R for Absolute Beginners - Part 2/2

Summary Statistics and Graphics in R

Isabel Duarte

Jan 2016

1 Introduction

This mini hands-on tutorial serves as a basic guide on how to use R to:

1. Calculate the most commonly used **Summary Statistics** (or Descriptive Statistics) measures, namely, the maximum, minimum, mean, median, mode, standard deviation and variance;
2. Data visualization, i.e., using R to graphically explore datasets, using basic plotting functions.

This document will guide you through some of the most fundamental R concepts, granting you some **practical experience** in exploring and summarizing data.

RStudio will be used has the development platform for this tutorial.

This protocol is divided into **6** parts, each one identified by a title, predicted execution time (in parenthesis), a brief task description and the R commands to be executed. These will always be inside grey text boxes, with the font colored according to the R syntax highlighting.

Some very simple exercises are suggested throughout the text, which will allow you to verify your level of comprehension and possible difficulties. These will be highlighted in bold font.

The last section of this tutorial, presents an exercise that makes use of the concepts and commands apprehend in this workshop. This should be regarded as a self-evaluation guide to identify individual difficulties.

Finally, there are 9 examples of R code - my R Graphics Cook Book - included as the Annex section to generate random data and plot it. These are very useful to experiment with R and its diverse data generation and graphical potential.

Keep Calm and keep up the Good Work!

2 Tutorial

2.1 Read/Load data into R (30 min)

In this exercise, we will use the data from a dataset titled “esoph”, which contains the results from a case-control study developed in France, where the association between esophageal cancer and alcohol consumption was, for the first time, evaluated.

First download the data file from the following url: http://188.226.199.29/r4absoluteBeginners-2015/esoph_cancer.txt (http://188.226.199.29/r4absoluteBeginners-2015/esoph_cancer.txt)

This can be done directly in R:

```
dataset_url <- "http://188.226.199.29/r4absoluteBeginners-2015/esoph_cancer.txt"
download.file (dataset_url, "esoph_cancer.txt")
```

[Note: Remember that you can obtain help about the R functions, simply by writing in the shell *?function*, for example, **?getwd**). In RStudio, you can also press **F1** when the cursor is in the function name, and the respective help-page will open in the lateral panel].

To make sure that you are downloading the data file into the correct directory, find your *current working directory* with the following command:

```
getwd ()                          # get working directory
setwd ("path/to/your/directory")  # set working directory
```

Alternatively, in RStudio you can verify/change your working directory in the menu **Tools > Global Options** or in the **Files** browser panel (*More* button).

Once the download is complete (it should take only a few seconds) and placed in the correct folder, this file will now be visible in your working directory.

```
list.files () # list all files
dir ()        # list all files
# in RStudio you can also browse the files graphically, in the Files browser panel
```

We will start by looking at the data in the file:

```
esoph_data <- read.table ("esoph_cancer.txt")
head (esoph_data) # print only the first lines of esoph_data
```

```
##      V1        V2       V3     V4        V5
## 1 agegp     alcgp    tobgp ncases ncontrols
## 2 25-34 0-39g/day 0-9g/day      0        40
## 3 25-34 0-39g/day    10-19      0        10
## 4 25-34 0-39g/day    20-29      0         6
## 5 25-34 0-39g/day      30+      0         5
## 6 25-34     40-79 0-9g/day      0        27
```

The data are divided into 5 columns: V1, V2, V3, V4 and V5, but the first row of data is different from all the others… it seems to be the name of the columns, i.e., the header of the table. To correct this issue, we will import the data again, but this time we will let R know that the first line is the header:

```
esoph_data <- read.table ("esoph_cancer.txt", header=TRUE)
head (esoph_data, n=10) # print the first 10 lines of esoph_data
```

```
##    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
```

```
tail (esoph_data)       # print only the last lines of esoph_data
```

```
##    agegp  alcgp    tobgp ncases ncontrols
## 83   75+  40-79    20-29      0         3
## 84   75+  40-79      30+      1         1
## 85   75+ 80-119 0-9g/day      1         1
## 86   75+ 80-119    10-19      1         1
## 87   75+   120+ 0-9g/day      2         2
## 88   75+   120+    10-19      1         1
```

```
class (esoph_data)      # find the type of data structure
```

```
## [1] "data.frame"
```

```
colnames (esoph_data)   # find the names of the columns
```

```
## [1] "agegp"     "alcgp"     "tobgp"     "ncases"    "ncontrols"
```

Now the 5 columns are correctly annotated:
agegp = age group; alcgp = alcohol group; tobgp = tobacco group; ncases = number of cases; ncontrols = number of controls.

Note that in RStudio, data tables can be graphically inspected; by clicking its name in the environment panel, a new tab opens in the script panel, showing its first 1000 lines.

2.2 Explore the Data (30 min)

We can start by exploring the data by just “looking” at it, without any summarizing statistical measures or plotting.

Firstly, lets find the total number of rows and the total number of columns. Since the *esoph_data* object is a *data.frame*, we can ask R for its dimension:

```
dim (esoph_data) # find the dimension of the data.frame esoph_data
```

```
## [1] 88  5
```

We can also infer these values indirectly by looking at the **length** of any column, for example the *agegp* column, and the **length** of any row, for example, row number 3:

```
length (esoph_data$agegp) # find the length of column ‘agegp', or esoph_data[,1]
length (esoph_data[3,])   # find the length of row 3
```

All these values must match.
Another very useful command to output all these info, plus the internal data **str**ucture of the object:

```
str (esoph_data)
```

```
## 'data.frame':    88 obs. of  5 variables:
##  $ agegp    : Factor w/ 6 levels "25-34","35-44",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ alcgp    : Factor w/ 4 levels "0-39g/day","120+",..: 1 1 1 1 3 3 3 3 4 4 ...
##  $ tobgp    : Factor w/ 4 levels "0-9g/day","10-19",..: 1 2 3 4 1 2 3 4 1 2 ...
##  $ ncases   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ ncontrols: int  40 10 6 5 27 7 4 7 2 1 ...
```

2.3 Summary statistics of the data (60 min)

Mathematically known as Descriptive Statistics, these measures allow the quantitative description of the data, effectively summarizing the data samples. The most commonly used measures are divided in 2 big groups:
1. **Central tendency measures**: mean (or average), mode and median;
2. **Variability or dispersion measures**: standard deviation, variance, maximum and minimum.

In R, calculating these values is very simple with the *summary* command:

```
summary (esoph_data)
```

```
##    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   120+     :21   10-19   :24   1st Qu.: 0.000   1st Qu.: 3.00  
##  45-54:16   40-79    :23   20-29   :20   Median : 1.000   Median : 6.00  
##  55-64:16   80-119   :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
```

As we can see, the categorical variables (from Wikipedia (https://en.wikipedia.org/wiki/Categorical_variable): “variables that can take on one of a limited, and usually fixed, number of possible values, thus assigning each individual to a particular group or *category*”) were tallied by group: 6 different *age groups*, 4 *alcohol groups* and

4 *tobacco groups*, each one properly identified. The numerical variables were described and summarized by showing their Minimum, Maximum, Mean and Median (equivalent to the second quartile), as well as the first and third quartile values.

These measures can also be computed individually, for each numerical variable of interest:

```
mean (esoph_data$ncases)       # mean/average of the nr of cases
median (esoph_data$ncontrols)  # median of all controls
mean (esoph_data$ncases[1:15]) # mean nr of cases in ages 25-34 (rows 1 to 15)
median (esoph_data[16:30,4])   # median nr of cases (column 4) in ages 35-44 (rows 16
to 30)
min (esoph_data$ncases)        # minimum nr of cases observed
max (esoph_data[,5])           # maximum nr of controls (column 5) observed
```

Exercise 1
How many cases of cancer were observed, on average, in subjects older that 75 (75+)?
(Answer: 1.181818)

Subsetting contiguous data is easy, one must only provide the range of rows containing the values of interest. However…

? How can we select data that are not contiguous? Imagine, for example, that we want to calculate the mean number of cancer cases observed in individuals that smoke over 30 grams of tobacco per day? These rows are not grouped contiguously… For these cases, we can use the *subset ()* function and ask R to "calculate the mean of the subset of the number of cases, where the tobacco group equals 30+". (As with any programming language, learning to *read* the code helps to understand it.)

```
# mean/average of the nr of cases, in smokers' "30+" group
mean (subset(esoph_data$ncases, esoph_data$tobgp == "30+"))
```

```
## [1] 1.55
```

Note: Remember that, in R, the functions are evaluated from inside out of the parenthesis (as in all mathematical expressions). Accordingly, in this case, the *mean ()* is applied to the output of the *subset ()* function.

? What if we want to select all the data for the tobacco group labeled 30+? How can we do that? Using the function *which ()*, R can select only the data pertaining to a particular group:

```
# all data concerning tobacco group "30+"
tobgp30_data <- esoph_data[which(esoph_data[,"tobgp"] == "30+"),]
```

? And how can we obtain, from the tobgp30_data, only the values where the number of cases is greater than zero, and sum all of them? In this case, we can use *which ()* to select and *sum ()* to add the values selected:

```
# all data from tobacco group "30+" where the nr of cases is higher than zero
tobgp30_noZero <- tobgp30_data[which(tobgp30_data[, "ncases"] > 0 ),]
sum (tobgp30_noZero$ncases) # sum all cases of cancer
```

```
## [1] 31
```

Exercise 2
How many cancer cases were observed in subjects that drink over 120 gram of alcohol (120+) per day?
(Answer: 45)

Finally, to complete the data exploring and summary, we must calculate the standard deviation and the variance (which is the square of the standard deviation).

```
sd (esoph_data$ncases)  # standard deviation of the total nr of cases
```

```
## [1] 2.753169
```

```
var (esoph_data$ncases) # variance of the number of cases
```

```
## [1] 7.579937
```

Or if we want to check the results…

```
# var of the nr of cases is the square of its standard deviation
sd (esoph_data$ncases)^2
# sd of the nr of cases is the square root of its variance
sqrt (var (esoph_data$ncases))
```

As briefly mentioned before, this dataset presents 2 types of variables: **continuous numerical variables** (the number of cases and the number of controls), and **discrete categorical variables** (the age group, the tobacco smoking group and the alcohol drinking group). Sometimes it is hard to “categorize” continuous variables, i.e. to group them in specific intervals of interest, and name these groups (also called **levels**).
Accordingly, imagine that we were interested in classifying the number of cancer cases according to their occurrence: *frequent*, *intermediate* and *rare*. This type of variable *recoding* into factors is easily accomplished using the function *cut ()*.

```
# factorize the nr of cases in 3 levels, equally spaced, and add the new variable,
 # as the new column cat_ncases, to our dataset
esoph_data$cat_ncases <- cut (esoph_data$ncases,3,labels=c("rare","med","freq"))
summary (esoph_data)
```

```
##    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   120+     :21   10-19   :24   1st Qu.: 0.000   1st Qu.: 3.00
##  45-54:16   40-79    :23   20-29   :20   Median : 1.000   Median : 6.00
##  55-64:16   80-119   :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
##  cat_ncases
##  rare:79
##  med : 8
##  freq: 1
##
##
##
```

Note that now, there is one extra column in the dataset, *cat_ncases*, with 3 categories (or levels): *rare*, *med* and *freq*.

2.4 Graphical display of the data (120 min)

Summarizing the data is important to gain some insights regarding the data. However, visualizing the data is fundamental to extract the relevant information which will ultimately allow us to interpret our experiments.

By default, R allows the plotting of several, highly customizable graphics. However, there are many graphical packages developed by the community that greatly expand its plotting potential. Here, we will focus on only the most common plots, which are included in the base installation of R.

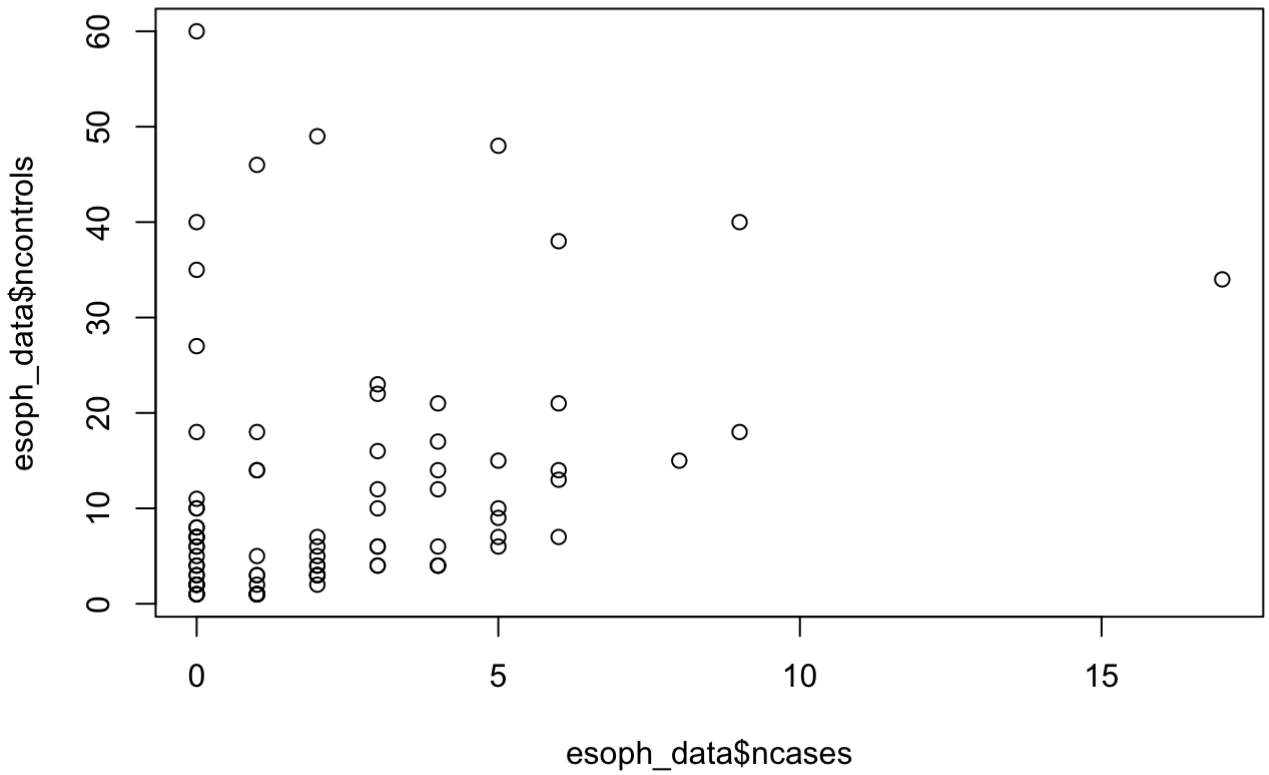
R graphics are created using a series of plotting commands of *high-level* and *low level*. *High level* commands create new plots, and the *low level* ones add information to an existing plot (the one currently opened by the graphics device).

Graphical parameters are customizable via the variable *par ()*, containing over **70** different customizable fields (for details: **?par**).

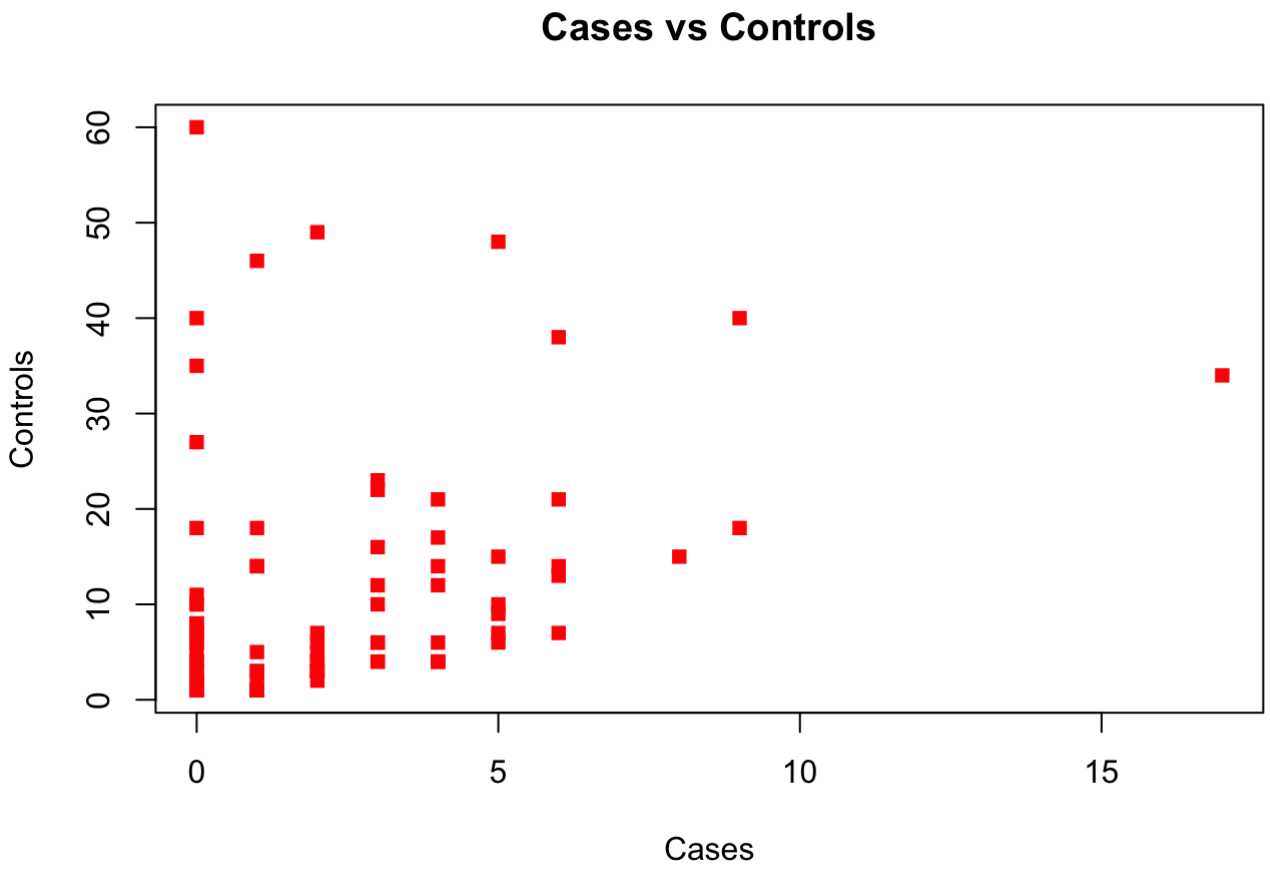
2.4.1 Scatterplots

These are xy dispersion plots using the *plot ()* function

```
# basic scatterplot
plot (esoph_data$ncases, esoph_data$ncontrols)
```



```
# put labels on axis, main title, change point type and color
plot(esoph_data$ncases, esoph_data$ncontrols, xlab="Cases", ylab="Controls",
     main="Cases vs Controls", pch=15, col="red")
```

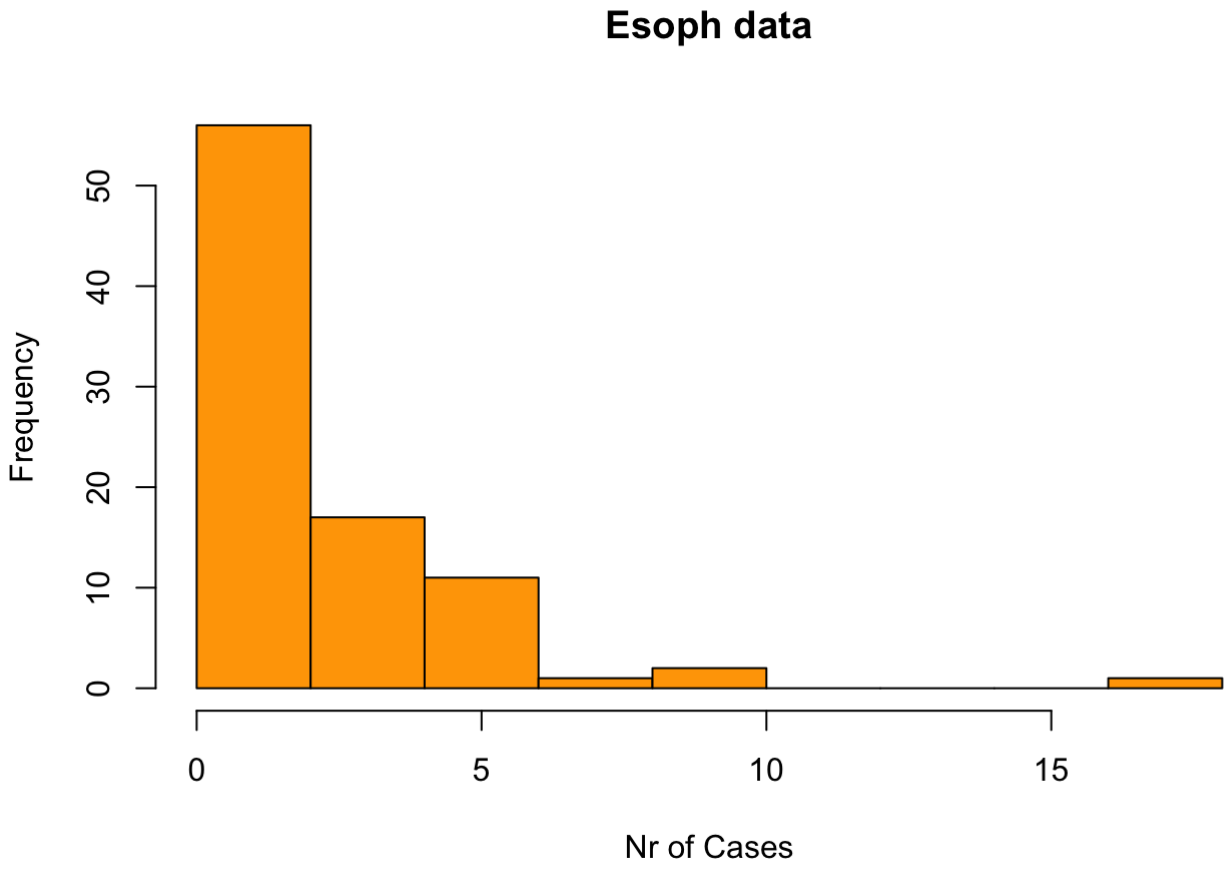


Note: The *demo("graphics")* command shows examples of available plots in R, together with the R code that can be used to generate it. The *colors ()* command shows the names of the available colors.

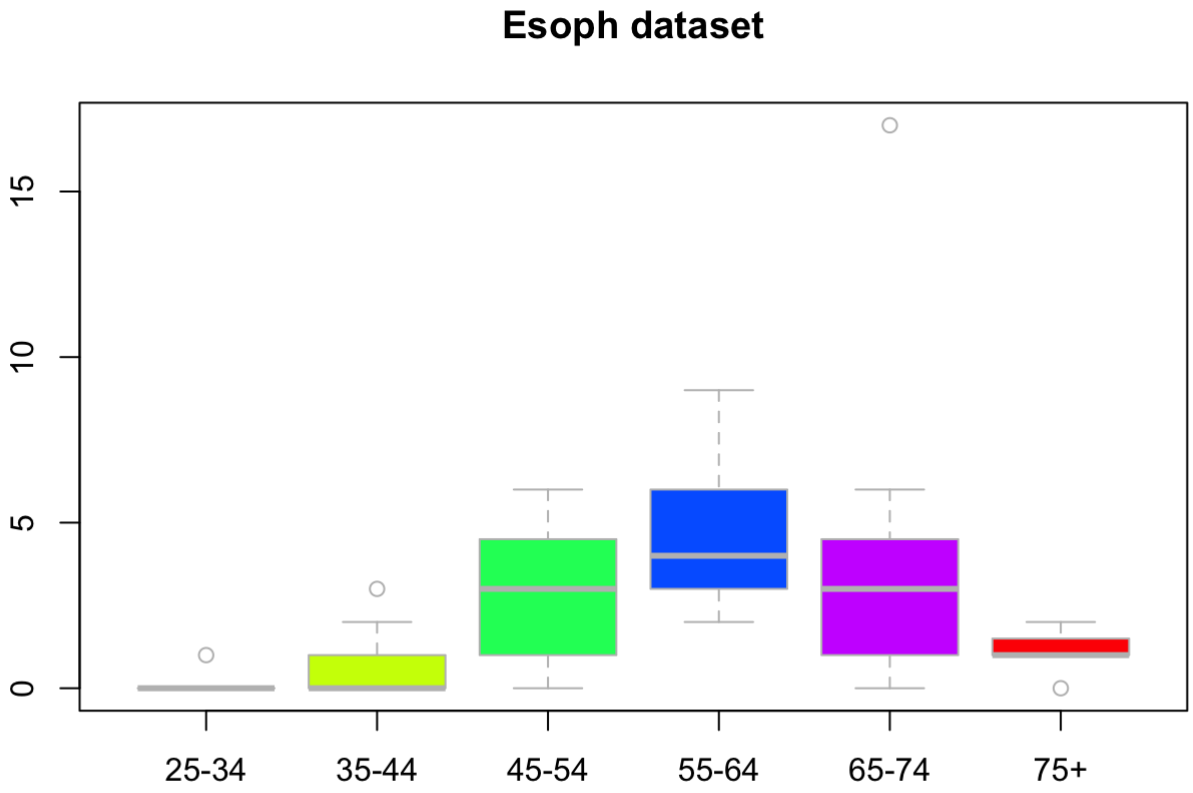
2.4.2 Histograms and Boxplots

The function *hist ()* shows the frequency (number of occurrences) of each observation; and the function *boxplot ()* shows the distribution of the occurrences in each category (agegp, alcgp and tobg).

```
# basic histogram, with labels, title and orange bars
hist(esoph_data$ncases, xlab="Nr of Cases", main="Esoph data", col="orange")
```



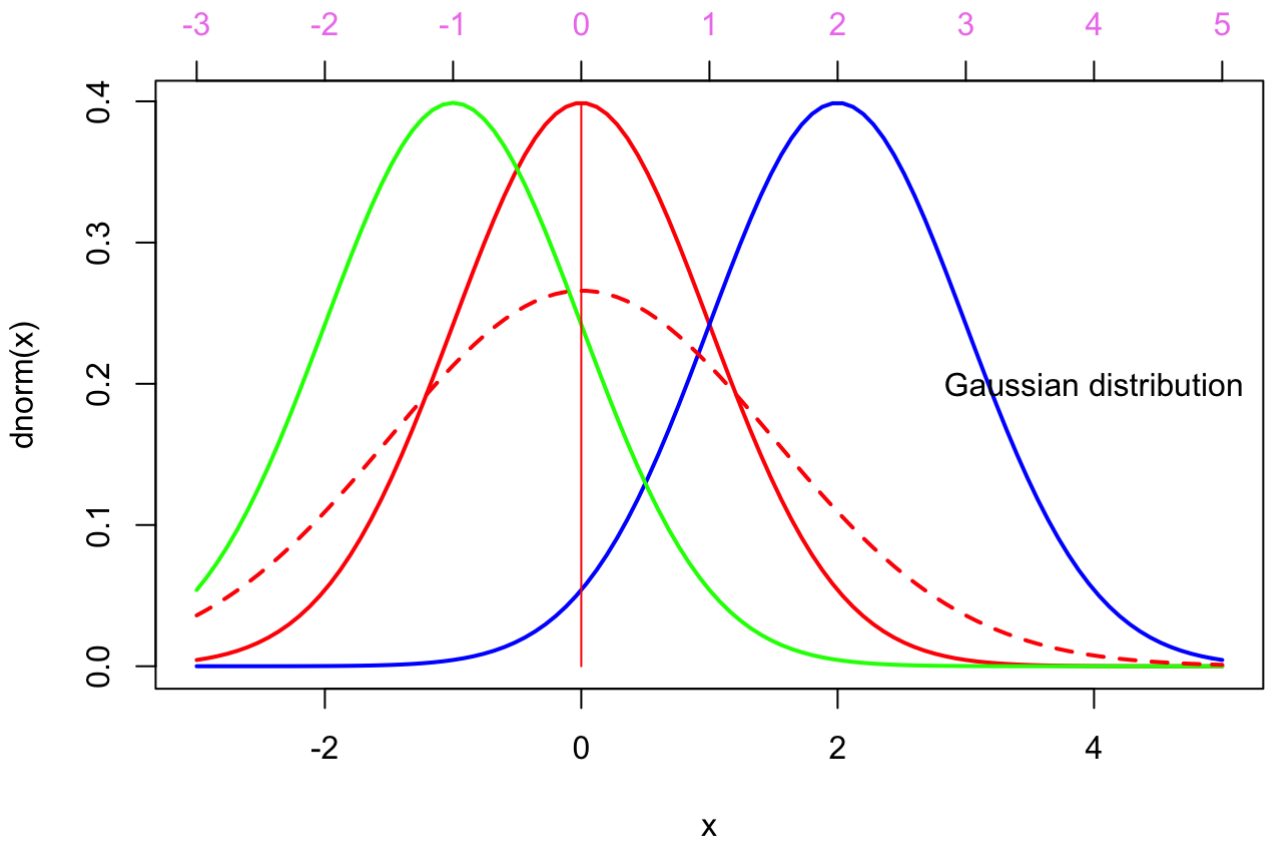

```
# basic boxplot of the cases per age group
boxplot (esoph_data$ncases ~ esoph_data$agegp,main="Esoph dataset",
         border="gray",lwd=1,col=rainbow(5))
```



2.4.3 Curves

These are continuous plots (usually of known statistical distributions, like the Gaussian (dnorm), gamma, beta, etc). Here we will see how to add lines and text to the plot (in specific locations/coordinates), as well as an extra axis on top with a different color.

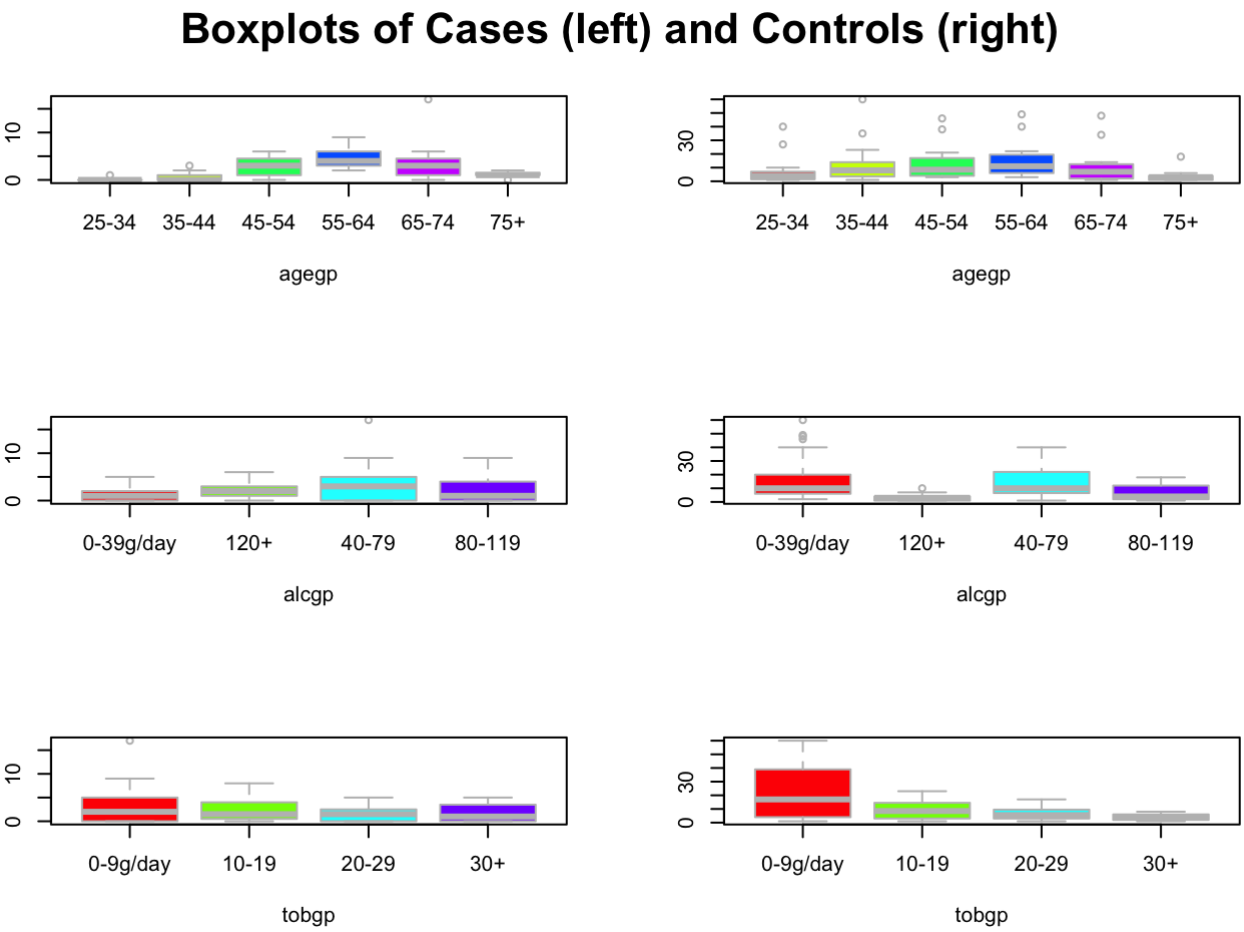
```
# multiple normal distribution curves, different mean and sd
curve(dnorm, from=-3, to=5, lwd=2, col="red")
curve(dnorm(x, mean=2), lwd=2, col="blue", add=TRUE)
curve(dnorm(x, mean=-1), lwd=2, col="green", add=TRUE)
curve(dnorm (x,mean=0,sd=1.5), lwd=2, lty=2, col="red", add=TRUE)
# add a vertical line at the mean of the distribution
lines(c(0,0), c(0,dnorm(0)), lty=1, col="red")
# add free text to the plot, in coordinates x=4, y=0.2
text(4,0.2,"Gaussian distribution")
# add extra axis, on top (side 3), from -3 to 5, with tick-marks from -3 to 5,
 # and colored violet
axis(3, -3:5, seq(-3,5), col.axis = "violet")
```



2.4.4 Multiple Graphs

To create a page with several plots located in side-by-side panels, we must use the function *par ()* with one of the following parameters: **par (mfrow=c(r,c))** or **par (mfcol=c(r,c))**. *mfrow* adds images per line, from right to left, and *mfcol* adds per column, from top to bottom.

```
# set the graphical display parameters to 3 rows and 2 columns
par(mfrow=c(3,2)) # mfrow adds plots per row, from left to right
# draw boxplots for cases and controls, per each group
boxplot (esoph_data$ncases ~ esoph_data$agegp, xlab="agegp",border="gray", lwd=1,
         col=rainbow(5))
boxplot (esoph_data$ncontrols ~ esoph_data$agegp,xlab="agegp", border="gray", lwd=1,
         col=rainbow(5))
boxplot (esoph_data$ncases ~ esoph_data$alcgp,border="gray",xlab="alcgp", lwd=1,
         col=rainbow(4))
boxplot (esoph_data$ncontrols ~ esoph_data$alcgp,border="gray", xlab="alcgp", lwd=1,
         col=rainbow(4))
boxplot (esoph_data$ncases ~ esoph_data$tobgp,border="gray",xlab="tobgp", lwd=1,
         col=rainbow(4))
boxplot (esoph_data$ncontrols ~ esoph_data$tobgp,border="gray", xlab="tobgp", lwd=1,
         col=rainbow(4))
# add a title outside of the plotting area
title("Boxplots of Cases (left) and Controls (right)",outer=TRUE, line=-2, cex.main=2
)
```



Once terminated the panel plots, we must revert the graphical parameters to its default values, so that we can go back to plotting one chart per page.

```
# reset the graphical display parameters to 1 row and 1 column
par (mfrow=c(1,1))
```

2.4.5 Export and Save plots

RStudio allows the visualization of the plots before exporting/saving them to an image file (i.e. *bitmap based*, like .jpeg and .png which become pixelated when zoomed in), or as .pdf (which is a vectorial format that can be zoomed and stretched to infinity, without loosing image quality). However, pdf files don't always export correctly, so they require some "trial and error" until one can get the "perfect" image.

Alternatively, we can export the plots directly in R. These will be saved in the current working directory (or any other folder you specify), with customized name and dimensions/resolution.

```
# Create a single pdf of figures, with one graph on each page
pdf ("esoph_boxplots.pdf", width=7, height=5) # Start graphics device
# draw boxplots for cases and controls, per each group
boxplot (esoph_data$ncases ~ esoph_data$agegp, main="Cases per Age group",xlab="ageg
p",
         border="gray", lwd=1, col=rainbow(5))
boxplot (esoph_data$ncontrols ~ esoph_data$agegp, xlab="agegp",main="Controls per Age
group",
         border="gray", lwd=1, col=rainbow(5))
boxplot (esoph_data$ncases ~ esoph_data$alcgp,border="gray",xlab="alcgp",
         main="Cases per Alcohol group",lwd=1, col=rainbow(4))
boxplot (esoph_data$ncontrols ~ esoph_data$alcgp,border="gray", xlab="alcgp",
         main="Controls per Alcohol group", lwd=1, col=rainbow(4))
boxplot (esoph_data$ncases ~ esoph_data$tobgp,border="gray", xlab="tobgp",
         main="Cases per Tobacco group", lwd=1, col=rainbow(4))
boxplot (esoph_data$ncontrols ~ esoph_data$tobgp,border="gray", xlab="tobgp",
         main="Controls per Tobacco group", lwd=1, col=rainbow(4))
dev.off () # close graphics device (stop writing to file)
```

3 Final Exercise (120 min)

R includes a package in its default base installation, named “The R Datasets Package”. This resource includes a diverse group of datasets, containing data from different fields: biology, physics, chemistry, economics, psychology, mathematics. These data are very useful to learn R. For more info about these datasets, run the following command: *library(help=datasets)*

The **Iris** dataset is particularly well known. It represents the measurements taken on these flowers in 3 iris species: *setosa*, *versicolor* and *virginica*. Explore and analyze this dataset, completing the following tasks:

1. Create a new folder titled “iris-R”. Navigate to this folder and create a new RStudio project with the same name. Make sure that the history file is always saved, and that the .RData file is automatically loaded when the project is opened and only saved when the user wants to.
2. Create a new document, of the type “R Script”, named “iris_analysis.R”. This file should be annotated, with appropriate comments for each command.
3. Summarize statistically the data, namely: the size of the sample, and for each variable its minimum, maximum, mean, median, standard deviation and variance. (Hint: Copy the dataset iris to a new object so that you can easily manipulate it; for example using the command *my.iris* <- *iris*).
4. Transform the variable **Petal.Length** into a discrete variable of 4 levels: “short” = min - 2 “med-short” = 2.1 - 3.5 “med-long” = 3.6 - 5 “long” = 5.1 - max
5. What is the frequency of “short” petals in iris “versicolor”? And of “long” petals in “setosa”? And “med-short” petals in “virginica”?
6. What is the variable that best correlates with the species of iris? For that, use a plot of your choosing that allows you to answer this question.
7. Change the following parameters of your previous plot: title, axis interval, type of point marker, text in the chart stating the chosen variable, line representing the perfect 100% correlation.

---

4 Correction of the Final Exercise (60 min)

The tutors will solve the final exercise together with the audience to pinpoint any major difficulties and discuss the alternative ways to answer the questions (yes… there is more than one correct way to solve the final exercise).

---

5 Annex: R Graphics Cook Book

Useful plotting examples to try at home.

5.1 Example 1: *plot ()*

```
# Create "x" and "y" vectors with 50 random numbers obtained from
 # a normal distribution with default parameters, i.e. mean=0 and sd=1
x <- rnorm(50); y <- rnorm(50)

# Create the object "group", with 50 random values obtained from
 # a binomial distribution, with one trial (size) and 0.5 probability
group <- rbinom(50, size=1, prob=.5)

# Basic Scatterplot
plot(x, y)
plot(x, y, xlab="X", ylab="Y", main="Y vs X", pch=15, col="red")
# Distinguish between two separate groups
plot(x, y, xlab="X", ylab="Y", main="Y vs X",pch=ifelse(group==1, 5, 19),
     col=ifelse(group==1, "red", "blue"))
 # pch is character used for the points; col is the color of the points

# The points argument can be:
# (1) two separate vectors where one vector is the x-coordinates and the other
# is the y-coordinates;
# (2) a two-column matrix;
# (3) a two-element list with x and y components.

plot(x, y, xlab="X", ylab="Y", main="Y vs X", type="n") # type n is NO PLOT
points(x[group==1], y[group==1], pch=5, col="red")
points(x[group==0], y[group==0], pch=19, col="blue")

plot(x, y, xlab="X", ylab="Y", main="Y vs X", type="n") # type n is NO PLOT
points(cbind(x,y)[group==1,], pch=5, col="red")         # cbind is column bind
points(cbind(x,y)[group==0,], pch=19, col="blue")
```

5.2 Example 2: Basic Line Graphs

```
# This exercise uses the x and y vectors from example 1
plot(sort(x), sort(y), type="l",lty=2, lwd=3, col="blue")
# sort the vectors, use lines instead of points; use line type 2, line width 3, color
blue

# Like points, the lines argument can be:
# (1) two separate vectors where one vector is the x-coordinate and the other is the
 # y-coordinate;
# (2) a two-column matrix;
# (3) a two-element list with x and y components.

plot(x, y, type="n")                # type n is NO PLOT
lines(sort(x), sort(y), type="b")   # type b is BOTH line and points
lines(cbind(sort(x),sort(y)), type="l", lty=1, col="blue")

# If there is only one component then the argument is plotted against its index
 # (same with plot and points)
plot(sort(x), type="n")
lines(sort(x), type="b", pch=8, col="red")
lines(sort(y), type="l", lty=6, col="blue")
```

5.3 Example 3: *hist ()* and *boxplot ()*

```
# Basic Histogram
hist(x, main="Histogram of X", col="deeppink4")

# Plot histogram along with a normal density
# Set freq=FALSE, so that the density histogram is plotted (area sums to 1)
hist(x, freq=FALSE, col="red", main="Histogram with Normal Curve")

# Uses the observed mean and standard deviation for plotting the normal curve
xpts <- seq(min(x), max(x), length=50)
ypts <- dnorm(xpts, mean=mean(x), sd=sd(x))
lines(xpts, ypts, lwd=3)

# Basic boxplot
boxplot(x, main="Boxplot of X", border="red", lwd=2)

# Side-by-Side Boxplots
boxplot(x~group, main="Boxplot of X by Group",names=c("Group 0", "Group 1"),
        border=c("red", "blue"), lwd=2)
```

5.4 Example 4: *curve ()*

The function *curve()* draws a curve corresponding to a given function. If the function is written within *curve()* it needs to be a function of x. If you want to use a multiple argument function, use x for the argument you wish to plot over.

```
# Plot a 5th order polynomial
curve(3*x^5-5*x^3+2*x, from=-1.25, to=1.25, lwd=2, col="blue")
# Plot the gamma density
curve(dgamma(x, shape=2, scale=1), from=0, to=7, lwd=2, col="red")
# Plot multiple curves, notice that the first curve determines the x-axis
curve(dnorm, from=-3, to=5, lwd=2, col="red")
curve(dnorm(x, mean=2), lwd=2, col="blue", add=TRUE)

# Add vertical lines at the mean of each function
lines(c(0, 0), c(0, dnorm(0)), lty=2, col="red")
lines(c(2, 2), c(0, dnorm(2, mean=2)), lty=2, col="blue")
```

5.5 Example 5: Multiple Graphs

```
# Figure with two plots side by side
par(mfrow=c(1,2))
plot(rnorm(100), main="Figure 1", pch=19, col="red")
plot(rnorm(100), main="Figure 2", pch=5, col="blue")

# Create layout
layout(matrix(c(1,1,2,3), 2, 2, byrow = TRUE), heights=c(.5,1))
layout.show(3) # View layout

# Create layout
layout(matrix(c(2,0,1,3),2,2, byrow=TRUE), widths=c(3,.5), heights=c(.5,3))
layout.show(3) # View layout

# Plot scatterplot and boxplots
x <- rnorm(100); y <- rnorm(100)

# Notice that the range of the scatterplot and boxplots have the same limits
par(mar=c(4,4,1,1),oma=c(0,0,1,0), font.axis=2, font.lab=2, cex.axis=1.5,cex.lab=1.5)
plot(x, y, xlim=c(-3,3), ylim=c(-3,3), xlab="X", ylab="Y", pch=17,col="darkgreen", ce
x=1.5)
box(lwd=2)
par(mar=c(0,4,0,1))
boxplot(x, horizontal=TRUE, ylim=c(-3,3), axes=FALSE, at=.75, border="red",lwd=3)
par(mar=c(4,0,1,0))
boxplot(y, ylim=c(-3,3), axes=FALSE, at=.75, border="blue", lwd=3)

# Add title in outer margin
title("Scatterplot and Boxplots of X and Y", outer=TRUE, line=-2, cex.main=2)
```

5.6 Example 6: Add *text ()* to a plot and to the margin with *mtext ()*

```
par(mfrow=c(1,2), oma=c(2,2,2,2)) # Add an outer margin to the figure
set.seed(123)   # "random" numbers with the same "seed" are always the same
x <- rnorm(10); y <- rnorm(10)

# Plot 1
plot(x, pch=19, col="red", main="Figure 1")
# Label each point with its index
text(1:10, x, label=1:10, pos=rep(c(4,2), c(2,8)), font=2, cex=1.5)
# Add fancy text to the plot region
text(1, 1, "This is Fancy Plot Text", family="HersheyScript", adj=0, cex=1.5)
# Add text to the margin
mtext("This is Margin Text for Figure 1", side=2, line=4)

# Plot 2
plot(x, pch=15, col="blue", main="Figure 2")
# Add plain text to the plot region
text(1, 1, "This is More Plot Text", family="mono", adj=0, cex=1)
# Add text to the margin
mtext("This is Margin Text for Figure 2", side=3, line=.5)
# Outer Margin, the \n can be include in character strings to add new lines
title("OUTER\nTITLE", outer=TRUE, line=-1)
mtext("This is Outer Margin Text", side=1, outer=TRUE, font=3)
```

5.7 Example 7: Add a *legend ()* to the plot

```
windows(width=9, height=6) # Fix window size
par(mfrow=c(1,2), oma=c(3,0,2,0)) # Add an outer margin to the figure
set.seed(789)
x1 <- rnorm(10); x2 <- rnorm(10, mean=2)
y1 <- rnorm(10); y2 <- rnorm(10, mean=2)

# PLOT 1, Use range to determine a plot region that is large enough for all the point
s
plot(range(x1,x2), range(y1,y2), main="Figure 1", type="n", xlab="X", ylab="Y")
points(x1, y1, col="red", pch=19) # Group 1
points(x2, y2, col="blue", pch=0) # Group 2
legend("topleft", c("Group 1","Group 2"), pch=c(19,0), col=c("red", "blue"),horiz=TRU
E, bty="n")
legend(locator(1), c("Group 1","Group 2"), pch=c(19,0), col=c("red", "blue"),title="L
egend")

# PLOT 2
plot(range(x1,x2), range(y1,y2), main="Figure 2", type="n", xlab="X", ylab="Y")
lines(sort(x1), sort(y1), col="red", type="o", pch=19) # Group 1
lines(sort(x2), sort(y2), col="blue", type="o", pch=0) # Group 2
legend(-2, 2.5, c("Group 1", "Group 2"), pch=c(19,0), col=c("red", "blue"),
       horiz=TRUE, bty="n", lty=1)

# Legend in figure margin
legend(1.5, -2.25, c("Group 1", "Group 2"), pch=c(19,0), col=c("red", "blue"),lty=1,
       bty="n", xpd=TRUE) # Legend in outer margin
legend(-5.25, -3, c("Group 1", "Group 2"), pch=c(19,0), col=c("red", "blue"),lty=1,
       horiz=TRUE, xpd=NA)
```

5.8 Example 8: Change the *axis ()*

```
# Plot with no axes
par(mar=c(5,5,5,5))
plot(1:10, axes=FALSE, ann=FALSE)

# Add an axis on side 2 (left)
axis(2)
# Add an axis on side 3 (top), specify tick mark location, and add labels
axis(3, at=seq(1,10,by=.5), labels=format(seq(1,10,by=.5), nsmall=3))
# Add an axis on side 4 (right), specify tick mark location and rotate labels
axis(4, at=1:10, las=2)
# Add axis on side 1 (bottom), with labels rotated 45 degrees
tck <- axis(1, labels=FALSE)
# Add box around the plot region
text(tck, par("usr")[3]-.5, labels=paste("Label", tck), srt=45, adj=1, xpd=TRUE) box(
)

mtext(paste("Side", 1:4), side=1:4, line=3.5, font=2) # Add axis labels
```

5.9 Example 9: Change *colors ()*

The function *colors()* returns a vector of built-in color names. *grep()* can be used to help find colors. Create a personal color palette using *palette()*. When the argument *col=number* is used, R uses the color indexed by that number.

```
# Create and use a custom color
burnt.orange <- rgb(red=204, green=85, blue=0, max=255)
plot(1:10, pch=15, col=burnt.orange, cex=3)
palette()   # Current palette
plot(1:10, pch=15, col=5, cex=3)
# Custom palette()
palette(c("red", "darkorange", "gold", "green3", "blue", "magenta3"))
plot(c(1,10), c(-3,3), type="n")

for(i in 1:length(palette()))
  {points(rnorm(10), col=i, pch=18, cex=1.5)}
palette("default") # Return to default, here "default" is a keyword
```