# 1 Introduction to this tutorial

R is an open source and free statistical programming software including many functions and many additional packages that can be loaded into it for more specialized analyses.

## 1.1 Learning Objectives

After successfully completing this tutorial, students will be able to:

1. Use R Studio to write scripts and save results
2. Conduct standard and vector arithmetic calculations
3. Perform basic analyses on data sets
4. Create simple statistical plots

In this tutorial, students should read all pages from page 1, reproduce example R commands, and do the exercises.

## 1.2 Installing R and R Studio

You will first install the basic R program and then install "R Studio" which provides a more user-friendly interface for R. Note that you must install "R" first, then "R Studio. Installation procedure may differ for personal and university computers.

**Install on an institutional computer**

If available, use an appropriate app store or contact the IT department if you do not have rights to install software.

**Install on your own computer**

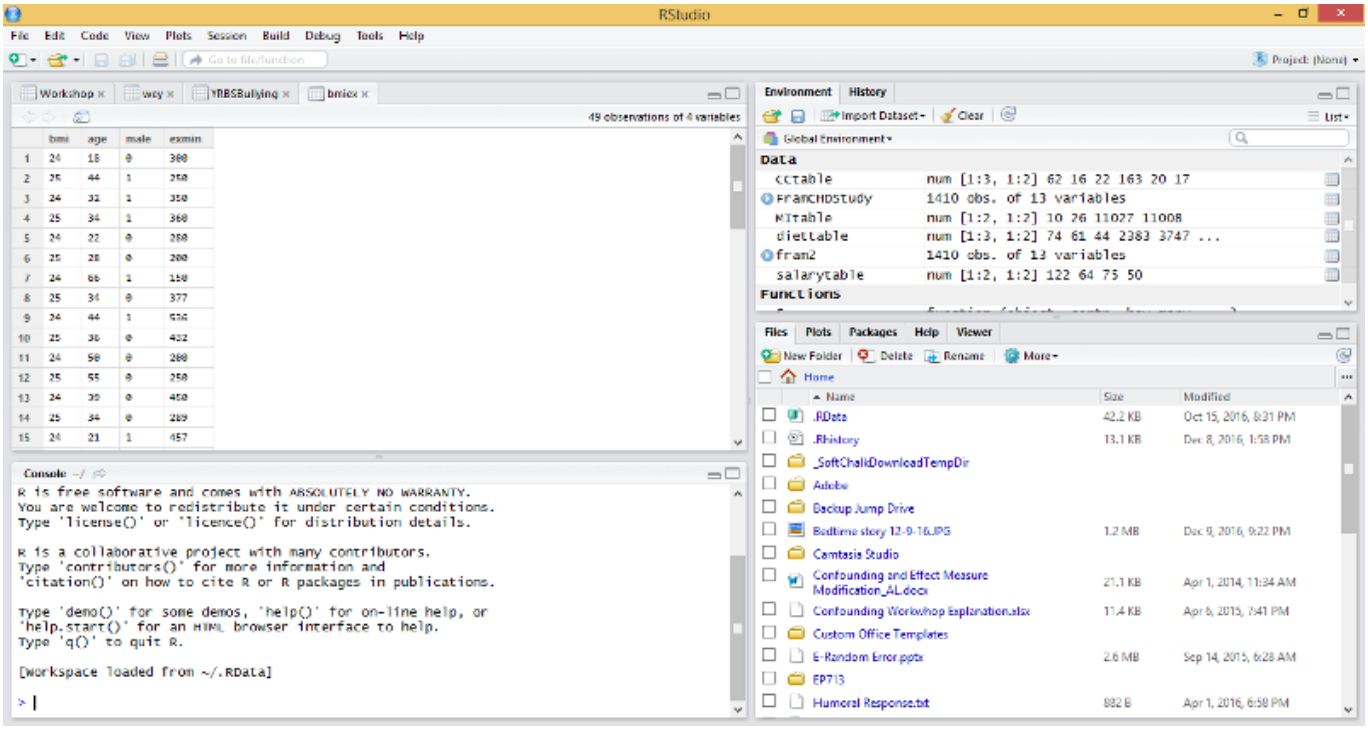
Download the software and read installation instructions from the following web sites:

- R on the Comprehensive R Archive Network: http://cran.r-project.org/
- R Studio URL: https://www.rstudio.com/products/rstudio/

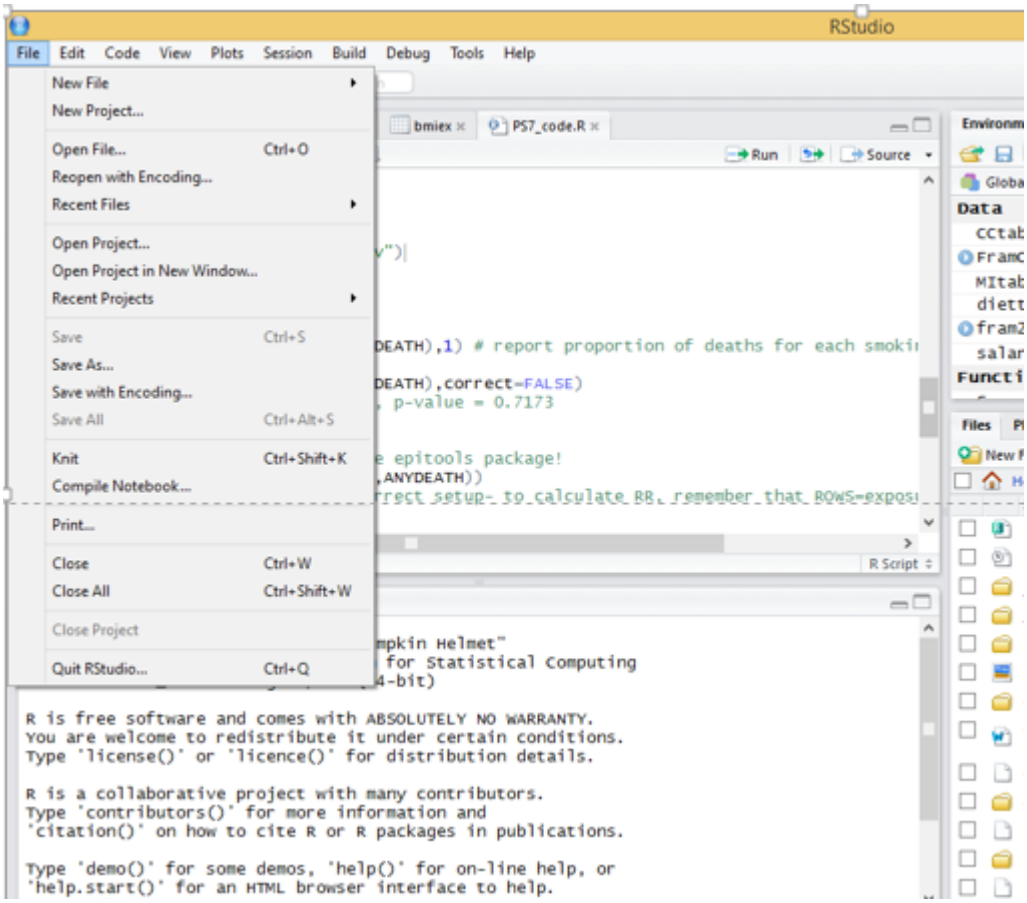
# 1.3 The R Studio Interface

The image below shows how the R Studio interface might look for an active user. Note that there are tabs at the top, and there are four sub-windows.

- The upper left window is for creating, running, and saving scripts which consist of a series of commands that are to be executed in sequence. This window is also used for viewing data sets, as illustrated in the image below.
- The sub-window at the lower left is the Console, where individual commands are entered and executed, and the results are shown. When a script is run from the editor, the output will appear in the Console window.
- The window at the upper right shows available data sets and objects when the "Environment" tab is selected, and it shows previously run commands when the "History" button is selected.
- The window at the lower right shows files and folders in the user's R folder in the image below, but this window also is used to display "Help" responses, and for displaying graphs and plots generated by R.




**Click on the "File" tab** at the upper left on the screen; you will see a menu that provides options for creating new files, saving, opening, printing, and quitting, as illustrated below.




Try hovering over each of the other tabs at the top of the interface, just to get an idea of the many functions that are available.

## 1.4 Credits

This document is inspired from a similar module originally created by members of the Boston University (Ching-Ti Liu, Jacqueline Milton, Avery McIntosh, Wayne W. LaMorte).

# 2 Entering Commands

The command prompt is "**>**" and can be found at the bottom of the **Console** window at the lower left. This is where you can enter commands for immediate execution.


- Commands are separated either by a new line or by a semi colon (;)
- Parentheses and brackets must be closed
- R is case sensitive
- R stores both data and output from data analysis (as well as everything else) in *objects*
- In the console, use arrow keys to traverse through the history of commands
  - "Up arrow" – traverse backwards (older commands)
  - "Down arrow" – traverse forward (newer commands)
- Text auto-completion is available using TAB key (e.g. for completing function or object names)

# 2.1 R as a Calculator

Enter the following examples at the command prompt of the Console to get a feel for this.


In this document, commands are shown in blue, comments in green, and results in black.

## 2.1.1 Simple operations

```
8+7

[1] 15

6*4

[1] 24
```

The notation [1] indicates the first element of the result. This is useful later when we have many resulting elements.

**Commonly Used Operators**

| Symbol | Meaning |
|---|---|
| + | add |
| - | subtract |
| * | multiply |
| / | divide |
| ^ | power (e.g. 2^3 is equal to 8) |

## *2.1.2* Comments

The # symbol indicates to R that what follows is a comment that should not be executed. You will find it useful to include comments in your scripts to annotate your programming.

```
# This is a comment that will not be executed.

>
```

## 2.1.3 Statistical functions

```
sqrt(81) # square root

[1] 9

1 + abs(-4) # absolute value

[1] 5
```

There are other statistical functions such as *exp()* for exponentiation, *log()* natural logarithm, or *log10()* for base-10 logarithm.

### 2.1.4 The Order of Operations

Keep in mind the rules for the order of operations: **"PEMDAS," meaning Parentheses, Exponents, Multiplication & Division, then Addition & Subtraction**

- Do calculations inside Parentheses first, e.g., $6 \times (5 + 3) = 6 \times 8 = 48$
- Then compute Exponents (Powers, Roots) before multiplication and division: $5 \times 2^2 = 5 \times 4 = 20$
- Then Multiply or Divide (before you Add or Subtract), e.g., $2 + 5 \times 3 = 2 + 15 = 17$
- **Otherwise just go left to right.**

### 2.1.5 Generating sequences

The colon operator ( : ) can be used to generate a sequence of integer values. More possibilities are available with the *seq(<start>, <end>, by=<step>)* function.

```
1:3

[1] 1 2 3

-2:1

[1] -2 -1 0 1

seq(1, 9, by = 2)      # matches 'end'

[1] 1 3 5 7 9

seq(1, 9, by = pi)     # stays below 'end'

[1] 1.000000 4.141593 7.283185

seq(1, 6, by = 3)

[1] 1 4
```

### 2.1.6 Exercise

1) Have a look at the following commands. For each command, first think about the result you expect. Then, run the command and describe your observations. Was your expectation correct? If not, why?

```
1+2

8/2-2*(2-3)

3*5*4/2

3+5*abs(-2)
```

2) Calculate the square root of 9

3) Calculate the log10 of the absolute value of -81

4) Create a sequence from 1 to 100 with a separation of 5

## 2.2 Assignment and variables

When performing calculations, we may want to save the intermediate results for later use. This can be achieved by assigning **objects (values)** to symbolic **variables** using an "**assign**" function. Once you assign an object a designation, it stays in the working memory until you close the program. To see what objects are in the working memory, type *ls()*, or select Show Workspace command from the drop down menu.

Write the following examples in an R script and run the current/selected line(s) by pressing CONTROL+ENTER keys or clicking on button named "Run". Save the script file from time to time.

So to create a variable called x with value of 2, we type

```
x <- 2

x # If I enter x, R returns 2

[1] 2
```

[Note that the symbol **<-** is made up from "less than" and "minus" with **NO space** between them.]

Although assignment using **<-** is recommended, one can also use the equals sign:

```
x=2
```

The arrow for the assignment symbol always points to the name assigned to the vector.

We can also assign multiple variables the same value and change variable values, as follows:

```
x <- y <- 3 # Both x and y are assigned values of 3

x # If I enter the updated x, R returns now 3

[1] 3

y # If I enter y, R returns 3 again.

[1] 3
```

We can manage the workspace with *ls()* that lists variables and *rm()* that removes variables from the computer RAM memory (useful if you have big data and little amount of memory)

```
ls()
[1] "x" "y"
rm(x)
ls()
[1] "y"
rm(list = ls()) # removes all objects from the workspace
ls()
character(0)
```

# 3 Classes and Objects

## 3.1 Definitions

Some of the main data classes (or types) in R are:

- numeric - e.g. 1, 25.5, 1e-6
- character - e.g. “ABCD”, “Hello World 24!”
- logical - TRUE, FALSE, NA (Not Applicable)
- factor - categorical values
- vector - a set of objects of the same class
- matrix - table of objects of the same class
- data frame - table of objects of same or different classes

R manipulates objects that are instances of classes (e.g. a particular value of a specific type):

- the object of value **1** is an instance of class *numeric*
- the object of value **“ABC“** is an instance of class *chr* (character)
- the object `ls` is an instance of class *function*

## 3.2 The *class()* and *str()* functions

The `class()` function returns the class of an object

```
class(1)

[1] "numeric"
```

The `str()` function displays the structure of an object

```
str("ABC")

chr "ABC"
```

# 4 Vectors

Vectors are sets of one or more objects of the same class (the atomic mode), e.g., numeric, integer, logical or character objects. For example, a numeric vector might consist of the numbers (1.2, 2.3, 0.2, 1.1). A vector can also have just a single object.

## 4.1 Concatenation

Vectors with multiple objects can be created using concatenation with the function `c()`. The c stands for *concatenation*. The objects themselves are placed inside the rounded parentheses, i.e. ( ), not square [ ] or curly { } brackets.

To create a vector named x, consisting of four numbers, namely 1.2, 2.3, 0.2 and 1.1, we can use the following R command

```
x <- c(1.2, 2.3, 0.2, 1.1)

x

[1] 1.2 2.3 0.2 1.1
```

## 4.2 Selecting Specific Elements of Data

If we want to select only some elements in the vector, then we can use **indices**. For example, if we want to know the first, the last, the third and the last three elements in vector x, then we can type

```
x[1]

[1] 1.2

x[ length(x) ]

[1] 1.1

x[3]

[1] 0.2

x[ c(2,3,4) ]

[1] 2.3 0.2 1.1
```

### 4.2.1 Exercise

1) Define x as shown below and write a code to get the second element of the vector

```
x <- c(1.2, 2.3, 0.2, 1.1)
```

2) What do you see in the screen if we type the following commands?

```
x[-1]

x[2:4]
```

3) Based on your observation, what does the negative sign do within the index?

4) Write a code to get together the first and third element

5) Write a code to get all elements but NOT the first and NOT the third elements.

# 4.3 Logical Vectors and Logical Operators

R allows us to create logical vectors and to manipulate logical quantities as well. To create logical vectors, you may use **TRUE** (or **T)**, **FALSE** (or **F)**, or **NA** (for **missing / not available**) directly, or type in the condition/logic operation. Note that in order to be used in arithmetic calculations, R treats **TRUE as 1** and **FALSE as 0**.

Let's look at some examples to see how these operators work.

```
1<2

[1] TRUE

! (1<3) # logical NOT (!)

[1] FALSE

1 != 3

[1] TRUE

(3 != 1) & (2 >= 1.9) # logical AND (&): TRUE & TRUE returns TRUE

[1] TRUE

(3 == 1) | (3 < 5) # logical OR (|): TRUE | TRUE returns TRUE

[1] TRUE

y <- c(TRUE, FALSE, 5>2)

y

[1] TRUE FALSE TRUE
```

## 4.3.1 Logical operators

| Symbol | Meaning |
|---|---|
| == | logical equals |
| != | not equal |
| ! | logical NOT |
| & | logical AND |
| \| | logical OR |
| < | less than |
| <= | less than or equal to |
| > | greater than |
| >= | greater than or equal to |

### 4.3.2 Logical evaluation

| Command | Result |
|---|---|
| TRUE | TRUE |
| FALSE | FALSE |
| ! TRUE | FALSE |
| ! FALSE | TRUE |
| TRUE & TRUE | TRUE |
| TRUE & FALSE | FALSE |
| FALSE & FALSE | FALSE |
| TRUE \| TRUE | TRUE |
| TRUE \| FALSE | TRUE |
| FALSE \| FALSE | FALSE |

### 4.3.3 Exercise

1) If you type the following commands into R, what will (x, y, z, w) be? Make first a guess and then test it.

```
x <- !(5>=3)

y <- ((2^4) > (2*3))

z<- x|y

w <- x&y
```

## 4.4 Vector Arithmetic, subsets and Functions

### 4.4.1 Vector arithmetic

Let's go back to a vector discussed above.

```
x <- c(1.2, 2.3, 0.2, 1.1)
```

This vector consists of four numbers. In some circumstances, we may want to apply certain operations or calculations to each element in the vector. For example, suppose we wanted to use the vector *x* to create a new vector *y* with elements that are 2 time each x plus 3. One could do this element by element with the following command:

```
y <- c( 2*x[1]+3, 2*x[2]+3, 2*x[3]+3, 2*x[4]+3 )

y

[1] 5.4 7.6 3.4 5.2
```

but a simpler way to do this is to use the following command:

```
2 * x + 3

[1] 5.4 7.6 3.4 5.2
```

### 4.4.2 Vector subsets

Logical operators can also be used to modify or select subsets of a data set. For example, in the previous example, we saw that x[c(2,3,4)], x[-1] and x[2:4] work exactly the same and select the last three elements of the vector x.

You can also use a logical vector to select elements

```
x[c(FALSE, TRUE, TRUE, TRUE)]
```

This instructs R to skip the first element and then select the next three, so it returns the following:

```
[1] 2.3 0.2 1.1
```

If we wanted to select the elements with values greater than 1, we could use the command:

```
x[x>1]

[1] 1.2 2.3 1.1
```

In summary, R can perform functions over entire vectors and can be used to select certain elements within a vector.

### 4.4.3 Important Statistical Functions in R

In addition to the elementary arithmetic operations, R can also use vector functions such as those listed below.

- `length(x)`
- `max(x)`
- `min(x)`
- `sum(x)`
- `mean(x)`
- `median(x)`
- `range(x)`
- `var(x)`
- `sd(x)`

Functions used earlier on single values such as *`sqrt()`* or *`log()`* can take a vector as input to perform a calculation for each element.

```
sqrt( c(4, 9, 16) )

[1] 2 3 4
```

### 4.4.4 Exercise

R contains a set of small datasets for testing purpose (from package named *datasets*; loaded by default). Here, we will use the dataset named "*Lengths of Major North American Rivers*" that is available as a vector object called *rivers*.

1) display the values of *rivers* and read its documentation using the following commands

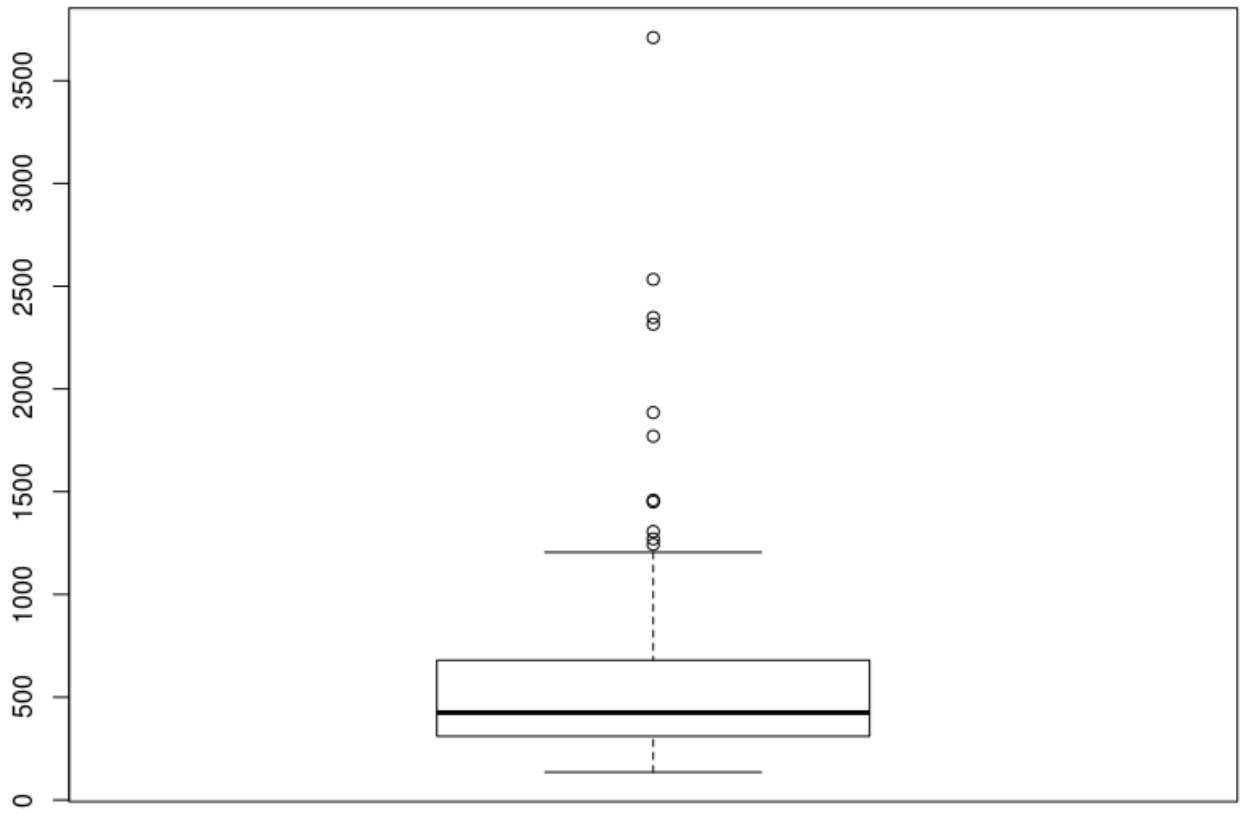
```
rivers

?rivers
```

2) calculate the number of elements using the appropriate function

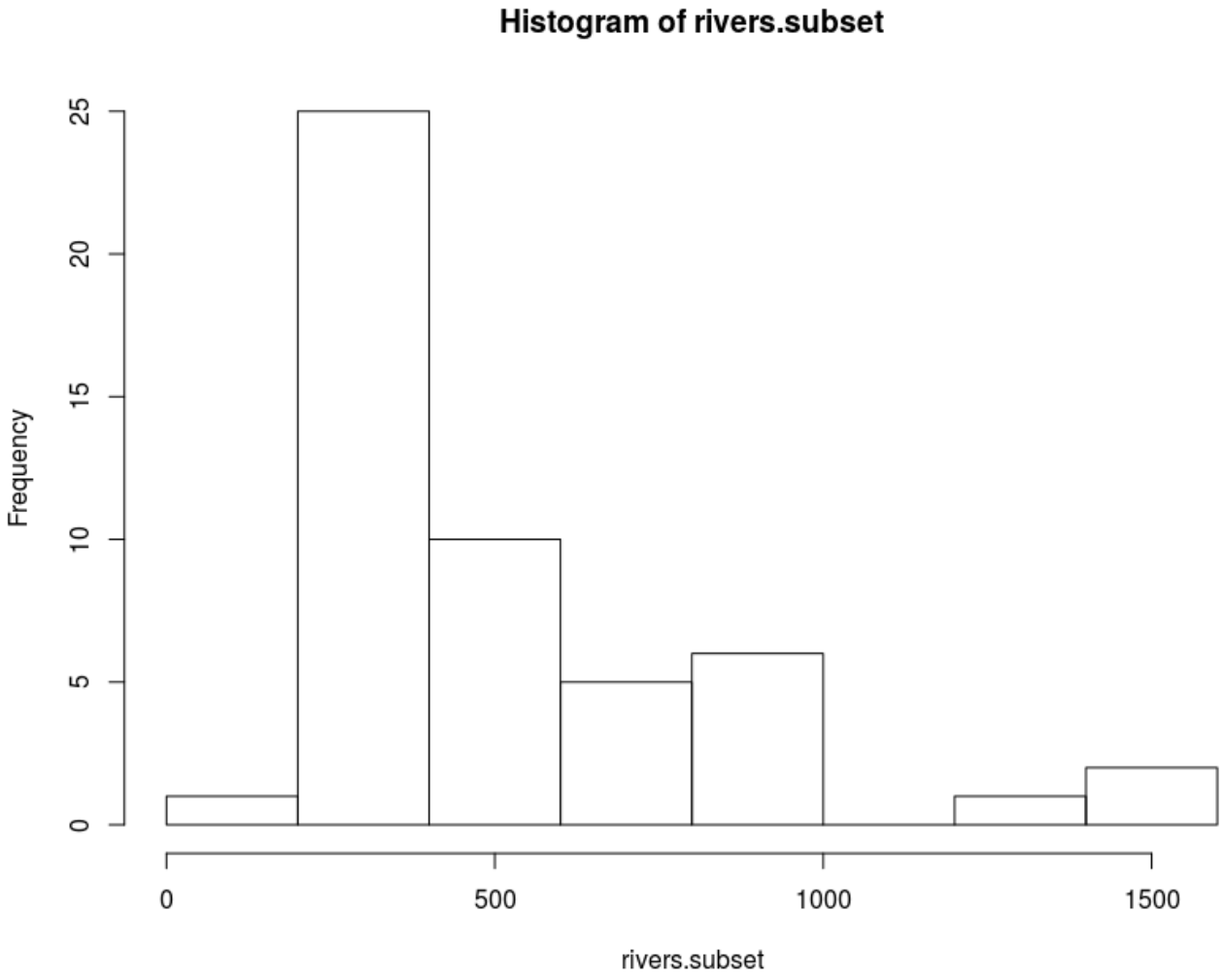
3) calculate the mean and standard deviation

4) create a boxplot of the vector values using function *boxplot()*




5) create a vector named *rivers.subset* containing the 50 first values

6) create a histogram of values in the subset using function *hist()*




7) what would you do to better visualize in the histogram the possible values?

8) Try the following solution. For what is the parameter *breaks* used?

```
hist( rivers,  breaks=20 )
```

9) create a boxplot of the values greater than 500

10) plot a histogram of the 25 first values

11) optional: create a barplot of 25 values that are greater than 500 (use *barplot()* )


Each time you create a new plot in R it replaces the previous content in the window (at the lower right in R Studio). However, you can review the previously created plots by using the arrow tab beneath the "Files" tab in the plot window. Note that you can also export the file and save it as an image file or as a PDF file, or you can copy the image to the clipboard and then paste it into another application, such as PowerPoint or Word.

# 5 Data Frames

The vectors that have been discussed previously in this module were one-dimensional, i.e., they consisted of a simple series of elements that you could imagine being organized in a single row or in a single column. Data frame objects present the information as table. In practical term, data frame may be seen as a collection of vectors of the same length, each vector being a column of the table.

## 5.1 Data Frame Indexing and creation

### 5.1.1 Data Frame Indexing

The data frame object *cars* is provided by the dataset "*Speed and Stopping Distances of Cars*". It has 50 rows and 2 columns named "speed" and "dist".

```
?cars

head(cars) # the 6 first rows (as data frame)

  speed dist
1     4    2
2     4   10
3     7    4
4     7   22
5     8   16
6     9   10
```

We can retrieve values at particular rows and columns:

```
cars[1,2]  # cell at row 1 and column 2

[1] 2

cars[5,2]  # cell at row 5 and column 2

[1] 16
```

We can retrieve a full row (as data frame) or column (as vector)

```
cars[,2]   # the second column

 [1]   2  10   4  22  16  10  18  26  34  17  28  14  20  24  28  …
[20]  26  36  60  80  20  26  54  32  40  32  40  50  42  56  76  …
[39]  32  48  52  56  64  66  54  70  92  93 120  85
cars[4,]   # the fourth row

  speed dist

4     7   22
```

Rows and columns can be indexed by name

```
colnames(cars)

[1] "speed" "dist"

cars$speed

 [1]  4  4  7  7  8  9 10 10 10 11 11 12 12 12 12 13 13 13 13 14 …

[26] 15 16 16 17 17 17 18 18 18 18 19 19 19 20 20 20 20 20 22 23 …

cars$speed[1:5]

[1] 4 4 7 7 8

cars[1:5, "speed"]

[1] 4 4 7 7 8

rownames(cars)

[1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12" …

[28] "28" "29" "30" "31" "32" "33" "34" "35" "36" "37" "38" "39"…

cars["3", "speed"]

[1] 7
```

### 5.1.2 Data Frame Creation

A data frame can be created from several vectors. We will use 2 vectors from the *state* dataset: *state.name* (names of 50 USA states) and *state.area* (area of 50 USA states).

```
head(state.name)

[1] "Alabama"    "Alaska"     "Arizona"    "Arkansas"   "California"

[6] "Colorado"

head(state.area)

[1]  51609 589757 113909  53104 158693 104247

state.combined <- data.frame(state.name, state.area)

head(state.combined)

  state.name state.area

1    Alabama      51609

2     Alaska      589757

3    Arizona      113909

…
```

Colunm and row names can be changed if not satisfactory or added if missing.

```
colnames(state.combined) <- c("state", "area")

rownames(state.combined)[1] <- "state_1"

head(state.combined)

             state   area
state_1    Alabama  51609
2           Alaska 589757
3          Arizona 113909
4         Arkansas  53104
5       California 158693
6         Colorado 104247
```

In order to change the 50 row names, we will use the function *paste( object1, object2, object3, …)* that converts the input objects as character vector. For example, *paste("ABC", 12)* will return "ABC 12" and *paste("ABC", 1:2)* will return a character vector of 2 elements: "ABC 1" and "ABC 2". Parameter *sep* allow to change the default white space separator.

```
rownames(state.combined) <- paste("state_", 1:50, sep="")

head(state.combined)

             state   area
state_1    Alabama  51609
state_2     Alaska 589757
state_3    Arizona 113909
state_4   Arkansas  53104
…
```

### 5.1.3 Data Frame import and export

Data Frames can also be created from Excel or CSV files. The simplest way is to use the "Import Data set" button from the Environment tab in R Studio.

Although not detailed here, the fllowing R commands may be used:

- *read.table(), read.delim(), read.csv()*

The *write.csv()* function can be used to export a table (write it into a file on the hard drive)

```
write.csv(state.combined, file="combined.csv")
```


The working directory, where files are read and written by default, can be viewed using *getwd()* function and changed using *setwd("path to directory")*.

# 5.2 Exploring and Manipulating a Data Frame

## 5.2.1 Exploring the Variables

We will use dataset *esoph* below: Data from a case-control study of (o)esophageal cancer in Ille-et-Vilaine, France.

Read the documentation to understand the data format

```
?esoph
```

Get the data frame dimensions as numbers of rows and columns, and display the 6 first rows:

```
nrow(esoph)# number of rows

[1] 88

ncol(esoph) # number of columns

[1] 5

dim(esoph) # number of rows and columns in a vector

[1] 88  5

dim(esoph)[1] # number of rows from the vector

[1] 88

dim(esoph)[2] # number of columns from the vector

[1] 5

head(esoph) # display first 6 rows

  agegp     alcgp    tobgp ncases ncontrols
1 25-34 0-39g/day 0-9g/day      0        40
2 25-34 0-39g/day    10-19      0        10
3 25-34 0-39g/day    20-29      0         6
4 25-34 0-39g/day      30+      0         5
5 25-34     40-79 0-9g/day      0        27
6 25-34     40-79    10-19      0         7
```

Compute descriptive statistics for each column:

```
summary(esoph)
```

```
   agegp          alcgp          tobgp          ncases          ncontrols     
 25-34:15   0-39g/day:23   0-9g/day:24   Min.   : 0.000   Min.   : 1.00  
 35-44:15   40-79    :23   10-19    :24   1st Qu.: 0.000   1st Qu.: 3.00  
 45-54:16   80-119   :21   20-29    :20   Median : 1.000   Median : 6.00  
 55-64:16   120+     :21   30+      :20   Mean   : 2.273   Mean   :11.08  
 65-74:15                                 3rd Qu.: 4.000   3rd Qu.:14.00  
 75+  :11                                 Max.   :17.000   Max.   :60.00  
```

Print the structure of the data frame, including column types

```
str(esoph)
```

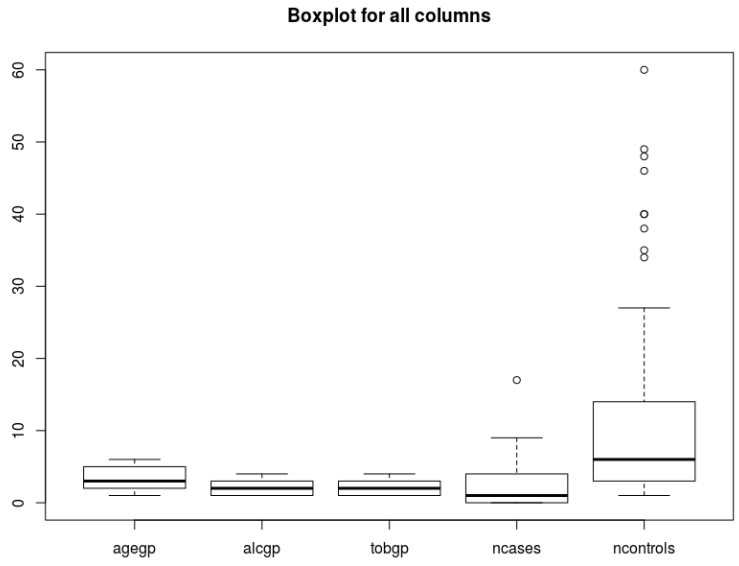
```
'data.frame':  88 obs. of  5 variables:
 $ agegp    : Ord.factor w/ 6 levels "25-34"<"35-44"<..: 1 1 1 1 1 ...
 $ alcgp    : Ord.factor w/ 4 levels "0-39g/day"<"40-79"<..: 1 1 1 ...
 $ tobgp    : Ord.factor w/ 4 levels "0-9g/day"<"10-19"<..: 1 2 3 ...
 $ ncases   : num  0 0 0 0 0 0 0 0 0 0 ...
 $ ncontrols: num  40 10 6 5 27 7 4 7 2 1 ...
```

```
class(esoph$agegp) # objects may have several classes
```

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

Show a boxplot for each column with custom main title:

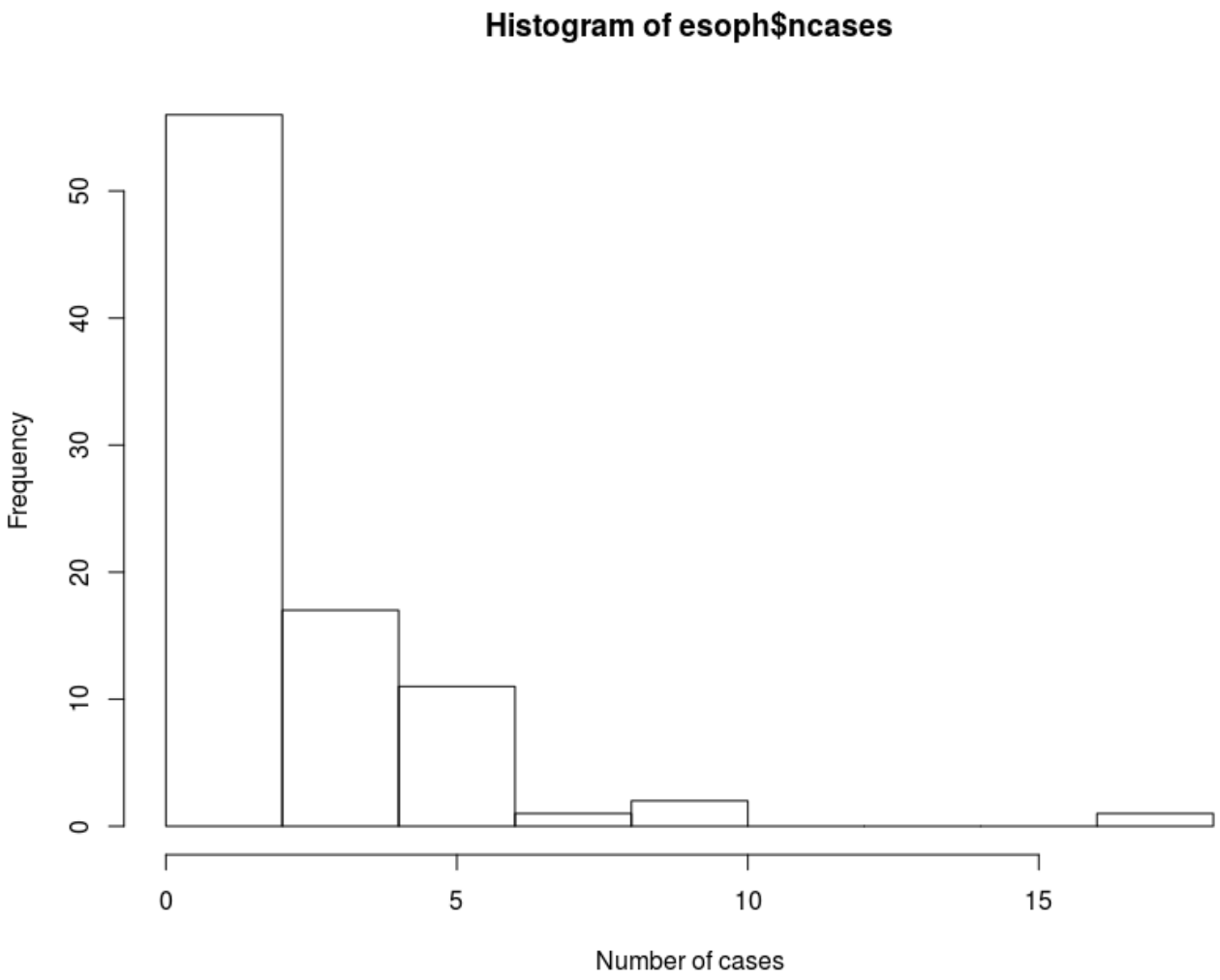
```
boxplot(esoph, main="Boxplot for all columns")
```




Note that factor variables have been plotted using arbitrarily given indices!

Show a histogram of values in column ncases with custom x axis label:

```
hist(esoph$ncases, xlab="Number of cases")
```




### 5.2.2 Exercise

1) Create a boxplot only for the numerical columns (the 2 last columns)

2) Create a simple plot only for the numerical columns and set axis labels using parameters *xlab* and *ylab*

3) Create a vector named *ntotal* with the sum of the cases and controls

4) Create a new data frame named *esoph2* composed by the data frame *esoph* with additional column *ntotal*

5) Create a vector called *case.percentages* containing percentage of cases (compared to total) and plot the vector in a barplot (use function *barplot()*)

6) Create a new column called *label* directly in the data frame *esoph2*, this column containing a text composed by number of cases and age group as the following examples: “1 x 25-34” or “3 x 35-44” (use function *paste()*)

# 6 Analyzing Data by Subsets

## 6.1 The *table()* Function

To compute proportions in different **categorical variables**, the *table()* or *prop.table()* functions can be used. For the preceding example, we can generate a contingency table of age and alcohol groups:

```
table(esoph$agegp, esoph$alcgp)

        0-39g/day 40-79 80-119 120+
  25-34         4     4      3    4
  35-44         4     4      4    3
  45-54         4     4      4    4
  55-64         4     4      4    4
  65-74         4     3      4    4
  75+           3     4      2    2
prop.table(table(esoph$agegp, esoph$alcgp))

         0-39g/day      40-79     80-119       120+
  25-34 0.04545455 0.04545455 0.03409091 0.04545455
  35-44 0.04545455 0.04545455 0.04545455 0.03409091
  45-54 0.04545455 0.04545455 0.04545455 0.04545455
  55-64 0.04545455 0.04545455 0.04545455 0.04545455
  65-74 0.04545455 0.03409091 0.04545455 0.04545455
  75+   0.03409091 0.04545455 0.02272727 0.02272727
```

In this case, we can appreciate the unbalanced experimental design for old persons and high tobacco consumption (n < 4).

## 6.2 The *tapply()* function

The *tapply()* function is useful for executing functions on subsets of a data frame, including **numeric variables**. It enables you to subset the data by one or more classifying factors and then performing some function by subset (e.g., computing the mean and standard deviation of a given variable). The basic structure of the tapply function is:

*tapply(<VAR>,<BY.VAR>,<FUN>)*

where *<VAR>* is the variable that you want to analyze, *<BY.VAR>* is the variable that you want to subset by, and *<FUN>* is the function or computation that you want to apply to *<VAR>*.

For the *esoph* dataset, the average number of cases per age groups is calculated as follows:

```
tapply(esoph$ncases, esoph$agegp, mean)

     25-34      35-44      45-54      55-64      65-74        75+
0.06666667 0.60000000 2.87500000 4.75000000 3.66666667 1.18181818
```

Or, the maximum number of controls per alcohol groups:

```
tapply(esoph$ncontrols, esoph$alcgp, max)

0-39g/day     40-79    80-119      120+
       60        40        18        10
```

Or, the maximum number of cases per age groups and alcohol groups:

```
# tapply expects a list object for several classifying factors

groups.list <- list(esoph$agegp, esoph$alcgp)

tapply(esoph$ncases, groups.list, max)

      0-39g/day 40-79 80-119 120+
25-34         0     0      0    1
35-44         1     3      0    2
45-54         1     6      6    4
55-64         4     9      9    6
65-74         5    17      6    3
75+           2     2      1    2
```

## 6.3 Logical vectors

Vectors of logical values can be used to index other vectors and thus data frames. They can be used to subset a data frame and apply a function.

Let's index age group 25-34 in variable *mask*, and make a boxplot of column "ncontrols" for this group:

```
mask <- esoph$agegp == "25-34"

mask # logical vector identifying age group 25-34

 [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[13]  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[25] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[37] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[49] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[61] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[73] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[85] FALSE FALSE FALSE FALSE
```

We will use this vector of logical values to select rows in the data frame. Each logical value of the vector corresponds to a row (first logical value to first row, second logical value to second row, …). Only rows associated with TRUE values will be selected.

```
esoph[ mask , ] # print rows selected by TRUE values in mask vector

   agegp     alcgp    tobgp ncases ncontrols
1  25-34 0-39g/day 0-9g/day      0        40
2  25-34 0-39g/day    10-19      0        10
3  25-34 0-39g/day    20-29      0         6
4  25-34 0-39g/day      30+      0         5
5  25-34     40-79 0-9g/day      0        27
6  25-34     40-79    10-19      0         7
7  25-34     40-79    20-29      0         4
8  25-34     40-79      30+      0         7
9  25-34    80-119 0-9g/day      0         2
10 25-34    80-119    10-19      0         1
11 25-34    80-119      30+      0         2
12 25-34      120+ 0-9g/day      0         1
13 25-34      120+    10-19      1         1
14 25-34      120+    20-29      0         1
15 25-34      120+      30+      0         2
boxplot( esoph[mask, "ncontrols"] ) # ncontrols for selected rows

# The same in 1 line but for 2 columns:

boxplot(esoph[ esoph$agegp == "25-34", c("ncases","ncontrols" ) ])
```

### 6.3.1 Exercise

We want to analyze the *airquality* data set: Daily air quality measurements in New York, May to September 1973.

1) For each month, print the average ozone measurement

2) For each month, print the maximum temperature

3) Display rows with Ozone not equal to NA using the *is.na()* function and the logical NOT

4) Display rows with Wind < 5

5) Display rows with Wind < 5 and Ozone not equal to NA

6) Create a new data frame called *summer* from measurements done in the first 15 days of a summer month (June to September; keep only the 4 first columns) and prints the number of rows

7) Show distributions of the first 4 columns of the summer data frame in a boxplot

# 7 Merging tables

We have seen above how to create a bigger table from smaller ones (including vectors) using the *data.frame()* function. Columns of the smaller tables must have the same number of rows and they are simply put next to each other in a new table.

If two tables must be merged but the order of their rows does not fit, we can check if values in one column from a table can be compared to one column of the other table (check if a common key exist) in order to align the rows. The common key may be any column or the row names.

Let's use as example below the *state.x77* and *USArrests* datasets that contain statistics on crimes in 50 USA states.

The *state.x77* is an object of class matrix with 50 rows and 8 columns giving statistics such as murder rate per 100,000 population (1976). For this demonstration, we will create a new data frame *my.state* using all the rows but only 3 columns from *state.x77* matrix.

```
my.state <- data.frame(state.x77[, c("Population", "Area", "Murder")])

head(my.state)

           Population   Area Murder
Alabama          3615  50708   15.1
Alaska            365 566432   11.3
Arizona          2212 113417    7.8
Arkansas         2110  51945   10.1
California      21198 156361   10.3
Colorado         2541 103766    6.8
```

The *USArrests* is a data frame with 50 rows on 4 columns about crimes and population in 1973 such as Murder arrests per 100,000 population. For this demonstration, we will use it to create a new data frame *my.USArrests* with different rows order based on UrbanPop column.

```
my.USArrests <- USArrests[order(USArrests$UrbanPop), ]

head(my.USArrests)

               Murder Assault UrbanPop Rape
Vermont           2.2      48       32 11.2
West Virginia     5.7      81       39  9.3
Mississippi      16.1     259       44 17.1
North Dakota      0.8      45       44  7.3
North Carolina   13.0     337       45 16.1
South Dakota      3.8      86       45 12.8
```

We want now to compare the Murder columns from *my.state* and *my.USArrests* data frames that do not have rows aligned.

```
my.US <- merge(my.state, my.USArrests, by="row.names")

head( my.US )
```

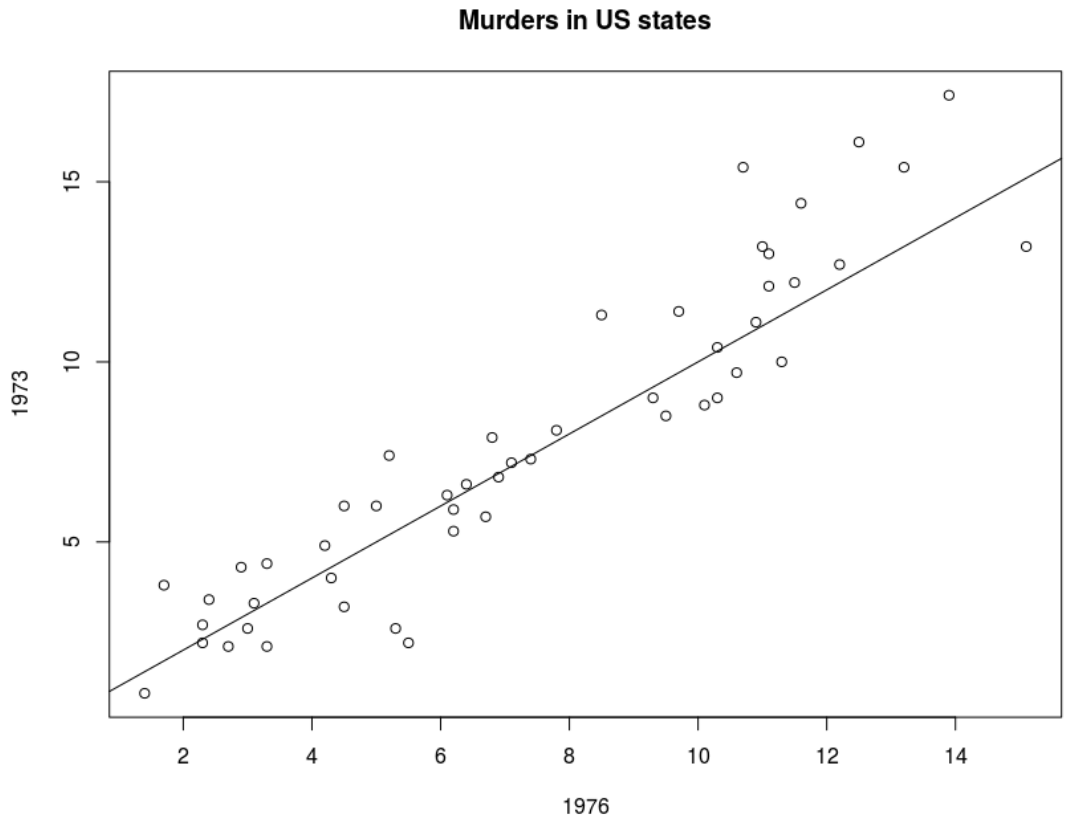
| | Row.names | Population | Area | Murder.x | Murder.y | Assault | UrbanPop | Rape |
|---|---|---|---|---|---|---|---|---|
| 1 | Alabama | 3615 | 50708 | 15.1 | 13.2 | 236 | 58 | 21.2 |
| 2 | Alaska | 365 | 566432 | 11.3 | 10.0 | 263 | 48 | 44.5 |
| 3 | Arizona | 2212 | 113417 | 7.8 | 8.1 | 294 | 80 | 31.0 |
| 4 | Arkansas | 2110 | 51945 | 10.1 | 8.8 | 190 | 50 | 19.5 |
| 5 | California | 21198 | 156361 | 10.3 | 9.0 | 276 | 91 | 40.6 |
| 6 | Colorado | 2541 | 103766 | 6.8 | 7.9 | 204 | 78 | 38.7 |

The *merge()* function expects to process a table x and a table y, as provided in its arguments (here, my.state and my. USArrests, respectively). Columns having identical names such as Murder are suffixed with x or y to keep track of their origin.

We can create a scatter plot to observe the variations. Points on the straight line (intercept 0, slope 1) indicate no changes between 1973 and 1976.

```
plot(my.US$Murder.x, my.US$Murder.y, main="Murders in US states", xlab="1976",  ylab="1973")

abline(0,1)
```



# 8 Advanced statistical functions

R includes functions for statistical tests and other advanced statistics. Many functions are already available in the R Stats package (accessible without additional installation). Other packages can provide additional functions. Below are presented a few common examples.

## 8.1 Correlation coefficients

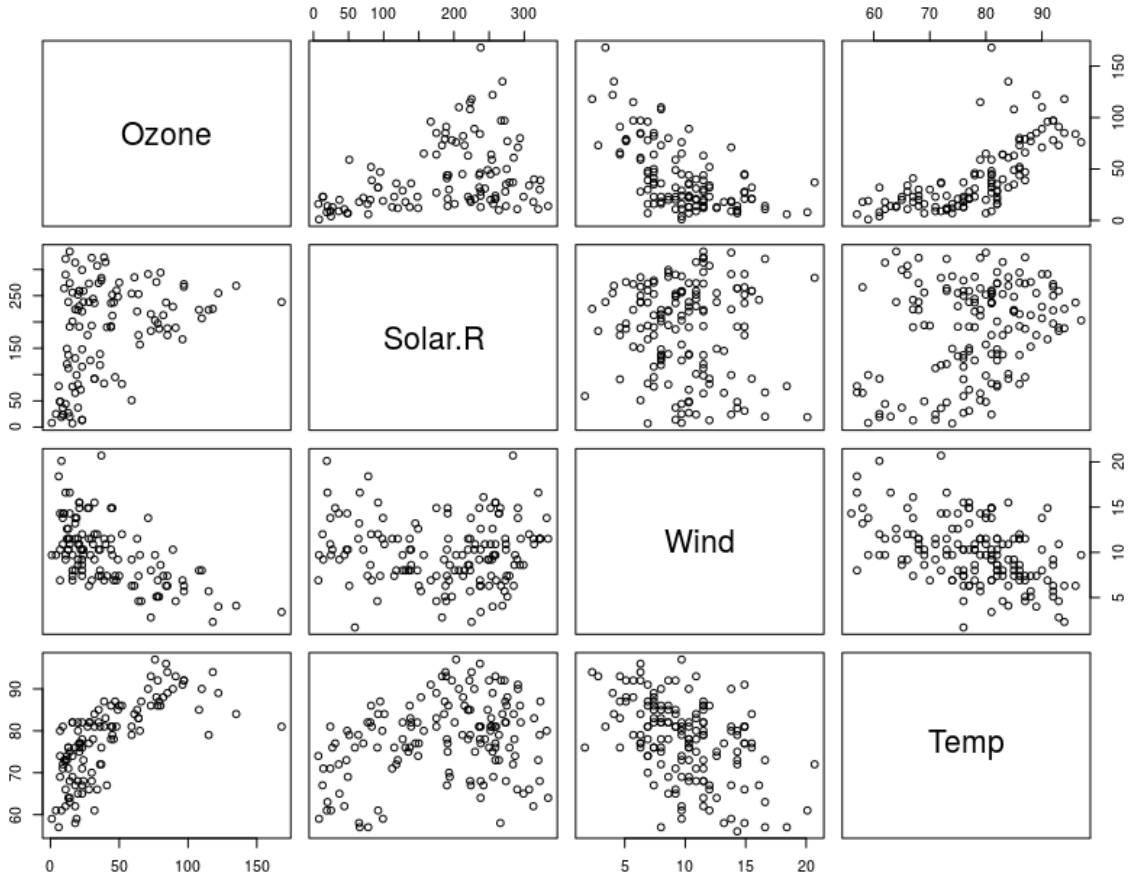
For this example, we use the airquality dataset

```
> head(airquality)  # NA values are present!

  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
5    NA      NA 14.3   56     5   5
6    28      NA 14.9   66     5   6
```

Scatter plots can show potential correlations between pairs of columns

```
pairs(airquality[,1:4])
```



We observe that Ozone and Temp are positively correlated but it is difficult to decide for Solar.R and Wind. We can use the *cor()* function to get correlation coefficients

```
# correlation between solar radiations (Solar.R) and average wind speed (Wind)

cor(airquality$Solar.R, airquality$Wind)

[1] NA
# Problem! We have to skip calculations involving NAs

cor(airquality$Solar.R, airquality$Wind, use="complete.obs")

[1] -0.05679167
# Ozone and Temp should be positively correlated

cor(airquality$Ozone, airquality$Temp, use="complete.obs")

[1] 0.6983603
```

## 8.2 Student's t-test

Student's t-test is "A two-sample location test of the null hypothesis such that the means of two populations are equal" (Wikipedia). Note that this test is not suitable for all types of data and examples below do not cover this potential issue.

Is the mean temperature significantly different in June or August?

```
> temp.june   <- airquality$Temp[airquality$Month==6]

> temp.august <- airquality$Temp[airquality$Month==8]

> t.test(temp.june, temp.august)


     Welch Two Sample t-test


data:  temp.june and temp.august
t = -2.8833, df = 58.926, p-value = 0.005486
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -8.246059 -1.489425
sample estimates:
mean of x mean of y
 79.10000  83.96774
```

The p-value being very small (<5%), we reject the null hypothesis that the mean temperatures are equal.

Is the mean temperature significantly different in July or August?

```
> temp.july <- airquality$Temp[airquality$Month==7]

> t.test(temp.july, temp.august)


     Welch Two Sample t-test


data:  temp.july and temp.august
t = -0.045624, df = 51.755, p-value = 0.9638
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -2.902417  2.773385
sample estimates:
mean of x mean of y
 83.90323  83.96774
```

The p-value not being very small (>5%), we can not reject the null hypothesis that the mean temperatures are equal.

# 9 Annexes

## 9.1 Installing Additional Packages

Many packages offering additional capabilities are freely available from the The Comprehensive R Archive Network (CRAN, https://cran.r-project.org/). However, many of these have to be installed manually after you have installed the basic R program.

For example, you may want to install the package tidyverse extending data analysis functions in R:

**From R Studio**

- Click on Tools menu and then Install Packages option
- Provide or search package name and click install
- If a CRAN mirror is asked, pick the closest to your physical location

If you are not administrator of the computer, R will install the package only for your user in the home directory (exact location depending on operating system).

Once you install the package, you still need to load it into R before use as follows:

```
library(tidyverse)
```

Then, the package is ready for you to use.

**From the console**

Use the following model to install a package from the console:

```
install.packages("packagename")
```

Installing packages may required a substantial amount of time, especially when installation of several packages is requested such as in the example below.

```
install.packages(c("tidyverse ", "ggvis"))
```