widehat{\beta}_1) = \sqrt{\widehat{\sigma}^2 (X'X)^{-1}_{2,2}}$$

where:

$$\widehat{\sigma}^2 = \frac{SSR}{n-3}$$

Then, using definitions of matrix of X (including observations of all regressors) and vector y (including all observations of dependent variable), the step-by-step calculation of the standard errors using matrix algebra:

```
#Estimate sigma2
n<-13
SSR<-sum(uhat^2)
s2hat<-SSR/(n-3)
s2hat
```

```
[1] 1909.878
```

```
#
#Estimate variance of OLS estimator
#
vbhat<-s2hat*solve(t(X)%*%X)
vbhat
```

```
           [,1]        [,2]        [,3]
[1,] 20.4044666 -3.87614336 -0.34917283
[2,] -3.8761434  0.97667279  0.03348679
[3,] -0.3491728  0.03348679  0.01370946
```

```
#
# Defining standard errors
#
seb0hat<-sqrt(vbhat[1,1])
seb1hat<-sqrt(vbhat[2,2])
seb2hat<-sqrt(vbhat[3,3])
seb0hat
```

```
[1] 4.51713
```

```
seb1hat
```

```
[1] 0.9882676
```

```
seb2hat
```

```
[1] 0.1170874
```

11.3 OLS estimation using function lm()

OLS estimation of regression:

$$Model(2): \qquad wage = \beta_0 + \beta_1 educ + \beta_2 exper + u$$

As seen for the simple regression model, function **lm()** can be used to directly calculate OLS estimates. Example:

```
Model2<-lm(wage~educ+exper, data=Wage_data)
Model2
```

```
Call:
lm(formula = wage ~ educ + exper, data = Wage_data)

Coefficients:
(Intercept)         educ        exper
     1.0737       1.9304       0.2007
```

11.3.1 Estimating model without a constant

$$Model(3): \qquad wage = \beta_1 educ + \beta_2 exper + u$$

```
Model3<-lm(wage~educ+exper-1 , data=Wage_data)
Model3
```

```
Call:
lm(formula = wage ~ educ + exper - 1, data = Wage_data)

Coefficients:
  educ   exper
2.1343  0.2191
```

11.3.2 Estimating model with a subset of observations

- Consider we want to estimate $Model(4)$ using only observations where $educ = 1$.

$$Model(4): \qquad W = wage = \beta_0 + \beta_1 exper + u$$

```
Model4<-lm(wage~exper,data=subset(Wage_data,educ==1))
Model4
```

```
Call:
lm(formula = wage ~ exper, data = subset(Wage_data, educ == 1))

Coefficients:
(Intercept)        exper
    6.86795      0.06035
```

- Consider we want to estimate $Model(4)$ but using observations where $educ \neq 1$.

```
Model4<-lm(wage~exper,data=subset(Wage_data,educ!=1))
Model4
```

```
Call:
lm(formula = wage ~ exper, data = subset(Wage_data, educ != 1))

Coefficients:
(Intercept)        exper
     8.5098       0.1645
```

11.3.3 Options after using function lm()

- Getting parameter estimates: Option 1

```
Model2<-lm(wage~educ+exper, data=Wage_data)
Model2
```

```
Call:
lm(formula = wage ~ educ + exper, data = Wage_data)

Coefficients:
(Intercept)         educ        exper
     1.0737       1.9304       0.2007
```

- Getting parameter estimates: Option 2

```
Model2<-lm(wage~educ+exper, data=Wage_data)
bhat<-Model1$coefficients
b0hat<-bhat[1]
b0hat
```

```
(Intercept)
    6.18513
```

```
b1hat<-bhat[2]
b1hat
```

```
    educ
1.440176
```

```
b2hat<-bhat[3]
b2hat
```

```
<NA>
  NA
```

- Getting fitted values and residuals:

```
  Model2<-lm(wage~educ+exper, data=Wage_data)
#
# Associated fitted values and residuals
#
What<-Model2$fitted
uhat<-Model2$residuals
```

- Getting coefficient of determination

```
Model2<-lm(wage~educ+exper, data=Wage_data)
R2<-summary(Model2)$r.quared
R2
```

```
NULL
```

- Getting adjusted coefficient of determination

```
Model2<-lm(wage~educ+exper, data=Wage_data)
adjR2<-summary(Model2)$adj.r.squared
adjR2
```

```
[1] 0.3436074
```

- Getting overall summary statistics from the estimation

```
Model2<-lm(wage~educ+exper, data=Wage_data)
summary(Model2)
```

```

Call:
lm(formula = wage ~ educ + exper, data = Wage_data)

Residuals:
     Min       1Q   Median       3Q      Max 
-14.0436  -2.0808  -0.4068   1.5915  31.2307 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  1.07374    0.37269   2.881  0.00402 ** 
educ         1.93037    0.08154  23.674  < 2e-16 ***
exper        0.20069    0.00966  20.774  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.606 on 1469 degrees of freedom
Multiple R-squared:  0.3445,	Adjusted R-squared:  0.3436 
F-statistic:   386 on 2 and 1469 DF,  p-value: < 2.2e-16
```

- Getting overall summary statistics from the estimation using package **stargazer**

```
Model2<-lm(wage~educ+exper, data=Wage_data)
stargazer(Model2 , type = "text")
```

```
===============================================
                        Dependent variable:
                    ---------------------------
                               wage
-----------------------------------------------
educ                         1.930***
                              (0.082)

exper                        0.201***
                              (0.010)

Constant                     1.074***
                              (0.373)
-----------------------------------------------
Observations                   1,472
R2                             0.344
Adjusted R2                    0.344
Residual Std. Error      3.606 (df = 1469)
F Statistic          386.018*** (df = 2; 1469)
===============================================
```

11.3.4 Reporting estimation of two models in a single table

- Example:

```
Model1<-lm(wage~educ, data=Wage_data)
Model2<-lm(wage~educ+exper, data=Wage_data)
stargazer(Model1 , Model2 , type = "text")
```

```
=======================================================================
                                    Dependent variable:
                    ---------------------------------------------------
                                           wage
                               (1)                       (2)
-----------------------------------------------------------------------
educ                         1.440***                  1.930***
                             (0.089)                   (0.082)

exper                                                  0.201***
                                                       (0.010)

Constant                     6.185***                  1.074***
                             (0.318)                   (0.373)
-----------------------------------------------------------------------
Observations                  1,472                     1,472
R2                            0.152                     0.344
Adjusted R2                   0.151                     0.344
Residual Std. Error     4.100 (df = 1470)          3.606 (df = 1469)
F Statistic         263.345*** (df = 1; 1470) 386.018*** (df = 2; 1469)
=======================================================================
```

11.4 Calculating variance inflating factors

11.4.1 By estimating auxiliary regression

We can calculate the variance inflating factor associated with the estimation of a parameter by running the appropriate auxiliary regression and its associated coefficient of determination.

11.4.2 Using function vif(), from package car

After running a regression using built-in function **lm()**, we can use function **vif()** from package **car**, to easily calculate the variance inflating factors. Remember in computer rooms on campus, package **car** is already installed. The only thing you need to do is activate it, by either running command **library** or ticking on the box next to the package name in the package tab (bottom right window). Example of how to use **vif()** function:

```
library(car)
```

```
Loading required package: carData
```

```
Model1<-lm(wage~educ+exper, data=Wage_data)
vif(Model1)
```

```
    educ    exper
1.091404 1.091404
```

11.5 Multiple regression estimation: logs, polynomial forms, interaction terms

11.5.1 Regressions with variables in log form

$$Model(1): \qquad log(wage) = \beta_0 + \beta_1 log(educ) + \beta_2 exper + u$$

```
Model1<-lm(log(wage)~log(educ)+exper, data=Wage_data)
Model1
```

```

Call:
lm(formula = log(wage) ~ log(educ) + exper, data = Wage_data)

Coefficients:
(Intercept)    log(educ)        exper
    1.56162      0.42834      0.01661
```

11.5.2 Regressions with variables polynomial forms

$$Model(2): \qquad wage = \beta_0 + \beta_1 educ + \beta_2 exper + \beta_3 exper^2 + u$$

In this case, just define variable $exper^2$ as a regressor. That is:

```
exper2<-Wage_data$exper^2
Model2<-lm(wage~educ+exper+exper2, data=Wage_data)
Model2
```

```

Call:
lm(formula = wage ~ educ + exper + exper2, data = Wage_data)

Coefficients:
(Intercept)         educ        exper       exper2
  -0.057261     1.932966     0.368841    -0.004435
```

Alternatively, you can avoid having to define $exper^2$ using the **I()**:

```
Model2<-lm(wage~educ+exper+I(exper^2), data=Wage_data)
Model2
```

```

Call:
lm(formula = wage ~ educ + exper + I(exper^2), data = Wage_data)

Coefficients:
(Intercept)         educ        exper   I(exper^2)
  -0.057261     1.932966     0.368841    -0.004435
```

11.5.3 Regressions with interaction terms

$$Model(3): \qquad ln(wage) = \beta_0 + \beta_1 educ + \beta_2 exper + \beta_3 (educ \cdot exper) + u$$

```
edex<-Wage_data$educ*Wage_data$exper
Model3<-lm(log(wage)~educ+exper+edex, data=Wage_data)
Model3
```

```

Call:
lm(formula = log(wage) ~ educ + exper + edex, data = Wage_data)

Coefficients:
(Intercept)         educ        exper         edex
   1.716558     0.101491     0.006283     0.003053
```

Alternatively, you can avoid having to define the interaction term by using the **I()**, again

```
Model3<-lm(log(wage)~educ+exper+I(educ*exper), data=Wage_data)
Model3
```

```

Call:
lm(formula = log(wage) ~ educ + exper + I(educ * exper), data = Wage_data)

Coefficients:
    (Intercept)             educ            exper  I(educ * exper)
       1.716558         0.101491         0.006283         0.003053
```

12 Simulating behavior of OLS estimator

12.1 Generating a sample from a given data generating process dgp

Consider generating a sample of 30 observations from the following dgp:

$$y_i = 10 + 1 \cdot x_{i1} + 1 \cdot x_{i2} + u_i \quad u_i/X \sim i.i.N(0, 81) \quad x_{i1} \sim U[0, 20] \quad x_{i2} \sim U[0, 20]$$

and then, estimating the following regression:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + u$$

```
# 1
n<-30

# 2
set.seed(12345)
x1 <- runif(n, min=0, max=20)
x2 <- runif(n, min=0, max=20)
y <- 10+1*x1+1*x2+rnorm(n, 0, 9)

# 3
Model <- lm(y~x1+x2)
Model
```

```

Call:
lm(formula = y ~ x1 + x2)

Coefficients:
(Intercept)           x1           x2
    14.6368       1.0392       0.7641
```

12.2 Monte Carlo experiments

Consider generating many samples samples of the same size from a given *data generating process* (dgp). All the parameters of the dgp need to be set. Example:

$$y_i = 10 + 1 \cdot x_{i1} + 1 \cdot x_{i2} + u_i \quad u_i/X \sim i.i.N(0, 81) \quad x_{i1} \sim U[0, 20] \quad x_{i2} \sim U[0, 20],$$

With each each sample, we wish to estimate the following regression model:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + u,$$

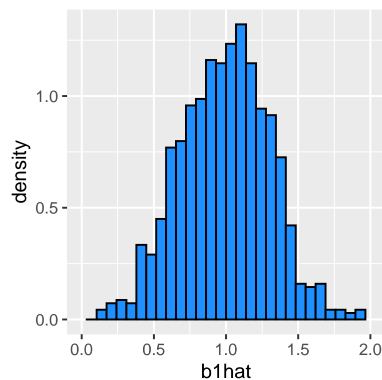
After each estimation, we want to save the estimates of each of our parameters, and creating a dataframe that includes all the estimates.

The following R script generates 1,000 samples of 30 observations each:

```r
# Sample size
n<-30

# Generating the values of the regressors (kept the same across samples)
set.seed(12345)
x1 <- runif(n, min=0, max=20)
x2 <- runif(n, min=0, max=20)
y <- 10+1*x1+1*x2+rnorm(n, 0, 9)


# Number of samples we want to generate
nsim <- 1000
b0hat <- numeric(nsim)
b1hat <- numeric(nsim)
b2hat <- numeric(nsim)

# Loop

for (i in 1:nsim){
  set.seed(12345+i)
  y <- 10+1*x1+1*x2+rnorm(n, 0, 9)
  Model <- lm(y~x1+x2)
  b0hat[i]<-Model$coefficients[1]
  b1hat[i]<-Model$coefficients[2]
  b2hat[i]<-Model$coefficients[3]
}

# Creating a dataframe with our estimates from the 1000 samples
MC_data<-data.frame(b0hat,b1hat,b2hat)
```

- To produce a density histogram with the 1,000 OLS estimates of β_1, for example, using function **ggplot()**, from **ggplot** package:

```r
library(ggplot2)
ggplot(MC_data, aes(x=b1hat)) +
  geom_histogram(aes(y =..density..), col="black", fill="dodgerblue1") +
  xlim(0,2)
```



13 Multiple regression model: inference

This section includes basic commands for inference based on *OLS* estimator using as example data file *Example.csv* and regression model:

$$wage = \beta_0 + \beta_1 educ + \beta_2 exper + u$$

To begin with we need to read the data file, located in the working directory, and create dataframe named *Wage_data* with it.

```
library(readr)
Wage_data<-read_csv(file="Example.csv")
names(Wage_data)
```

```
[1] "wage"  "educ"  "exper"
```

For inference we will need the statistical tables. In the case of the multiple regression model, the tables of the t-distribution and the F-distribution will be needed. Please see *Section 4*.

13.1 Statistics for significance test of each regressor (t-values, p-values)

Please revise pages 42-43 for the case of the simple regression model. Example:

```
Model2<-lm(wage~educ+exper, data=Wage_data)
library(stargazer)
stargazer(Model2 ,  type = "text" , report="vcstp*")
```

```
===============================================
                        Dependent variable:
                    ---------------------------
                               wage
-----------------------------------------------
educ                          1.930
                             (0.082)
                           t = 23.674
                          p = 0.000***

exper                         0.201
                             (0.010)
                           t = 20.774
                          p = 0.000***

Constant                      1.074
                             (0.373)
                            t = 2.881
                          p = 0.005***

-----------------------------------------------
Observations                  1,472
R2                            0.344
Adjusted R2                   0.344
Residual Std. Error      3.606 (df = 1469)
F Statistic          386.018*** (df = 2; 1469)
===============================================
Note:               *p<0.1; **p<0.05; ***p<0.01
```

13.2 Confidence intervals

13.2.1 Confidence intervals using function confint()

Calculating the confidence interval (default=95% confidence level) for our regression parameters can be done using function **confint()**. This function needs to be used after estimating a model using built-in function **lm()**. Example:

```
Model2<-lm(wage~educ+exper , data=Wage_data)
confint(Model2)
```

```
                2.5 %    97.5 %
(Intercept) 0.3426679 1.8048040
educ        1.7704305 2.0903198
exper       0.1817372 0.2196369
```

- Calculating the confidence interval changing the default 95% confidence level to, say, 99% confidence level, can de done using option **level**. Example:

```
Model2<-lm(wage~educ+exper , data=Wage_data)
confint(Model2, level=.99)
```

```
                0.5 %    99.5 %
(Intercept) 0.1124924 2.0349794
educ        1.7200722 2.1406781
exper       0.1757709 0.2256032
```

13.2.2 Confidence intervals using function lm() and stargazer() function

Recall, function **stargazer()** also allows us to calculate confidence intervals by including option **ci.custom**.

```
Model2<-lm(wage~educ+exper , data=Wage_data)
stargazer(Model2, ci.custom = list(confint(Model2)), type = "text")
```

```
===============================================
                        Dependent variable:
                    ---------------------------
                               wage
-----------------------------------------------
educ                         1.930***
                          (1.770, 2.090)

exper                        0.201***
                          (0.182, 0.220)

Constant                     1.074***
                          (0.343, 1.805)
-----------------------------------------------
Observations                   1,472
R2                             0.344
Adjusted R2                    0.344
Residual Std. Error      3.606 (df = 1469)
F Statistic          386.018*** (df = 2; 1469)
===============================================
```

13.3 Inference with F statistic using function linearHypothesis()

This sub-section includes function **linearHypothesis()** from the **car** package. This function calculates the $F - value$ for any linear hypothesis we wish to test.

Important: You should be able to calculate the *F-value* from the expression of the *F-statistic* step by step using previous commands (that is estimating the model we are testing, calculating SSR, estimating the restricted model and calculating $RSSE$ and then calculating the F_value.

13.3.1 Testing one restriction

- Significance test of a regressor. Example, testing significance of regressor *educ*:

$$Test: \qquad H_0: \beta_1 = 0 \qquad vs \qquad H_1: \beta_1 \neq \beta_2$$

```
Model2<-lm(wage~educ+exper , data=Wage_data)
library(car)
linearHypothesis(Model2, c("educ=0"))
```

```
Linear hypothesis test

Hypothesis:
educ = 0

Model 1: restricted model
Model 2: wage ~ educ + exper

  Res.Df   RSS Df Sum of Sq      F    Pr(>F)
1   1470 26386
2   1469 19099  1    7286.9 560.47 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

  Very important:

  Notice that even if we are testing whether parameter $\beta_1 = 0$ (proper way to state the null hypothesis of regressor *educ* not being significant!), the way we enter this restriction into the function **linearHypothesis()** implies writing $educ = 0$!!!!

  This is very stardard in econometric software, but we should be aware of the proper way to write an hypothesis. Hypothesis are about parameters, not about variables!!!

- Testing equality of two parameters. Example:

$$Test: \qquad H_0: \beta_1 = \beta_2 \qquad vs \qquad H_1: \beta_1 \neq \beta_2$$

```
Model1<-lm(wage~educ+exper , data=Wage_data)
library(car)
linearHypothesis(Model1, c("educ=exper"))
```

```
Linear hypothesis test

Hypothesis:
educ - exper = 0

Model 1: restricted model
Model 2: wage ~ educ + exper

  Res.Df   RSS Df Sum of Sq      F    Pr(>F)
1   1470 25287
2   1469 19099  1      6188 475.95 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

13.3.2 Testing two restrictions

- Example:

$$Test: \qquad H_0: \beta_1 = \beta_2 = 0 \qquad vs \qquad H_1: not\ H_0$$

```
Model1<-lm(wage~educ+exper , data=Wage_data)
library(car)
linearHypothesis(Model1, c("educ=0","exper=0"))
```

```
Linear hypothesis test

Hypothesis:
educ = 0
exper = 0

Model 1: restricted model
Model 2: wage ~ educ + exper

  Res.Df   RSS Df Sum of Sq      F    Pr(>F)
1   1471 29136
2   1469 19099  2     10037 386.02 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

14 Multiple regression model: prediction

14.1 Point prediction

```
model2 <- lm(wage~educ+exper , data=Wage_data)
predict(model2, newdata=data.frame(educ=c(5),exper=c(10)))
```

```
       1
12.73248
```

14.2 Interval prediction

```
model1 <- lm(wage~educ, data=Wage_data)
predict(model1, data.frame(educ=c(5),exper=c(10)), interval="prediction")
```

```
       fit      lwr      upr
1 13.38601 5.336038 21.43598
```