Methods and tools for health statistics

R laboratories

Eduart Murati, Fabrizio Carinci

Version 1.1, February 2023 (A.Y. 2022-2023)

Table of contents

1. Introduction to R

Firstly, we need to install **R** and **RStudio** :

- Download the latest version of **R** from The Comprehensive R Archive Network (CRAN).
- Download the latest version of **RStudio** here: R Studio.
  Select the free **RStudio Desktop version** and download the Installer specific for your OS.
  R should be installed first, R Studio detects your R installation automatically.

1.1. Basics

RStudio is a free and open source integrated development environment (IDE) for R. It includes a console, where the user can type R commands, and an environment which shows data and variables loaded into the sytem. The default display of the main user interface of RStudio is composed of four panels:

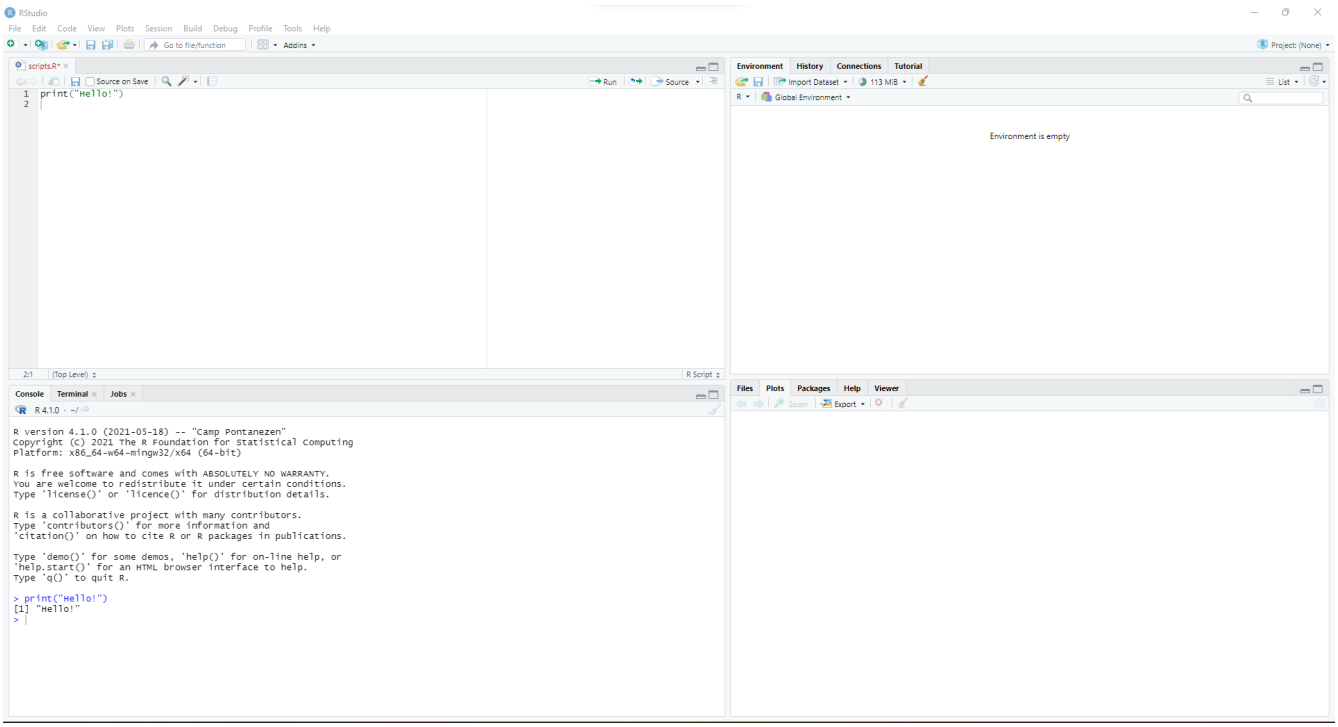


Figure 1. Rstudio interface.

The panel on the upper left corner is called the **source panel**. This area allows editing, running and viewing the datasets in an Excel sheet fashion. The upper right side is called **environement panel**. This area allows displaying the contents of the objects imported as datasets, parameters, lists, functions ect. On the lower left side there is the **console panel**, which displays R commands that have been run, and allows adding more in a free text format. When you launch RStudio, it will show the version of R that you're running at the top of the console. Every plot generated will show up in the lower right side of RStudio. In this area the user can browse all files, the documentation of the packages. The small window also allows managing R packages in an interactive way. Before starting a new analysis, it is good practice to open a new script:

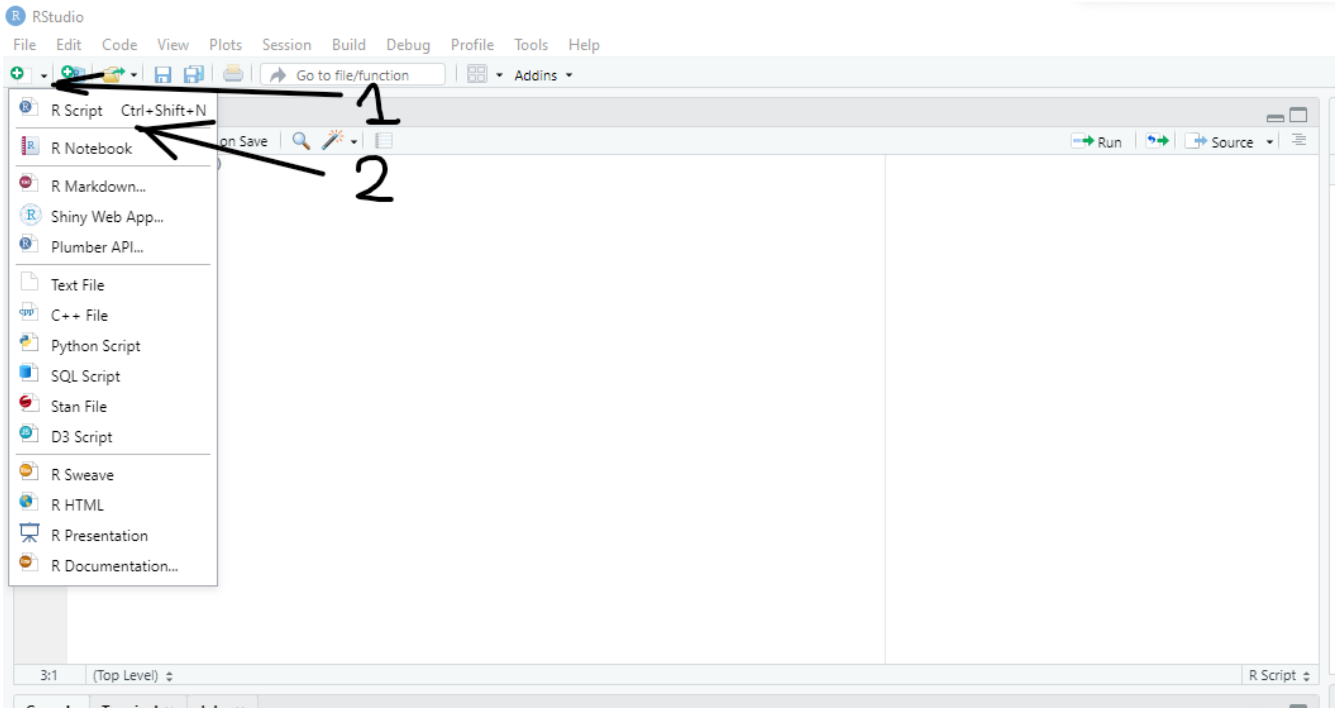


Figure 2. Rstudio new script.

To run the code line in the editor, just click run or use the combination **CTRL+ENTER** from the keyboard:

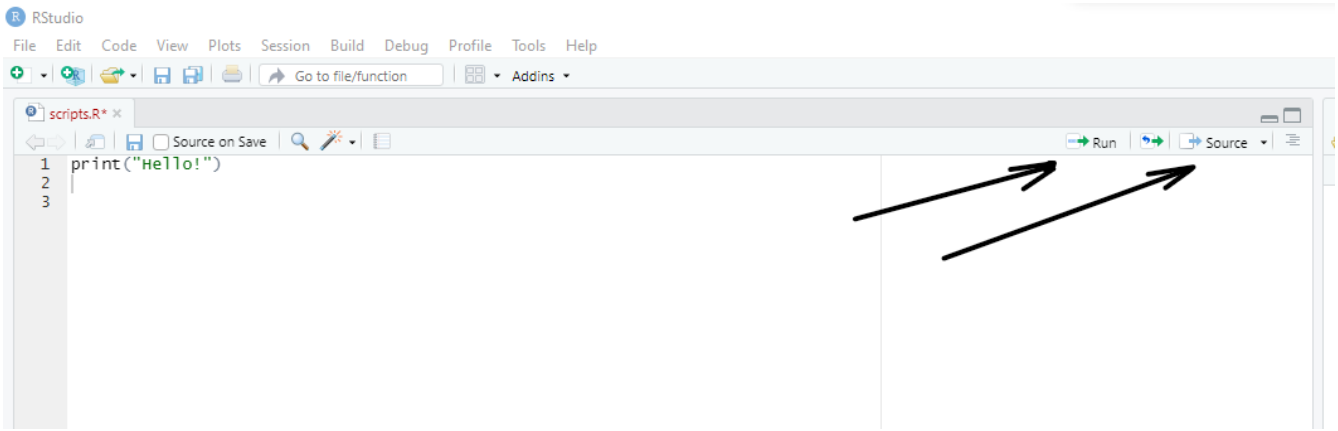


Figure 3. Run a command.

Use **Run** if you want to execute the line where the cursor is positioned, or **Source** if you want to execute all the code present in a window of the script editor.

A fundamental command that should be run before any other parts of the code is to set the **working directory**. The **working directory** is a file path where the user defines the default location of any files that must be read or saved in R. For example, if you import or save a dataset, the system will directly use the working directory as a reference location. To check or change your current working directory, use:

```
getwd() ①
setwd("C:/....") ②
```

① Return current working directory.

② Change current working directory.

You can also set the path of your **working directory** directly from the RStudio interface:

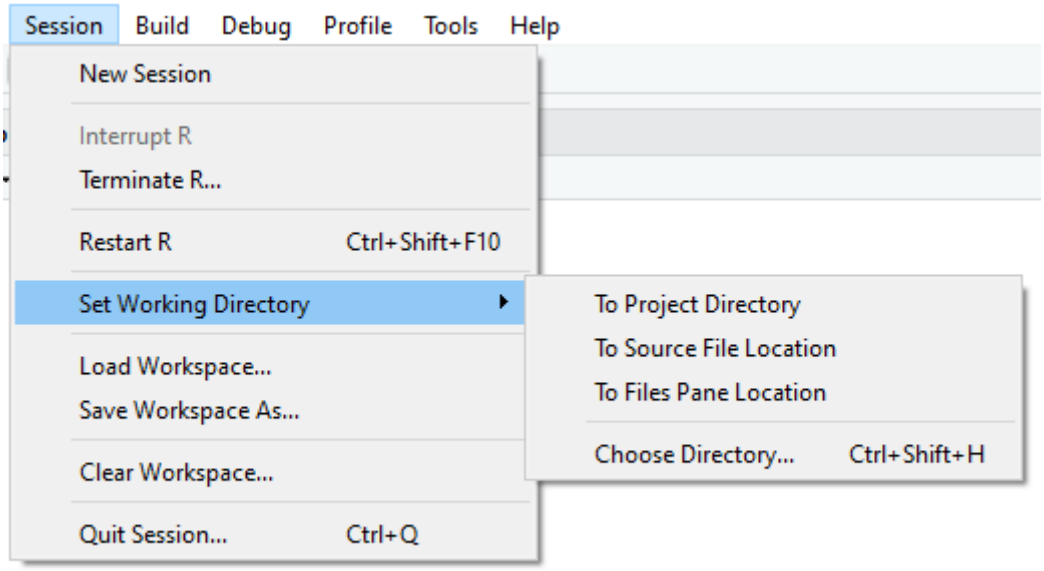


Figure 4. Set working directory.

Let's now start with some basic commands:

```
# This is a comment ①
print("Hello!")
1+1*3/3 ②
```

① R treats the hashtag character, # , in a special way; R will not run anything that follows a hashtag on a line. This make hashtags very useful for adding comments and annotations to your code.

② R can be used as a calculator.

The basic arithmatic operations in R are:

```
+       Addition
-       Subtraction
*       Multiplication
/       Division
^       Exponent
%%      Modulus (Remainder from division)
%/%     Integer Division
```

← is an assignment operator in R. It assigns the value on the right side to the variable on the left.

The **functions** in R are composed as: fun(p1,p2...), where: **fun** is the name and p1,p2,... are the parameters you pass to the function.

For example, the following functions can be used to sum two numbers:

```
sum_numbers<-function(x,y){  ①
  output=x+y
  return(output)
}
sum_numbers(3,4)

sum(3,4)                     ②
```

① Personalized function

② Build in function, with R installation.

In R you can use **help** or **?** to access to the documentation of a specific function. When this command is run, a webpage on the CRAN will open with explanation of what the function does, what its arguments are, and what are the associated functions.

For example, let's check some information about **sum** function:

```
help(sum)
?sum
```

Another useful R function is **source("file")**. This command submits directly all source in the file to R for execution. For instance, this can load all functions from the file ".R" or even execute all the contents of any text file. The functions will appear immediately in the global environment of R.
The R environment refers to your current workspace. It can be thought like the contents of the memory in terms of all assigned values, which will be used in all consecutive steps.

Load the functions and install the packages with:

```
source("methods_lab_functions.R") ①

installLibraries(c("readxl", "writexl" )) ②

install.packages("package_name") ③
```

① Load the functions inside **methods_lab_functions.R** file.

② Use the function **installLibraries** loaded from **methods_lab_functions.R** file to install the listed packages.

③ Basic function included with the R installation, to install packages from **CRAN**.

To be correctly loaded, the file **methods_lab_functions.R** shoud be present in the working directory.

RCurl requires curl (libcurl-devel on linux).

The **packages** are collections of **functions**, data, and compiled code in a well-defined format, created to add specific functionalities. There are more than 10,000 packages avaible ready to install from **CRAN**.
There are a set of **basic packages** which are considered part of the **R source code** and automatically available as part of R installation. Base packages contain the **basic functions** that allow R to work and enable standard statistical and graphical functions. For example, all functions that we have been using so far in our examples. The R directories in which packages are stored are called "libraries".

You can check the libraries loaded in your current R session by typing into the console:

```
sessionInfo() ①
search() ②
```

① Print version information about R, the OS and attached or loaded packages

② Gives a list of attached packages

The installation of a package is not sufficient for using its functions. The user needs to load the package with **library(package_name)** function in every new R session.

Vectors are the among most basic data structures in R. You can create a vector with the use of **c()** function. For example, create two vectors:

```
v1<-c(3,5,2,4,6,8,45,6,89)
v2 <- c("red","blue","red","orange","black", "white", "black","blue", "red" )
```

The vector **v1** is of a numeric class and **v2** vector of a character class. You can check the class of an object with **class()** function. For example:

```
class(v1)
[1] "numeric"

class(v2)
[1] "character"
```

For the numeric vectors, there are some build in functions very useful to describe the data:

```
min(v1)    ①
max(v1)    ①
mean(v1)   ②
median (v1)③
length(v1) ④
summary(v1)⑤
```

① Returns the (regular or parallel) maxima and minima of the input values.

② Generic function for the (trimmed) arithmetic mean.

③ Compute the sample median.

④ Get the length of vector.

⑤ Generic function used to produce result summaries of the results of various model fitting functions.

An essential data structure that is frequently used in R is the **dataframe**. This convenient data structure can include vectors of different type. You can create a dataframe with the function **data.frame()**. Create a data frame by typing:

```
df<-data.frame(v1,v2)
```

A categorical variable can be created in R with **as.factor()** function. The as.factor function is used to encode a vector as a factor (the terms 'category' and 'enumerated type' are also used for factors). Let's try to transform v2 vector into a categorical variable:

```
c <- as.factor(v2)
class(c)
```

In this case, the **c** variable will have three entries and two levels(categories), **red** and **blue**. You can also trnasform the categorical variable into a numeric varible with the **as.numeric()** function:

```
n<-as.numeric(c)
```

In this case it returns the index numbers of the elements, after they have been sorted by alphabetic order. So, the vector **n** will be composed by 2,1 and 2.

1.2. Vectorization and loops

One of the most important features of R is its vectorized capabilities. Vectorization indicates that a function operates on a whole vector of values at the same time and not just on a single value. Many of the basic functions in R, are already vectorized.
Take for example two vectors:

```
x<-c(2,4,5)
y<-c(3,5,8)

x*y       ①
sum(x)    ②
```

① Multiply the vectors.

② Sum the x vector

The output:

```
> x*y
[1]  6 20 40

> sum(x)
[1] 11
```

Another way of iterating through a vector, similar to other programming languages is to use the loops.
For example, if you want to replicate the vectorized product of two vectors using a loop, you can utilize the **for** statement:

```
z<-c()                    ①
  for (i in 1:length(x)){ ②
    z[i]<-x[i] * y[i]     ③
  }
  print(z)
```

① Initialize a vector called z.

② Repeat the oparation inside {}, start from 1 to the length(x) = 3.

③ The i-th element of z = i-th element of x multiplied by the i-th elemento of y.

The vectorization is also more efficient compared to the loops. For instance, let's create two functions:

```
vectorized_prod<- function(x,y) {  ①
  y <- x * y
  return(y)
}
vectorized_prod(x,y)


loop_prod <-function(x,y){         ②
  z<-c()
  for (i in 1:length(x)){
    z[i]<-x[i] * y[i]
  }
  print(z)
}
loop_prod(x,y)
```

① Vectorized function.

② Function with **for** loop.

Compare the efficiency between vectorized operations and loops installing and loading the **microbenchmark package**:

```
install.packages("microbenchmark")

library(microbenchmark)

microbenchmark(vectorized_prod(x,y),
               loop_prod(x,y),
               times = 1000)
```

The output:

```
Unit: nanoseconds
                  expr   min    lq    mean median    uq    max neval
 vectorized_prod(x, y)   500   800  1440.7   1300  1500  19000  1000
       loop_prod(x, y) 45100 49100 70642.3  53100 87500 239000  1000
```

As you can see, running 1000 times each function, the vectorized one results faster.

A very convenient and efficient function in R is the **ifelse()** function. It's basically a vectorized version of an if ... else control structure every programming language has. While using if ... else statement you need to run:

```
z<-c()
for (i in 1:length(x)){
  if (x[i]<4) {z[i]<-0}
  else  {z[i]<-1}
}
print(z)
```

The same result can be archived with the vectorized version using only a single row of code:

```
z<-ifelse(x<4,0,1)

print(z)
```

1.3. Import and export the data

R has different built in functions to loading datasets, depending from the type of file used as input. For example, you can import data from text, csv, excel and Rda files. Almost every single type of file that you want to get into R requires its own function. A text file with .txt exstension and csv files, can be easily imported with the basic R function **read.table()**:

```
df <- read.table("file_name.txt", header = TRUE)  ①
df <- read.table("file_name.csv",header=TRUE,sep=",")  ②
```

① If the first row contains the columns names then set header=FALSE, otherwise set it TRUE (by default is setted to FALSE)

② If the first row contains the columns names then set header=FALSE, otherwise set it TRUE (by default is setted to FALSE). You also have to declare the separator type (by default is empty "").

You can also import data through RStudio interface:

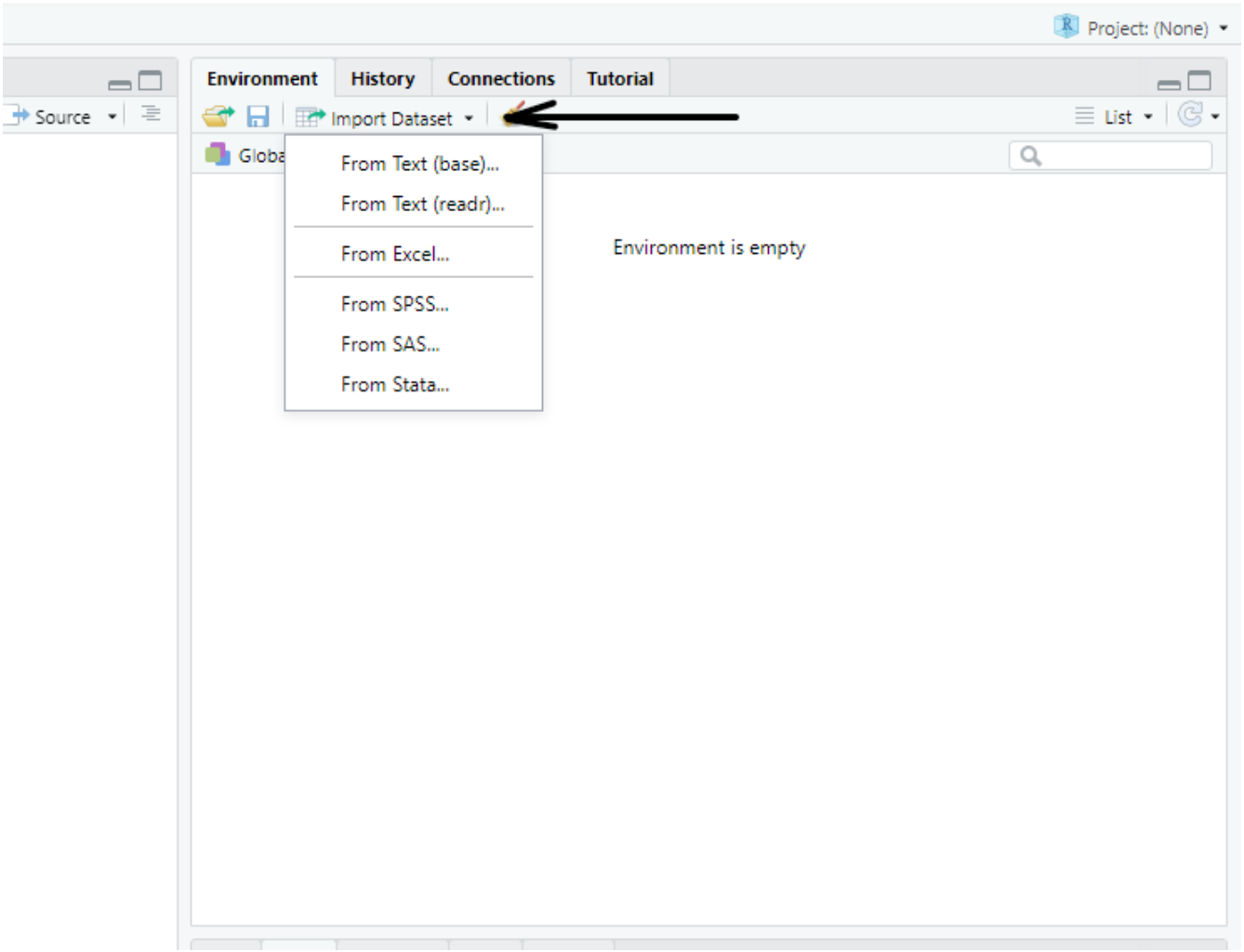


Figure 5. Import data with RStudio.

There are also two specific functions to read csv files, **read.csv()** and **read.csv2()**. They are similar to **read.table()** with the difference that the **Header** is setted to **TRUE** by default.

```
df <- read.csv("file_name.csv", header = TRUE,sep=",")
df <- read.csv2("file_name.csv",header=TRUE,sep=",")
```

An easy way to save you dataframe in to a **.txt** or **csv** file is:

```
write.csv(df, file = "file_name.txt")
write.csv(df, file = "file_name.csv")
```

There are several packages that import excel files. We will use for this the **readxl** package. The readxl package allows R users to easily load Excel files, as follows:

```
library(readxl)

df <- read_excel("test.xlsx",sheet = 1,col_names = TRUE)
```

The simplest way to save a dataframe in an excel format is:

```
library(writexl)

write_xlsx(df, "prova.xlsx")
```

If your data file is one that has been saved in R as an **.rdata** file, you can import your data with the built in **load()** function:

```
df<-load("file_name.RDA")
```

You can also save a dataframe with **.RDA** extension by simply typing:

```
save(df, file="file_name.RDA")
```

Some other useful basic funcions frequenly used in R are the following:

```
save.image() ①
list.files() ②
rm(df) ③
rm(list=ls()) ④
options(scipen=999) ⑤
```

① Save the objects of the workspace(environment) directly in the working directory.

② List of the files in the working directory.

③ Remove df object from the environement.

④ Clear memory by removing the objects from your environment.

⑤ Prevent scientific notation, by simply using a large positive value like 999. A penalty to be applied when deciding to print numeric values in fixed or exponential notation.

1.4. Introduction to hospital data

In almost all the laboratory sessions we will deal with an open source dataset from New York hospitals. The data can be downloaded here, searching for example, for **Hospital Inpatient Discharges (SPARCS De-Identified) 2017**: NY Hospital Data

Complete the following steps to have all the files you need:

- Export the dataset(e.g. for 2017: Hospital_Inpatient_Discharges*SPARCS_De-Identified*_2017.csv).

- Download guides in attachment(s) for each year e.g. 2017:
  - NYSDOH_SPARCS_De-Identified_Overview_2017.pdf
  - NYSDOH_SPARCS_De-Identified_Data_Dictionary_2017.pdf
  - NYSDOH_SPARCS_De-Identified_Intro_to_SPARCS.pdf

After download, it is possible to import the NY hospital dataset by typing into the console:

```
hospdata<-read.csv("NYSDOH_HospitalInpatientDischarges_SPARCS_De-Identified_2017.csv",
                   stringsAsFactors = F,header=T,sep=",") ①
class(dataset) ②
```

① Load the downloaded dataset as a data frame, use first row as header and consider comma as separator.

② Check the class of the data.

You can invoke the viewer in a console by calling the **View()** function on the data frame you want to look at. For instance, to view the hospdata dataset, run the command:

```
View(hospdata)
```

You can also open the data viwer by clicking on the data on the environment panel:

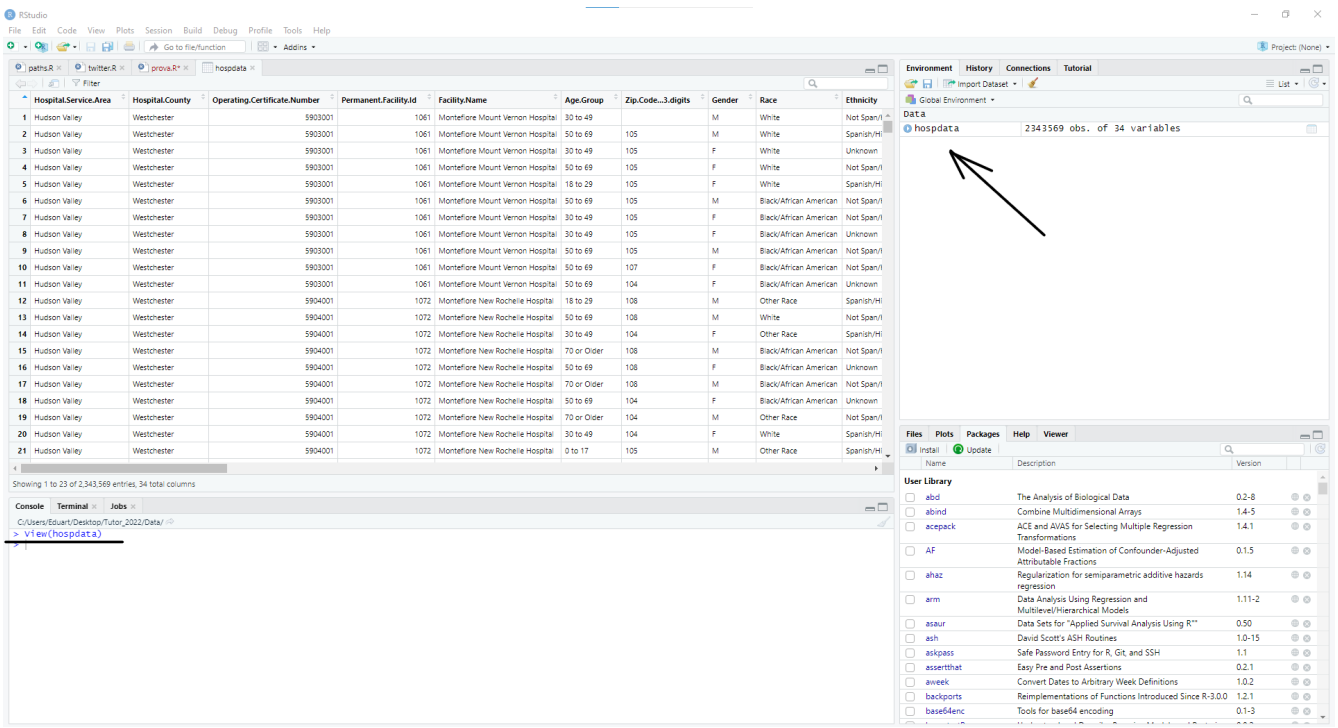
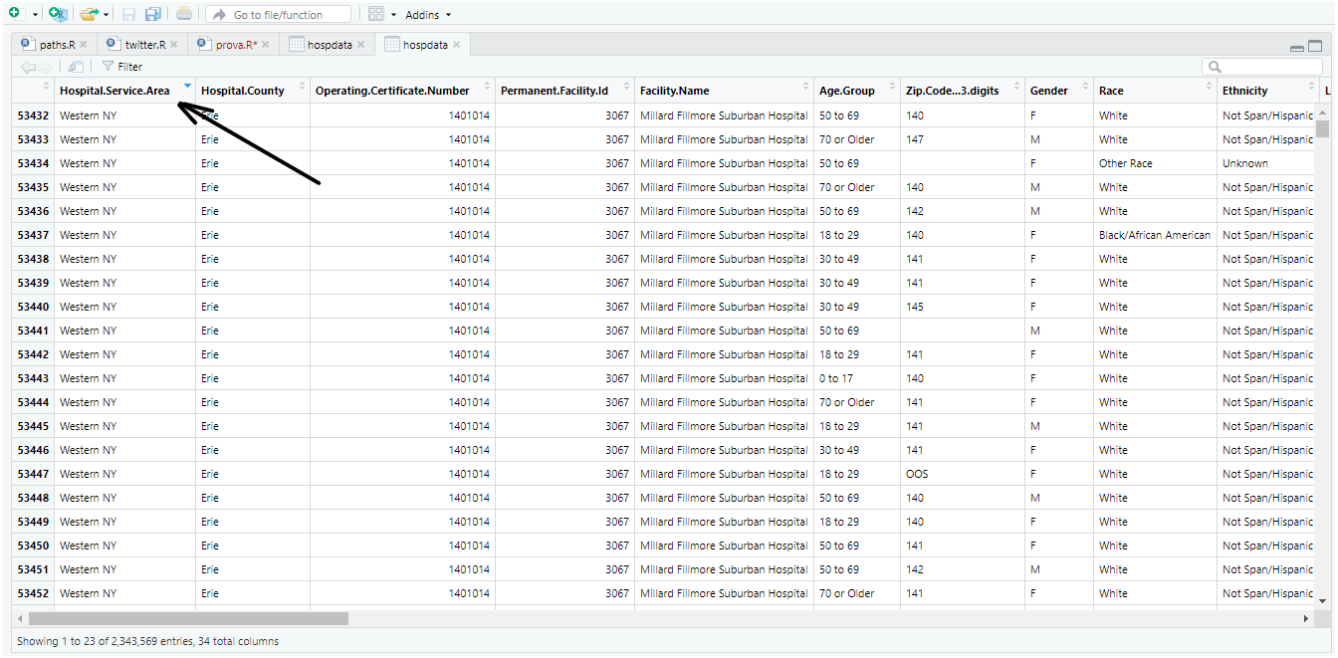


Figure 6. Data viewer.

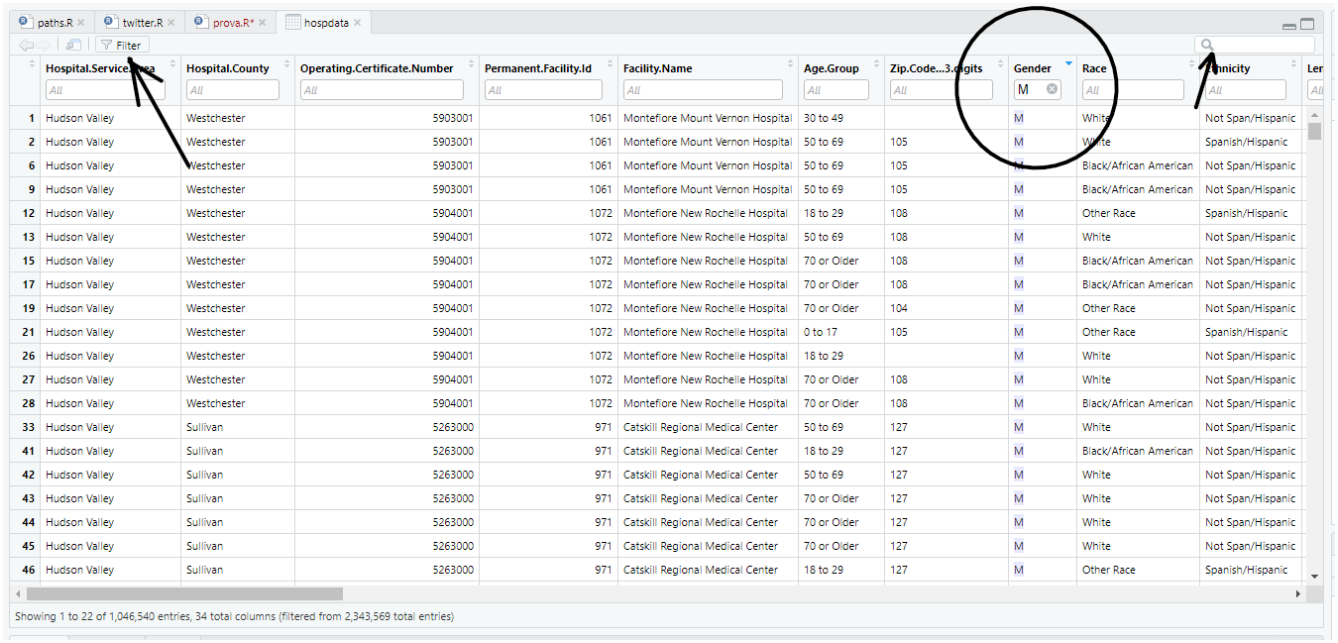
You can sort the data by any column, by just by clicking on the column. You can also reverse the order by clicking again on the same column:



	Hospital.Service.Area	Hospital.County	Operating.Certificate.Number	Permanent.Facility.Id	Facility.Name	Age.Group	Zip.Code...3.digits	Gender	Race	Ethnicity
53432	Western NY	Erie	1401014	3067	Millard Fillmore Suburban Hospital	50 to 69	140	F	White	Not Span/Hispanic
53433	Western NY	Erie	1401014	3067	Millard Fillmore Suburban Hospital	70 or Older	147	M	White	Not Span/Hispanic
53434	Western NY	Erie	1401014	3067	Millard Fillmore Suburban Hospital	50 to 69		F	Other Race	Unknown
53435	Western NY	Erie	1401014	3067	Millard Fillmore Suburban Hospital	70 or Older	140	M	White	Not Span/Hispanic
53436	Western NY	Erie	1401014	3067	Millard Fillmore Suburban Hospital	50 to 69	142	M	White	Not Span/Hispanic
53437	Western NY	Erie	1401014	3067	Millard Fillmore Suburban Hospital	18 to 29	140	F	Black/African American	Not Span/Hispanic
53438	Western NY	Erie	1401014	3067	Millard Fillmore Suburban Hospital	30 to 49	141	F	White	Not Span/Hispanic
53439	Western NY	Erie	1401014	3067	Millard Fillmore Suburban Hospital	30 to 49	141	F	White	Not Span/Hispanic
53440	Western NY	Erie	1401014	3067	Millard Fillmore Suburban Hospital	30 to 49	145	F	White	Not Span/Hispanic
53441	Western NY	Erie	1401014	3067	Millard Fillmore Suburban Hospital	50 to 69		M	White	Not Span/Hispanic
53442	Western NY	Erie	1401014	3067	Millard Fillmore Suburban Hospital	18 to 29	141	F	White	Not Span/Hispanic
53443	Western NY	Erie	1401014	3067	Millard Fillmore Suburban Hospital	0 to 17	140	F	White	Not Span/Hispanic
53444	Western NY	Erie	1401014	3067	Millard Fillmore Suburban Hospital	70 or Older	141	F	White	Not Span/Hispanic
53445	Western NY	Erie	1401014	3067	Millard Fillmore Suburban Hospital	18 to 29	141	M	White	Not Span/Hispanic
53446	Western NY	Erie	1401014	3067	Millard Fillmore Suburban Hospital	30 to 49	141	F	White	Not Span/Hispanic
53447	Western NY	Erie	1401014	3067	Millard Fillmore Suburban Hospital	18 to 29	OOS	F	White	Not Span/Hispanic
53448	Western NY	Erie	1401014	3067	Millard Fillmore Suburban Hospital	50 to 69	140	M	White	Not Span/Hispanic
53449	Western NY	Erie	1401014	3067	Millard Fillmore Suburban Hospital	18 to 29	140	F	White	Not Span/Hispanic
53450	Western NY	Erie	1401014	3067	Millard Fillmore Suburban Hospital	50 to 69	141	F	White	Not Span/Hispanic
53451	Western NY	Erie	1401014	3067	Millard Fillmore Suburban Hospital	50 to 69	142	M	White	Not Span/Hispanic
53452	Western NY	Erie	1401014	3067	Millard Fillmore Suburban Hospital	70 or Older	141	F	White	Not Span/Hispanic

Figure 7. Order the data.

You can apply filters by clicking the filter icon and you can search for text across all the columns by typing in the search box. For istance, let's filter the data by males:



	Hospital.Service.Area	Hospital.County	Operating.Certificate.Number	Permanent.Facility.Id	Facility.Name	Age.Group	Zip.Code...3.digits	Gender	Race	Ethnicity
1	Hudson Valley	Westchester	5903001	1061	Montefiore Mount Vernon Hospital	30 to 49		M	White	Not Span/Hispanic
2	Hudson Valley	Westchester	5903001	1061	Montefiore Mount Vernon Hospital	50 to 69	105	M	White	Spanish/Hispanic
6	Hudson Valley	Westchester	5903001	1061	Montefiore Mount Vernon Hospital	50 to 69	105	M	Black/African American	Not Span/Hispanic
9	Hudson Valley	Westchester	5903001	1061	Montefiore Mount Vernon Hospital	50 to 69	105	M	Black/African American	Not Span/Hispanic
12	Hudson Valley	Westchester	5904001	1072	Montefiore New Rochelle Hospital	18 to 29	108	M	Other Race	Spanish/Hispanic
13	Hudson Valley	Westchester	5904001	1072	Montefiore New Rochelle Hospital	50 to 69	108	M	White	Not Span/Hispanic
15	Hudson Valley	Westchester	5904001	1072	Montefiore New Rochelle Hospital	70 or Older	108	M	Black/African American	Not Span/Hispanic
17	Hudson Valley	Westchester	5904001	1072	Montefiore New Rochelle Hospital	70 or Older	108	M	Black/African American	Not Span/Hispanic
19	Hudson Valley	Westchester	5904001	1072	Montefiore New Rochelle Hospital	70 or Older	104	M	Other Race	Not Span/Hispanic
21	Hudson Valley	Westchester	5904001	1072	Montefiore New Rochelle Hospital	0 to 17	105	M	Other Race	Spanish/Hispanic
26	Hudson Valley	Westchester	5904001	1072	Montefiore New Rochelle Hospital	18 to 29		M	White	Not Span/Hispanic
27	Hudson Valley	Westchester	5904001	1072	Montefiore New Rochelle Hospital	70 or Older	108	M	White	Not Span/Hispanic
28	Hudson Valley	Westchester	5904001	1072	Montefiore New Rochelle Hospital	70 or Older	108	M	Black/African American	Not Span/Hispanic
33	Hudson Valley	Sullivan	5263000	971	Catskill Regional Medical Center	50 to 69	127	M	White	Not Span/Hispanic
41	Hudson Valley	Sullivan	5263000	971	Catskill Regional Medical Center	18 to 29	127	M	Black/African American	Not Span/Hispanic
42	Hudson Valley	Sullivan	5263000	971	Catskill Regional Medical Center	50 to 69	127	M	White	Not Span/Hispanic
43	Hudson Valley	Sullivan	5263000	971	Catskill Regional Medical Center	70 or Older	127	M	White	Not Span/Hispanic
44	Hudson Valley	Sullivan	5263000	971	Catskill Regional Medical Center	70 or Older	127	M	White	Not Span/Hispanic
45	Hudson Valley	Sullivan	5263000	971	Catskill Regional Medical Center	70 or Older	127	M	White	Not Span/Hispanic
46	Hudson Valley	Sullivan	5263000	971	Catskill Regional Medical Center	18 to 29	127	M	Other Race	Spanish/Hispanic

Figure 8. Filter the data.

If you try to sort or filter the NY hospital data it seems that nothing heppens, this is just because the dataset is heavy. Just wait!

You can access the vector's or dataframe's values by using square brackets. For instance, run the code:

```
gender[2]           ①
hospdata[1,2]       ②
hospdata[1:10,]     ③
hospdata[,1:2]      ④
hospdata[,"Gender"] ⑤
```

① Returns the second element of the vector gender.

② Returns the element in the first row of the second column.

③ Select the first 10 rows of hospdata.

④ Select the first 2 columns of hospdata.

⑤ Returns the column Gender from hospdata.

2. Hospital data

2.1. Hospital data management

In this session we will start exploring the open source dataset of New York hospitals. The first steps involve clearing the environment:

```
rm(list=ls())
```

We should start loading the dataset by typing into console:

```
hospdata<-read.csv("NYSDOH_HospitalInpatientDischarges_SPARCS_De-Identified_2017.csv",
                   stringsAsFactors = F,header=T,sep=",") ①
class(dataset) ②
```

① Load the downloaded dataset as a data frame, use first row as header and consider comma as separator.

② Check the class of the data.

You may need to re-encode a variable, by changing the values. For istance, let's create a **dummy variable** (with 0 and 1 entries) for the variable **Gender** in the hospdata dataset. In this case we will assign 1 for the males and 0 for females. We will use the basic build in **ifelse()** function:

```
hospdata$males<-0  ①
hospdata$males<-ifelse(hospdata$Gender=="U",NA,hospdata$males) ②
hospdata$males<-ifelse(hospdata$Gender=="M" &
!is.na((hospdata$Gender)),1,hospdata$males) ③
```

① Create the variable named males assigning 0 values.

② If the variable **Gender** is equal to **U** we assign **NA** to the variable **males**.

③ If the variable Gender is equal to **M** and is not **NA** we assign **1** to the variable males. **Note**: 0 entries assigned to the females after excluding males and NA's.

To access one variable in a dataset, use the dollar sign **$**.

The **ifelse()** function returns a value with the same shape as condition which is filled with elements selected from either **yes** or **no** depending if the condition is **TRUE** or **FALSE**. In general, the use of the function can be reported as:

```
ifelse(condition, yes, no)
```

If you need to know and store the unique catogories of a factor variable, you can save the levels by assigning them to a vector. For example, let's store the levels of the variable **Gender** from the

hospdata dataset:

```
gender<- levels(factor(hospdata$Gender))
gender  ①
```

① Outputs: "F" "M" "U"

You can access the vector's values by using square brackets. For instance, check the second element of the vector **gender**:

```
gender[2]  ①
```

① Outputs: "M"

You can also check number of unique entries of a vector by using the **length()** and **unique()** functions by typing:

```
length(unique(gender)) ①
```

① Outputs: 3

Using the **length()** function make possible to get or even set the length of an object in R. The **unique()** function returns a vector with the duplicate elements or rows removed.

If you neeed to store the levels of a categorical variable into the working space you can use the **assign()** function. For example, you can store in the global envoirment the levels of the vector **gender** into a vector called **gen_lev** using the command:

```
assign("gen_lev",gender,envir=.GlobalEnv)  ①
```

The **assign()** is useful to assign a value to a name in a specific environment.

In a dataset with a lot of entries like hospdata, it can be difficult to explore all variables without filtering or searching text using specific words. A function that searches for matches to an argument pattern within each element of a character vector is the **grep()** function. This function outputs the index number of entry or entries matching the text being searched. For instance, we can search the **M** entry in the gender variable:

```
grep("M",gender)  ①
gender[2]  ②
```

① Outputs: 2

② Access to the entry of the variable that corrisponds to the index 2.

If you need to perform a replacement of entries of all matches, you can use the **gsub** function. For example, remove comma and different entries from **0-9**. digits, from **Total.Charges** variable:

```
hospdata$cost<-as.numeric(gsub("[^0-9.]","",hospdata$Total.Charges))
View(hospdata[1,c("Total.Charges","cost")]) ①
```

① View the first row. The original value **114,168.00** is replaced with **114168**.

Before closing your work session, it is possible to save essential information in a **.txt** fil with **sink()** function. For instance, we can save the levels of the gender variable in a text file called levels_of_gender:

```
sink("levels_of_gender.txt") ①
cat ("\n")  ②
cat ("Gender\n") ③
cat ("\n")
print(gender)
sink() ④
```

① Create a .txt file named **levels_of_gender**.

② Skip the row.

③ Write **Gender** and skip the row.

④ Ends the diversion.

In other words the **sink()** function diverts the R output to a connection and also stops such diversions.

2.2. Transformation of the dataset

The dataset from the NY hospitals, as shown from the loading time, is very heavy to manage. Some attribuites of the dataset contain text entries that are repeated for every observation, which increases the computing power required to process data significantly. To scale the original dataset down, we need to process the data through the following function:

```
create_hospdata<-function(input,output,value_labels="value_labels.txt") { ①

  hospdata<-read.csv(file=input,stringsAsFactors = F,header=T,sep=",")
  print(names(hospdata))

  hospdata$males<-0  ②
  hospdata$males<-ifelse(hospdata$Gender=="U",NA,hospdata$males) ③
  hospdata$males<-ifelse(hospdata$Gender=="M" & !is.na((hospdata$Gender))
                         ,1,hospdata$males) ④

  ny_counties<-levels(factor(hospdata$Hospital.County)) ⑤
  ny_areas<-levels(factor(hospdata$Hospital.Service.Area))  ⑤
  ny_age<-levels(factor(hospdata$Age.Group))    ⑤
  ny_adm_types<-levels(factor(hospdata$Type.of.Admission)) ⑤
  ny_races<-levels(factor(hospdata$Race))  ⑤
```

```
ny_ethnicities<-levels(factor(hospdata$Ethnicity)) ⑤
ny_diags<-levels(factor(hospdata$CCS.Diagnosis.Description)) ⑤
ny_procs<-levels(factor(hospdata$CCS.Procedure.Description))  ⑤
ny_dispositions<-levels(factor(hospdata$Patient.Disposition))  ⑤
ny_severities<-levels(factor(hospdata$APR.Severity.of.Illness.Description)) ⑤
ny_risks<-levels(factor(hospdata$APR.Risk.of.Mortality)) ⑤

ny_counties[2]  ⑥
length(unique(ny_counties)) ⑥

assign("ny_counties",ny_counties,envir=.GlobalEnv)  ⑦
assign("ny_areas",ny_areas,envir=.GlobalEnv)  ⑦
assign("ny_age",ny_age,envir=.GlobalEnv)   ⑦
assign("ny_adm_types",ny_adm_types,envir=.GlobalEnv)  ⑦
assign("ny_races",ny_races,envir=.GlobalEnv)  ⑦
assign("ny_ethnicities",ny_ethnicities,envir=.GlobalEnv)  ⑦
assign("ny_diags",ny_diags,envir=.GlobalEnv)   ⑦
assign("ny_procs",ny_procs,envir=.GlobalEnv)   ⑦
assign("ny_dispositions",ny_dispositions,envir=.GlobalEnv)  ⑦
assign("ny_severities",ny_severities,envir=.GlobalEnv)  ⑦
assign("ny_risks",ny_risks,envir=.GlobalEnv)  ⑦

hospdata$ny_hosp_id<-as.numeric(as.factor(hospdata$Facility.Name))  ⑧
hospdata$ny_county<-as.numeric(as.factor(hospdata$Hospital.County))  ⑧
hospdata$ny_area<-as.numeric(as.factor(hospdata$Hospital.Service.Area))  ⑧
hospdata$cl_age<-as.numeric(as.factor(hospdata$Age.Group))  ⑧
hospdata$zipcode<-as.numeric(as.factor(hospdata$Zip.Code...3.digits))   ⑧
hospdata$adm_type<-as.numeric(as.factor(hospdata$Type.of.Admission))  ⑧
hospdata$race<-as.numeric(as.factor(hospdata$Race))  ⑧
hospdata$ethnicity<-as.numeric(as.factor(hospdata$Ethnicity))  ⑧
hospdata$los<-hospdata$Length.of.Stay
hospdata$disposition<-as.numeric(as.factor(hospdata$Patient.Disposition))  ⑧
hospdata$diagnosis<-as.numeric(as.factor(hospdata$CCS.Diagnosis.Description))  ⑧
hospdata$procedure<-as.numeric(as.factor(hospdata$CCS.Procedure.Description))  ⑧
hospdata$drg<-as.numeric(as.factor(hospdata$APR.DRG.Code))  ⑧
hospdata$mdc<-as.numeric(as.factor(hospdata$APR.MDC.Code))  ⑧
hospdata$severity<-as.numeric(as.factor(
                   hospdata$APR.Severity.of.Illness.Description))  ⑧
hospdata$risk<-as.numeric(as.factor(hospdata$APR.Risk.of.Mortality))  ⑧
hospdata$payment_type<-as.numeric(as.factor(hospdata$Payment.Typology.1))  ⑧
hospdata$cost<-hospdata$Total.Charges

hospdata$surgical<-0  ⑨
hospdata$surgical<-ifelse(hospdata$APR.Medical.Surgical.Description==
                          "Not Applicable", NA,hospdata$surgical) ⑨
hospdata$surgical<-ifelse(hospdata$APR.Medical.Surgical.Description=="Surgical" &
                          !is.na((hospdata$surgical)),1,hospdata$surgical) ⑨

grep("infarction",ny_diags) ⑩
ny_diags[] ⑩
```

```
  ny_dispositions[6]  ⑪
  hospdata$dead<-0  ⑪
  hospdata$dead<-ifelse(hospdata$disposition==6,1,hospdata$dead)  ⑪

  hospdata$ami<-0 ⑫
  hospdata$ami<-ifelse(hospdata$diagnosis==8,1,hospdata$ami)  ⑫

  hospdata$cost<-as.numeric(gsub("[^0-9.]","",hospdata$cost)) ⑬
  hospdata$los <-as.numeric(gsub("[^0-9.]","",hospdata$los))  ⑬

  ny_hospdata<-hospdata[,c("ny_hosp_id","ny_county","ny_area","cl_age",  ⑭
                           "males","zipcode","adm_type","race","ethnicity","los",  ⑭

"disposition","diagnosis","procedure","drg","mdc","severity",  ⑭
                           "risk","payment_type","cost","surgical",   ⑭
                           "dead","ami")]  ⑭

  write.csv(ny_hospdata,output,row.names=FALSE)  ⑮

  sink(value_labels)  ⑯
  cat ("\n")
  cat ("Counties\n")
  cat ("####################################\n")
  cat ("\n")
  print(ny_counties)
  cat ("\n")
  cat ("Areas\n")
  cat ("####################################\n")
  cat ("\n")
  print(ny_areas)
  cat ("\n")
  cat ("Age\n")
  cat ("####################################\n")
  cat ("\n")
  print(ny_age)
  cat ("\n")
  cat ("Adm Types\n")
  cat ("####################################\n")
  cat ("\n")
  print(ny_adm_types)
  cat ("\n")
  cat ("Races\n")
  cat ("####################################\n")
  cat ("\n")
  print(ny_races)
  cat ("\n")
  cat ("Ethnicities\n")
  cat ("####################################\n")
  cat ("\n")
  print(ny_ethnicities)
  cat ("\n")
```

```
  cat ("Diagnoses\n")
  cat ("###################################\n")
  cat ("\n")
  print(ny_diags)
  cat ("\n")
  cat ("Procedures\n")
  cat ("###################################\n")
  cat ("\n")
  print(ny_procs)
  cat ("\n")
  cat ("Dispositions\n")
  cat ("###################################\n")
  cat ("\n")
  print(ny_dispositions)
  cat ("\n")
  cat ("Risks\n")
  cat ("###################################\n")
  cat ("\n")
  print(ny_risks)
  sink()
  rm(hospdata) ⑰
}
```

① Create function with 3 parameters.

② Create the variable named males assigning 0 values.

③ If the variable **Gender** is equal to **U** we assign **NA** to the variable **males**.

④ If the variable Gender is equal to **M** and is not **NA** we assign **1** to the variable males. **Note**: 0 entries are for females after excluding males and NA's.

⑤ Save the levels of selected variable. Assign the unique levels of the variable considering it as factor variable.

⑥ Get access to the rows and columns of a dataframe through **[]** symbols. Here, show 2° element of ny_counties and show the number of unique values of the variable.

⑦ Store the levels into the working space. For example the first row: assign ny_counties values to ny_counties variable and store in global envoirment.

⑧ Turn the levels of a variable ,firstly transformed in a factor variable, in numeric entries and assign the index number.

⑨ Create a dummy variable for surgical.The not available entries were recorded as Not Applicable in the original variable.

⑩ **Ami** stands for **Acute Myocardial Infarction**. Grep function search for matches. Output the index number of **Acute myocardial infarction** from the **ny_diags** stored before.

⑪ Create a dummy variable for **expired** patients (dead). Disposition = 6 matches with expired.

⑫ Create a dummy variable for **Acute myocardial infarction-AMI**. The index for ami equal to 8 was checked before (with **grep("infarction",ny_diags)**) from ny_diags variable.

⑬ Remove comma and different entries, exept **0-9**. digits, from cost variable.

⑭ Create a dataframe with the selected variables. Include all the rows and the specified selected columns.

⑮ Save the re-encoded and reduced data frame in .csv format in the working directory. Row names setted as false because first column does not contain the names of the observations.

⑯ Save the levels of factor variables in a text file (.txt).

⑰ Remove hospdata (the complete dataset) from Global environment to clear the memory.

The arguments needed to run the function are the **NYSDOH**, the name of the new csv dataset and the name of the text file with the stored levels. The output will be a reduced **.csv** dataset saved directly into the working directory and a text file with the levels of the factor variables, saved inside the function.

We can now check how the function works by typing:

```
create_hospdata(input=
                "NYSDOH_HospitalInpatientDischarges_SPARCS_De-Identified_2017.csv",
                output= "ny_hospdata_2017.csv",
                value_labels= "value_labels.txt")
```

Now, you can load the reduced dataset:

```
ny_hospdata<-read.csv("ny_hospdata_2017.csv")
```

Save the data in a **.RDA** format to load them quicker the next times:

```
ny_hospdata_2017<-ny_hospdata ①
amidata_2017<-amidata

save(ny_hospdata_2017,file="ny_hospdata_2017.Rda") ②
save(amidata_2017,file="amidata_2017.Rda")
```

① Rename datasets.

② Save to rda file. The RDA format is lighter and loads quicker.

3. Life expactancy and inital analysis of hospital data

3.1. Life expectancy

In this section we will perform some demographic analysis. The package including all functions for demographic analysis, e.g. lifetable calculations, Lee-Carter modelling and functional data analysis of mortality rates is called **demography**. It also includes some special R objects defined as demogdata, which can be extracted and analysed.

Load the the package:

```
library(demography) ①
demography:: ②
```

① Load **demography** package.

② A simple way to check what a package contains and to access to the functions of a package, **RStudio** will suggest the functions after typing **::**.

It may happen that two different packages have functions with the same name. In this case, it is better to specify the package and funtion with **package::function**.

3.1.1. Demogdata management

The **fr.mort** object from demograpgy package is a **demogdata** defined object. It includes the mortality rates of the total, female and male population of France from 1816 to 2006.

Explore the object by typing in the console:

```
fr.mort ①
class(fr.mort) ②
names(fr.mort) ③
fr.mort$year ④
```

① Show the object.

② Show class of the object. This is a special R object.

③ Return the names of the variables.

④ Show the content of one variable.

Continue to explore and extract data by typing:

```
p<-as.data.frame(fr.mort$rate) ①
names(p) ②
m_x<-p$total.2000 ③
x<-as.data.frame(fr.mort$age) ④
names(x)<-c("x") ⑤
life_table<-as.data.frame(cbind(x,m_x)) ⑥
```

① Turn **rate** columns into a data frame called **p**.

② Show the names of the columns.

③ Assign total.2000 vector from p dataframe to m_x.

④ Assign age vector from fr.mort object touni x and turn **AGE** columns into a data frame.

⑤ The name of the variable assigned automatically is fr.mort$age, we rename the column of x, calling it x.

⑥ Combines by columns a sequence of vector, matrix or dataframe.Binds m_x and x by columns. It exists also rbind, it binds the data by rows.

Generic X-Y plotting with R:

```
plot(life.expectancy(fr.mort),ylab="Life expectancy") ①
```

① **plot()** is a generic function for plotting R objects. Here we plot life.expectancy from demography package.

The output plot for life expactancy will be the following:

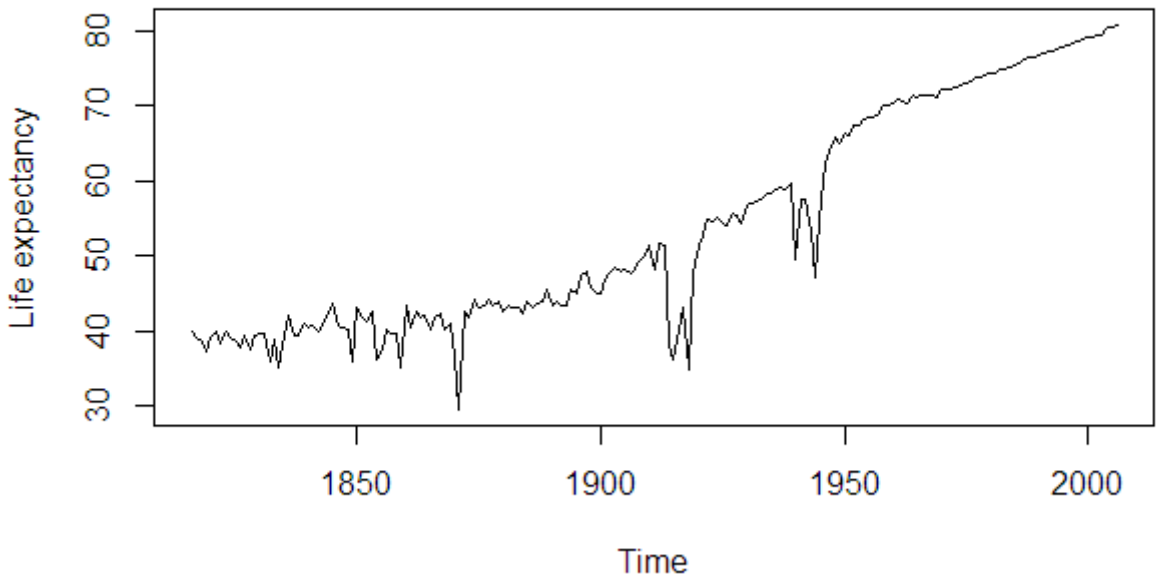


Figure 9. Life expectancy in France from 1816 to 2006.

We can also create a life table using the **lifetable()** function from demography package:

```
lifetable(fr.mort,years=2000,type="period")
```

To reproduce the life table with selected columns and personalized names, type:

```
z<-extract.years(fr.mort,2006) ①
a<-lifetable(z) ②
names(a)
w<-data.frame(a$age,a$mx,a$qx,a$lx,a$Lx,a$Tx,a$ex)  ③
names(w)<-c("Age","Mortality death rate",                ④
            "Probability of death","Suvivers at age x",
            "Person years lived at age x",
            "Total number of years lived from age x",
            "Life expectancy at age x")
print (w)
```

① Demogdata object for france mortality for the year 2006.

② Generate lifetables from z.

③ Create a dataframe with only selected columns.

④ Rename the columns of w.

Now, costruct the cohort from specific age values at each year in the data series by typing in your console:

```
lifetable(fr.mort,type="cohort",age=0, years=1900:1905,max.age=10) ①
trend<-as.data.frame(life.expectancy(fr.mort,type='period',age=0)) ②
plot(trend)
```

① Create life tables from 0 to 10 years old for the years from 1900 to 1905.

② Life expectancy at birth for cross-section of time.

We can also create our own function for life tables by looping over columns and rows. For exaple, we can compute **the number of persons surviving to exact age x (lx)**:

```
my_lifetable<- function(input) { ①
  for (i in 1:nrow(input)) {
     if(i==1) {
       input[1,c("l_x")]<-100000 ②
     }
     else {
       input[i,c("l_x")]<-round(input[i-1,c("l_x")]-input[i-1,c("m_x")]* ③
       input[i-1,c("l_x")])
     }
  }
return(input)
}

lt<-my_lifetable(life_table) ④
print(lt)
```

① Declare the function with one parameter.

② For the first row set lx to 100000.

③ Otherwise, calculate lx.

④ **life_table** contains the age and age specific mortality rates.

EXERCISE 3.1
Complete the calculation of the life table from the function **my_lifetable**.

Lx=0.5*(lx+1 + lx)
Tx=sum Lx from i to nrow of Lx

3.2. Initial data analysis

Clear the environment, load the functions, install and load the packages for the analysis:

```
rm(list=ls())

source("methods_lab_functions.R")

installLibraries(c("gmodels","epitools","Hmisc"))

library(gmodels)
library(epitools)
library(Hmisc)
```

Also, load the data from NY hospitals:

```
load("ny_hospdata_2017.Rda")
```

We can use a set of basic tecniques of data processing e.g. filtering the data through a selection of subset of rows. For example, we can obtain a dataset only with patients coded with a specific diagnosis by using the following:

```
amidata<-ny_hospdata[ny_hospdata$ami==1,]  ①
cordata<-ny_hospdata[ny_hospdata$diagnosis==69,]  ②
```

① Select the rows with only Acute myocardial infarction-AMI inpatients.

② Select only patients with Coronary Atherosclerosis.

Here is another example on how to create a meaningful clinical dummy variable for SURGICAL interventions:

```
amidata$ptca<-NA  ①
amidata$ptca<-ifelse(amidata$procedure==185,1,amidata$ptca)  ②
amidata$ptca<-ifelse(amidata$procedure!=185 &
!is.na((amidata$procedure)),0,amidata$ptca)  ③
```

① Create ptca variable assigning NA entries

② If procedure is equal to 185 assign 1

③ Otherwise assign 0 (if procedure **is not equal to 185** and **is not NA**).

We can also re-encode the values of a variable as following:

```
①
amidata$risky<-NA
amidata$risky<-ifelse(amidata$risk=="4",0,amidata$risky)
amidata$risky<-ifelse(amidata$risk=="5",1,amidata$risky)
amidata$risky<-ifelse(amidata$risk=="3",2,amidata$risky)
amidata$risky<-ifelse(amidata$risk=="2",3,amidata$risky)


②
amidata$severe<-NA
amidata$severe<-ifelse(amidata$severity=="4",0,amidata$severe)
amidata$severe<-ifelse(amidata$severity=="5",1,amidata$severe)
amidata$severe<-ifelse(amidata$severity=="3",2,amidata$severe)
amidata$severe<-ifelse(amidata$severity=="2",3,amidata$severe)
```

① Invert the values of the variable **risky** for a miningful interpretation of the values. The original levels include: Extreme=2, Major=3, Minor=4, Moderate=5. The natural order can be set as: 0 → Minor, 1 → Moderate, 2 → Major and 3 → Extreme.

② Invert the values of the variable **severity**. The original levels include: Extreme=2, Major=3, Minor=4, Moderate=5. The natural order can be set as: 0 → Minor, 1 → Moderate, 2 → Major and 3 → Extreme.

Save the the two subsets including the new variables:

```
save(amidata,file="amidata_2017.Rda")
save(cordata,file="cordata_2017.Rda")
```

3.2.1. Data exploration

We can start computing some quick analysis by creating contingency tables that displays the frequency distribution of the variables. Check frequencies by using **table** and **prob.table** built-in functions:

```
table(ny_hospdata$ami,ny_hospdata$dead)  ①
table(amidata$ami,amidata$dead)
table(amidata$severe,amidata$dead)

prop.table(table(ny_hospdata$ami,ny_hospdata$dead))   ②
prop.table(table(ny_hospdata$ami,ny_hospdata$dead),1) ③
prop.table(table(ny_hospdata$ami,ny_hospdata$dead),2) ④

round(prop.table(table(ny_hospdata$ami,ny_hospdata$dead)),2) ⑤
```

① Table with absolute frequencies.

② Table with relative frequencies. The sum of all the cells is equal to 1.

③ Table with relative frequencies, row total. The sum of each row is equal to 1.

④ Table with relative frequencies, column total. The sum of each column is equal to 1.

⑤ Round the values of the cells to 2 decimal places.

To remove the observations with **not avaible** records of cl_age, males, risky, severe and ptca variables, type:

```
amidata_limited<-amidata[!is.na(amidata$cl_age) & !is.na(amidata$males) &
!is.na(amidata$risky) & !is.na(amidata$severe) & !is.na(amidata$ptca),]
```

If we put | instead of **&** operator, it removes rows with **NA** entries in at least one of the selected variables.

Start elaborating some descriptive analysis of the NY Dataset and interpret the source code below row by row. Also, start planing some possible data analyses of interest based upon specific research questions.

```
sum(ny_hospdata$males,na.rm=T)  ①
Hmisc::describe(as.factor(ny_hospdata$males))  ②
m_ami<-aggregate(amidata$dead,by=list(amidata$ny_hosp_id),FUN="mean")  ③
m_ami_nrow<-aggregate(amidata$dead,by=list(amidata$ny_hosp_id),FUN="NROW")  ④
table(amidata$dead,amidata$ny_hosp_id)
names(m_ami)<-c("ny_hosp_id","cr")  ⑤
names(m_ami_nrow)<-c("ny_hosp_id","n")  ⑥
m_ami<-merge(m_ami,m_ami_nrow,by=c("ny_hosp_id"))  ⑦
m_ami$cr<-m_ami$cr*100  ⑧
hosp_county<-aggregate(ny_county ~ ny_hosp_id,ny_hospdata,FUN="unique")  ⑨
names(hosp_county)<-c("ny_hosp_id","ny_county")

z_ami<-merge(m_ami,hosp_county,by=c("ny_hosp_id"),all.x=TRUE)  ⑩
z_ami$county_up<-ifelse(z_ami$ny_county>20,2,1)  ⑪
m_ami_area<-mean(amidata$dead) ⑫

m_ami_area<-aggregate(amidata$dead,by=list(amidata$ny_area),FUN="mean")  ⑬
names(m_ami_area)<-c("ny_area","mortality")
m_ami_area$mortality<-round(m_ami_area$mortality*1000,2)
```

① Sum the entries of the variable **males** , ignoring **NAs**.

② Describe function from Hmisc package: turn a table with absolute and relative frequencies and including also the NA's and distinct entries.

③ Split the data into subsets and return the result in a convenient form. FUN parameter computes summary statistics which can be applied to all data subsets. Death rates (mean of dead variable) for each distinct hopital_id. Group by hospitals_id using as function the mean of death variable.

④ Number admissions for each hospital_id.

⑤ Rename the variables of m_ami.

⑥ Rename the variables of m_ami_row.

⑦ Merge the dataframes by the id of the hospitals.

⑧ Death rates moltipied by 100.

⑨ Turn unique hospital id and the county number.

⑩ Merge the data by ny_hosp_id. It also includes rows with no matching assigning **NA**.

⑪ Create a new variable called county_up with entries: if county higher than 20 assign 2, otherwise 1.

⑫ Mean of the variable **dead**.

⑬ Aggregate the deaths by ny areas. Split data in areas and turn death rates of each area.

3.2.2. Bivariate analysis

The bivariate data analysis includes testing the relations existing between two variables, using appropriate tests and statistics like **Chi-square** test and the **odds ratios** for categorical variables and independent sample **t-test** for continuous variables.

For instance, let's check the assosiation between gender and death in the **amidata** dataset, by typing into the console:

```
chisq.test(amidata$males,amidata$dead, correct=FALSE) ①

table(amidata$males,amidata$dead)  ②
CrossTable(amidata$males,amidata$dead)  ③
CrossTable(amidata$males,amidata$dead,format="SPSS")  ④
CrossTable(amidata$males,amidata$dead,chisq=TRUE)  ⑤

oddsratio(amidata$males,amidata$dead)  ⑥
oddsratio(amidata$males,amidata$dead, method = "wald")  ⑦
riskratio(amidata$males,amidata$dead)  ⑧
```

① **Chi square test** of independence, to check the releation between gender and death.

② Table with absolute frequencies.

③ Table with absolute and relative frequencies using the function **CrossTable** from **gmodels** package.

④ Similar to spss tables.

⑤ **Chi square test** of independence.

⑥ **Odds Ratio**, by default: median-unbiased estimate & mid-p exact CI. The function is included in the **epitools** package.

⑦ **Odds Ratio**: Unconditional MLE & normal approximation (Wald) CI.

⑧ **Risk Ratio** estimated by Unconditional MLE & normal approximation (Wald) CI.

The **Odds Ratio** can also be collected by fitting a logistic regression model:

```
logit_model<-glm(dead~males,family = binomial("logit"),data=amidata) ①

summary(logit_model) ②

round(cbind(Beta=coef(logit_model),confint(logit_model), ③
            P=coef(summary(logit_model))[,4],
            exp(cbind(OR=coef(logit_model),
            confint(logit_model)))),4)
```

① Fit a logistic regression model with gender as independet variable.

② Results of estimated model from basic command of R.

③ Show selected results: Coefficients, C.I. of the coefficients, Exponential(Coefficiets)=Odd Ratios, C.I. of the OR and P-value of wald test

To evaluate a statistical test that determines whether there is a statistically significant difference between the mean cost (mean of the total estimated cost for the discharges) of the survivors and not survivors, type:

```
t.test(amidata$cost[amidata$dead==1], amidata$cost[amidata$dead==0])
```

Study the relationship direction, strength, and linearity using scatterplots:

```
plot (amidata$los, amidata$cost, xlab="Length of Stay", ylab="Cost", main = "LOS -
COST") ①

abline(lm(cost~los, data=amidata),col='red') ②
```

① Scatter plot.

② Add a red regression line.

Interpret and distiunguish, from the picture below, the linear relationship:

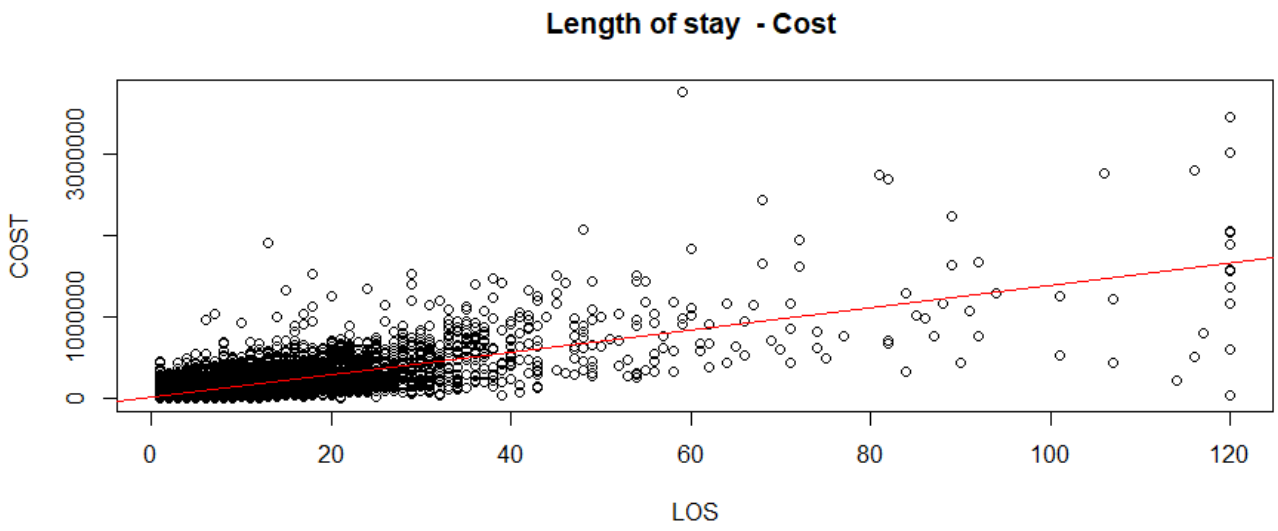


Figure 10. Scatter plot.

The linear relationship can be quantified via the **Pearson's correlation coefficient** and the **simple linear regression** model:

```
cor(amidata$cost, amidata$los) ①

linear_model<- glm(cost~ los,family = gaussian(link = identity),data=amidata) ②

summary(linear_model) ③
```

① Pearson correlation coefficient.

② Simple linear regression model.

③ Results af the simple linear regression model.

4. Standardization

By the end of present session, you will be able to compute direct and indirect standardized death rates using R.

Before starting, you need to install and load all the packages below:

```
source("methods_lab_functions.R")

installLibraries(c("ggplot2","gmodels"))

library(gmodels)
require(ggplot2)
```

Now, load the data:

```
load(file="amidata_2017.Rda")
```

4.1. Direct and indirect standardization

The computation of a standardized mortality ratio requires operations with matrices and the structure of the tables are very important. Let's try to create two matrices of the same dimension and cumpute the following operation:

```
x <- matrix(c(2,5,6,7,98,3,5,9), nrow = 2) ①
y<-matrix(c(4,5,7,9,9,4,6,8), nrow = 2)    ②


x/y ③

④
    [,1]      [,2]     [,3]      [,4]
[1,]  0.5 0.8571429 10.88889 0.8333333
[2,]  1.0 0.7777778  0.75000 1.1250000
```

① Create a 2x4 matrix called x.

② Create a 2x4 matrix called y.

③ Divide elements of a matrix with the corresponding elements of other matrix.

④ The output.

Try to compute the same opration with matrices of different dimensions:

```
x <- matrix(c(2,5,6,7), nrow = 2)  ①
y<-matrix(c(4,5,7,9,9,4), nrow = 3)  ②


rownames(x)<-c("B","C")    ③
rownames(y)<-c("A","B","C")
colnames(x)<-c("0","1")    ④
colnames(y)<-c("0","1")

x/y   ⑤

Error in x/y : non-conformable arrays ⑥
```

① Create a 2x2 matrix called x.

② Create a 3x2 matrix called y.

③ Rename the rows of matrix x.

④ Rename the columns of matrix x.

⑤ Divide elements of a matrix with the corresponding elements of other matrix.

⑥ The output.

As you can see, the operation can not be computed. This can happen when the structure of the population we are considering to compute the age-gender standardized death rates is different from the reference population. For example, no deaths for a specific age class.

To avoid the errore above, we will apply the following technique to adapt the structure of a matrix to the other one:

```
Standard_matrix<-matrix(c(rep(0,6)),3)    ①

rownames(Standard_matrix) <- rownames(y)
colnames(Standard_matrix) <- colnames(y)

# Adapt the structure of the matrix x

x_adj<-merge(as.data.frame.matrix(x),as.data.frame.matrix(Standard_matrix),  ②
             by="row.names",all.y=TRUE)
x_adj<-x_adj[2:3]    ③
x_adj[is.na(x_adj)] <- 0
x_adj<-x_adj[,c(sort(names(x_adj)))] ④

x_adj <- as.matrix(x_adj) ⑤
rownames(x_adj)<-rownames(Standard_matrix) ⑥
colnames(x_adj)<-colnames(Standard_matrix)  ⑦

x_adj ⑧

  0 1
A 0 0
B 2 6
C 5 7

x_adj/y ⑨
```

① Create a 3x2 standard matrix with 0 entries. Same dimension as the bigger matrix.

② Merge the smaller matrix with the standard matrix by the row names keeping all the rows from the standard matrix.

③ Select the 2nd and 3d column.

④ Order by names.

⑤ Transform the dataframe into a matrix.

⑥ Assign to the rows the names of the rows from the standard matrix.

⑦ Assign to the columns the names of the columns from the standard matrix.

⑧ The adapted matrix.

⑨ Divide elements of the structure adapted matrix with the corresponding elements of other matrix y.

This method, does not alternate the result if the matrices have the same shape:

```
①

x <- matrix(c(2,5,6,7,6,7), nrow = 3)
y<-matrix(c(4,5,7,9,4,6), nrow = 3)


rownames(x)<-c("A","B","C")
rownames(y)<-c("A","B","C")
colnames(x)<-c("0","1")
colnames(y)<-c("0","1")

x/y ②

Standard_matrix<-matrix(c(rep(0,6)),3)

rownames(Standard_matrix) <- rownames(y)
colnames(Standard_matrix) <- colnames(y)

x_adj<-
merge(as.data.frame.matrix(x),as.data.frame.matrix(Standard_matrix),by="row.names",all
.y=TRUE)
x_adj<-x_adj[2:3]
x_adj[is.na(x_adj)] <- 0
x_adj<-x_adj[,c(sort(names(x_adj)))]

x_adj <- as.matrix(x_adj)
rownames(x_adj)<-rownames(Standard_matrix)
colnames(x_adj)<-colnames(Standard_matrix)

x_adj/y ③
```

① Create two 3x2 matrices.

② Divide elements of a matrix with the corresponding elements of other matrix.

③ Divide elements of the adapted matrix with the corresponding elements of other matrix.

4.1.1. Direct standardization

In order to compute the direct standardization, you need to check the age and gender specific death rates and frequencies. If some age classes contain few observations you can re-encode the age classes. Specifically, you can create a new variable assigning to class 3 the age classes 1,2 and 3 of **cl_age variable** as follows:

```
amidata$recl_age<-ifelse(amidata$cl_age %in% c(1,2,3),3,amidata$cl_age)
```

Firstly, you need to create a common structure for the denominator and nominator of the direct standardized formula:

```
n_terms<-length(levels(as.factor(amidata$recl_age)))*
                length(levels(as.factor(amidata$males)))  ①
std_m<-matrix(c(rep(0,1,n_terms)),
                length(levels(as.factor(amidata$recl_age))))  ②
rownames(std_m) <- levels(as.factor(amidata$recl_age))  ③
colnames(std_m) <- levels(as.factor(amidata$males))  ④
```

① Moltiplication of the numbers of the levels of the 2 variables (3*2).

② Create a matrix with only zeroes of the same shape as the table recl_age-males.

③ Assign the correct names of the rows.

④ Assign the correct names of the columns.

The contingency table for age classes and gender can be obtained as follows:

```
pop_ij<-table(amidata[,c("recl_age")],amidata[,c("males")])
totpop<-sum(pop_ij) ①
```

① **N**=Number of total population (sum of all the cells of the table)

You can now calculate the direct standardized rate for one hospital e.g. hosp_id=19:

```
target_group<-subset(amidata,amidata$ny_hosp_id=="19")  ①
outcome_intarget<-subset(target_group,target_group$dead==1)  ②
label_target_group<-"Hospital 19" ③
num_data<-table(outcome_intarget[,c("recl_age")],outcome_intarget[,c("males")])  ④
den_data<-table(target_group[,c("recl_age")],target_group[,c("males")])  ⑤
```

① Select from amidata only patients of hosp_id 19.

② Select from patients of hosp_id 19 only the expired ones (dead).

③ Create the label name.

④ Age - gender table of dead patients of hosp_id 19. The cells contain contain dij.

⑤ Age - gender table of patients of hosp_id 19. The cells contain nij.

The tables **num_data** and **den_data** have different dimensions.

You need to uniform the numerator and the denominator structure, to be able to compute matrix operations. Substantially, you need matrices with the same dimensions:

```
①
res<-merge(as.data.frame.matrix(num_data),
           as.data.frame.matrix(std_m),
           by="row.names",all.y=TRUE)
res<-res[2:3]
res[is.na(res)] <- 0
res<-res[,c(sort(names(res)))]

res <- as.matrix(res)
rownames(res)<-rownames(std_m)
colnames(res)<-colnames(std_m)
res<-rowsum(res,row.names(res))

②
den_data<-
merge(as.data.frame.matrix(den_data),as.data.frame.matrix(std_m),by="row.names",all.y=
TRUE)
den_data<-den_data[,2:3]
den_data<-den_data[,c(sort(names(den_data)))]
rownames(den_data) <- levels(as.factor(amidata$recl_age))
colnames(den_data) <- levels(as.factor(amidata$males))
den_data[is.na(den_data)] <- 0
```

① Adapt num_data structure if needed.

② Adapt den_data structure if needed.

If the numerator has 0 rows than apply directly: **res←as.data.frame.matrix(std_m)**

At this point, you can compute the standardised rates. Calculate the rates by typing:

```
asr_ijt<-as.matrix(res/den_data)  ①
cr_t<-((sum(num_data)/sum(den_data) ))*100  ②
sr_t<-sum(unclass(asr_ijt)*unclass(pop_ij))/totpop*100 ③
sr_t.se<- sqrt((sr_t/100*(1-sr_t/100))/(sum(den_data)))  ④
sr_t.ll95  <- sr_t - 1.96 * sr_t.se  ⑤
sr_t.ul95  <- sr_t + 1.96 * sr_t.se  ⑥

output<-data.frame(label_target_group,round(cr_t,2),round(sr_t,2),
                   round(sum(den_data),2),round(sr_t.ll95,2),round(sr_t.ul95,2))  ⑦
names(output1)<-c("Unit","CR","SR","N","Lower95","Upper95")  ⑧
output  ⑨
```

① Age - gender table with dij/nij cells.

② Crude rate=sum(dij)/sum(nij) *100

③ Standardized rate=sum(dij/nij * Nij)/N *100

④ Standard error.

⑤ Lower 95% C.I.

⑥ Upper 95% C.I.

⑦ Create dataframe with the useful informations.

⑧ Rename the columns.

⑨ Show results.

4.1.2. Indirect standardization

We adopt the **AHRQ** formulas for indirect standardization. The computation will include **regression standardization** using vital status as the outcome of a logistic regression model. Firstly, fit a logistic regression model that predicts death adjusting by age and gender:

```
DLogit<-glm(dead~cl_age+males,family = binomial("logit"),data=amidata)

round(cbind(Beta=coef(DLogit),confint(DLogit),P=coef(summary(DLogit))[,4],
            exp(cbind(OR=coef(DLogit),confint(DLogit)))),4)  ①

summary(DLogit)  ②
```

① Show selected values: coefficients, C.I. of the coefficients,P-value from wald test, exp(coefficiets)=Odd Ratios and C.I. of the ORs.

② Show results of the model using a build in function of R.

The linear combination of the predictors are obtained by:

```
amidata$p<- predict(DLogit,newdata=amidata,type="response")
```

The total number of deaths from **ami patients** divided by the number of observations (average outcome in the entire sample:reference population rate) can be computed with:

```
Ybar<-sum(amidata$dead,na.rm=TRUE)/dim(amidata)[1]
```

You can obtain the average of the predicted probabilities for all dataset:

```
mean(amidata$p)  ①
mean(amidata$p,na.rm=TRUE)  ②
sum(amidata$p,na.rm=TRUE)/dim(amidata)[1]  ③
sum(amidata$p,na.rm=TRUE)/sum(!is.na(amidata$p))  ④
```

① With missing values (can non be produced).

② Without missing values.

③ Divided by the number of observations.

④ Divided by the number of observations WITHOUT missing values.

Now, calculate **Standardized Rates** for hosp_id = 1 by typing into console:

```
O1<-mean(amidata[amidata$ny_hosp_id=="1",c("dead")],na.rm=TRUE)  ①
E1<-mean(amidata[amidata$ny_hosp_id=="1",c("p")],na.rm=TRUE)  ②
N1<-dim(amidata[amidata$ny_hosp_id=="1",])[1]  ③
P1j<-amidata[amidata$ny_hosp_id=="1",c("p")]  ④

rar1<-Ybar*(O1/E1)  ⑤
rar1.se<-sqrt( (Ybar/E1)^2 *(1/N1)^2 * sum(P1j*(1-P1j),na.rm=TRUE))  ⑥
rar1.ll95  <- rar1 - 1.96 * rar1.se  ⑦
rar1.ul95  <- rar1 + 1.96 * rar1.se  ⑧
```

① Oi=(1/ni)Σ(Yij) - **OBSERVED RATE** at hosp_id 1.

② Ei = (1/ni)Σ(Pij) - **EXPECTED RATE** at hosp_id 1

③ Number of patients at hosp_id 1.

④ Predicted probability from logit of Y on X for patients at hosp_id 1.

⑤ Risk adjusted rate (indirect standardized rate).

⑥ Standard error of risk adjusted rate.

⑦ Lower 95% C.I.

⑧ Upper 95% C.I.

You can store the labels, crude death rate, standardized rate, the number of observations and C.I. by:

```
output<-data.frame("HOSPITAL 1",round(O1*100,2),round(rar1*100,2),
                    N1,round(rar1.ll95*100,2),round(rar1.ul95*100,2))
names(output)<-c("Unit","CR","SR","N","Lower95","Upper95")
```

4.2. Funnel plot

You can also explore the mortality rates and check for outliers using the **Funnel Plots**:

```
m_ami<-aggregate(amidata$dead,by=list(amidata$ny_hosp_id),FUN="mean")
m_ami_nrow<-aggregate(amidata$dead,by=list(amidata$ny_hosp_id),FUN=NROW)
names(m_ami)<-c("ny_hosp_id","cr")
names(m_ami_nrow)<-c("ny_hosp_id","n")
m_ami<-merge(m_ami,m_ami_nrow,by=c("ny_hosp_id"))
m_ami$cr<-m_ami$cr*100
hosp_county<-aggregate(ny_county ~ ny_hosp_id,amidata,FUN="unique")
names(hosp_county)<-c("ny_hosp_id","ny_county")

z_ami<-merge(m_ami,hosp_county,by=c("ny_hosp_id"),all.x=TRUE)
z_ami$county_up<-ifelse(z_ami$ny_county>20,2,1)


funnel_plot(title="AMI Mortality Rates", ①
            names=z_ami$ny_hosp_id,  ②
            names_outliers=1, ③
            plot_names=1, ④
            in_colour=1, ⑤
            colour_var=z_ami$county_up, ⑥
            colour_values=c("indianred2","blue"), ⑦
            rate=z_ami$cr, ⑧
            unit=100, ⑨
            population=z_ami$n, ⑩
            binary=1, ⑪
            p_mean="weighted", ⑫
            filename="",  ⑬
            graph="ami_mortality", ⑭
            pdf_height=3.5,  ⑮
            pdf_width=6,  ⑯
            ylab="Indicator per 100",  ⑰
            xlab="Total N",  ⑱
            selectlab="County",  ⑲
            selectlev=c("1","2"),  ⑳
            dot_size=1.5, ⑳
            dfout="dfout")  ⑳
```

① The title of graph.

② Variable for group names.

③ Differently plot outliers by names (0=No;1=Yes).

④ Plot names instead of dots.

⑤ Colour points within boundaries (0=No;1=Yes).

⑥ Variable for colouring groups (numeric).

⑦ List of colours for all levels of colour_var.

⑧ Value of standardized rates.

⑨ Denominator of rates.

⑩ Total number of subjects for each rate.

⑪ Binary variable (0=No;1=Yes).

⑫ Unweighted, Weighted or specified value.

⑬ Output graph filename.

⑭ Name of graph object.

⑮ Height pdf output.

⑯ Width pdf output.

⑰ y axis label.

⑱ x axis label.

⑲ Label for group legend.

⑳ Values of group legend.Scaling factor for dots in the funnel plot. Funnel dataset saved in global output object.

You can print the plot by typing:

```
print(ami_mortality)
```

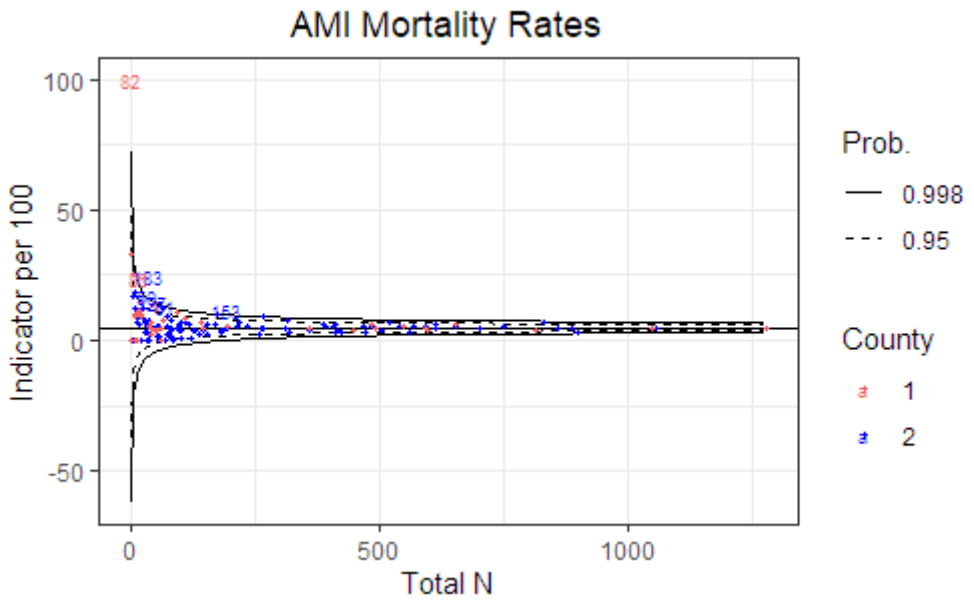


Figure 11. Acute Myocardial Infarction Mortality Rates in the State of New York - Year 2017.

EXERCISE 4.1
Using the code to obtain direct standardized rates on one hospital, generalise the approach to produce a table with results for all hospitals simultaneously.

You can use a function for a loop, or vectorize the approach (preferred) using matrix operations.

EXERCISE 4.2
a. Create a function to compute Risk Adjusted Rates(Indirect standardization) for AMI Mortality by Hospital.
b. Produce funnel plots from NY dataset using the table obtained above. Replicate the approach for a different outcome.

5. Multivariable analysis, GEE and logistic regression model

5.1. Generalized Estimating Equation (GEE) model

The GEE model example presented here is based on the scientific paper: **Carinci F, Massi Benedetti M, Klazinga N, Uccioli L. Lower extremity amputation rates in people with diabetes as an indicator of health systems performance**, available at: Lower extremity amputation rates .

Let's import the data:

```
retros_final<-read.table("amputations.csv",sep=";",header=TRUE) ①

retros_final$value<-as.numeric(retros_final$value)  ②
retros_final$Year<-as.character(retros_final$Year)  ③
retros_final<-retros_final[retros_final$Country!="USA"|
              (retros_final$Country==c("USA") & retros_final$Year>2009),] ④
```

① Importy data.

② Transform amputation rate into a numeric variable.

③ Transform **Year** into a character variable.

④ Exclude US data.

We need to create the reference value, to compare rates of different countries:

```
refo<-data.frame(matrix(ncol=2,nrow=1))  ①
names(refo)<-c("value","Year")
refo$value<-14
refo$Year<-"2000"
```

① Create data frame 1x2.

We also need to transform the **Country** variable into a factor to create a line chart plot:

```
retros_final$Country<-as.factor(retros_final$Country)
```

We can now draw a line chart with time trends. It will include the amputation rates of all countries from 2000 to 2011.

```
gar<-ggplot(data=retros_final,aes(x=Year,y=value,group=Country)) +
  geom_line(aes(linetype=Country)) +
  geom_point()+
  scale_linetype_manual(values=retros_final$Country) +
  xlab("Year") +
  ylab("Rate") +
  labs(title="Indicator Rates")+
  geom_text(data=retros_final[retros_final$Year=="2000",],aes(label=Country),
            hjust=1.5,size=3.0)+
  geom_text(data=retros_final[retros_final$Year=="2001" & retros_final$Country
            %in% c("ITA"),],aes(label=Country),hjust=1.5,size=3.0)+
  geom_text(data=retros_final[retros_final$Year=="2002" & retros_final$Country
            %in% c("LUX"),],aes(label=Country),hjust=1.5,size=3.0)+
  geom_text(data=retros_final[retros_final$Year=="2004" & retros_final$Country
            %in% c("HUN"),],aes(label=Country),hjust=1.5,size=3.0)+
  geom_text(data=retros_final[retros_final$Year=="2005" & retros_final$Country
            %in% c("NZL","NLD","POL"),],aes(label=Country),hjust=1.5,size=3.0)+
  geom_text(data=retros_final[retros_final$Year=="2006" & retros_final$Country
            %in% c("CHE","GBR","USA"),],aes(label=Country),hjust=1.5,size=3.0)+
  geom_text(data=retros_final[retros_final$Year=="2007" & retros_final$Country
            %in% c("CAN","DNK","FIN","FRA","DEU","KOR","NOR","SWE","USA"),],
            aes(label=Country),hjust=1.5,size=3.0)+
  geom_text(data=retros_final[retros_final$Year=="2008" & retros_final$Country
            %in% c("AUS"),],aes(label=Country),hjust=1.5,size=3.0)+
  geom_text(data=retros_final[retros_final$Year=="2009" & retros_final$Country
            %in% c("SVN"),],aes(label=Country),hjust=1.5,size=3.0)+
  geom_text(data=retros_final[retros_final$Year=="2010" & retros_final$Country
            %in% c("USA"),],aes(label=Country),hjust=1.5,size=3.0)+
  geom_text(data=retros_final[retros_final$Year=="2011" & retros_final$Country
            %in% c("PRT"),],aes(label=Country),hjust=1.5,size=3.0)+
  geom_text(data=refo,aes(1,value,label="OECD",hjust=0.5),inherit.aes=FALSE,
            size=3)+
  geom_line(data=try,aes(x=as.numeric(factor(Year)),y=value),colour="red")+
  theme(legend.position = "none")

print(gar)
```

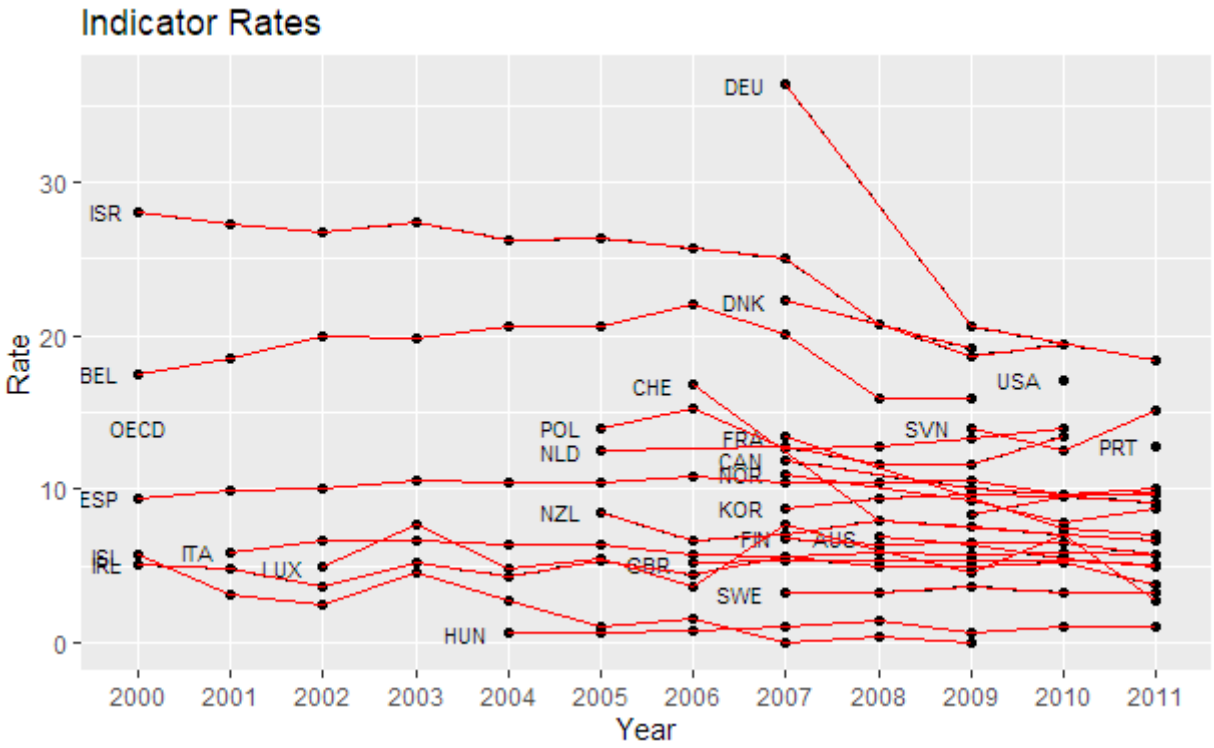


Figure 12. The amputation rates of different countries.

Drawing a turnip chart with a target object **gr_retro** can be also useful. It includes the box plots for every year and the trend of the mean amputation rates.

```
mpv_beeswarm(input=retros_final,
             gr="gr_retro",
             output="retros_by_year",
             vars=c("value"),labels=c(),units=c("100000"),ylim=c(),box=1,
             xlab="Year",meanscale=0,logscale=0,
             groupvar="Year",normalize=0,spacing=0.7,cex=1.5,
             main="Lower Extremity Amputation Rates in Diabetes, OECD 2000-2011")
print(gr_retro)
```

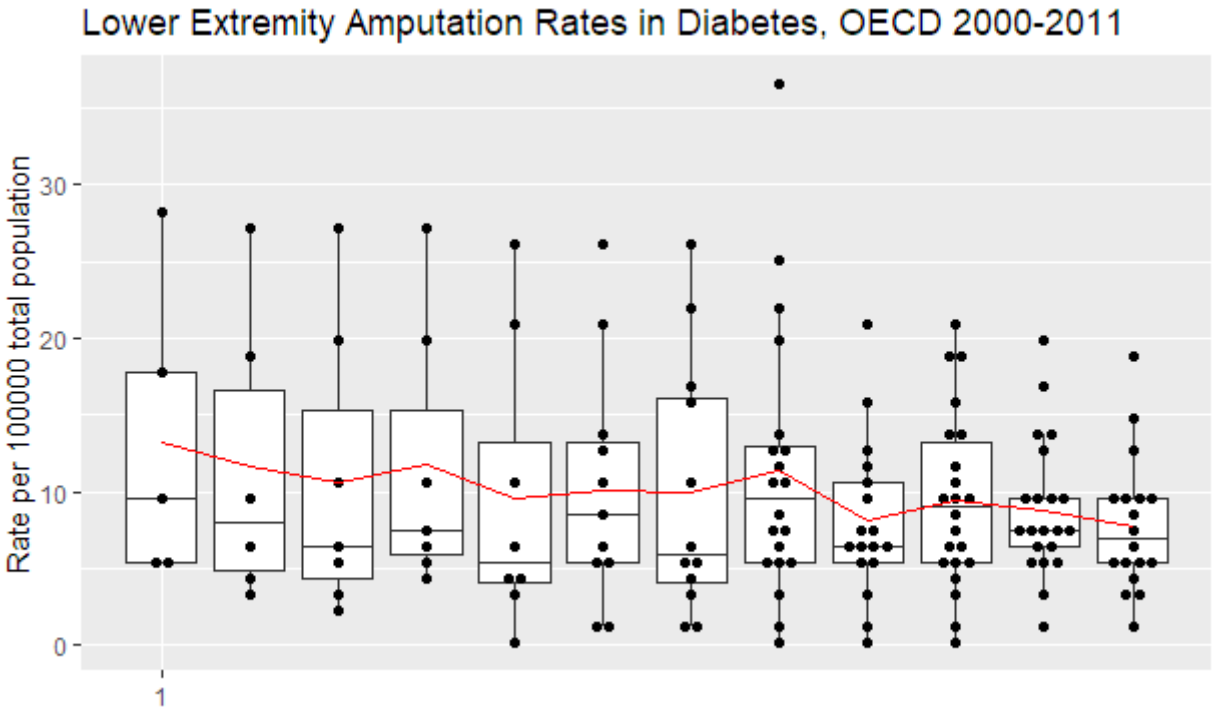


Figure 13. The amputation rates in diabetes, 2000-2011.

Before running the GEE model we need to prepare the data as follows:

```
retros_final$Year<-as.numeric(retros_final$Year)  ①
retros_final<-retros_final[order(retros_final$Country),]  ②
retros_final$Year<-retros_final$Year-2000  ③
retros_final$Country<-as.numeric(factor(retros_final$Country))  ④
```

① Transform year into a numeric variable.

② Order data by alphabetic order of the countries.

③ Subtract baseline year to compute interpretable intercept.

④ Transform country in numeric.

First run a GLM model to derive initial estimates from a limited pool of covariates:

```
modglm<-glm(value~primary_tax+Year,data=retros_final,family=gaussian)
```

Now run the GEE model using the same covariates:

```
modgee<-geeglm(value~primary_tax+Year,data=retros_final,
               id=Country,family=gaussian,corstr="exchangeable")
```

The results can be compared by typing:

```
summary(modglm)
summary(modgee)
```

The contents of GEE model can be checked by typing:

```
names(modgee)
```

The function below is very useful to compute confidence intervals:

```
confint.geeglm <- function(object, parm, level = 0.95, ...) {
  cc <- coef(summary(object))
  mult <- qnorm((1+level)/2)
  citab <- with(as.data.frame(cc),
                cbind(Lower=Estimate-mult*Std.err,
                      Upper=Estimate+mult*Std.err))
  rownames(citab) <- rownames(cc)
  citab[parm,]
}

cbind(coef(summary(modglm)),confint(modglm))
cbind(coef(summary(modgee)),confint.geeglm(modgee))
```

More covariates can be added to the GLM model, for example **registry** and **coding** variables:

```
summary(glm(value~primary_tax+registry+coding+Year,data=retros_final,
            family=gaussian)) ①
summary(geeglm(value~primary_tax+registry+coding+Year,data=retros_final,id=Country,
               family=gaussian,corstr="exchangeable"))  ②
```

① GLM model.

② GEE model.

A subgroup analysis can be performed as following:

```
summary(geeglm(value~Year,data=retros_final[retros_final$primary_tax==0,],
               id=Country,family=gaussian,corstr="exchangeable")) ①
summary(geeglm(value~Year,data=retros_final[retros_final$primary_tax==1,],
               id=Country,family=gaussian,corstr="exchangeable"))  ②
```

① Primary tax=0.

② Primary tax=1.

EXERCISE 5.1:
a. Formally check PRIMARY TAXATION as a potential effect modifier of the relation between ANNUAL INCREASE and AMPUTATION RATES

b. Find a way to show all regression coefficients of AMPUTATION RATES from TWO LEVELS OF PRIMARY TAXATION in the SAME MODEL

5.2. Logistic regression model

Firstly, it is highly advisable to clear the memory, load the packages, functions and data:

```
list.files()
rm(list=ls())
options(scipen=999)

source("methods_lab_functions.R")

library(Hmisc)
library(gmodels)
require(ggplot2)
library(gridExtra)
library(plyr)
library(grid)
library(geepack)
library(caret)
library(geepack)
library(forestplot)
library(Epi)
library(MatchIt)
library(epitools)
library(ROCR)
library(ROCit)
library(pROC)
library(cutpointr)
library(bootLR)


load(file="amidata_2017.Rda")
```

Recreate the dummy variables for **severity of illness** and **risk**:

```
amidata$d_severe<-0
amidata$d_severe<-ifelse(amidata$severe>=2,1,amidata$d_severe)
amidata$d_severe<-ifelse(is.na(amidata$severe)==TRUE,NA,amidata$d_severe)

amidata$d_risky<-0
amidata$d_risky<-ifelse(amidata$d_risky>=2,1,amidata$d_risky)
amidata$d_risky<-ifelse(is.na(amidata$risky)==TRUE,NA,amidata$d_risky)
```

You can plot a log-odds graph to explore the risk patterns associated with specific levels of risk factors. Below, we use the **emplogit** function present in the source **methods_lab_functions.R** functions. You can create the plot by typing:

```
emplogit(amidata$cost,amidata$dead,xlab="Cost",ylab="Logit(Dead)")
```

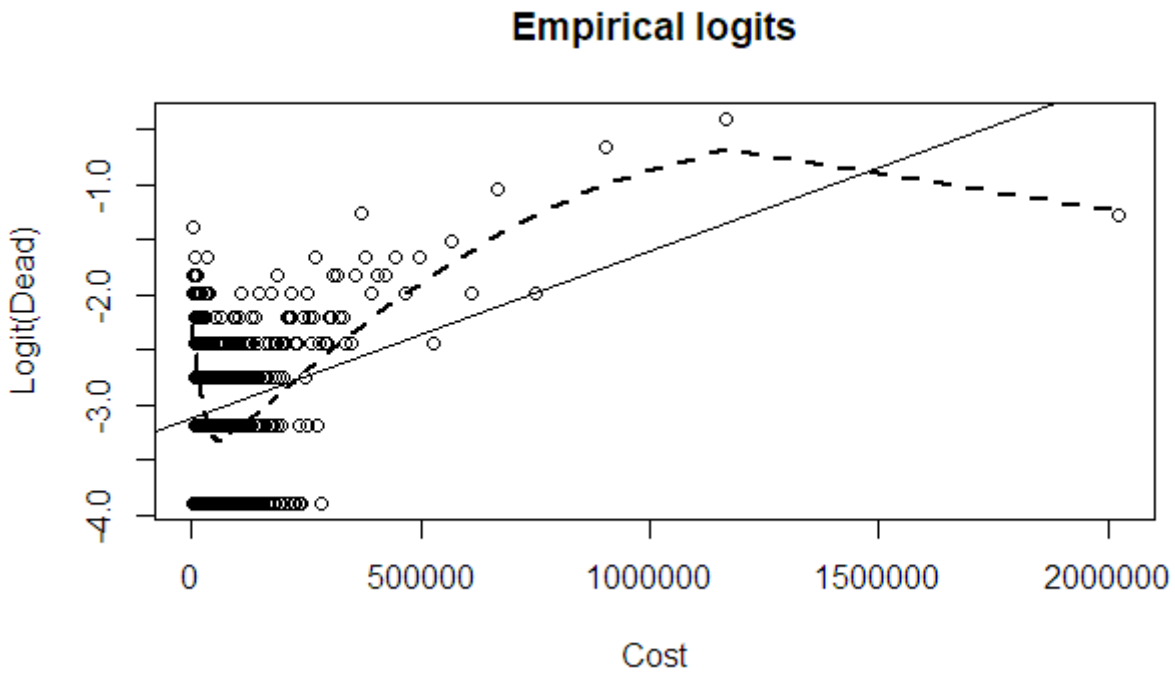


Figure 14. Cost - logit(dead) plot.

We will now build a multivariable model called **Logit_reduced** that predicts mortality, adjusting by **age**, **sex** and **risk**:

```
Logit_reduced<-glm(dead~cl_age+males+d_risky,
                   family = binomial("logit"),data=amidata)

round(cbind(Beta=coef(Logit_reduced),confint(Logit_reduced),
      P=coef(summary(Logit_reduced))[,4],exp(cbind(OR=coef(Logit_reduced),
      confint(Logit_reduced)))),4)  ①

round(cbind(BIC=BIC(Logit_reduced),AIC= AIC(Logit_reduced),
      "-2*Log-Likelihood"= -2 * as.numeric(logLik(Logit_reduced))))  ②
```

① Results of Coefficients + 95% C.I. and exp(coefficients)=ORs + 95% C.I..

② Cumpute BIC, AIC and -2 log-likelihood statistics.

Create a another model called **Logit_full** adding other variables to the base model and check the results:

```
Logit_full<-glm(dead~cl_age+males+surgical+d_risky+d_severe+ptca,
                family = binomial("logit"),data=amidata)

round(cbind(Beta=coef(Logit_full),confint(Logit_full),
      P=coef(summary(Logit_full))[,4],exp(cbind(OR=coef(Logit_full),
      confint(Logit_full)))),4)

round(cbind(BIC=BIC(Logit_full),AIC= AIC(Logit_full),
      "-2*Log-Likelihood"= -2 * as.numeric(logLik(Logit_full))))  ①
```

To access the model, we can carry out a formal test using the likelihood ratio as follows:

```
ndiffpars<-length(Logit_full$coefficients)-length(Logit_reduced$coefficients)  ①
LL1<-(-2*as.numeric(logLik(Logit_full)))  ②
LL2<-(-2*as.numeric(logLik(Logit_reduced)))  ③
LR<-pchisq(LL2-LL1,ndiffpars,lower.tail=FALSE)  ④
message(paste("-2 LogLik FULL:",LL1))  ⑤
message(paste("-2 LogLik REDUCED:",LL2)) ⑥
message(paste("Likelihood Ratio:",LL2-LL1,";
              P(chi-square)=",LR,"; df=",ndiffpars))  ⑦
```

① Difference of number of variables- degrees of freedom.

② -2 * log-likelihood of full model.

③ -2 * log-likelihood of reduced model.

④ Chi Square test of: -2 * (log-likelihood of reduced model -log-likelihood of full model).

⑤ Show -2 * log-likelihood of full model.

⑥ Show -2 * log-likelihood of reduced model.

⑦ Show Likelihood ratio, p-value and degrees of freedom.

The prediction accuracy of the models above can be computed by calculating the concordant percentage of pairs from expected and observed events. For instance, the prediction accuracy considering a threshold of 0.5 for the probability of outcomes can be obtained as follows:

```
amidata$p<- predict(Logit_reduced,newdata=amidata,type="response")  ①
predicted.value <- ifelse(amidata$p> 0.5,1,0)  ②
discordant <- mean(predicted.value != amidata$dead,na.rm=TRUE)  ③
message(paste('Accuracy of the REDUCED model:',1-discordant))  ④
```

① Create a variable with prediction probabilities from the fitted model.

② Create vector with 1 entry if prediction probability higher than 0.5 and 0 otherwise.

③ Nr. of discordances between observed and expected values divided by nr of observations.

④ Show prediction accuracy (1-discordance).

EXERCISE 5.2: Apply different models, compute the accuracy and assess the goodness of fit through the likelihood ratio test.

6. Model assessment and validation

Firstly, clear the memory, load the packages, functions and data:

```
list.files()
rm(list=ls())
options(scipen=999)

source("methods_lab_functions.R")

library(Hmisc)
library(gmodels)
require(ggplot2)
library(gridExtra)
library(plyr)
library(grid)
library(geepack)
library(caret)
library(geepack)
library(forestplot)
library(Epi)
library(MatchIt)
library(epitools)
library(ROCR)
library(ROCit)
library(pROC)
library(cutpointr)
library(bootLR)


load(file="amidata_2017.Rda")
```

Recreate the dummy variables for **severity of illness** and **risk**:

```
amidata$d_severe<-0
amidata$d_severe<-ifelse(amidata$severe>=2,1,amidata$d_severe)
amidata$d_severe<-ifelse(is.na(amidata$severe)==TRUE,NA,amidata$d_severe)


amidata$d_risky<-0
amidata$d_risky<-ifelse(amidata$d_risky>=2,1,amidata$d_risky)
amidata$d_risky<-ifelse(is.na(amidata$risky)==TRUE,NA,amidata$d_risky)
```

Fit two multivariable models called **Logit_reduced** and **Logit_full** that predict mortality:

```
Logit_reduced<-glm(dead~cl_age+males+d_risky,
                   family = binomial("logit"),data=amidata)

Logit_full<-glm(dead~cl_age+males+surgical+d_risky+d_severe+ptca,
                family = binomial("logit"),data=amidata)
```

A more reliable way, compared to the accuracy, to assess the discrimination ability of the model can be obtained through **Receiver Operating Characteristic (ROC) Curve** and * Area Under the Curve (AUC)*. We will compute the predicted scores and graph thresholds in the ROC curve using the ROCR package:

```
amidata$p<- predict(Logit_reduced,newdata=amidata,type="response")

pr <- prediction(amidata[!is.na(amidata$p),c("p")],
                 amidata[!is.na(amidata$p),c("dead")]) ①
prf <- performance(pr, measure = "tpr", x.measure = "fpr") ②
plot(prf) ③
abline(a=0, b= 1)  ④
title("ROC Curve Reduced Model")  ⑤
```

① Transforms the input data into "prediction object" used from ROCR package.

② An bject of ROCR package that contains True positive rates and False positive rates

③ Plot the curve.

④ Add a line with intercept 0 and slope 1.

⑤ Title of the plot.

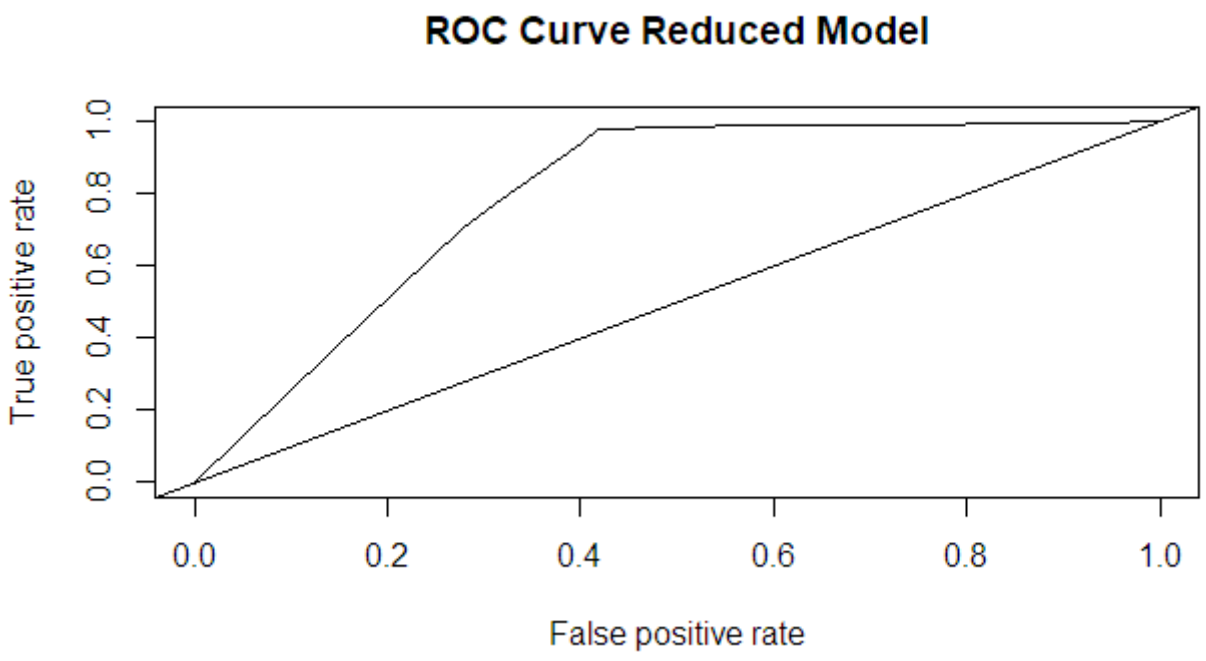


Figure 15. Area Under the Curve for the reduced model.

The area under the curve(AUC) can be calculated as follows:

```
auc <- performance(pr, measure = "auc")
message(paste(("AUC REDUCED:"),auc@y.values[[1]]))
```

We can also compare the models by the AUC and ROC curves using the following codes:

```
①
pr2 <- prediction(amidata[!is.na(amidata$p2),c("p2")],
                  amidata[!is.na(amidata$p),c("dead")])
prf2 <- performance(pr2, measure = "tpr", x.measure = "fpr")
plot(prf2)
abline(a=0, b= 1)
title("ROC Curve Full Model")

②
auc2<- performance(pr2, measure = "auc")
message(paste(("AUC FULL:"),auc2@y.values[[1]]))

③
plot(prf, col="red", lwd = 2)
plot(prf2, col="green", add=T)
legend(0.4,0.4,c('Reduced model','Full Model'),lty=c(1,1),lwd=c(1,1),
       col=c('red', 'green'), cex = 1.3,bty = "n")
abline(a=0, b= 1)
title("Roc curves")
```

① ROC cuve and AUC for the full model.

② Compute area under the curve (AUC) for the full model.

③ Plot the ROC curves of both models.

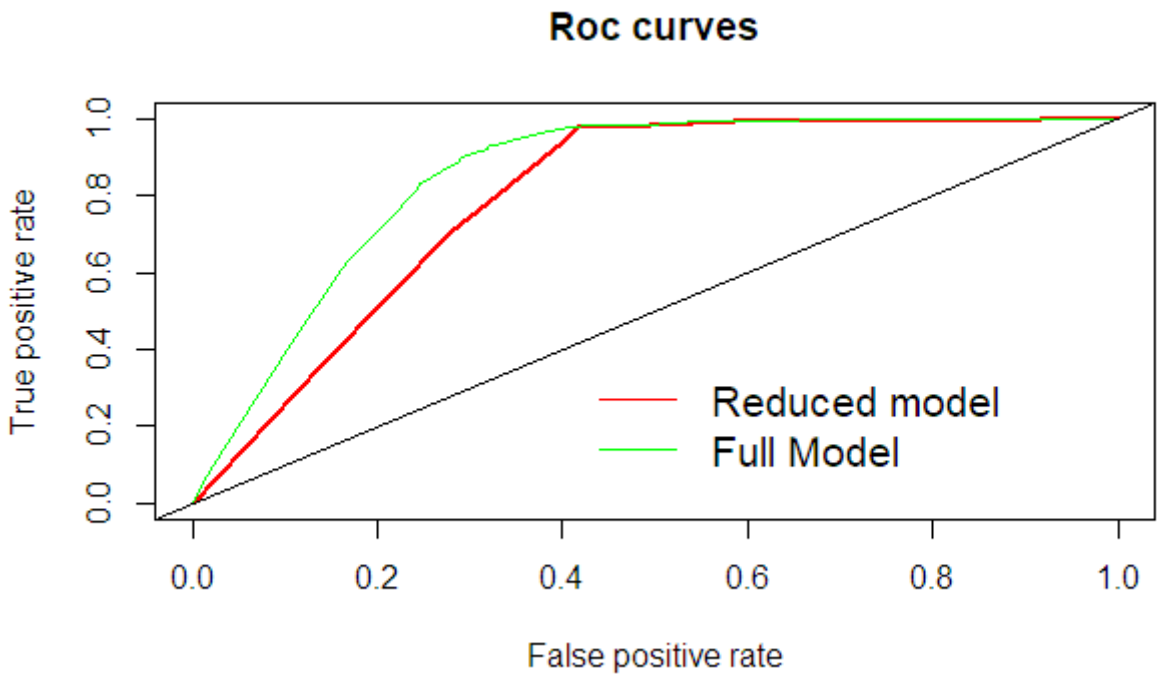


Figure 16. ROC curve comparision.

In some cases the 0.5 cutpoint may not be the best choice or can even not be raccomended. A different cutpoint can achieve a better discrimination ability of a binary model. We can identify a better cutpoint using the **cutpointr** package, by maximizing a specific metric. For example let's maximize the Youden index (**J** = **sensitivity** + **specificity - 1**) and compute the optimal cutpoint for the reduced model:

```
optimal_cut_1<-cutpointr(data=amidata_pr,x=p,class=dead,na.rm=TRUE,
               method=maximize_metric,metric=youden,use_midpoint=TRUE)
optimal_cut_1$optimal_cutpoint
summary(optimal_cut_1)
```

We can now create a dummy variable by using a prediction probability higher or equal to the optimal cut point. If the condition is met, the dummy variable will be equal to 1, otherwise 0.

```
amidata_pr$d_score_1<-ifelse(amidata_pr$p>=optimal_cut_1$optimal_cutpoint,1,0)
amidata_pr$d_score_1<-ifelse(is.na(amidata_pr$p),NA,amidata_pr$d_score_1)
```

Another package that can be used for model assessment is the **pROC** package, which contains functions for the optimal cutpoint, AUC and C.I. of AUC. For example, run the code below for the reduced model:

```
pROC_1<- pROC::plot.roc(amidata_pr$dead,amidata_pr$p,
                        main="Confidence interval of a threshold", percent=T,
                        ci=TRUE, of="thresholds",  ①
                        thresholds="best",  ②
                        print.thres=optimal_cut_1$optimal_cutpoint,
                        print.thres.col="red")
ci_auc_1<-ciAUC(rocit(score=amidata_pr$p,amidata_pr$dead))  ③
```

① Compute AUC (of threshold).

② Select the (best) threshold.

③ AUC and Confidence Intervals.

Here is what you get:

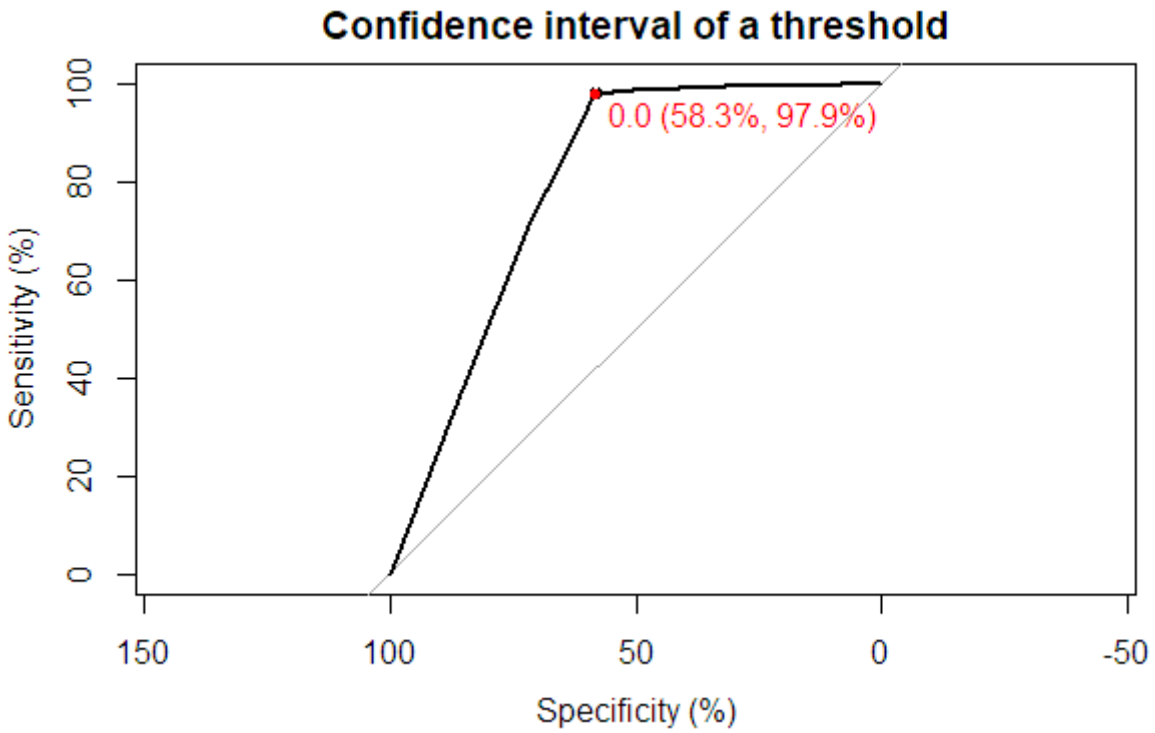


Figure 17. AUC with optimal thresholds and C.I.

We can also plot the AUC with optimal thresholds and C.I. of AUC for both models in the same figure.

```
      ①
optimal_cut_2<-cutpointr(data=amidata,x=p2,class=dead,na.rm=TRUE,
               method=maximize_metric,metric=youden,use_midpoint=TRUE)
summary(optimal_cut_2)

amidata$d_score_2<-ifelse(amidata$p>=optimal_cut_2$optimal_cutpoint,1,0)
amidata$d_score_2<-ifelse(is.na(amidata$p2),NA,amidata$d_score_2)

pROC_2<- pROC::plot.roc(amidata$dead,amidata$p2,
                        main="Confidence interval of a threshold", percent=T,
                        ci=TRUE, of="thresholds",
                        thresholds="best",
                        print.thres=optimal_cut_2$optimal_cutpoint,
                        print.thres.col="red")

ci_auc_2<-ciAUC(rocit(score=amidata$p2,amidata$dead))

      ②
plot(pROC_1,col="red",lty=2)
text(round(optimal_cut_1$specificity*100,1)-10,round(optimal_cut_1$sensitivity*100,1)-
2,
           paste(round(optimal_cut_1$optimal_cutpoint,4),"
(",round(optimal_cut_1$specificity*100,1),
           ",",round(optimal_cut_1$sensitivity*100,1),")",sep=""),cex=0.65,col="red")
      ③
plot(pROC_2,type="shape",col="blue",lty=3,add=TRUE)
text(round(optimal_cut_2$specificity*100,1)-8,round(optimal_cut_2$sensitivity*100,1)-
2,
           paste(round(optimal_cut_2$optimal_cutpoint,4),"
(",round(optimal_cut_2$specificity*100,1),
           ",",round(optimal_cut_2$sensitivity*100,1),")",sep=""),cex=0.65,col="blue")

      ④
legend("bottomright", legend = c(
       paste("Score pROC_1 AUC: ",format(round(ci_auc_1$AUC*100,1),nsmall=1),
       " (",round(ci_auc_1$lower*100,1),"-",round(ci_auc_1$upper*100,1),")",sep=""),
       paste("Score pROC_2 AUC: ",format(round(ci_auc_2$AUC*100,1),nsmall=1),
       " (",round(ci_auc_2$lower*100,1),"-",round(ci_auc_2$upper*100,1),")",sep="")
       ), inset=c(.02,.04),col = c("red","blue"),lty=c(2,3),lwd=c(2,2))
```

① Compute optimal cut-point, a dummy varible using of the optimal cut-point and AUC for the full model.

② Plot first model.

③ Add the second model.

④ Add the legend.

The figure that will be obtained is the following:

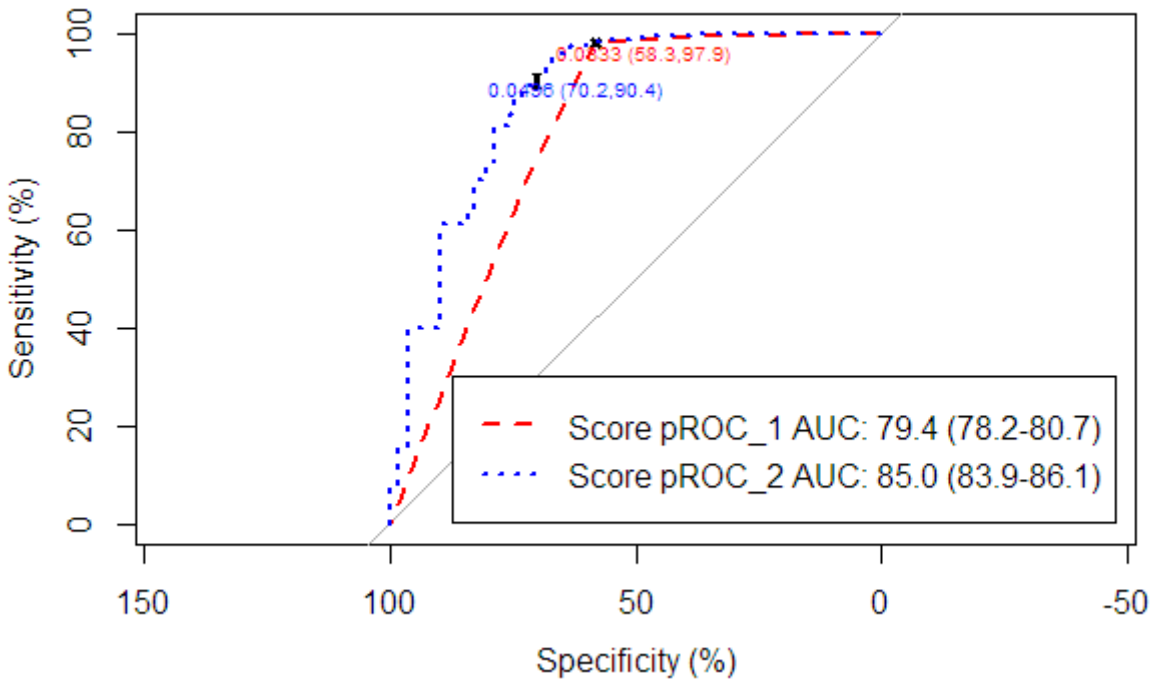


Figure 18. AUC with optimal thresholds and C.I. for both models

6.1. Model validation

The prediction abitity of a model evaluated on the same dataset can lead to an optimistic high accuracy. The main goal of a model is to predict the data from a training dataset to a new one. In this session we will focus on the discrimination ability of the model using different cutpoints outside the traning dataset. We will split the data in two parts, by assigning randomly half of the observations to the traning dataset and the other half to the test dataset:

```
split=0.50  ①
training_set<-createDataPartition(y=amidata$dead,p=split,list=FALSE)  ②

amidata_train <-amidata[training_set,]  ③
amidata_test  <-amidata[-training_set,]  ④
```

① Assign to split 0.5.

② Results are random rows id's for the training set containing half of deaths.

③ Select rows from amidata using the random id's generated.

④ Select rows from amiodata, out of training set.

Now, let's fit two models using the two generated datasets and check the results:

```
Logit_full_train<-glm(dead~cl_age+males+surgical,family = binomial("logit"),
                      data=amidata_train)  ①
Logit_full_test<-glm(dead~cl_age+males+surgical,family = binomial("logit"),
                     data=amidata_test)  ②

round(cbind(Beta=coef(Logit_full_train),confint(Logit_full_train),
            P=coef(summary(Logit_full_train))[,4],exp(cbind(OR=coef(Logit_full_train),
            confint(Logit_full_train)))),4)
round(cbind(Beta=coef(Logit_full_test),confint(Logit_full_test),
            P=coef(summary(Logit_full_test))[,4],exp(cbind(OR=coef(Logit_full_test),
            confint(Logit_full_test)))),4)
```

① Fit a logistic model considering the training dataset.

② Fit a logistic model considering the test dataset.

As you can see, the results of the two models are similar. You can build the confusion matrix for the first model and check the results:

```
amidata_train$predictions<-
predict(Logit_full_train,newdata=amidata_train,type="response")  ①
optimal_cut<-cutpointr(data=amidata_train,x=predictions,class=dead,na.rm=TRUE,
                       method=maximize_metric,metric=youden,use_midpoint=TRUE)  ②
amidata_train$predictions.value<-
ifelse(amidata_train$predictions>optimal_cut$optimal_cutpoint,1,0)  ③
confusionMatrix(as.factor(amidata_train$predictions.value),
                as.factor(amidata_train$dead), positive = "1") ④
```

① Calculate the predicition probabilities for every observation.

② Calculate the optimal cut-point maximizing the Youden index.

③ Create a dummy variable called **predictions.value** where: if prediction probabilities are higher than the optimal cut-point contains 1, otherwise 0.

④ Compute the confusion matrix with the new dummy variable.

It will be noted that the model reaches an accuracy of only 56% but a specificity of 73% with the optimal cutpoint. You can plot the Youden index for different cutpoins, by typing:

```
plot_metric(optimal_cut)
```

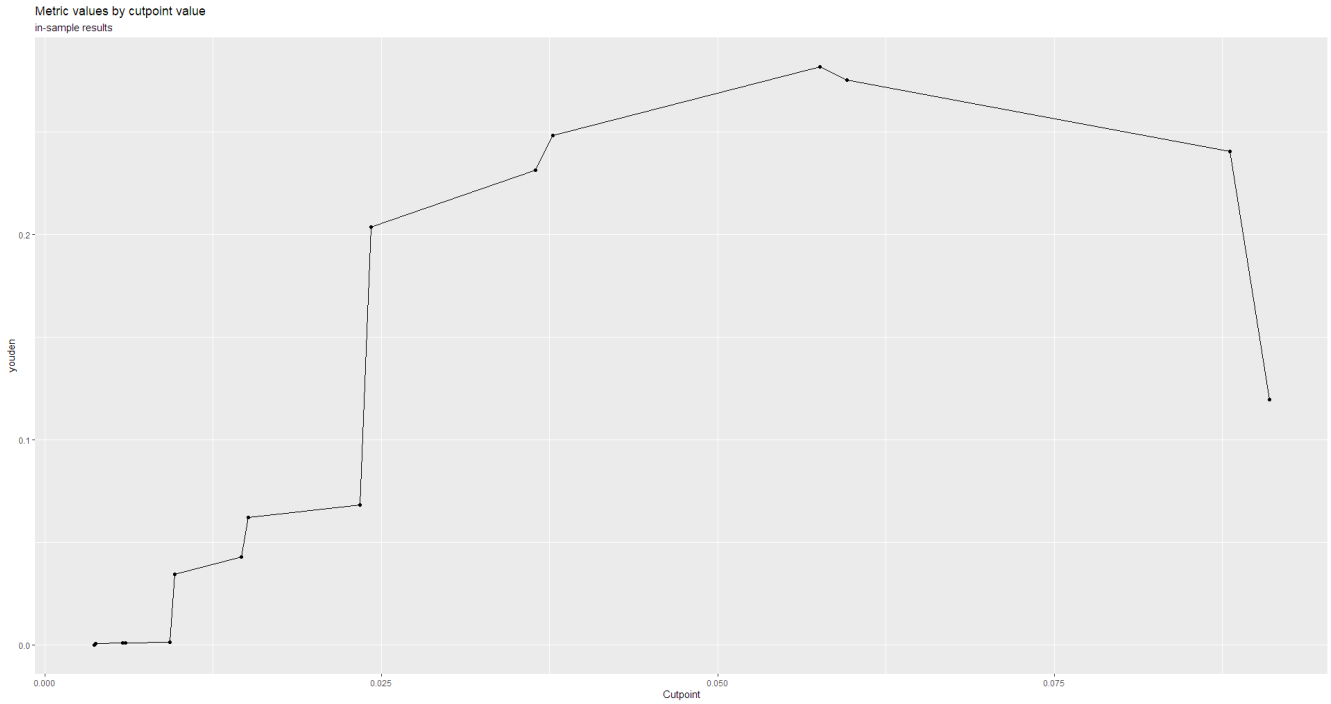


Figure 19. Youden-cutpoint figure

We will now check what heppens if we consider a 0.5 cutpoint instead of an optimal one:

```
amidata_train$predictions.value<-ifelse(amidata_train$predictions>0.5,1,0)
confusionMatrix(as.factor(amidata_train$predictions.value),
                as.factor(amidata_train$dead),positive = "1")
```

The model now reaches a very high accuracy, more than 95%, but the specificity and Youden index are very low. Even with a high accuracy the discrimination ability of the model is very poor.

If you run **table(amidata_train$predictions.value)**, you can observe that the model couldn't predict even one event with the 0.5 cutpoint.

We can now validate the accuracy of the model, applying the training model on test dataset:

```
amidata_test$predictions<- predict(Logit_full_train,newdata=amidata_test,
                                   type="response")  ①
amidata_test$predictions.value<-ifelse(amidata_test$predictions>
                                       optimal_cut$optimal_cutpoint,1,0)
confusionMatrix(as.factor(amidata_test$predictions.value),
                as.factor(amidata_test$dead),positive = "1")
```

① Create a variable with predicted probabilities by predicting the events in the test dataset.

6.1.1. Cross validation

We can also validate the model by spliting the datasert in k parts, training the model on the k-1 parts and testing it on the k-th part. This tecnique is called * k-Fold Cross Validation*. For instance, we can use the 10 folds cross validation technique, by typing:

```
train(make.names(dead)~cl_age+males+surgical,data=amidata,method="glm",
                       family="binomial",trControl=trainControl
                       (method="repeatedcv", umber=10,repeats=10),
                       na.action=na.exclude)
```

The procedure for 10-fold cross validation from the code above can be summarized as:

1. Split the dataset into 10 groups randomly
2. For each unique group:
    - Keep the group as a test data set
    - Use the remaining groups as a training data set
    - Fit a model on the training set and evaluate it on the test set
    - Store the evaluation score and discard the model
3. Summarize the results using the evaluation scores

If you consider the accuracy, the cutpoint will be setted to 0.5 by default!

Instead of the accuracy, we can evaluate the model using the AUC:

```
train(make.names(dead)~cl_age+males+surgical,data=amidata,method="glm",
                      family="binomial",trControl=trainControl
                      (method="repeatedcv",number=10,repeats=10,classProbs = TRUE,
                      summaryFunction = twoClassSummary),metric="ROC",
                      na.action=na.exclude)
```

Another resampling method for the model validation is the **Botstrap**. Bootstrap generates new samples with replacement from a dataset and with these new bootstrap samples, the bootstrap evaluates the accuracy of a sample statistic of choice by calculating its estimate, standard error, and confidence interval. The models may be developed in bootstrap samples and tested in the original sample.

We can perform a bootstrap model validation typing:

```
①
train(make.names(dead)~cl_age+males+surgical,data=amidata,method="glm",
                       family="binomial",
                       trControl=trainControl(method="boot",number=100,
                       classProbs = TRUE,summaryFunction = twoClassSummary),
                       metric="ROC",na.action=na.exclude)
②
train(make.names(dead)~cl_age+males+surgical,data=amidata,method="glm",
                       family="binomial",
                       trControl=trainControl(method="boot632",number=100,
                       classProbs = TRUE,summaryFunction = twoClassSummary),
                       metric="ROC",na.action=na.exclude)
```

① We obtain: Optimism= average (bootstrap performance - test performance). Estimated performance = Apparent performance - Optimism. We generate 100 samples with replacement of same size as dataset.

② Bootstrap .632 uses different weights: Estimated performance= 0.368*Apparent performance+ 0.632*Optimism.

EXERCISE 6.1: Apply different models, validate them and compare accuracy using ROC curves.

7. Propensity scores

Before starting, remember to clear the memory, load the packages, functions and the data:

```
list.files()
rm(list=ls())
options(scipen=999)
source("methods_lab_functions.R")

library(Hmisc)
library(gmodels)
require(ggplot2)
library(gridExtra)
library(plyr)
library(grid)
library(geepack)
library(caret)
library(geepack)
library(forestplot)
library(Epi)
library(MatchIt)
library(epitools)
library(ROCR)
library(ROCit)
library(pROC)
library(cutpointr)
library(bootLR)
library(survival)

load(file="amidata_2017.Rda")
```

We can now fit a logistic regression model on **death**, adjusted by **age**, **gender**, **risk**, **severity of illness** and **surgical**:

```
DLogit<-glm(dead~cl_age+males+risky+severe+ptca,
            family = binomial("logit"),data=amidata)
round(cbind(Beta=coef(DLogit),confint(DLogit),
            P=coef(summary(DLogit))[,4]
            ,exp(cbind(OR=coef(DLogit),
            confint(DLogit)))),4)
```

Now, fit a model on **ptca** adjusted by **age**, **gender**, **risk** and **severity of illness**:

```
PropLogit<-glm(ptca~cl_age+males+risky+severe,family = binomial("logit"),data=amidata)
round(cbind(Beta=coef(PropLogit),confint(PropLogit),P=coef(summary(PropLogit))[,4],exp
(cbind(OR=coef(PropLogit),confint(PropLogit)))),4)
```

The **propensity scores** for each observation can be collected from the predicted probabilities estimated from the model that predicts the **ptca**:

```
amidata$p<- predict(PropLogit,newdata=amidata,type="response")
```

7.1. Applications of the propensity scores

We can group the subjects by levels of propensity scores, categorised in five groups according to their quintiles. In this way, we create subgroups with similar characteristics in terms of covariates used for the propensity scores.

```
amidata$p_group<-
cut(amidata$p,quantile(amidata$p,probs=seq(0,1,0.20),na.rm=TRUE),labels=FALSE,include.
lowest=TRUE,right=TRUE)
```

We can now run a model using death as an outcome in each group, adjusted only by **ptca**.

Let's check the results:

```
for (i in 1:5) {
 DLogit<-glm(dead~ptca,family=binomial("logit"),data=amidata[amidata$p_group==i,])
 curmodel<-data.frame(round(cbind(Beta=coef(DLogit),confint(DLogit),

P=coef(summary(DLogit))[,4],exp(cbind(OR=coef(DLogit),confint(DLogit)))),4))
 names(curmodel)<-c("beta","lcl_beta","ucl_beta","p","or","lcl_or","ucl_or")
 curmodel<-curmodel[rownames(curmodel)=="ptca",c("or","lcl_or","ucl_or")]
 rownames(curmodel)<-paste("ptca STRATUM",i)
 if (i>1) {alld<-rbind(alld,curmodel)} else {alld<-curmodel}
}
print(alld)
```

The propensity scores can also be used as a covariate in a logistic regression model. In this way, we adjust the model using the predictors included in the propensity scores:

```
DLogit<-glm(dead~ptca+p,family=binomial("logit"),data=amidata)
round(cbind(Beta=coef(DLogit),confint(DLogit),P=coef(summary(DLogit))[,4],
            exp(cbind(OR=coef(DLogit),confint(DLogit)))),4)
```

Another appropriate technique can be using a stratified logistic regression model, with strata corresponding to groups of observations with similar propensity scores. The stratified logistic regression model can be carried out with the **clogistic** function from the **EPI** package, by typing into the console:

```
cLogistic<-clogistic(dead~cl_age+males+risky+severe+ptca,
                     strata=p_group,data=amidata[amidata$p_group>2,])
cbind(Beta=cLogistic$coefficients,confint(cLogistic),
      exp(cbind(OR=cLogistic$coefficients,confint(cLogistic))))
```

The model may not converge if all groups are considered!

The propensity scores can also be applied to balance the characteristics between case and control subjects. We can create a balanced dataset using the **matchit** function from the **MatchIt** package and check summary information:

```
amidata_nona<-na.omit(amidata[,c("dead","cl_age","males",
                                 "risky","severe","ptca",
                                 "p_group")]) ①

p.match<-matchit(formula = ptca~cl_age+males+risky+severe,  ②
                 data    = amidata_nona,  ③
                 method  = "nearest",  ④
                 distance= "logit",  ⑤
                 ratio   = 1)  ⑥

summary(p.match)
```

① Remove Na's from amidata.

② The model for the propensity scores.

③ Cannot have missing values. Use complete dataset without NA's.

④ Greedy match.

⑤ Distance defined by usual propensity score from logistic model

⑥ 1:1 match by default.

The balanced versus original data can be compared by plotting:

```
plot(p.match,type = "jitter",interactive=F)
plot(p.match,type = "hist",interactive=F)
```

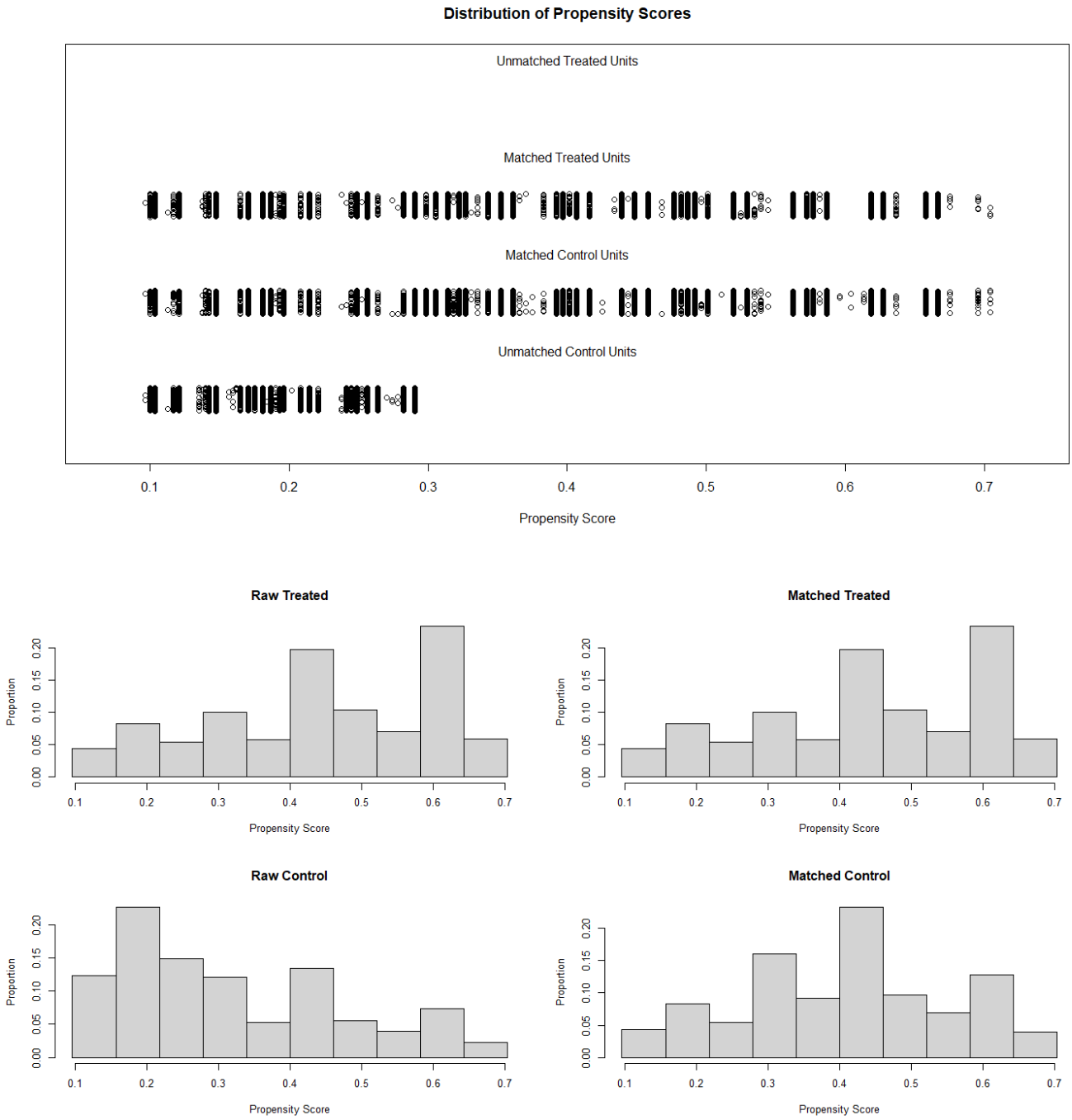


Figure 20. Distribution of Propensity Scores.

Now, you can store the balanced dataset by typing:

```
dat.match <- match.data(p.match)
```

The balanced data can be observed by the frequencies in the table of treated and control groups, considering the five groups.

```
CrossTable(dat.match$p_group,dat.match$ptca)
```

A regular (not conditional) logistic regression model can also be fitted using the new dataset. We do not need to include the age, gender, risk and severity anymore, because the data are already

adjusted for those characteristics.

```
DMLogit<-glm(dead~ptca,data=dat.match,family=binomial(link = "logit"))
round(cbind(Beta=coef(DMLogit),confint(DMLogit),
            P=coef(summary(DMLogit))[,4],exp(cbind(OR=coef(DMLogit),
            confint(DMLogit)))),4)
```

A GEE model can be applied to the binary outcomes, specifying the appropriate link function. For example, considering the **logit** link function use the following code in the console and check the results of the GEE model:

```
modgee<-geeglm(dead~ptca,data=amidata[!is.na(amidata$p_group),],
               id=p_group,family=binomial(link="logit"),
               corstr="exchangeable")

summary(modgee)
cc <- coef(summary(modgee))
citab <- with(as.data.frame(cc),cbind(OR=round(exp(Estimate),3),
             LCL=round(exp(Estimate-1.96*Std.err),3),
             UCL=round(exp(Estimate+1.96*Std.err),3)))
rownames(citab) <- rownames(cc)
print(citab)
```

The **Inverse Probability of Treatment Weighting (IPTW)** is another technique for balancing the data. It uses weights based on the propensity scores, creating samples in which the baseline characteristics are similar among treated and not treated subjects. For instance, you can obtain the Average Treatment Effect (ATE) weights by running:

```
amidata$weight<-ifelse(amidata$ptca==1,1/amidata$p,1/(1-amidata$p)) ①
tapply(amidata$weight,amidata$ptca,summary) ②
```

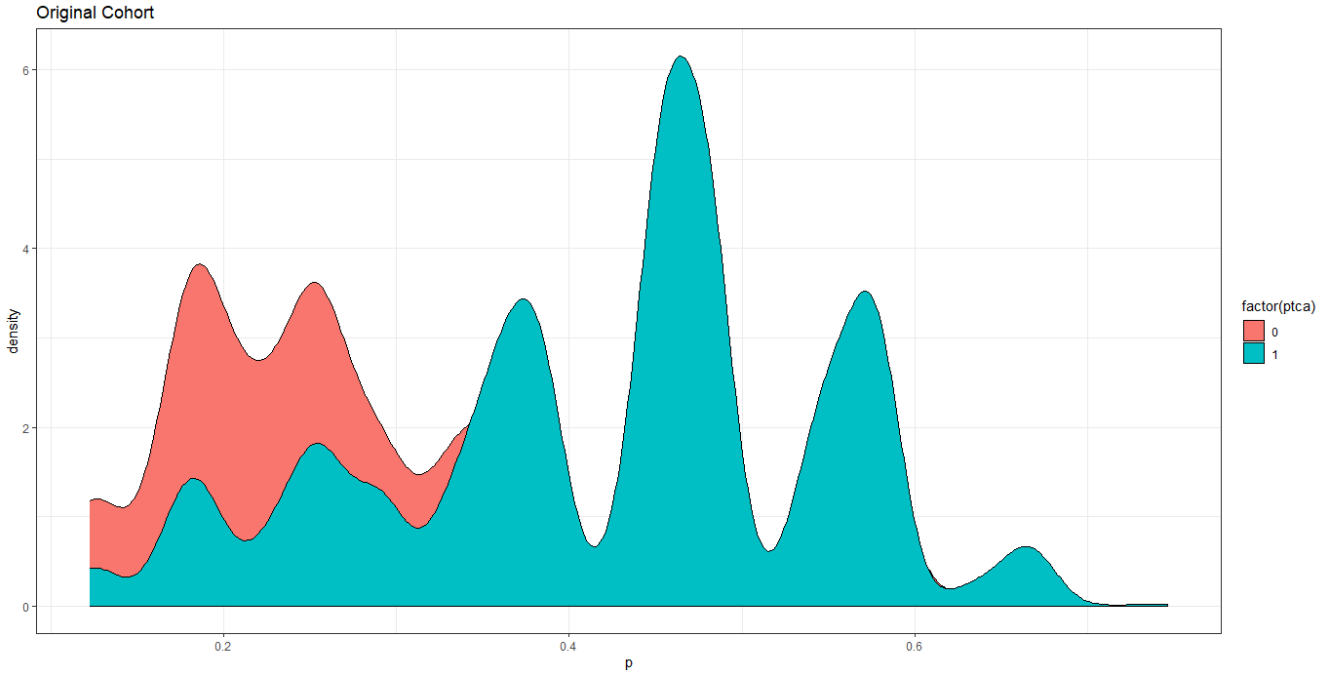
① Wi= (Zi/pi) + 1-Zi)/(1-pi where Z is the treatment assignment and p is the propensity score.

② Show summarizing statistics of weights for each level of ptca.

We can compare the distribution of the propensity scores between the original cohort and the weighted one:

```
ggplot(data = amidata,
       mapping = aes(x=p,fill=factor(ptca),weight=NULL)) +
        layer(geom="density",stat="density",position="identity") +
        theme_bw() +
        theme(legend.key = element_blank()) +
        labs(title = "Original Cohort")

ggplot(data = amidata,
       mapping = aes(x=p,fill=factor(ptca),weight=weight)) +
        layer(geom="density",stat="density",position="identity") +
        theme_bw() +
        theme(legend.key = element_blank()) +
        labs(title = "Inverse Probability of Treatment Weighting (IPTW)")
```



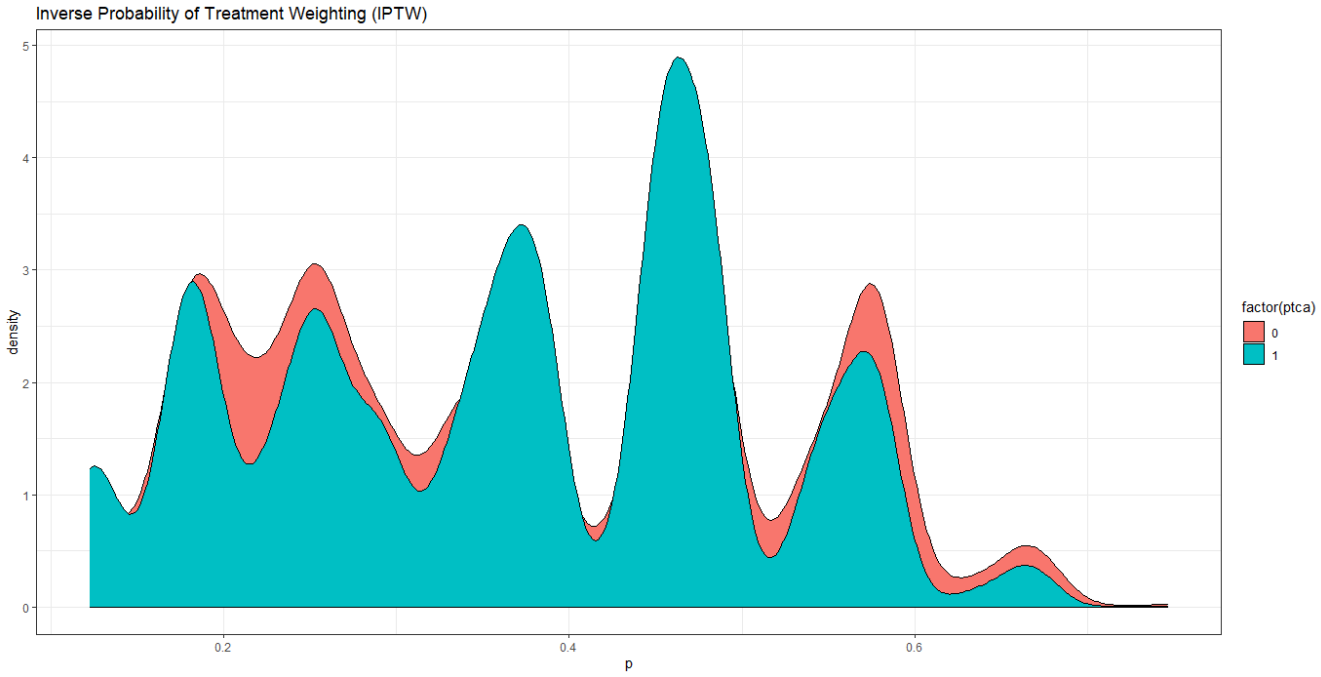


Figure 21. Distribution of Propensity Scores.

We can now apply weights to a weighted logistic regression model and check the results as follows:

```
WLogit<-glm(dead~ptca,weights=weight,data=amidata,
            family=binomial(link="logit"))
round(cbind(Beta=coef(WLogit),confint(WLogit),
            P=coef(summary(WLogit))[,4],
            exp(cbind(OR=coef(WLogit),
            confint(WLogit)))),4)
```

The Average Treatment effect for Treated (ATT) inverse probability of treatment Weighting can be obtained by typing:

```
amidata$weight_att<-ifelse(amidata$ptca==1,1,amidata$p/(1-amidata$p))  ①
tapply(amidata$weight_att,amidata$ptca,summary)  ②
```

① Create the variable with the weights, for treatment group: 1 for control group: p/(1-p).

② Show summarizing statistics of weights for each level of ptca.

Similarly, to the **ATE** , you can check the distribution of the propensity scores using the **ATT** weights:

```
ggplot(data = amidata,
       mapping = aes(x=p,fill=factor(ptca),weight=weight_att)) +
        layer(geom="density",stat="density",position="identity") +
        theme_bw() +
        theme(legend.key = element_blank()) +
        labs(title = "Inverse Probability of Treatment Weighting (IPTW)")
```

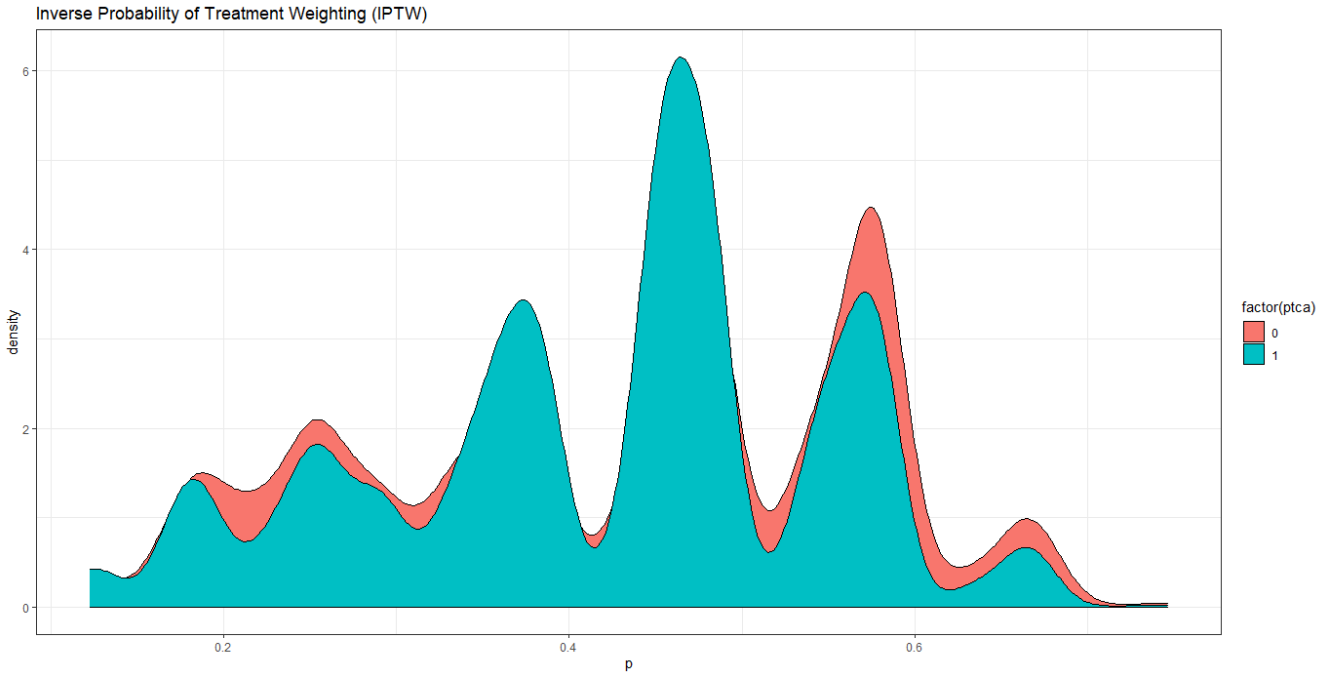


Figure 22. ATT-distribution of Propensity Scores.

We can also fit a weighted logistic regression model using the **ATT** weights:

```
WLogit<-glm(dead~ptca,weights=weight_att,data=amidata,family=binomial(link="logit"))
round(cbind(Beta=coef(WLogit),confint(WLogit),P=coef(summary(WLogit))[,4],
            exp(cbind(OR=coef(WLogit),confint(WLogit)))),4)
```

EXERCISE 7.1:
Try to build your propensity score model for the indicator **STROKE MORTALITY** (coded as "Acute Cerebrovascular Disease"). Select the most appropriate procedure or mix of procedures - see literature.

EXERCISE 7.2:
Apply simple definitions to resolve the stratified propensity method above.