Corso Base R

Gianluca Mastrantonio, Stefano Moro, Alessio Pollice, Giovanna Jona lasinio

Contents

1 Introduzione

Questo file serve come introduzione ad **R** al corso di Probabilità e statistica e contiene le basi della programmazione in R e i comandi che maggiormente verranno usati nel corso.
R è disponibile per praticamente tutti i sistemi operativi, e si può scaricare gratuitamente da R-cran (Comprehensive R Archive Network) all'indirizzo

https://cran.r-project.org

è open-source e ad eccezione di pochi pacchetti che vengono installati direttamente con R, la maggior parte delle sue funzionalità viene implementata dai singoli ricercatori/scienziati, che le sviluppano per i loro progetti e poi le rendono pubbliche.

Una volta aperto R vi avrete il controllo delle console

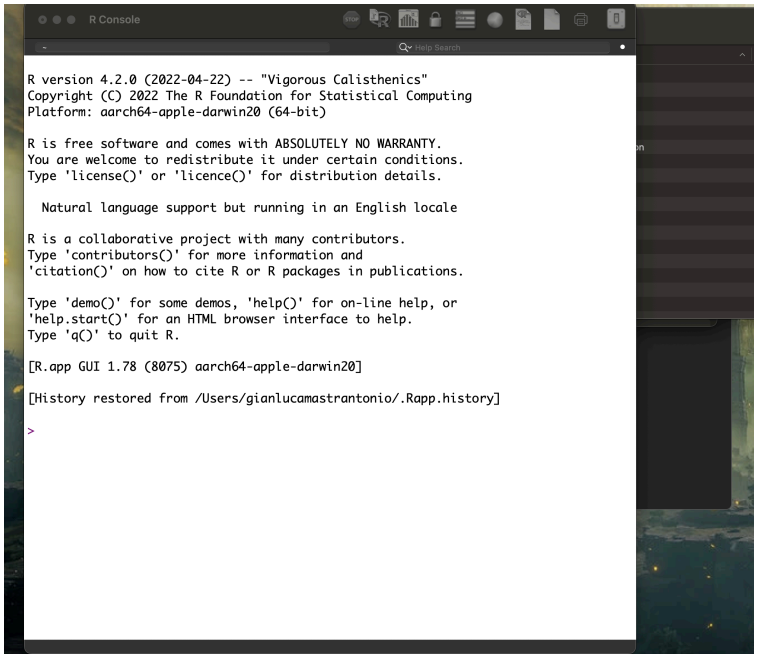


Figure 1: R-console on Mac

e il simbolo ">" nella console ci indica che R è pronto a ricevere comandi. Oltre alla console avremo bisogno anche dello **Script**, che è un file con estensione **.R** in cui scrivere e salvare i comandi (fate attenzione che nello script si salvano solo i comandi, ma non i risultati). Come si apre un nuovo script dipende dal sistema utilizzato, ma in generale è tra le opzioni di R, per esempio vedete la figura sotto

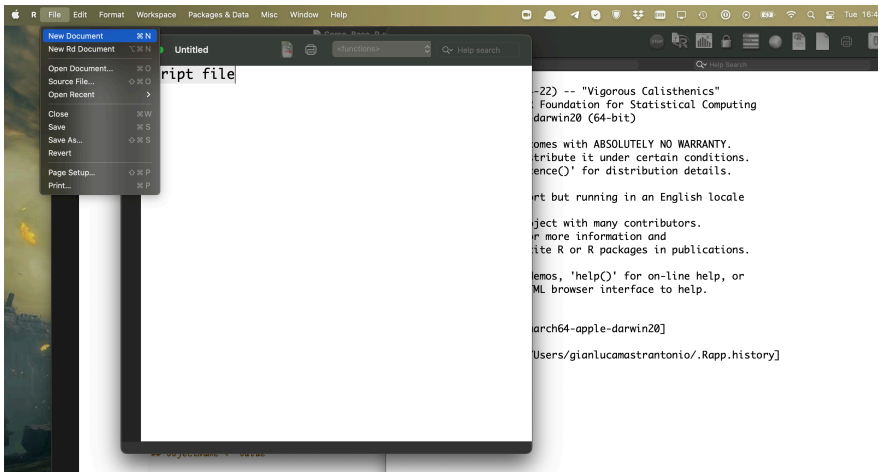


Figure 2: Creazione script on Mac

C'è anche un altro modo per utilizzare R, che è Rstudio. Rstudio è un IDE (integrated development environment) per R, anch'esso gratuito, disponibile su

https://posit.co/download/rstudio-desktop/

che permette di avere tutte le funzionalità di R in un'unica applicazione. Questo è molto comodo, soprattutto all'inizio, quando non si è ancora ferrati nell'utilizzo di R

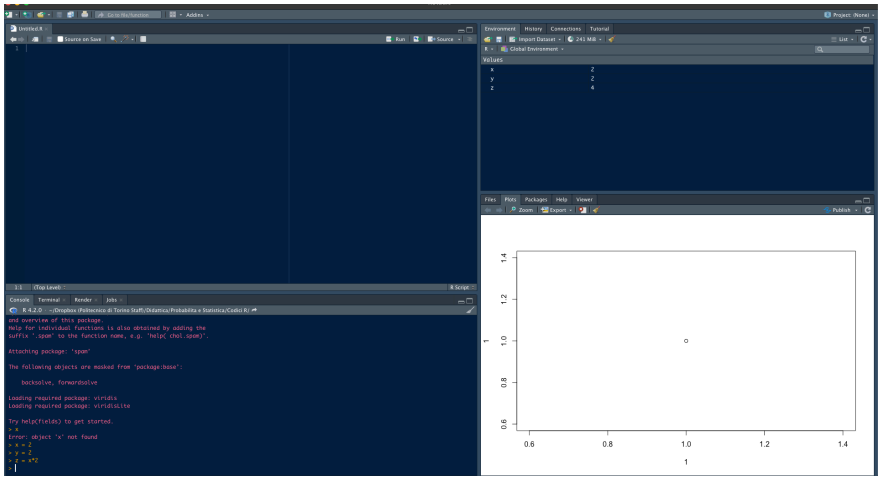


Figure 3: Rstudio

Lanciado RStudio/R si apriranno i pannelli di default:

- La console (a sinistra in basso)
- Lo script (a sinistra in alto)
- L'Environment e la History (a destra in alto)
- Files/plots/Packages/Help (a destra in basso)

In R ogni operazione su file avviene a partire dalla cartella di lavoro corrente. Per sapere in quale cartella si sta lavorando è sufficiente digitare nella Console il comando `getwd()`,

```
getwd()
```

```
## [1] "/Users/gianlucamastrantonio/Dropbox (Politecnico di Torino Staff)/Didattica/Probabilita e Statis
```

che restituisce l'indicazione del percorso relativo alla cartella di lavoro, dove risiedono tutti i dati su cui si lavora e dove vengono salvati tutti i risultati prodotti. Tutti gli **oggetti** (dati, risultati, funzioni, etc.) vengono conservati in un formato interno ad R ed in un unico file con estensione `.Rdata`, che può essere salvato con il comando

```
save.image("nome file.Rdata")
```

Qualora si volesse cambiare la cartella di lavoro è sufficiente utilizzare il comando

```
setwd("PAthToDirectory")
```

oppure spostarsi nella cartella Files nella parte in basso a destra dello schermo di RStudio, selezionare prima il percorso dall'icona ..., quindi "Set As Working Directory" dal tasto "More".

Nella Console un comando già eseguito puo essere richiamato tramite il tasto "freccia su". I tasti "freccia su" e "freccia giù" permettono di scorrere in avanti e indietro la "storia" dei comandi gia utilizzati. Nel codice R le righe di comando precedute da # sono considerate commenti e vengono ignorate da R.

2 Primi comandi

Andiamo alla *Console* dove potremo interagire direttamente con R. Facciamo un'assegnazione e poi ispezioniamo l'oggetto che abbiamo creato.

```
x <- 3 * 4
x
```

```
## [1] 12
```

l'assegnazione si può fare sia con "<-" che con "=". I due comandi sono molto simili, ma non fanno esattamente la stessa cosa, ma per un utente di basso/medio livello di R, i due comandi possno essere considerati identici. I nomi delle variabili sono **CaseSensitive**, non possono iniziare con un numero, e non possono avere caratteri speciali come :, , etc, ma possono avere il punto, anche se questo è fortemente sconsigliato. Se vogliamo salvare questo comando, per avere la storia di tutte le operazioni fatte, dobbiamo copiarlo nello script. Se avete un comando nello script di R, potete mandarlo direttamente nella consolo con il comando "Cmd+Return" sul mac (oppure "Ctrl+Enter" su Windows).

A differenza di altri linguaggi di programmazione, l'assegnazione copia i valori e non indica che due variabili rappresentano lo stesso oggetto. Per esempio

```
y <- 12
z <- x
```

stiamo dicendo che z ha lo stesso valore di x, ma non è x. Quindi, se cambio valore a z, x rimane inalterato

```
z <- 2
x
```

```
## [1] 12
```

```
z
```

```
## [1] 2
```

è possibile anche vedere tutto quello che è stato create in R usando il comando

```
ls()
```

```
## [1] "x" "y" "z"
```

oppure nella sezione in alto a detra di Rstudio.

Quando assegnate un valore ad un oggetto, R non lo mostra automaticamente sulla console. Potete forzarlo a farlo inserendo le parentesi oppure digitando sulla console il nome dell'oggetto.

```
peso_lb <- 121 # R non mostra niente
(peso_lb <- 121) # R vi mostra il valore assegato all'oggetto
```

```
## [1] 121
```

```
peso_lb # stessa cosa scrivendo il nome
```

```
## [1] 121
```

Adesso che R ha `peso_lb` in memoria possiamo usarlo per fare delle semplici operazioni aritmetiche. Ad esempio, potrebbe darsi che un vostro collega americano vi abbia passato il peso di un animale con il valore riportato in *pounds*. Ma noi siamo europei e usiamo il Sistema Interazionale e abbiamo bisogno di avere la misura in kg. Il peso in libbre è circa 2.2 volte il peso in kg.

```
peso_lb / 2.2
```

```
## [1] 55
```

Potremmo anche cambiare una variabile assegnandogli un nuovo valore

```
peso_lb <- 126.5
peso_lb / 2.2
```

```
## [1] 57.5
```

Oppure potrei assegnare il valore del peso in kg ad una nuova variabile, chiamandola magari `peso_kg`.

```
peso_kg <- peso_lb / 2.2
```

e poi cambiare il valore di `peso_lb` a 220.

```
peso_lb <- 220
```

Possiamo cancellare un oggetto con il comando

```
rm(x)
ls()
```

```
## [1] "peso_kg" "peso_lb" "y"       "z"
```

oppure cancellare tutto con

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

```
## character(0)
```

2.1 Operatori Logici

E' possibile porre ad R delle domande riguardo gli oggetti appena creati. Per farlo si possono usare questi simboli:

- `==` vuol dire "è uguale a"
- `!=` vuol dire "è diverso da"
- `<` vuol dire "è minore di"
- `>` vuol dire "è maggiore di"
- `<=` vuol dire "è minore o uguale a"
- `>=` vuol dire "è maggiore o uguale a"

Ricreiamo un oggetto y, visto che abbiamo cancellato tutto il contenuto di r

```
y = 10
```

e poi interroghiamo R

```
y == 2
```

```
## [1] FALSE
```

```
y <= 30
```

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

```
y !=5
```

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

2.2 Le funzioni di R & l'*help page*!

R ha un incredibile archivio di funzioni che vengono usate tutte con la stessa sintassi: nome della funzione con le parentesi intorno a ciò di cui la funzione ha bisogno per fare ciò che è stata creata per fare. Scrivendo una funzione di questo tipo si dice "stò richiamando una funzione". `verbo( nome = qualcosa, aggettivo = qualcos'altro, etc...)`. Proviamo ad usare la funzione `seq()`, che permette di creare delle sequenze regolari di numeri. Scrivete `se` e poi premete il tasto TAB. Come visto anche prima RStudio vi consiglierà tutte le funzioni disponibili che iniziano con `se`. Specificate che volete usare proprio `seq()` finendo di scrivere

il nome o scorrendo con le frecce.
Inseriamo gli *argument* `1, 10` e vediamo cosa esce.

```
seq(1,10)
```

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

In questo caso R ha già capito da solo a quali argument si riferiscono i valori che abbiamo inserito ma possiamo anche esplicitarli, magari aggiungendone degli altri (**Ricordatevi di guardare l'help della funzione che state usando**). Potete richiamarlo in qualsiasi momento mettendo un punto interrogativo prima del nome della funzione.

```
?seq()
```

Nell'*Help page* troverete moltissime informazioni:
- in alto a sinistra c'è il nome della funzione seguito da due graffe che racchiudono il nome della libreria che contiene quella funzione. Se fa parte dei comandi base di R troverete `{base}`.
- **Description** -> una breve descrizione di cosa fa la funzione.
- **Usage** -> vi mostra le impostazioni di default della funzione. Possono essere modificate cambiando gli arguments.
- **Arguments** -> vengono descritti in ordine, uno per uno, tutti gli argument che possono essere utilizzati nella funzione.
- **Details** -> di solito è presente una descrizione più dettagliata dei metodi che stanno dietro al comando. Esplorando tutte le possibili varianti.
- **Value** -> viene descritto l'output della funzione.
- **Reference** -> ogni funzione di R presente nel CRAN (il grande mondo di R) è stata controllata ed associata ad una pubblicazione scientifica.
- **See Also** -> vi consiglia altre voci dell'*Help* che sono in qualche modo legate alla funzione che state cercando.
- **Examples** -> vi fornisce alcuni esempi di come utilizzare la funzione.

C'è anche la possibilità di cercare una funzione di cui non ricordate perfettamente il nome, basta aggiungere un punto interrogativo. R riporterà tutte le funzioni che contengono quei caratteri.

```
??get
```

Ora riutilizziamo la funzione esplicitando gli argument.

```
seq(from = 1, to = 10, by = 2)
```

```
## [1] 1 3 5 7 9
```

Questa dicitura ci ricorda una cosa in più di come funziona R. In ogni funzione è possibile specificare l'argument nella forma `name = value`, ma nel caso non lo facessimo R tenterà di risolvere la questione autonomamente prendendo in cosiderazione la posizione che abbiamo indicato. Tuttavia esistono anche delle funzioni che non hanno la necessità di esplicitare degli argument, ad esempio:

```
date()
```

```
## [1] "Sun Dec 11 14:31:39 2022"
```

Se lanciamo il nome della funzione senza le parentesi nella console, R ci fa vedere il contenuto della funzione. Questo non è molto utili con le funzioni base, perchè sono compilate e quindi non si riesce a vedere il loro contenuto, ma sarà utile per funzioni presenti in pacchetti

```
date
```

```
## function ()
## .Internal(date())
## <bytecode: 0x106114d48>
## <environment: namespace:base>
```

```
seq
```

```
## function (...)
## UseMethod("seq")
## <bytecode: 0x10638dc48>
## <environment: namespace:base>
```

Qualche funzione per cui si può vedere il contenuto sono **data** o **sd**

```
sd
```

```
## function (x, na.rm = FALSE)
## sqrt(var(if (is.vector(x) || is.factor(x)) x else as.double(x),
##     na.rm = na.rm))
## <bytecode: 0x1582a2da8>
## <environment: namespace:stats>
```

```
data
```

```
## function (..., list = character(), package = NULL, lib.loc = NULL,
##     verbose = getOption("verbose"), envir = .GlobalEnv, overwrite = TRUE)
## {
##     fileExt <- function(x) {
##         db <- grepl("\\.[^.]+\\.(gz|bz2|xz)$", x)
##         ans <- sub(".*\\.", "", x)
##         ans[db] <- sub(".*\\.([^.]+\\.)(gz|bz2|xz)$", "\\1\\2",
##             x[db])
##         ans
##     }
##     my_read_table <- function(...) {
##         lcc <- Sys.getlocale("LC_COLLATE")
##         on.exit(Sys.setlocale("LC_COLLATE", lcc))
##         Sys.setlocale("LC_COLLATE", "C")
##         read.table(...)
##     }
##     stopifnot(is.character(list))
##     names <- c(as.character(substitute(list(...))[-1L]), list)
##     if (!is.null(package)) {
##         if (!is.character(package))
##             stop("'package' must be a character vector or NULL")
##     }
##     paths <- find.package(package, lib.loc, verbose = verbose)
##     if (is.null(lib.loc))
##         paths <- c(path.package(package, TRUE), if (!length(package)) getwd(),
##             paths)
##     paths <- unique(normalizePath(paths[file.exists(paths)]))
##     paths <- paths[dir.exists(file.path(paths, "data"))]
##     dataExts <- tools:::.make_file_exts("data")
##     if (length(names) == 0L) {
##         db <- matrix(character(), nrow = 0L, ncol = 4L)
##         for (path in paths) {
##             entries <- NULL
##             packageName <- if (file_test("-f", file.path(path,
##                 "DESCRIPTION")))
##                 basename(path)
##             else "."
```

```
##             if (file_test("-f", INDEX <- file.path(path, "Meta",
##                 "data.rds"))) {
##                 entries <- readRDS(INDEX)
##             }
##             else {
##                 dataDir <- file.path(path, "data")
##                 entries <- tools::list_files_with_type(dataDir,
##                   "data")
##                 if (length(entries)) {
##                   entries <- unique(tools::file_path_sans_ext(basename(entries)))
##                   entries <- cbind(entries, "")
##                 }
##             }
##             if (NROW(entries)) {
##                 if (is.matrix(entries) && ncol(entries) == 2L)
##                   db <- rbind(db, cbind(packageName, dirname(path),
##                     entries))
##                 else warning(gettextf("data index for package %s is invalid and will be ignored",
##                   sQuote(packageName)), domain = NA, call. = FALSE)
##             }
##         }
##         colnames(db) <- c("Package", "LibPath", "Item", "Title")
##         footer <- if (missing(package))
##             paste0("Use ", sQuote(paste("data(package =", ".packages(all.available = TRUE))")),
##                 "\n", "to list the data sets in all *available* packages.")
##         else NULL
##         y <- list(title = "Data sets", header = NULL, results = db,
##             footer = footer)
##         class(y) <- "packageIQR"
##         return(y)
##     }
##     paths <- file.path(paths, "data")
##     for (name in names) {
##         found <- FALSE
##         for (p in paths) {
##             tmp_env <- if (overwrite)
##                 envir
##             else new.env()
##             if (file_test("-f", file.path(p, "Rdata.rds"))) {
##                 rds <- readRDS(file.path(p, "Rdata.rds"))
##                 if (name %in% names(rds)) {
##                   found <- TRUE
##                   if (verbose)
##                     message(sprintf("name=%s:\t found in Rdata.rds",
##                       name), domain = NA)
##                   thispkg <- sub(".*/([^/]*)/data$", "\\1", p)
##                   thispkg <- sub("_.*$", "", thispkg)
##                   thispkg <- paste0("package:", thispkg)
##                   objs <- rds[[name]]
##                   lazyLoad(file.path(p, "Rdata"), envir = tmp_env,
##                     filter = function(x) x %in% objs)
##                   break
##                 }
##                 else if (verbose)
```

```
##                   message(sprintf("name=%s:\t NOT found in names() of Rdata.rds, i.e.,\n\t%s\n",
##                     name, paste(names(rds), collapse = ",")),
##                     domain = NA)
##             }
##             if (file_test("-f", file.path(p, "Rdata.zip"))) {
##                 warning("zipped data found for package ", sQuote(basename(dirname(p))),
##                   ".\nThat is defunct, so please re-install the package.",
##                   domain = NA)
##                 if (file_test("-f", fp <- file.path(p, "filelist")))
##                   files <- file.path(p, scan(fp, what = "", quiet = TRUE))
##                 else {
##                   warning(gettextf("file 'filelist' is missing for directory %s",
##                     sQuote(p)), domain = NA)
##                   next
##                 }
##             }
##             else {
##                 files <- list.files(p, full.names = TRUE)
##             }
##             files <- files[grep(name, files, fixed = TRUE)]
##             if (length(files) > 1L) {
##                 o <- match(fileExt(files), dataExts, nomatch = 100L)
##                 paths0 <- dirname(files)
##                 paths0 <- factor(paths0, levels = unique(paths0))
##                 files <- files[order(paths0, o)]
##             }
##             if (length(files)) {
##                 for (file in files) {
##                   if (verbose)
##                     message("name=", name, ":\t file= ...", .Platform$file.sep,
##                       basename(file), "::\t", appendLF = FALSE,
##                       domain = NA)
##                   ext <- fileExt(file)
##                   if (basename(file) != paste0(name, ".", ext))
##                     found <- FALSE
##                   else {
##                     found <- TRUE
##                     zfile <- file
##                     zipname <- file.path(dirname(file), "Rdata.zip")
##                     if (file.exists(zipname)) {
##                       Rdatadir <- tempfile("Rdata")
##                       dir.create(Rdatadir, showWarnings = FALSE)
##                       topic <- basename(file)
##                       rc <- .External(C_unzip, zipname, topic,
##                         Rdatadir, FALSE, TRUE, FALSE, FALSE)
##                       if (rc == 0L)
##                         zfile <- file.path(Rdatadir, topic)
##                     }
##                     if (zfile != file)
##                       on.exit(unlink(zfile))
##                     switch(ext, R = , r = {
##                       library("utils")
##                       sys.source(zfile, chdir = TRUE, envir = tmp_env)
##                     }, RData = , rdata = , rda = load(zfile,
```

```
##                       envir = tmp_env), TXT = , txt = , tab = ,
##                       tab.gz = , tab.bz2 = , tab.xz = , txt.gz = ,
##                       txt.bz2 = , txt.xz = assign(name, my_read_table(zfile,
##                         header = TRUE, as.is = FALSE), envir = tmp_env),
##                       CSV = , csv = , csv.gz = , csv.bz2 = ,
##                       csv.xz = assign(name, my_read_table(zfile,
##                         header = TRUE, sep = ";", as.is = FALSE),
##                         envir = tmp_env), found <- FALSE)
##                   }
##                   if (found)
##                     break
##                 }
##                 if (verbose)
##                   message(if (!found)
##                     "*NOT* ", "found", domain = NA)
##             }
##             if (found)
##                 break
##         }
##         if (!found) {
##             warning(gettextf("data set %s not found", sQuote(name)),
##                 domain = NA)
##         }
##         else if (!overwrite) {
##             for (o in ls(envir = tmp_env, all.names = TRUE)) {
##                 if (exists(o, envir = envir, inherits = FALSE))
##                   warning(gettextf("an object named %s already exists and will not be overwritten",
##                     sQuote(o)))
##                 else assign(o, get(o, envir = tmp_env, inherits = FALSE),
##                   envir = envir)
##             }
##             rm(tmp_env)
##         }
##     }
##     invisible(names)
## }
## <bytecode: 0x10704dad8>
## <environment: namespace:utils>
```

2.3 Scrivere Funzioni

Si possono anche creare delle funzioni su R con la sintassi
"nome funzione <- function(arguments)
{

code-block

return(output_variable)

}"

```
area_rettangolo <- function(lato1, lato2)
{
  area = lato1*lato2
  return(area)
}
```

e utilizzarla come ogni altra funzione di R

```
area_rettangolo(lato1 = 1, lato2 = 2)
```

```
## [1] 2
```

Fate solo attenzione a non sovrascrivere delle funzioni che già esistono in R, per esempio se io sovrascrivessi la funzione **sd**, che è presente nel pacchetto **stats**, con la seguente

```
sd <- function(var)
{
  return(var^0.5)
}
```

se volessi usarte la versione presente in **stats** devo indicarlo a R con "stats::sd"

```
sd
```

```
## function(var)
## {
##   return(var^0.5)
## }
```

```
stats::sd
```

```
## function (x, na.rm = FALSE)
## sqrt(var(if (is.vector(x) || is.factor(x)) x else as.double(x),
##     na.rm = na.rm))
## <bytecode: 0x1582a2da8>
## <environment: namespace:stats>
```

2.4 Pacchetti

Come detto precedentemente, non tutte ciò che ci servirà sarà presente in R,ma dovrà essere scaricato. Per esempio immaginiamo di dover usare le funzioni presenti in 'ggplot2', che è una libreria utile per fare grafici. Questa non è presente in R base e per utilizzarla dobbiamo richiamarla con il comando

```
library(ggplot2)
```

A me non da errore perchè ho già la libraria installata, ma nel caso vuoi non l'aveste dovete farlo con il comando

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

o dal comando "Install packages" che su trova sotto il menù "Tools" in Rstudio.

2.5 if e Cicli for e while

Come in tutti i linguaggi di programmazione, R ha i cicli for e while, che si implementano con **for (condition) { code-block } and while (condition) { code-block }**

```
for(i in 1:10)
{

}
```

i.e., in ciclo di lunghezza 10 sull'indice i, e

```
i = 0
while (i < 9)
{
```

```
  i = i + 1
}
```

un ciclo while che continua fintanto che i<9. Per esempio, possiamo

```
# calcolare i primi 12 valori della sequenza di Fibonacci
myx <- c(1,1)
for (i in 1:10)
{
  myx <- c(myx, myx[i] + myx[i+1])
  print(myx)
}
```

```
## [1] 1 1 2
## [1] 1 1 2 3
## [1] 1 1 2 3 5
## [1] 1 1 2 3 5 8
## [1]  1  1  2  3  5  8 13
## [1]  1  1  2  3  5  8 13 21
## [1]  1  1  2  3  5  8 13 21 34
##  [1]  1  1  2  3  5  8 13 21 34 55
##  [1]  1  1  2  3  5  8 13 21 34 55 89
##  [1]   1   1   2   3   5   8  13  21  34  55  89 144
```

Possiamo anche creare porzioni di codice che vengono lanciate solo se una condizione è verificata, utilizzando il comando **if**, che ha sintassi **if (condition) { code-block }**

```
x = 2
if(x == 2)
{
  print("x=2")
}
```

```
## [1] "x=2"
```

```
if(x != 2)
{
  print("x!=2")
}
```

I due **if** possono essere messi insieme in un unico **if/else**

```
x = 2
if(x == 2)
{
  print("x=2")
}else{
  print("x!=2")
}
```

```
## [1] "x=2"
```

2.6 Quantili, Densità, Cumulate e Simulazioni da distribuzioni note

Molto spesso avremo bisogno di ottenere il quantile, la cumulata, la densità/probabilità o simulare da distribuzioni note. I comandi per fare queste oparazioni sono nella forma **NomecomandoNomedistribuzione**. Per esempio

- **rnorm** simula un campione da una normale

- **dnorm** calcola la densità di una normale
- **pnorm** calcola la cumulata di una normale
- **qnorm** calcola il quantile di una normale

e per saperne i parametri, quardate l'help

```
?rnorm
?dnorm
?pnorm
?qnorm
```

Nella figura sottostante potete vedere a cosa corrispondono

```
## Loading required package: MASS
```

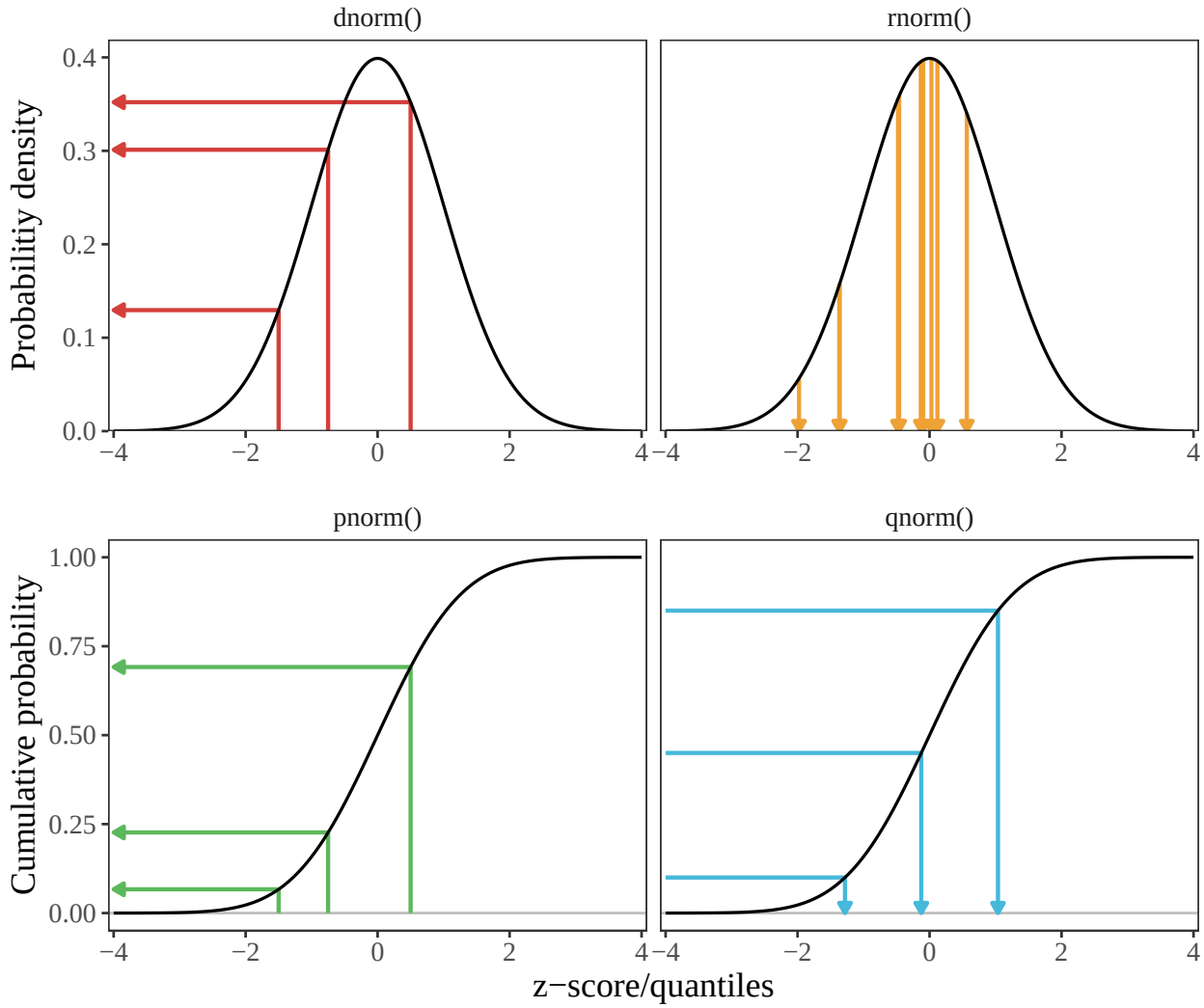
```
##
## Attaching package: 'patchwork'
```

```
## The following object is masked from 'package:MASS':
##
##     area
```

```
## -- Attaching packages ---------------------------------------------------------------------------
```

```
## v tibble  3.1.6     v dplyr   1.0.9
## v tidyr   1.2.0     v stringr 1.4.0
## v readr   2.1.2     v forcats 0.5.1
## v purrr   0.3.4
```

```
## -- Conflicts ------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## x dplyr::select() masks MASS::select()
```



per una distribuzione di Poisson possiamo utilizzare

- **rpois** simula un campione da una Poisson
- **dpois** calcola la probabilità di una Poisson (per uniformità usa sempre il suffisso **d**)
- **ppois** calcola la cumulata di una Poisson
- **qpois** calcola il quantile di una Poisson

e **dgamma** per la gamma, **dexp** per l'esponenziale etc...

3 Matrici, Vettore, Liste, DataFrame

Matrici, Vettore, Liste, DataFrame sono il modo di R per salvare gli oggetti in contenitori strutturati. Prima di introdurli discutiamo i diversi tipi di oggetti che si possono creare in R.

Naturalmente in R si possono avere oggetti che rappresentano numeri

```
x <- 2
```

e il **tipo** si può vedere con la seguente sintassi

```
str(x)
```

```
##  num 2
```

Ci sono motlo dipi diversi, ma i più comuni e utili sono

- `character`: stringhe
- `factor`: stringhe con un limitato numero di valori possibili
- `logic`: vero/false
- `numeric`: interi e numeri con virgola
- `complex`: numeri complessi

Vediamo come crearrli in R:

```
x1 <- "Ciao"
x2 <- as.factor("Ciao")
x3 <- TRUE # or FALSE
x4 <- 10
x5 <- 10 +2i

str(x1)
```

```
##  chr "Ciao"
```

```
str(x2)
```

```
##  Factor w/ 1 level "Ciao": 1
```

```
str(x3)
```

```
##  logi TRUE
```

```
str(x4)
```

```
##  num 10
```

```
str(x5)
```

```
##  cplx 10+2i
```

R accetta and T e F come TRUE e FALSE. La differenza tra caratteri e fattori sarà più chiara quando lavoreremo con vettori o matrici.

3.1 Vettori

I vettori si creano in R utilizzando la sintassi "c(element1, element2, ...)".

```
x <- c(1,2,3,4)
x
```

```
## [1] 1 2 3 4
```

```
y <- c(6,7,8)
c(y,x,y)
```

```
##  [1] 6 7 8 1 2 3 4 6 7 8
```

Naturalmente l'ultimo vettore viene perso, non essendo stato assegnato ad alcun oggetto.
Un vettore non può contenere elementi di tipi diversi, e se questo succede R cerca il tipo che possa rappresentarli tutti

```
x <- c(1,"2",3,4)
x
```

```
## [1] "1" "2" "3" "4"
```

Le operazioni aritmetiche `+`, `-`, `*`, `/`, `^` vengono effettuate sui vettori elemento per elemento:

```r
x<-c(1,2,3,4,5,6)
y<-c(10,11,12,100,-5,-6)
x+y
```

```
## [1]  11  13  15 104   0   0
```

```r
x*x
```

```
## [1]  1  4  9 16 25 36
```

```r
x*y
```

```
## [1]  10  22  36 400 -25 -36
```

```r
x/x
```

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

Se i vettori che compaiono nella stessa espressione non sono della stessa dimensione, il vettore risultante avrà la dimensione del vettore più numeroso. I vettori meno numerosi sono "riutilizzati" a partire dal primo elemento il numero di volte necessario a raggiungere la dimensione del vettore più numeroso:

```r
x<- c(10.4,5.6,3.1,6.4,21.7)
y<- c(x,0,x)
v<- 2*x+y+1
```

```
## Warning in 2 * x + y: longer object length is not a multiple of shorter object length
```

```r
v
```

```
##  [1] 32.2 17.8 10.3 20.2 66.1 21.8 22.6 12.8 16.9 50.8 43.5
```

il vettore `2*x` e ripetuto 2.2 volte, mentre lo scalare `1` e ripetuto 11 volte. Se un vettore viene ripetuto un numero di volte frazionaria, R vi da un warning.

Altre espressioni aritmetiche dall'ovvio significato sono: `log()`, `exp()`, `sin()`, `cos()`, `tan()`, `sqrt()` (radice quadrata). Le funzioni `max()` e `min()` selezionano rispettivamente il maggiore e il minor elemento di un vettore, mentre `length(x)`, `sum(x)`, `prod(x)`, `mean(x)` e `var(x)` restituiscono rispettivamente la numerosità, la somma, il prodotto, la media e la varianza corretta degli elementi di `x`.

Gli elementi di un vettore/matrice possono essere recuperati utilizzando le parentesi quadrate

```r
x <- c(30,29,28)
x[1]
```

```
## [1] 30
```

```r
x[2]
```

```
## [1] 29
```

```r
x[3]
```

```
## [1] 28
```

```r
x[c(2,1)]
```

```
## [1] 29 30
```

e in questo modo possiamo anche cambiarne i valori

```r
x
```

```
## [1] 30 29 28
```

```
x[1] <- 10
x
```

```
## [1] 10 29 28
```

e possiamo eliminare un elemento dal vettore

```
x[-1]
```

```
## [1] 29 28
```

```
x[-c(1,2)]
```

```
## [1] 28
```

Vediamo la differenza tra un carattere e una lista. Creiamo un vettore di caratteri e trasformiamoli in fattori

```
x <- c("A", "B", "C")
y <- as.factor(x)
str(x)
```

```
##  chr [1:3] "A" "B" "C"
```

```
str(y)
```

```
##  Factor w/ 3 levels "A","B","C": 1 2 3
```

Proviamo a modificare i valori di x

```
x[2] <- "A"
x[1] <- "D"
x
```

```
## [1] "D" "A" "C"
```

se provassimo a fare la stessa cosa su y, R ci da un errore

```
y[2] <- "A"
y[1] <- "D"
```

```
## Warning in `[<-.factor`(`*tmp*`, 1, value = "D"): invalid factor level, NA generated
```

```
y
```

```
## [1] <NA> A    C
## Levels: A B C
```

Possiamo cambiare il secondo valore di y perchè A fa parte dei valori ammessi (i **levels** che vediamo in str(y)), ma non possiamo aggiungere un valore D, visto che non fa parte dei **levels**, e come risultato otteniamo un **NA** (Not Available).

3.2 Matrici

Si possono trasformare i vettori in matrici tramite la funzione `matrix()`:

```
x<-c(1,2,3,4,5,6)
y<-c(10,11,12,100,-5,-6)
M<-matrix(c(x,y),ncol=2)
M
```

```
##      [,1] [,2]
## [1,]    1   10
## [2,]    2   11
```

```
## [3,]    3   12
## [4,]    4  100
## [5,]    5   -5
## [6,]    6   -6
```

```
matrix(c(17,2,3,4,5,1,2,3,4),ncol=2)
```

```
## Warning in matrix(c(17, 2, 3, 4, 5, 1, 2, 3, 4), ncol = 2): data length [9] is not a sub-multiple or
```

```
##      [,1] [,2]
## [1,]   17    1
## [2,]    2    2
## [3,]    3    3
## [4,]    4    4
## [5,]    5   17
```

```
matrix(c(17,2,3,4,5,1,2,3,4),ncol=3)
```

```
##      [,1] [,2] [,3]
## [1,]   17    4    2
## [2,]    2    5    3
## [3,]    3    1    4
```

Si noti come nel secondo esempio, poiché la numerosita del vettore è dispari e la matrice ha due colonne, l'ultimo elemento inserito da R nella matrice e nuovamente l'elemento iniziale del vettore.

Le dimensioni di una matrice sono restituite dalla funzione `dim()`:

```
dim(M)
```

```
## [1] 6 2
```

Anche per le matrici le operazioni aritmetiche vengono effettuate elemento per elemento:

```
M*M
```

```
##      [,1]  [,2]
## [1,]    1   100
## [2,]    4   121
## [3,]    9   144
## [4,]   16 10000
## [5,]   25    25
## [6,]   36    36
```

```
M/M
```

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

Il prodotto matriciale viene invece effettuato con l'operatore `%*%`:

```
M1<-matrix(c(1,2,2,1),ncol=2)
M2<-matrix(c(3,4,5,6,7,8),ncol=3)
M1%*%M2
```

```
##      [,1] [,2] [,3]
## [1,]   11   17   23
```

```
## [2,]   10   16   22
```

```
M2%*%M1
```

```
## Error in M2 %*% M1: non-conformable arguments
```

La funzione `t()` effettua l'operazione di trasposizione:

```
t(M2)%*%M1
```

```
##      [,1] [,2]
## [1,]   11   10
## [2,]   17   16
## [3,]   23   22
```

Si possono ottenere sottoinsiemi degli elementi di una matrice tramite la notazione `[i,j]`. Si noti che le notazioni `[i,]` e `[,j]` vengono utilizzate in R per indicare rispettivamente un'intera riga e un'intera colonna di una matrice:

```
M[3,2]
```

```
## [1] 12
```

```
M[1,]
```

```
## [1]  1 10
```

```
M[,2]
```

```
## [1]  10  11  12 100  -5  -6
```

```
M[c(1,2,5),]
```

```
##      [,1] [,2]
## [1,]    1   10
## [2,]    2   11
## [3,]    5   -5
```

Analogamente è possibile escludere elementi o sottoinsiemi di elementi di una matrice:

```
M[-c(1,2,5),]
```

```
##      [,1] [,2]
## [1,]    3   12
## [2,]    4  100
## [3,]    6   -6
```

```
M[-1,2]
```

```
## [1]  11  12 100  -5  -6
```

Gli elementi di matrici e vettori possono essere selezionati anche tramite gli operatori logici, indicati in R con i simboli `<`, `>`, `<=`, `>=`, `==` (=):

```
m<-c(M)
m
```

```
##  [1]   1   2   3   4   5   6  10  11  12 100  -5  -6
```

```
m[m<=10]
```

```
## [1]  1  2  3  4  5  6 10 -5 -6
```

```
y<-rep(0,times=12)
y
```

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

```
y[c(1,12)]<-1
y
```

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

```
m[y>0]
```

```
## [1]  1 -6
```

Matrici e vettori possono essere uniti per riga o per colonna rispettivamente con i comandi `rbind()` e `cbind()`:

```
X1<-matrix(1:10,ncol=2)
Y1<-matrix(1:20,ncol=4)
Z1<-cbind(X1,Y1)
Z1
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    6    1    6   11   16
## [2,]    2    7    2    7   12   17
## [3,]    3    8    3    8   13   18
## [4,]    4    9    4    9   14   19
## [5,]    5   10    5   10   15   20
```

```
rbind(X1,Y1)
```

```
## Error in rbind(X1, Y1): number of columns of matrices must match (see arg 2)
```

```
rbind(X1,Y1[,3:4])
```

```
##       [,1] [,2]
##  [1,]    1    6
##  [2,]    2    7
##  [3,]    3    8
##  [4,]    4    9
##  [5,]    5   10
##  [6,]   11   16
##  [7,]   12   17
##  [8,]   13   18
##  [9,]   14   19
## [10,]   15   20
```

3.3 Liste

Una lista in R equivale ad un insieme di oggetti (tra i quali possono essere incluse delle altre liste). Una lista viene definita dal comando `list()`:

```
Alessio<-list(lavoro="prof",sta.civ="cel",eta=52, misure=c(185,80,44))
```

Gli elementi di una lista possono essere recuperati con doppie aprentesi quadrate

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

```
## [1] "prof"
```

```
Alessio[[2]]
```

```
## [1] "cel"
```

```
Alessio[[3]]
```

```
## [1] 52
```

```
Alessio[[4]]
```

```
## [1] 185  80  44
```

o con il nome preceduto dal dollaro

```
Alessio$eta
```

```
## [1] 52
```

```
Alessio$misure
```

```
## [1] 185  80  44
```

```
Alessio$misure[2]
```

```
## [1] 80
```

3.4 Data frames

Un data frame è il tipo di oggetto che si usa abitualmente in R per immagazzinare una matrice di dati. La struttura di un data frame è quella di una lista di vettori tutti della stessa lunghezza ed eventualmente di diverso tipo (elementi numerici, stringhe di caratteri, logici). La distribuzione di R contiene diversi data frame interni utilizzabili. Per richiamarli si usa il comando `data()`:

```
data(iris)
iris
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
## 1            5.1         3.5          1.4         0.2     setosa
## 2            4.9         3.0          1.4         0.2     setosa
## 3            4.7         3.2          1.3         0.2     setosa
## 4            4.6         3.1          1.5         0.2     setosa
## 5            5.0         3.6          1.4         0.2     setosa
## 6            5.4         3.9          1.7         0.4     setosa
## 7            4.6         3.4          1.4         0.3     setosa
## 8            5.0         3.4          1.5         0.2     setosa
## 9            4.4         2.9          1.4         0.2     setosa
## 10           4.9         3.1          1.5         0.1     setosa
## 11           5.4         3.7          1.5         0.2     setosa
## 12           4.8         3.4          1.6         0.2     setosa
## 13           4.8         3.0          1.4         0.1     setosa
## 14           4.3         3.0          1.1         0.1     setosa
## 15           5.8         4.0          1.2         0.2     setosa
## 16           5.7         4.4          1.5         0.4     setosa
## 17           5.4         3.9          1.3         0.4     setosa
## 18           5.1         3.5          1.4         0.3     setosa
## 19           5.7         3.8          1.7         0.3     setosa
## 20           5.1         3.8          1.5         0.3     setosa
## 21           5.4         3.4          1.7         0.2     setosa
## 22           5.1         3.7          1.5         0.4     setosa
## 23           4.6         3.6          1.0         0.2     setosa
## 24           5.1         3.3          1.7         0.5     setosa
## 25           4.8         3.4          1.9         0.2     setosa
```

```
## 26          5.0         3.0          1.6         0.2     setosa
## 27          5.0         3.4          1.6         0.4     setosa
## 28          5.2         3.5          1.5         0.2     setosa
## 29          5.2         3.4          1.4         0.2     setosa
## 30          4.7         3.2          1.6         0.2     setosa
## 31          4.8         3.1          1.6         0.2     setosa
## 32          5.4         3.4          1.5         0.4     setosa
## 33          5.2         4.1          1.5         0.1     setosa
## 34          5.5         4.2          1.4         0.2     setosa
## 35          4.9         3.1          1.5         0.2     setosa
## 36          5.0         3.2          1.2         0.2     setosa
## 37          5.5         3.5          1.3         0.2     setosa
## 38          4.9         3.6          1.4         0.1     setosa
## 39          4.4         3.0          1.3         0.2     setosa
## 40          5.1         3.4          1.5         0.2     setosa
## 41          5.0         3.5          1.3         0.3     setosa
## 42          4.5         2.3          1.3         0.3     setosa
## 43          4.4         3.2          1.3         0.2     setosa
## 44          5.0         3.5          1.6         0.6     setosa
## 45          5.1         3.8          1.9         0.4     setosa
## 46          4.8         3.0          1.4         0.3     setosa
## 47          5.1         3.8          1.6         0.2     setosa
## 48          4.6         3.2          1.4         0.2     setosa
## 49          5.3         3.7          1.5         0.2     setosa
## 50          5.0         3.3          1.4         0.2     setosa
## 51          7.0         3.2          4.7         1.4 versicolor
## 52          6.4         3.2          4.5         1.5 versicolor
## 53          6.9         3.1          4.9         1.5 versicolor
## 54          5.5         2.3          4.0         1.3 versicolor
## 55          6.5         2.8          4.6         1.5 versicolor
## 56          5.7         2.8          4.5         1.3 versicolor
## 57          6.3         3.3          4.7         1.6 versicolor
## 58          4.9         2.4          3.3         1.0 versicolor
## 59          6.6         2.9          4.6         1.3 versicolor
## 60          5.2         2.7          3.9         1.4 versicolor
## 61          5.0         2.0          3.5         1.0 versicolor
## 62          5.9         3.0          4.2         1.5 versicolor
## 63          6.0         2.2          4.0         1.0 versicolor
## 64          6.1         2.9          4.7         1.4 versicolor
## 65          5.6         2.9          3.6         1.3 versicolor
## 66          6.7         3.1          4.4         1.4 versicolor
## 67          5.6         3.0          4.5         1.5 versicolor
## 68          5.8         2.7          4.1         1.0 versicolor
## 69          6.2         2.2          4.5         1.5 versicolor
## 70          5.6         2.5          3.9         1.1 versicolor
## 71          5.9         3.2          4.8         1.8 versicolor
## 72          6.1         2.8          4.0         1.3 versicolor
## 73          6.3         2.5          4.9         1.5 versicolor
## 74          6.1         2.8          4.7         1.2 versicolor
## 75          6.4         2.9          4.3         1.3 versicolor
## 76          6.6         3.0          4.4         1.4 versicolor
## 77          6.8         2.8          4.8         1.4 versicolor
## 78          6.7         3.0          5.0         1.7 versicolor
## 79          6.0         2.9          4.5         1.5 versicolor
```

```
## 80          5.7         2.6          3.5         1.0 versicolor
## 81          5.5         2.4          3.8         1.1 versicolor
## 82          5.5         2.4          3.7         1.0 versicolor
## 83          5.8         2.7          3.9         1.2 versicolor
## 84          6.0         2.7          5.1         1.6 versicolor
## 85          5.4         3.0          4.5         1.5 versicolor
## 86          6.0         3.4          4.5         1.6 versicolor
## 87          6.7         3.1          4.7         1.5 versicolor
## 88          6.3         2.3          4.4         1.3 versicolor
## 89          5.6         3.0          4.1         1.3 versicolor
## 90          5.5         2.5          4.0         1.3 versicolor
## 91          5.5         2.6          4.4         1.2 versicolor
## 92          6.1         3.0          4.6         1.4 versicolor
## 93          5.8         2.6          4.0         1.2 versicolor
## 94          5.0         2.3          3.3         1.0 versicolor
## 95          5.6         2.7          4.2         1.3 versicolor
## 96          5.7         3.0          4.2         1.2 versicolor
## 97          5.7         2.9          4.2         1.3 versicolor
## 98          6.2         2.9          4.3         1.3 versicolor
## 99          5.1         2.5          3.0         1.1 versicolor
## 100         5.7         2.8          4.1         1.3 versicolor
## 101         6.3         3.3          6.0         2.5  virginica
## 102         5.8         2.7          5.1         1.9  virginica
## 103         7.1         3.0          5.9         2.1  virginica
## 104         6.3         2.9          5.6         1.8  virginica
## 105         6.5         3.0          5.8         2.2  virginica
## 106         7.6         3.0          6.6         2.1  virginica
## 107         4.9         2.5          4.5         1.7  virginica
## 108         7.3         2.9          6.3         1.8  virginica
## 109         6.7         2.5          5.8         1.8  virginica
## 110         7.2         3.6          6.1         2.5  virginica
## 111         6.5         3.2          5.1         2.0  virginica
## 112         6.4         2.7          5.3         1.9  virginica
## 113         6.8         3.0          5.5         2.1  virginica
## 114         5.7         2.5          5.0         2.0  virginica
## 115         5.8         2.8          5.1         2.4  virginica
## 116         6.4         3.2          5.3         2.3  virginica
## 117         6.5         3.0          5.5         1.8  virginica
## 118         7.7         3.8          6.7         2.2  virginica
## 119         7.7         2.6          6.9         2.3  virginica
## 120         6.0         2.2          5.0         1.5  virginica
## 121         6.9         3.2          5.7         2.3  virginica
## 122         5.6         2.8          4.9         2.0  virginica
## 123         7.7         2.8          6.7         2.0  virginica
## 124         6.3         2.7          4.9         1.8  virginica
## 125         6.7         3.3          5.7         2.1  virginica
## 126         7.2         3.2          6.0         1.8  virginica
## 127         6.2         2.8          4.8         1.8  virginica
## 128         6.1         3.0          4.9         1.8  virginica
## 129         6.4         2.8          5.6         2.1  virginica
## 130         7.2         3.0          5.8         1.6  virginica
## 131         7.4         2.8          6.1         1.9  virginica
## 132         7.9         3.8          6.4         2.0  virginica
## 133         6.4         2.8          5.6         2.2  virginica
```

```
## 134          6.3         2.8          5.1         1.5  virginica
## 135          6.1         2.6          5.6         1.4  virginica
## 136          7.7         3.0          6.1         2.3  virginica
## 137          6.3         3.4          5.6         2.4  virginica
## 138          6.4         3.1          5.5         1.8  virginica
## 139          6.0         3.0          4.8         1.8  virginica
## 140          6.9         3.1          5.4         2.1  virginica
## 141          6.7         3.1          5.6         2.4  virginica
## 142          6.9         3.1          5.1         2.3  virginica
## 143          5.8         2.7          5.1         1.9  virginica
## 144          6.8         3.2          5.9         2.3  virginica
## 145          6.7         3.3          5.7         2.5  virginica
## 146          6.7         3.0          5.2         2.3  virginica
## 147          6.3         2.5          5.0         1.9  virginica
## 148          6.5         3.0          5.2         2.0  virginica
## 149          6.2         3.4          5.4         2.3  virginica
## 150          5.9         3.0          5.1         1.8  virginica
```

Il data frame iris è composto da cinque colonne di 150 elementi ciascuna. Le prime quattro variabili sono numeriche, mentre la quinta è qualitativa (rappresentata da stringhe di caratteri). Per ottenere la lista di tutti i dataset disponibili nel package di base di R è sufficiente utilizzare il comando `data()` senza argomenti. Inoltre a ciascun nome corrisponde una voce dell'help con la descrizione del dataset (provare `help(iris)`).

Le cinque colonne del data frame iris hanno altrettanti nomi visualizzabili con il comando `names()`:

```
names(iris)
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
```

Tutte le operazioni introdotte per le matrici possono essere utilizzate con i data frame, ad esempio:

```
rbind(iris[1:5,c(2,4)],iris[21:25,c(2,4)])
```

```
##    Sepal.Width Petal.Width
## 1          3.5         0.2
## 2          3.0         0.2
## 3          3.2         0.2
## 4          3.1         0.2
## 5          3.6         0.2
## 21         3.4         0.2
## 22         3.7         0.4
## 23         3.6         0.2
## 24         3.3         0.5
## 25         3.4         0.2
```

Alcuni comandi utili quando si lavora con i dataframe sono

```
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
tail(iris)
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
## 145          6.7         3.3          5.7         2.5 virginica
## 146          6.7         3.0          5.2         2.3 virginica
## 147          6.3         2.5          5.0         1.9 virginica
## 148          6.5         3.0          5.2         2.0 virginica
## 149          6.2         3.4          5.4         2.3 virginica
## 150          5.9         3.0          5.1         1.8 virginica
```

```
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
summary(iris)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   setosa    :50
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300   versicolor:50
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300   virginica :50
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
```

Tra questi il più importante di tutti è 'summary()' che da un'idea di base di cosa c'è nel dataframe, con statistiche di base e un'idea del tipo di variabile.

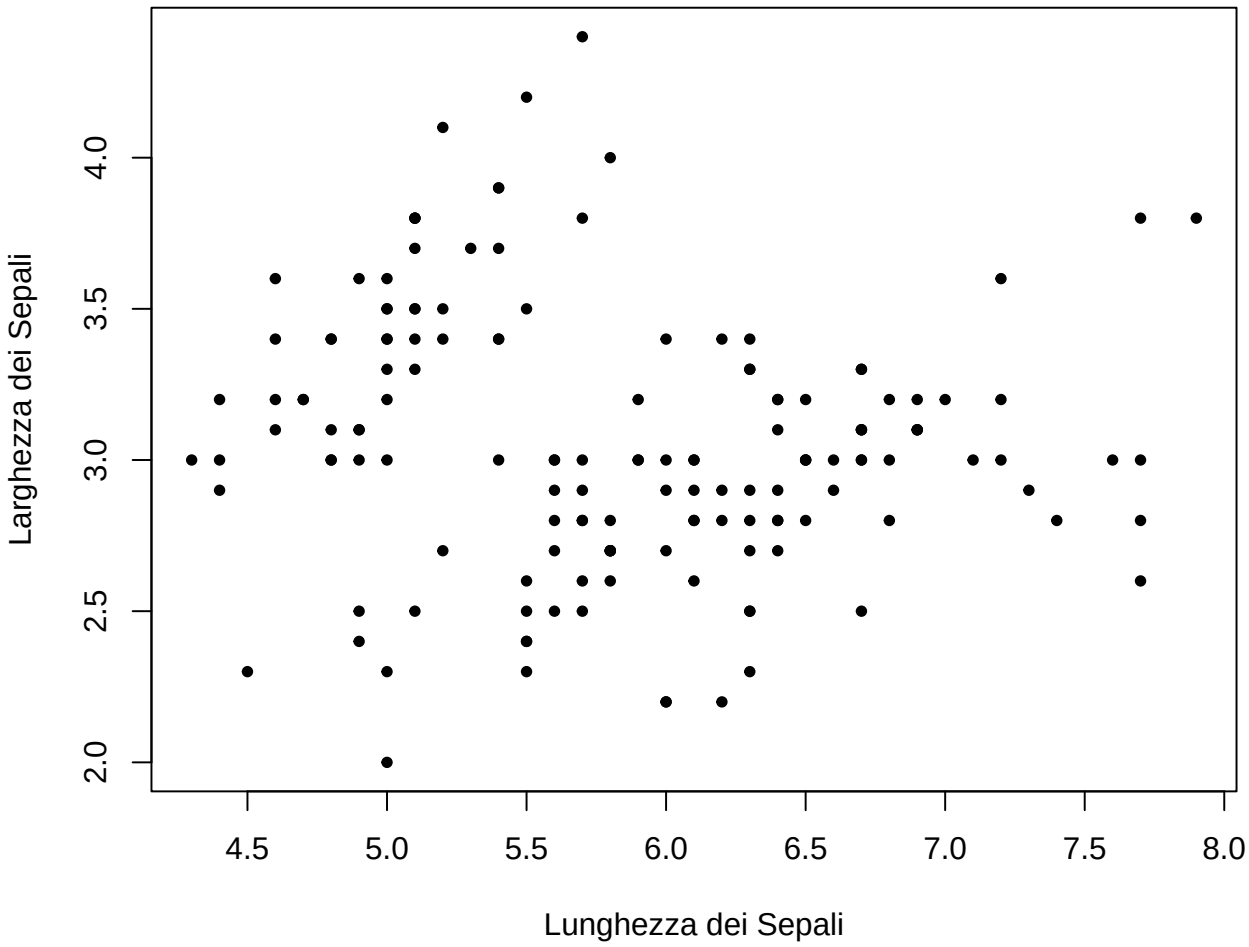
4 Visualizzazione dei dati

Vediamo come fare dei grafici di base

4.1 Grafici a dispersione (scatterplot)

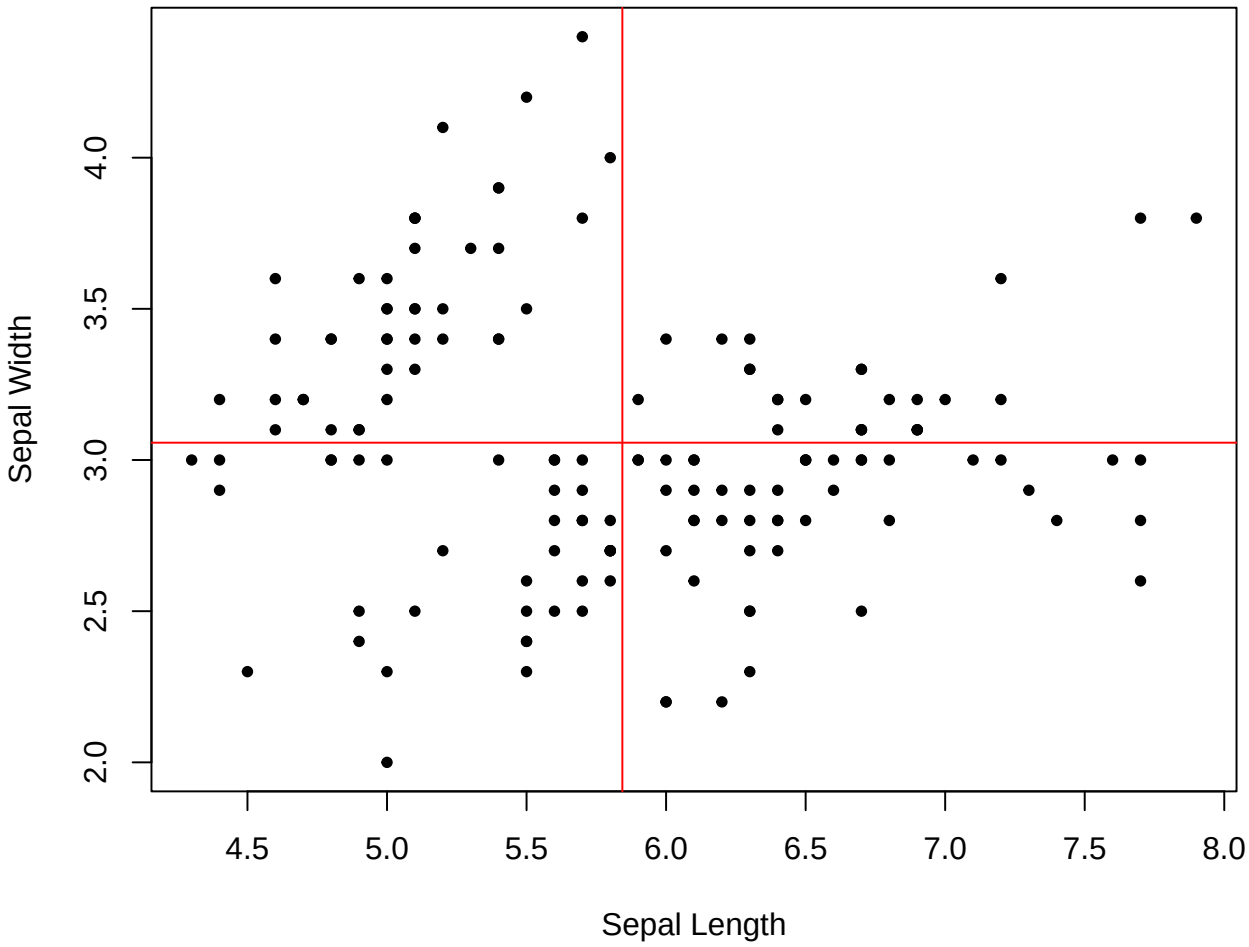
Per prima cosa proviamo a costruire un grafico xy con due variabili prese da i dati iris. Useremo la funzione `plot()`. Questa funzione è una delle più utilizzate in R poichè è stata molto implementata. Infatti, nella maggior parte dei casi, è in grado di riconoscere automaticamente l'oggetto che volete raffigurare (in computer science viene definita una funzione *polimorfa*). Man mano che la userete vi renderete conto di cosa vuol dire.

```
plot(iris$Sepal.Length, iris$Sepal.Width,
     xlab = "Lunghezza dei Sepali", ylab = "Larghezza dei Sepali", pch = 20)
```



Ora costruimo due rette parallele rispettivamente ad x ed y e che intercettino l'asse in corrispondenza della media delle due variabili usando la funzione `abline()`.

```
plot(iris$Sepal.Length, iris$Sepal.Width,
     xlab = "Sepal Length", ylab = "Sepal Width", pch = 20)
abline(h = mean(iris$Sepal.Width), col = "red")
abline(v = mean(iris$Sepal.Length), col = "red")
```



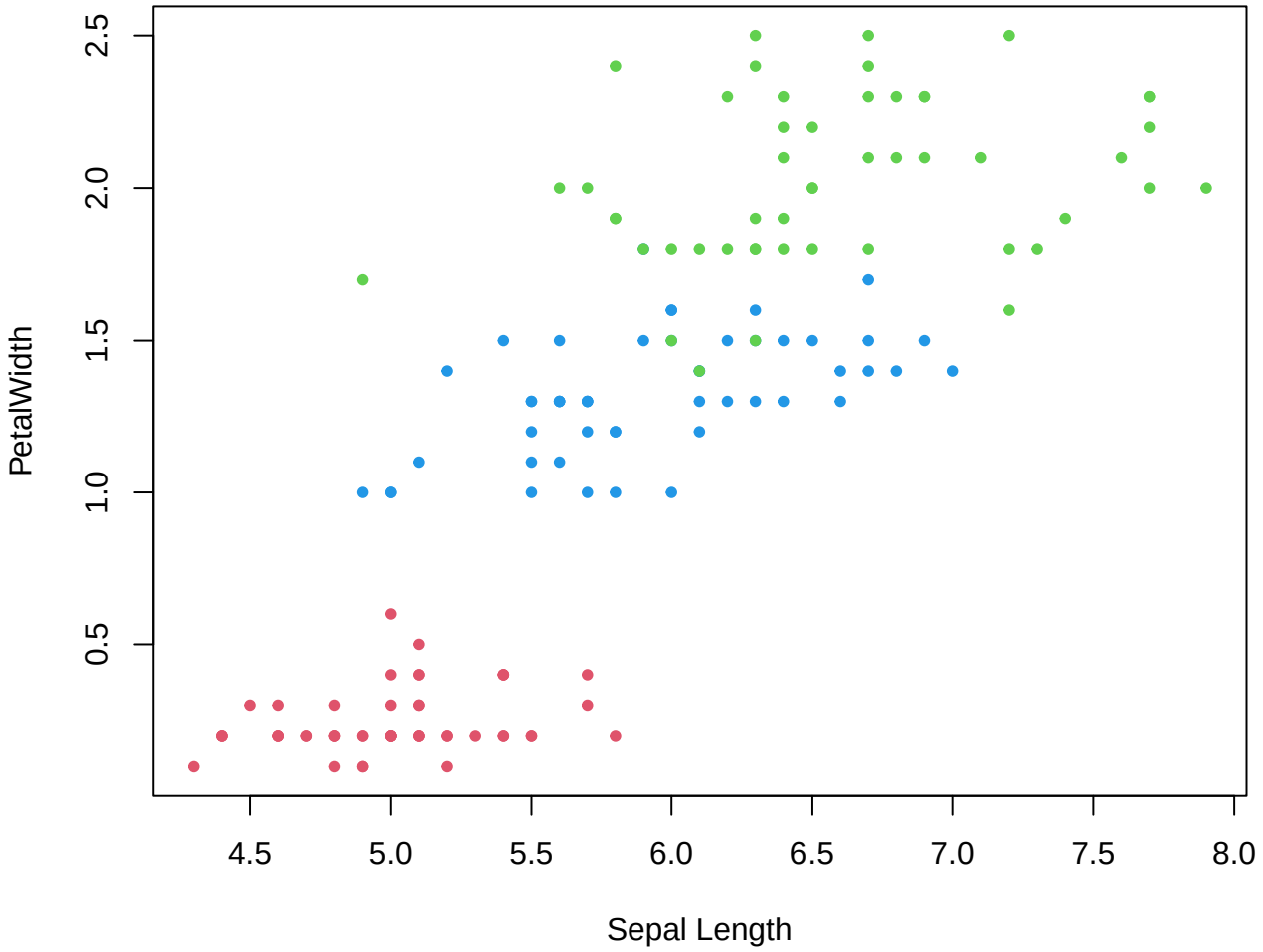
Come vedete le rette sono state inserite automaticamente all'ultimo grafico presente nella finestra grafica. Esistono altre funzioni che aggiungono elementi ad un grafico preesistente. Per aggiungere dei punti, ad esempio, vedremo a breve come usare `points()`. Se volessimo aggiungere una spezzata potremmo usare la funzione `lines()` etc…
Tuttavia nel grafico manca un'informazione. Infatti, questi dati si riferiscono a tre specie differenti. Aggiungiamo il fattore specie colorando i pallini a seconda della specie. Per farlo creiamo un vettore di colori.

```
#v ettore con colore rosso per setosa, blu per versicolor e verde per virginica,
# costruisco un grafico con i
# primi 50 elementi uguali a 2 (rosso), i secondi 50 uguali a 4(blu)
# e gli ultimi 50 uguali a 3 (verde).
# Faccio questo perche' le tre specie sono raggruppate (prime 50 una specie etc.)
vettorecol <- rep(c(2,4,3), each = 50)
```

In R esistono molti modi per richiamare i colori. Alcuni colori base possono essere richiamati con i numeri: 2 corrisponde al rosso; 4 al blu e 3 al verde. In alternativa posso usare dei caratteri, in un certo senso richiamando il nome del colore: `"red"`, `"blue"`, `"green"`. Una lista di colori è disponibile a questo **link**.
Per creare la lista di colori abbiamo usato i numeri e la funzione `rep()`. Questa funzione permette di creare una ripetizione di elementi. Ha tre arguments principali: `x =` cosa deve essere ripetuto, `times =` quante volte deve essere ripetuto, `each =` quante volte deve essere ripetuto ogni elemento di x. Nel nostro caso abbiamo 50 osservazioni per ogni specie e noi vogliamo associare lo stesso colore ad ognuna delle osservazioni appartenenti alla stessa specie. Per fortuna il nostro dataset è già ordinato con in sequenza le cinquanta righe appartenenti ad ognuna delle specie quindi basterà dire alla funzione `rep()` di ripetere lo stesso colore per 50 volte. Quindi nel nostro caso:
- `x =` alla sequenza di colori che vogliamo usare. Per poterli inserire dobbiamo concatenarli in un vettore con

la funzione `c()`.
- `times =` sarà uguale ad 1, che corrisponde al default della funzione quindi possiamo anche non iserirlo.
- `each =` sarà uguale a 50 perchè abbiamo 50 osservazioni per ogni specie.
Ora inseriamo questo nuovo vettore nel nostro grafico all'interno dell'argument `col =` che gestisce le opzioni dei colori.

```
plot(iris$Sepal.Length, iris$Petal.Width,
     xlab = "Sepal Length", ylab = "PetalWidth",
     col = vettorecol, pch = 20)
```



Esistono molti altri modi per selezionare le variabili da plottare ed ottenere lo stesso grafico:
- `plot(iris[,1],iris[,4])`
- `plot(iris[,c(1,4)])`
- `plot(iris$Petal.Width ~ iris$Sepal.Length)`. In questo caso stiamo utilizzando il simbolo `~`. Questo simbolo indica *in funzioni di* e viene usato molto nei modelli.
Anche l'assegnazione dei colori può essere effettuata in altri modi. Ad esempio potremmo usare i nomi delle specie. Partiamo sempre dallo stesso grafico di base e poi assegniamo un colore al nome di ogni specie. Creiamo un vettore che contenga i nomi delle specie. Essendo la variabile specie un factor, per farlo, possiamo usare la funzione `levels()` che richiama i livelli contenuti in una variabile qualitativa di tipo factor.

```
#### vettore dei nomi delle specie
# aggiungo le parentesi tonde per mostrare sulla console
# l'output dell'operazione che sto facendo
(etic <- levels(iris$Species))
```

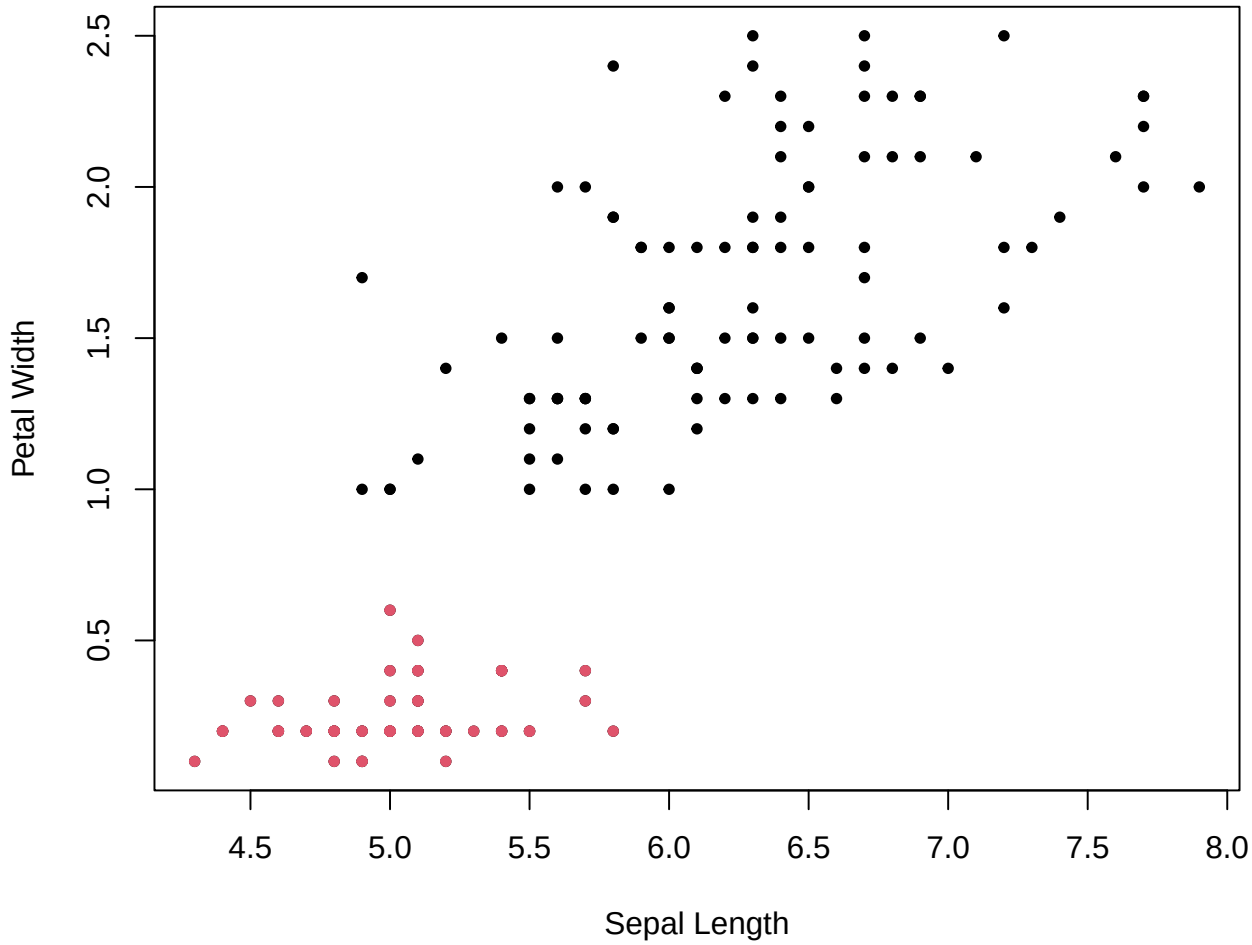
```
## [1] "setosa"     "versicolor" "virginica"
```

In questo modo ho creato un vettore con tre elementi. Nella posizione 1 c'è setosa, nella 2 versicolor e nella tre virginica. Posso usare queste posizioni per creare delle varibili logiche (TRUE/FALSE) da usare nel grafico. Selezioniamo la specie setosa.

```
w <- iris$Species == etic[1]
```

Ora partendo dallo stesso grafico di prima coloriamo i punti appartenenti a setosa di un colore differente. Per farlo possiamo utilizzare la funzione `points()`. Noi vogliamo cambi solo il colore dei punti corrispondenti alla specie setosa. Il vettore logico che abbiamo creato ci permette di fare esattamente ciò applicando il nuovo colore solo ai punti in cui la condizione che abbiamo definito (Specie == etic[1], che corrisponde a setosa) è verificata.

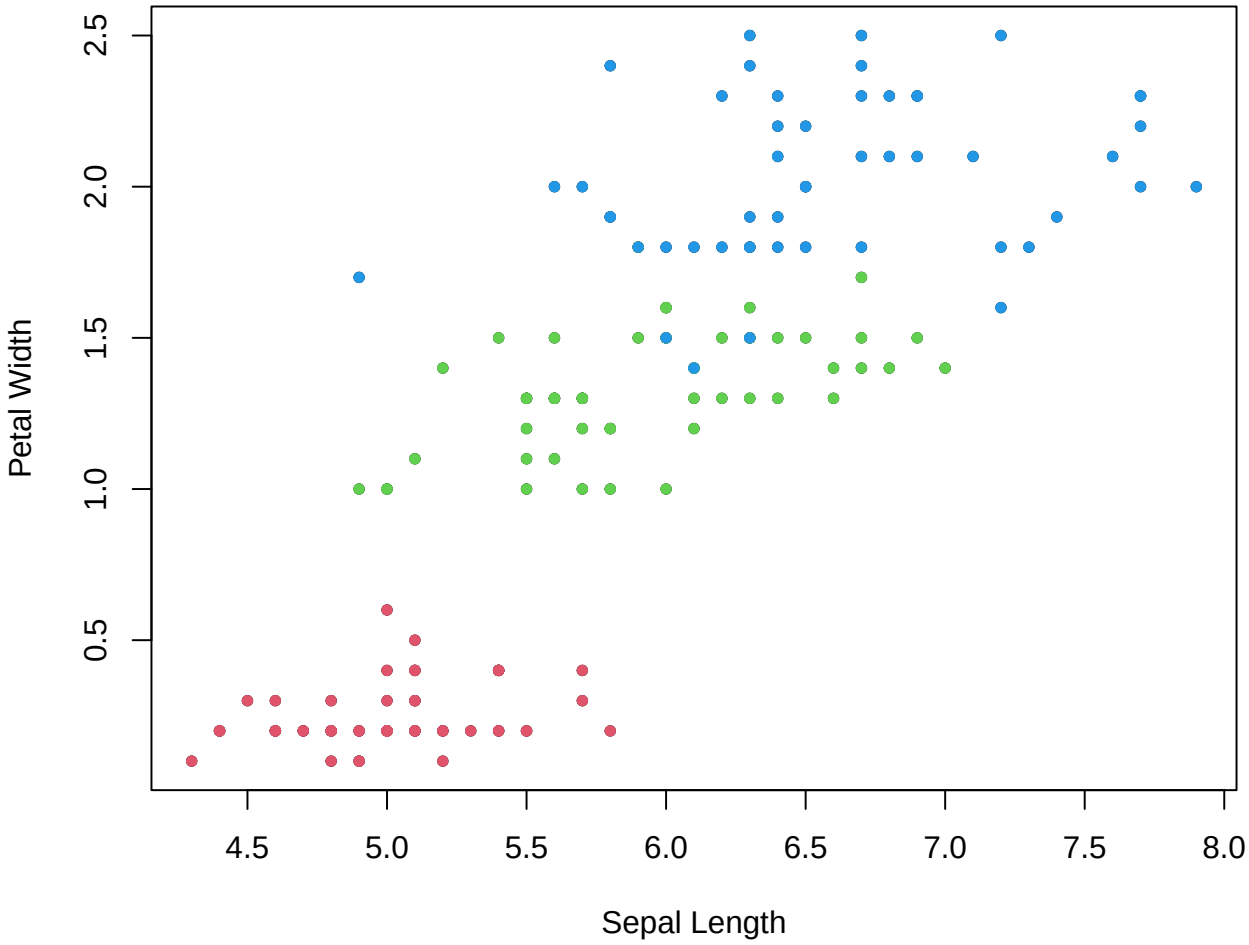
```
plot(iris[,1], iris[,4], xlab = "Sepal Length", ylab = "Petal Width", pch = 20)
#con la variabile logica seleziono le righe di iris che contengono
# la specie e le coloro come voglio ad
# esempio in rosso (col=2)
points(iris[w,1],iris[w,4], pch = 20, col = 2)
```



Possiamo fare la stessa cosa con le altre due specie, attribuendo un colore diverso ed aggiungendolo al grafico.

```
w <- iris$Species == etic[1]
w1 <- iris$Species == etic[2]
w2 <- iris$Species == etic[3]
```

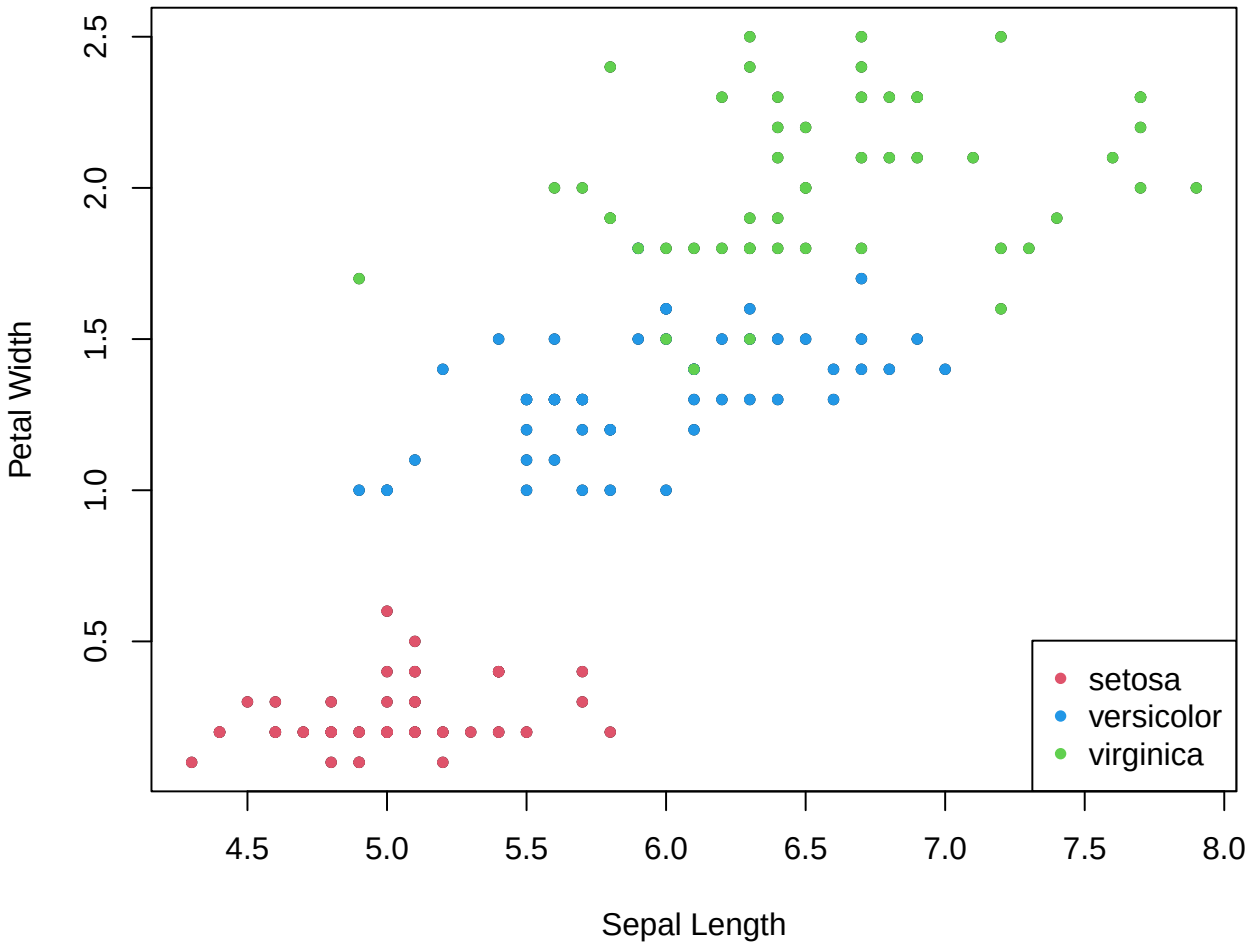
```r
plot(iris[,1], iris[,4], xlab = "Sepal Length", ylab = "Petal Width", pch = 20)
#con la variabile logica seleziono le righe di iris che contengono
# la specie e le coloro come voglio ad
# esempio in rosso (col=2)
points(iris[w,1],iris[w,4], pch = 20, col = 2)
points(iris[w1,1],iris[w1,4], pch = 20, col = 3)
points(iris[w2,1],iris[w2,4], pch = 20, col = 4)
```



Adesso aggiungiamo la leggenda al nostro grafico. Per farlo possiamo utlizzare la funzione `legend()`

```r
#### aggiungo la legenda
legend("bottomright", etic ,pch = 20, col = c(2,4,3))
```

`legend()` funziona come `points()` o `abline()` e aggiunge automaticamente la legenda all'ultimo grafico che abbiamo creato.

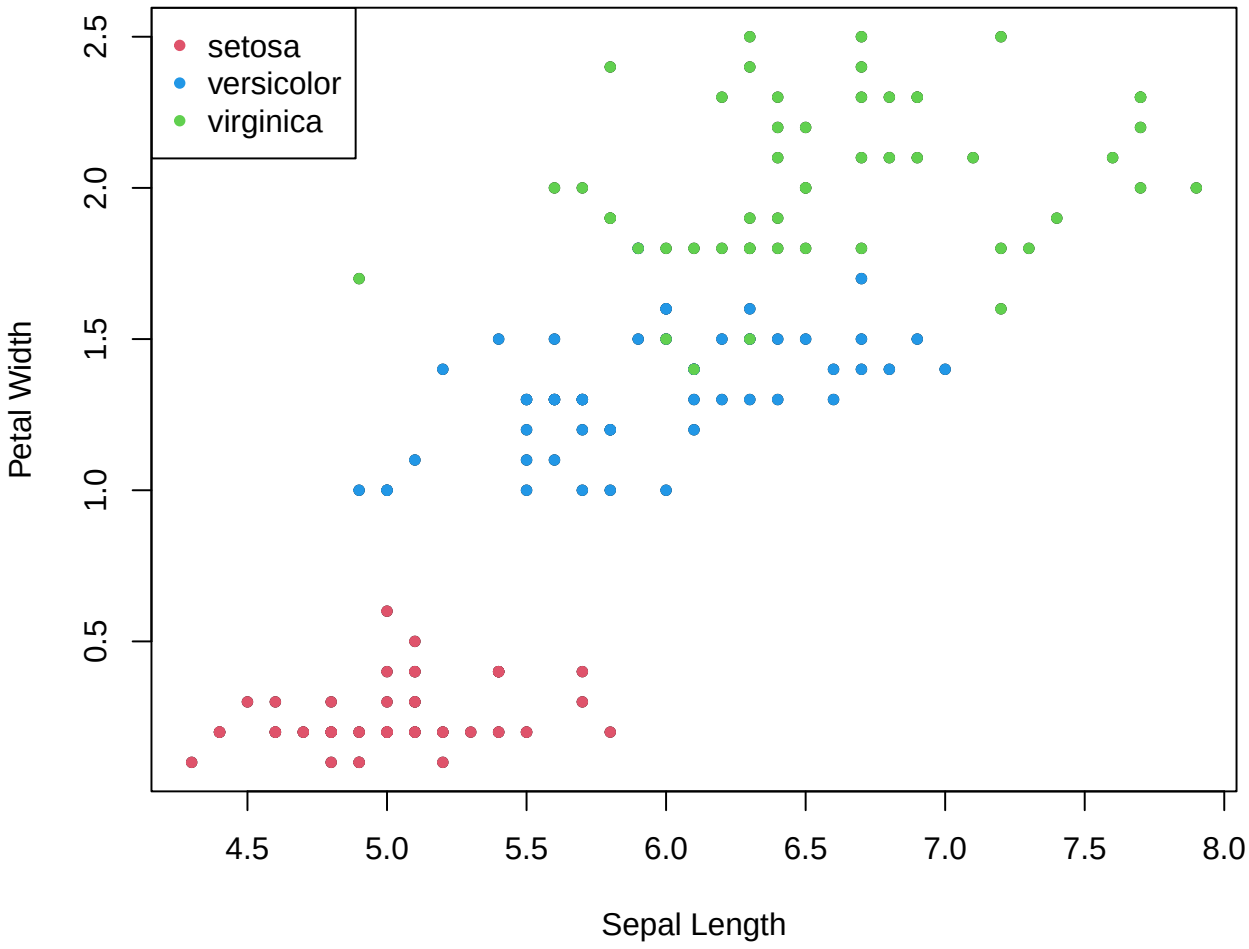


In alternativa posso anche usare questa dicitura.

```
#### oppure
legend("bottomright",levels(iris$Species),pch = 20, col = c(2,4,3))
```

Esistono quattro posizioni predefinite per la legenda: bottomright, bottomleft, topright e topleft. Spostiamo la legenda nell'angolo in alto a sinistra.

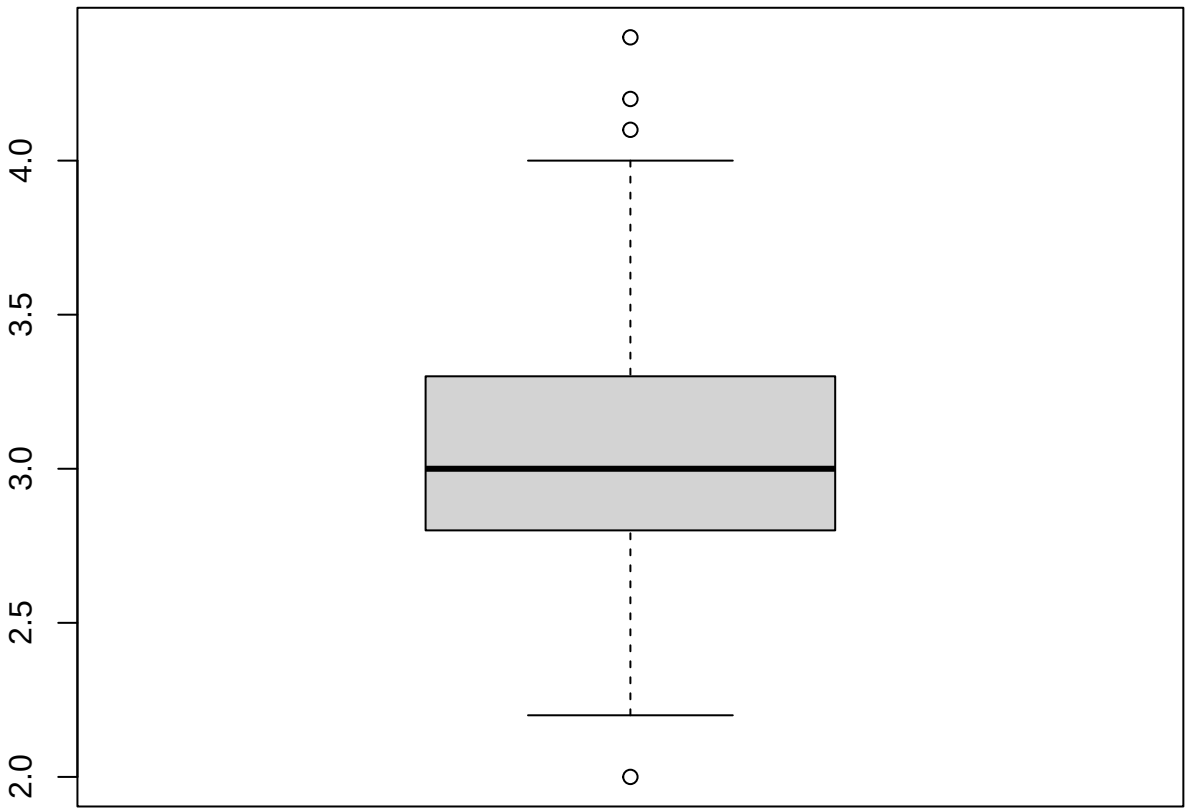
```
legend("topleft",levels(iris$Species), pch = 20, col = c(2,4,3))
```



4.1.1 Boxplot

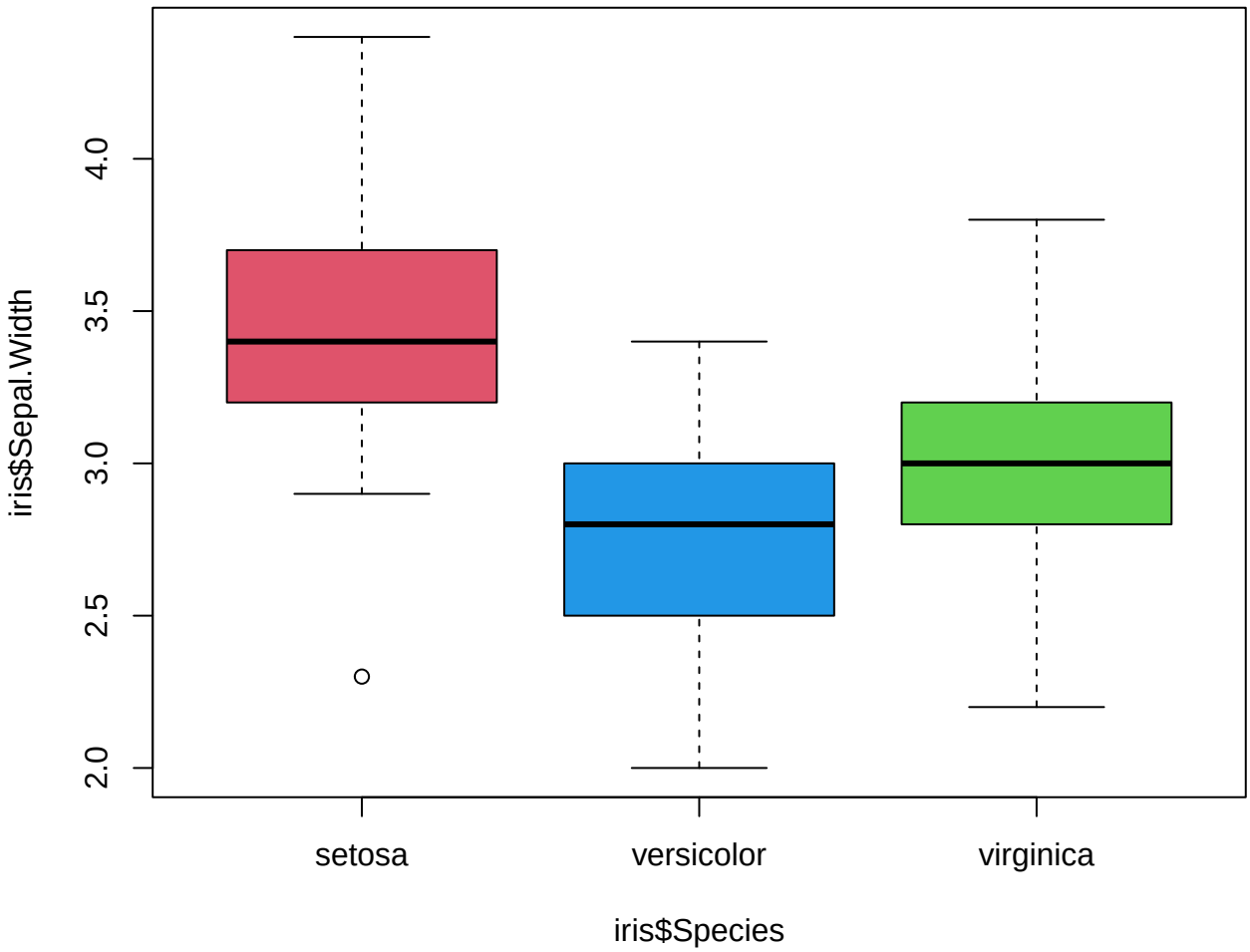
Adesso vediamo come costruire dei boxplot su R. Approfittimone per rivedere cos'è un boxplot e a cosa serve. Il boxplot è una rappresentazione grafica compatta utilizzata per rappresentare la distribuzione di una variabile continua tramite gli **indici di posizione**. La distribuzione viene rappresentata tramite un rettangolo (da qui boxplot) i cui estremi sono il primo e terzo quartile. La scatola è tagliata a metà da una linea che rappresenta il 50esimo percentile (ovvero la mediana). Oltre alla scatola, di solito, vengono rappresentati anche dei baffi che sono lunghi 1,5 volte la distanza interquartile. Le osservazioni che ricatono al di fuori di questo intervallo solo di solito identificate come **outliers** (o valori anomali). Pare che a John Wilder Tukey venne chiesto perché nella determinazione dei valori adiacenti superiore ed inferiore fosse stata scelta una distanza limite dai quartili pari a 1.5 e lui avrebbe risposto "perché 1 è poco e 2 troppo". Costruiamo il boxplot della lunghezza dei petali.

```
boxplot(iris$Sepal.Width)
```



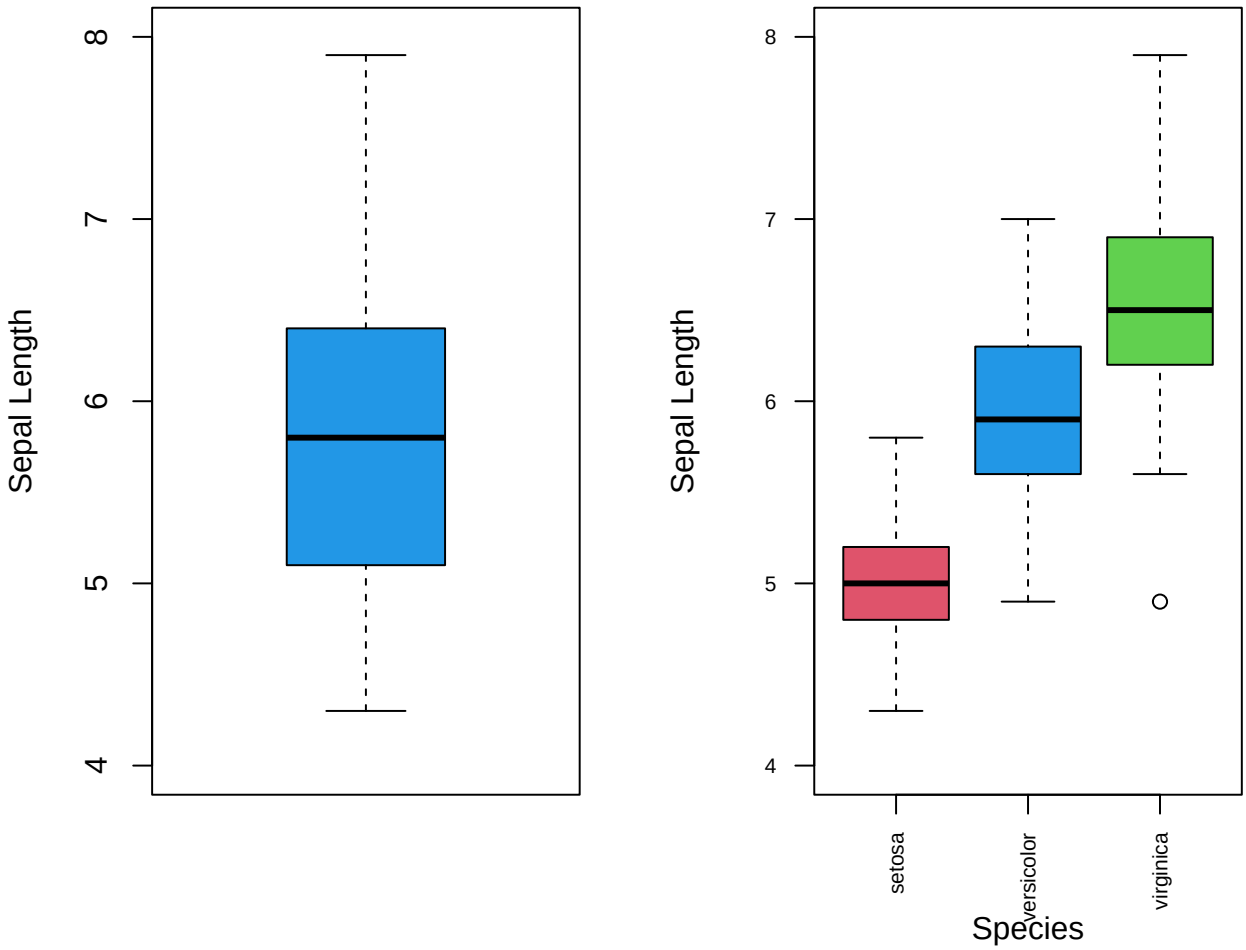
Ora rifacciamo il plot ma in fuzione della specie. Come si diceva ad R “in funzione di”?

```
boxplot(iris$Sepal.Width~iris$Species, col = c(2,4,3))
```



Ora impariamo ad affiancare i due grafici uno accanto all'altro usando le impostazioni della finestra grafica. Per farlo possiamo utilizzare la funzione `par()`. Questa funzione gestisce tutte le impostazioni della finestra grafica. Tramite l'argument `mfrow =` è possibile dividere la suddetta in celle della stessa grandezza, indicando il numero di righe e di colonne. `par(mfrow = c(2,2))` dividerà la finestra in due righe e due colonne (4 celle), `par(mfrow = c(2,3)` in 2 righe e 3 colonne, e così via. Dividiamo la nostra finestra in due colonne.
Nota bene: questa suddivisione non si applica ai grafici creati con `ggplot` che hanno un modo totalmente differente per gestire la finestra grafica

```
par(mfrow = c(1,2))
boxplot(iris$Sepal.Length, ylab = "Sepal Length",
        main = "", col = 4, ylim = c(4,8))
boxplot(iris$Sepal.Length ~ iris$Species,
        ylab = "Sepal Length", xlab = "Species",
        col = c(2,4,3), las = 2,
        cex.axis = 0.7, ylim = c(4,8))
```



Diamo maggiori dettagli sugli argument della funzione `plot()` che abbiamo usato: `ylim =` delimita il volore minimo e massimo dell'asse y; `ylab =` permette di indicare il label dell'asse y; `main =` indica il titolo da dare al grafico; `col =` gestisce i colori; `las =` indica se i label dell'asse x debbano essere plottati verticalmente (2) od orizzontalmente (1); `cex.axis =` gestisce la grandezza dei valori sugli assi.