Note di R

Federico Reali

06/06/2019

Contents

Premessa

Questo libro racchiude le note riguardanti il laboratorio di R del corso di Probabilità e Statistica Matematica dell'anno accademico 2018/2019.

Il materiale contenuto si intende ad uso degli studenti. Può contenere errori, typos ed imprecisioni.

Questo e-book è stato ottenuto grazie usando il pacchetto **bookdown** (Xie, 2019) e la guida (Xie, 2015).

Chapter 1

Primi passi con R

In questa prima parte vedremo quali sono i passi base per istallare R e Rstudio sul nostro computer.

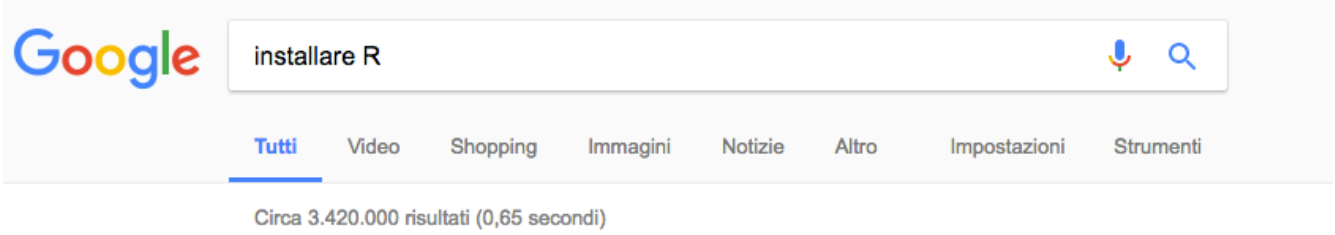
1.1 Perché R

> R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS.
>
> The R Foundation

R è un versatile linguaggio di programmazione che sta particolarmente a cuore alla comunità che si occupa di statistica, per questo esistono molte pagine volte ad insegnarne l'uso, così come molti blog e forum dove gli utenti possono cercare risposte alle loro domande realtive all'uso.

R è un software libero disponibile per le maggiori piattaforme, inclusi Windows, Mac e diverse distribuzioni di Linux.

Per esempio, possiamo trovare molte guide su come installare R sul nostro computer Windows, Mac o Linux. Una semplice ricerca su google produce innumerevoli guide e tutorial su come farlo.



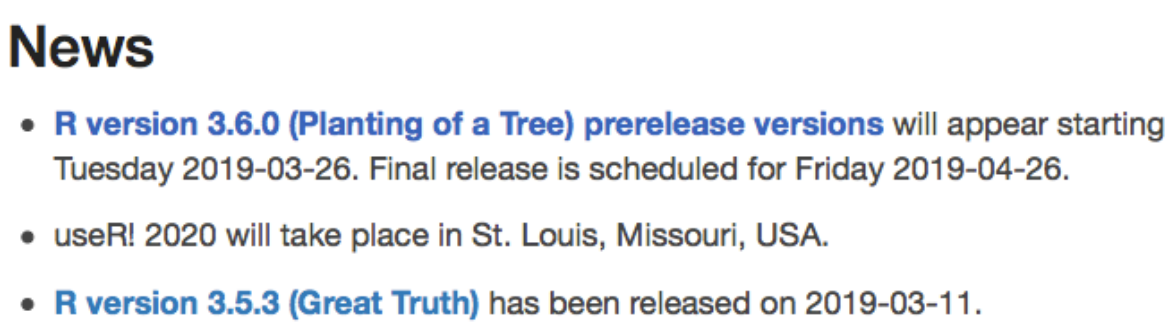


Figure 1.1: Release di R

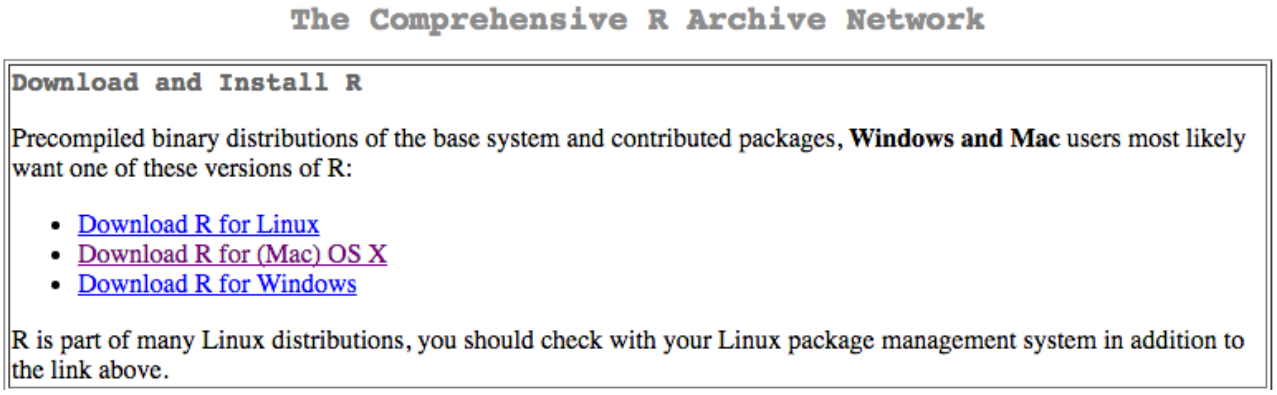


Figure 1.2: Il server cran ospitato presso l'Università di Padova

Noi seguiremo una via molto diretta che segue due passi fondamentali:

- l'istallazione di R vera e propria;
- l'istallazione di un interfaccia.

Installeremo la [versione] (https://www.r-project.org/) di R rilasciata l'11/03/2019 e tra le innumerevoli interfacce per R, noi useremo R Studio. Si noti che tra pochi giorni sarà disponibile la versione 3.6.0.

1.2 Installare R

Procediamo con l'istallazione di R

1.2.1 Scaricare R

Le varie release di R sono disponibili attraverso il sito cran, dove è possibilie selezionare il server più vicino da cui fare il download dei file di istallazione.

Ad esempio possiamo scegliere il server relativo all'Università di Padova. Da qui troviamo i link per scaricare le release di R per le varie piattaforme.

R è un progetto open source, quindi oltre agli installer è possibile scaricare il codice sorgente di R, ed è possibile apportare modifiche e/o compilarlo personalmente.

Noi scaricheremo invece l'applicazione relativa al vostro sistema operativo (nelle immagine Mac) e procederemo con l'installazione.

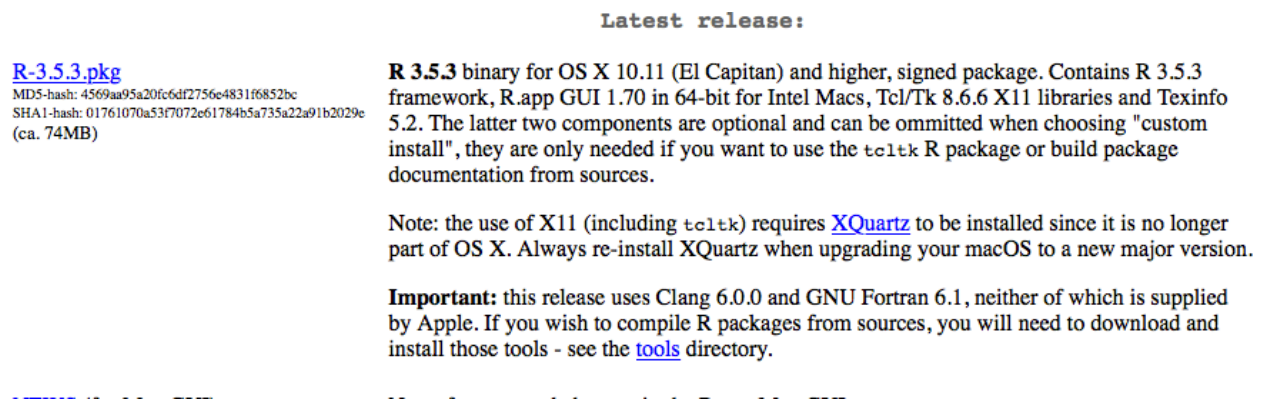


Figure 1.3: Example image

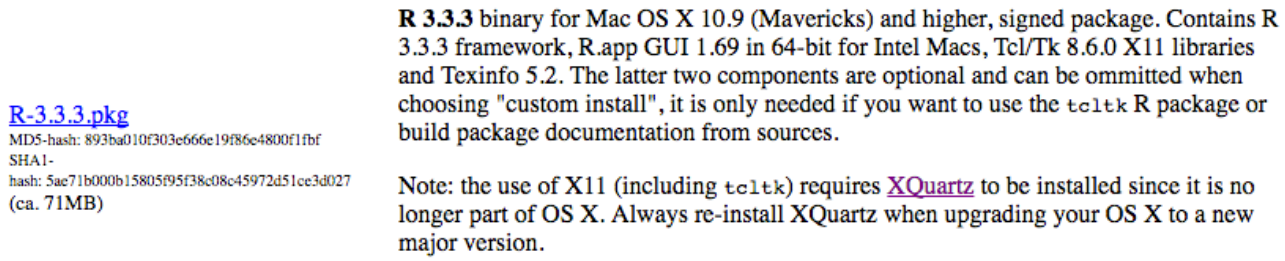


Figure 1.4: Example image

La pagina di download permette di scaricare anche versioni meno recenti di R. Questo può essere importante qualora si usino delle estensioni (chiamate ***pacchetti***), che potrebbero non essere state aggiornate per l'ultima versione. Noi considereremo per questo tutorial l'istallazione della versione 3.5.3.

Si noti che l'istallazione su Mac richiede l'istallazione di XQuartz

Alcuni computer meno aggiornati potrebbero non supportare la versione più recente (tipo il mio). In tal caso, si consiglia l'istallazione della versione 3.3.3.

Nel caso in cui si stesse installando R sulla distribuzione linux Ubuntu, è possibile farlo da terminale attraverso i comandi:

```
sudo apt-get update
sudo apt-get install r-base
```

Il sito descrive la procedura anche per le altre distribuzioni.

1.3 Installare R

Nell'istallazione su Mac o Windows si segue la procedura standard una volta scaricati i file. Per installare R su linux, basta usare i comandi descritti precedentemente per Ubuntu, o utilizzare gli equivalenti per le altre distribuzioni.

1.4 Usare R

Una volta istallato R, è già possibile usarne tutte le funzionalità. Infatti andando in **Applicazioni** si può già trovare R tra esse. Pigiando la relativa icona si avvia la **R console**, una interfaccia molto semplice che però permette di usufruire di **tutte** le potenzialità del software.

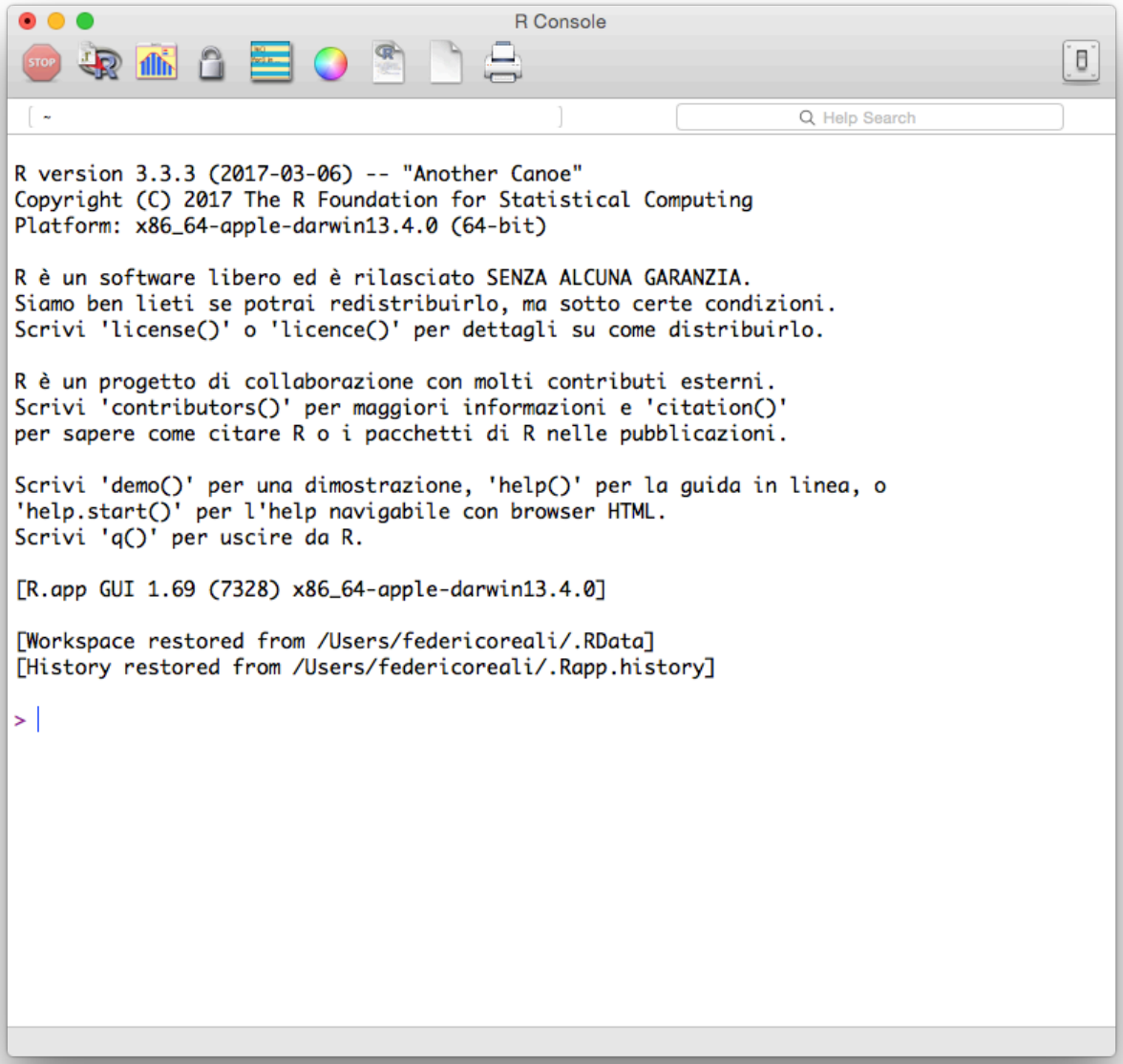


Figure 1.5: Example image

Installers for Supported Platforms

Installers	Size	Date	MD5
RStudio 1.2.1335 - Windows 7+	126.9 MB	2019-04-08	d0e2470f1f8ef4cd35a669aa323a2136
RStudio 1.2.1335 - Mac OS X 10.12+ (64-bit)	121.1 MB	2019-04-08	6c570b0e2144583f7c48c284ce299eef
RStudio 1.2.1335 - Ubuntu 14/Debian 8 (64-bit)	92.2 MB	2019-04-08	c1b07d0511469abfe582919b183eee83
RStudio 1.2.1335 - Ubuntu 16 (64-bit)	99.3 MB	2019-04-08	c142d69c210257fb10d18c045fff13c7
RStudio 1.2.1335 - Ubuntu 18 (64-bit)	100.4 MB	2019-04-08	71a8d1990c0d97939804b46cfb0aea75
RStudio 1.2.1335 - Fedora 19+/RedHat 7+ (64-bit)	114.1 MB	2019-04-08	296b6ef88969a91297fab6545f256a7a
RStudio 1.2.1335 - Debian 9+ (64-bit)	100.6 MB	2019-04-08	1e32d4d6f6e216f086a81ca82ef65a91
RStudio 1.2.1335 - OpenSUSE 15+ (64-bit)	101.6 MB	2019-04-08	2795a63c7efd8e2aa2dae86ba09a81e5
RStudio 1.2.1335 - SLES/OpenSUSE 12+ (64-bit)	94.4 MB	2019-04-08	c65424b06ef6737279d982db9eefcae1

Figure 1.6: RStudio - Versions

Ad esempio digitando

```
2+2
```

```
## [1] 4
```

oppure

```
3*5
```

```
## [1] 15
```

è possibile utilizzare la console come calcolatrice. Ovviamente questo tipo di uso sfrutta solo una minima parte delle potenzialità del software.

Sebbene tutto quello che faremo sarà possibile farlo anche direttamente da console, noi useremo un software che si interfaccia ad R, ma che presenta un'interfaccia grafica più completa. Tale programma è R Studio.

1.5 Installare R Studio

Come R, anche R Studio è disponibile gratuitamente per utenti singoli, anche se sono previste anche versioni commerciali a pagamento. Inoltre è possibile scaricare pacchetti di installazione sia per Windows, Mac e alcune distribuzioni di Linux.

Una volta scaricati i pacchetti di installazione, la procedura automatica dovrebbe installare R Studio per tutte le piattaforme (speriamo!).

1.6 Usare R Studio

Una volta installato R Studio, avviamolo e (se l'istallazione è andata a buon fine) dovrebbe apparire la schermata iniziale:

L'interfaccia grafica si presenta con 3 principali finestre che riportano la console di R (la stessa che abbiamo visto prima singolarmente), la finestra **Enviroment** e **History** e quella contenente **Files**, **Plots**, **Packages**, **Help** e **Viewer**.

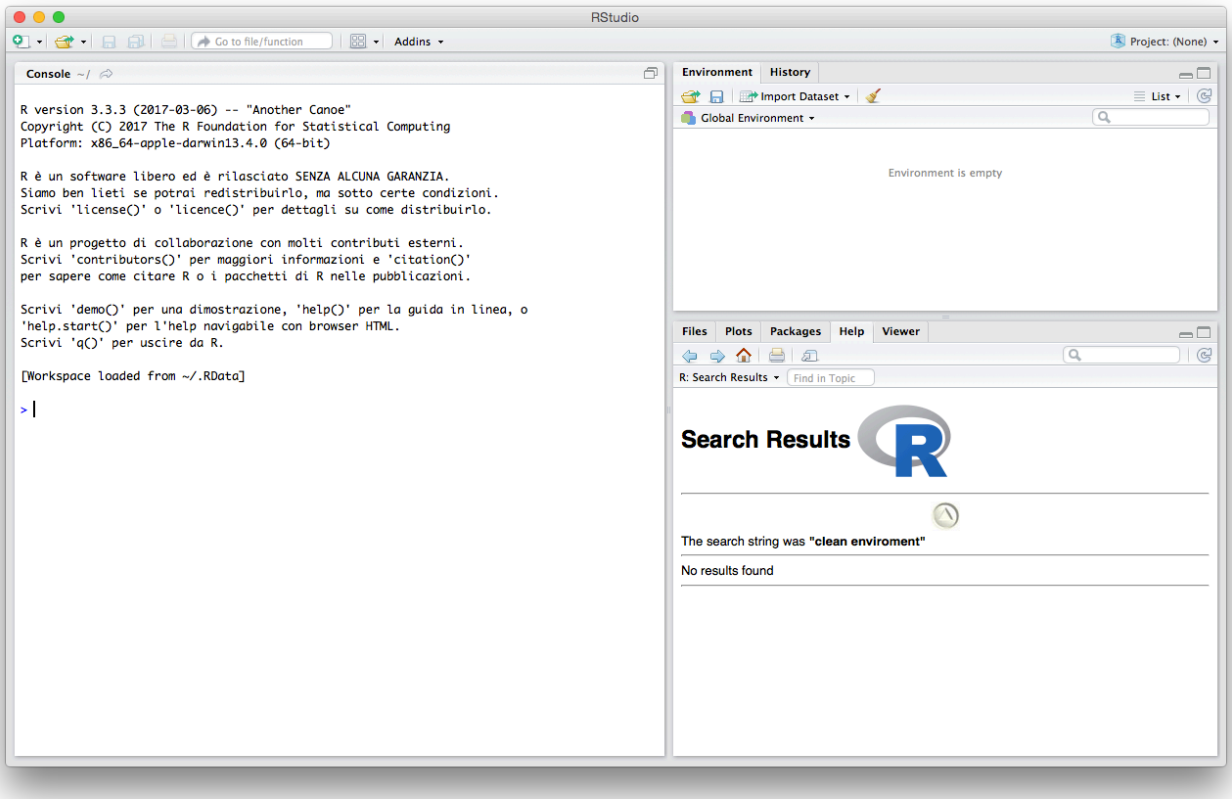


Figure 1.7: RStudio - Versions

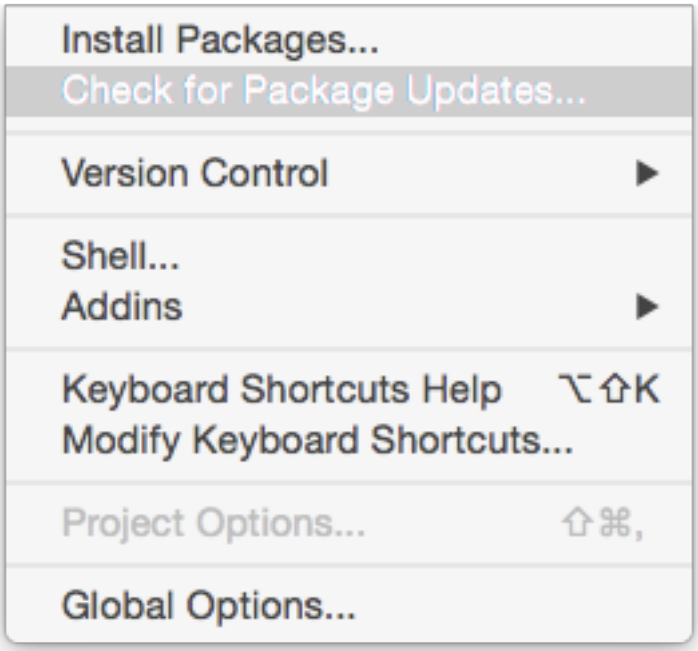


Figure 1.8: RStudio - Versions

Per prima cosa controlliamo che tutti i pacchetti contenuti nell'installazione siano aggiornati. Selezionando da Tools:

Se risultano pacchetti da aggiornare, selezionateli tutti e aggiornateli. In caso non vi siano, R Studio vi dovrebbe avvisare.

1.6.1 Cartelle

E' possibile usare degli appositi comandi per capire in quale cartella sta lavorando R. Questo è essenziale se si vuole salvare dei file o leggerne il contenuto.

```
getwd()
```

```
## [1] "/Users/federicoreali/Dropbox/Bonaccorsi/Calcolo delle probabilita e statistica matematica/2019/Rbo
```

Questo comando *get working directory* restituisce il path della cartella corrente. R può facilmente leggere o salvare file in questa cartella.

E' possibile impostare una cartella diversa. Ad esempio possiamo creare una cartella relativa al corso di **Calcolo delle probabilità e statistica matematica (CPeSM)** e possiamo creare lì dentro una cartella chiamata R, dove possiamo salvare i nostri file.

```
setwd('IL-Vostro-path/CPeSM/R/Lezione1')
```

Una volta capito dove siamo e dove dovremmo essere, possiamo iniziare ad usare una delle funzionalità per cui abbiamo scelto R Studio: la possibilità di creare degli script nella stessa interfaccia grafica e di lanciare comandi dagli stessi.

Selezioniamo **File|New File|R Script** apriamo in R Studio una nuova finestra che permette di editare (scrivere) uno script.

Ora non rimane che spendere un po' di tempo a familiarizzare con l'interfaccia e con R!

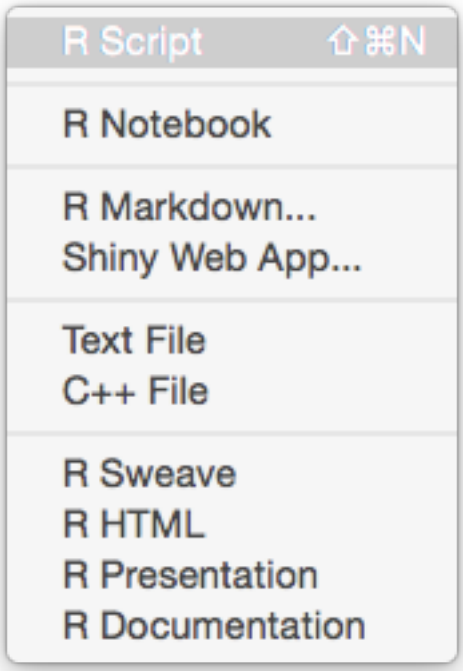


Figure 1.9: RStudio - Versions

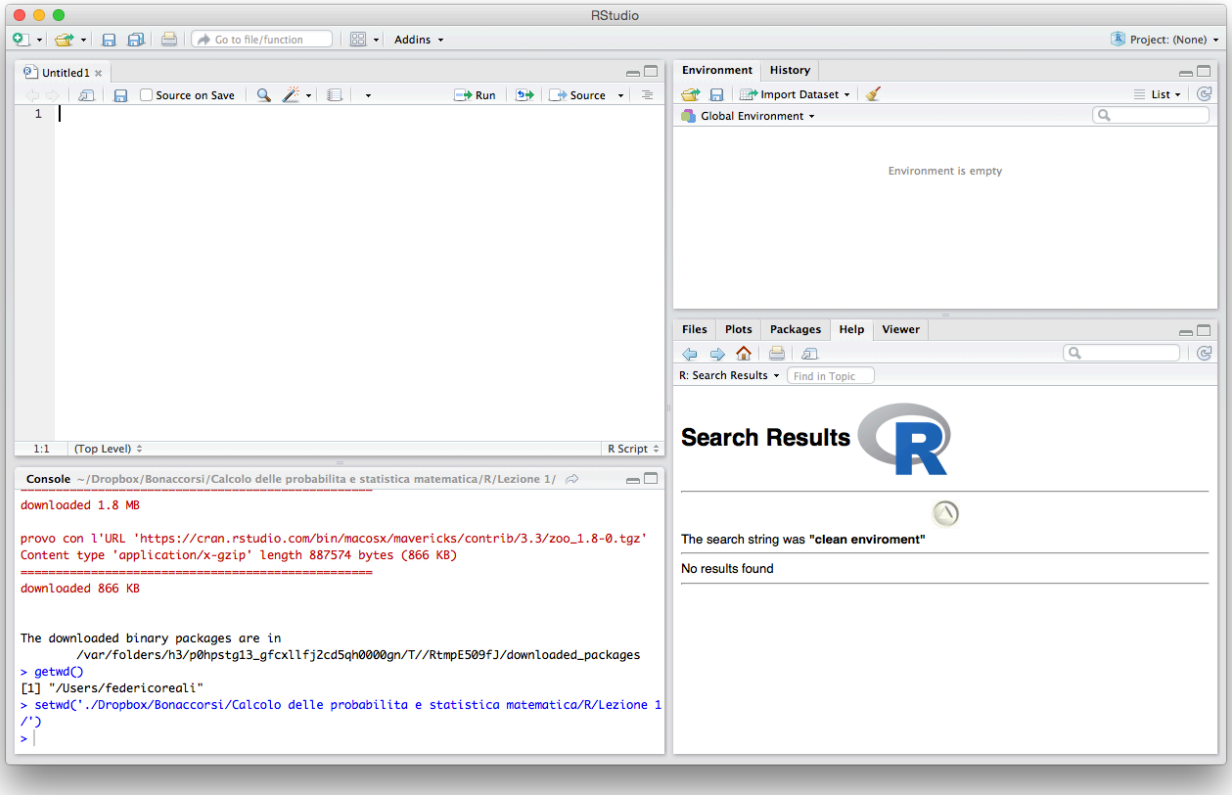


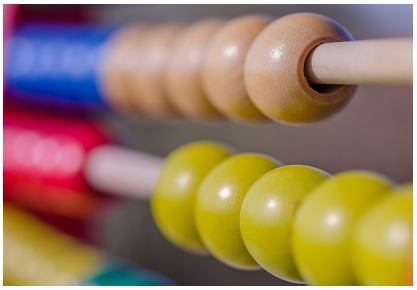
Figure 1.10: RStudio - Versions

1.7 Link utili

- A questo link è possibile reperire delle utili dispense sull'uso di R.
- A questo link è possibile reperire un utile formulario che descrive moltissimi comandi di R.

Chapter 2

Primi comandi di R



2.1 Operazioni Aritmetiche

Come abbiamo già visto è possibile usare R per svolgere operazioni matematiche, usando i simboli standard.

Ad esempio, l'uso di `+` più tra due numeri farà comparire la loro somma. In modo analogo è possibile farne la differenza (`-`), il prodotto (`*`) e la divisione(`/`).

```
2+2
```

```
## [1] 4
```

```
2-2
```

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

```
2*2
```

```
## [1] 4
```

```
2/2
```

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

Le potenze si possono esprimere usando il simbolo `^`. Lo stesso simbolo si può utilizzare per il calcolo delle radici, anche se è possibile richiamare la radice quadrata usando il comando `sqrt()`. Si noti che tale funzione accetta anche valori complessi.

Per la divisione intera tra due numeri è possibile usare il simbolo `%/%`, mentre `%%` restituisce il resto di tale divisione.

Un altro comando di base che può risultare utile è il calcolo del valore assoluto ottenuto con il comando `abs()`.

In modo analogo sono già definite alcune delle funzioni che sono alla base dei calcoli matematici più frequenti. Le vedremo nei prossimi paragrafi.

2.2 Esponenziale e logaritmi

E' possibile calcolare i valori della funzione esponenziale usando il comando `exp()` mentre per il calcolo dei logaritmi è possibile usare più funzioni: `log(), log10()` e `log2()` che restituiscono il logaritmo naturale, base 10 e base 2 rispettivamente. Il comando `log()` permette anche di specificare una base differente da ***e***, che è quella di default. Ad esempio, `log( 5 , base = 3)` restituisce il logaritmo in base 3 di 5.

2.3 Funzioni Trigonometriche

Le principali funzioni trigonometriche sono già implementate in R e si possono richiamare usando i comandi elencati sotto, i cui nomi richiamano le funzioni stesse. Va notato che l'input di queste funzioni è atteso in radianti, **non gradi**.

```
sin() # seno
cos() # cosen
tan() # tangente
```

E' possibile utilizzare le stesse funzioni ma con desinenza ***pi*** (`sinpi()`, `cospi()` e `tanpi()`) se si intende considerare dei multipli di pi-greco.

Anche le funzioni inverse sono già implementate e si possono richiamare con i comandi `acos(), asin(), atan()`.

2.4 Assegnazione e vettori

Fin ora abbiamo usato R per calcolare dei singoli valori, in modo analogo a come avremmo usato una calcolatrice. Tuttavia R permette di fare molto di più.

Come primo passo per scoprirne le potenzialità vedremo la possibilità di assegnare valori e di richiamarli successivamente, così come di lavorare con vettori o matrici, invece che con singoli valori.

Questa parte, così come alcune successive sono anche coperte dalle utili dispense, di cui ho già consigliato la lettura.

E' possibile assegnare un valore ad una variabile usando la freccina `->`. Stesso risultato si può ottenere usando `=` oppure `assign()`, tuttavia il primo è decisamente il comando più usato e di più facile lettura. Qui è possibile leggere una discussione sulle differenze nell'uso di `=` o di `->`.

```
x  <- 6
```

Usando il precedente comando viene assegnato il valore 6 alla variabile x. In questo modo è possibile richiamarla successivamente, così come è possibile scrivere espressioni più complesse che coinvolgono x a prescindere dal suo valore. Questo sarà particolarmente utile se dovremo definire delle funzioni a cui è possibile passare per argomento diversi valori. Si noti che il comando `6 -> x` produce gli stessi risultati del precedente.

Oltre ad assegnare un singolo valore, è possibile considerati dei vettori. Il comando

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

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

assegna a y i valori contenuti nel comando `c()`. Tale comando combina i valori in un vettore colonna (ma viene visualizzato come riga). L'accesso ai vettori avviene attraverso indicandone il nome, seguito da parentesi quadre. Ovviamente è possibile svolgere operazioni aritmetiche anche tra vettori, usando la stessa sintassi vista prima. Di default R considera operazioni puntuali tra vettori, cioè l'operazione richiesta viene svolta entrata per entrata. Questo richiede le dimensioni siano le stesse. Qualora non lo fossero, il contenuto del vettore più piccolo viene ripetuto un numero di volte sufficienti da rendere possibile l'operazione. Questo viene anche segnalato da un warning.

E' possibile ottenere le dimensioni di un vettore usando il comando `length()`.

Un comando utile tra vettori è `t()` che traspone il vettore passato per argomento.

```
z <- t(y)
y
```

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

```
print(z)
```

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

```
z
```

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

Chiamare una variabile senza indicare operazioni ne permette la stampa a video. Inserendo l'assegnazione tra parentesi tonde si ottiene lo stesso risultato. Anche il comando `print()` permetta la stampa a video. Inoltre questo comando, diversamente dai precedenti, permette di mostrare l'output anche quando è chiamata dentro una funzione o uno script.

Dal comando precedente vediamo che y e z hanno stessi valori ma dimensioni diverse. Tuttavia R permette di svolgere operazioni matematiche su di essi senza warning o errori.

Per svolgere operazioni tra vettori, come il prodotto righe per colonne si può considerare il comando `%*%`. In questo caso il risultato dipende dall'ordine.

```
y %*% z
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    2    4    6    8
## [3,]    3    6    9   12
## [4,]    4    8   12   16
```

```
z %*% y
```

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

2.5 Sequenze

Nel caso si debba indicare una successione di valori è possibile farlo senza indicarli espressamente tutti, purché essi siano una sequenza regolare.

Ad esempio con `a:b` possiamo usare i due punti per indicare una sequenza di valori con passo 1 da a fino a b . Per indicare un passo diverso possiamo usare il comando `seq(a,b,passo)` che produrrà una sequenza di elementi da a fino a b distanziati di un valore uguale al *passo* indicato. Se il passo viene omesso è considerato 1 di default.

Per tutte le funzioni è possibile accedere all'help di R usando il comando `help(Nome_funzione)`. Leggendo l'help di `seq()` vediamo ad esempio che possiamo anche indicare in ordine diverso i parametri, a patto di specificarli usando il nome indicato nell'help. Ad esempio `seq(to = b, by = passo, from = a)` restituisce lo stesso output di `seq(a,b,passo)`.

Inoltre è possibile ripetere un valore o un vettore usando il comando `rep()`, ad esempio `rep(1,5)` restituisce cinque volte il valore 1. Nel caso di un vettore, il comando ripeterà 5 volte il vettore.

2.6 Operazioni su vettori

Oltre alle operazioni che abbiamo già visto esistono altre funzioni di R appositamente pensate per i vettori. Ad esempio le funzioni `min` e `max` restituiscono, rispettivamente, il minimo ed il massimo valore contenuto in un vettore. Per accedere alla posizione di tali valori si combinano i precedenti comandi con `which`. Ad esempio

```
x <- c(3, 5, 7, 1, 3, 3, 9, 8)
min(x)
```

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

```
which.min(x)
```

```
## [1] 4
```

```
max(x)
```

```
## [1] 9
```

```
which.max(x)
```

```
## [1] 7
```

Altre funzioni molto utili per lavorare con i vettori sono la funzione `sum()` che calcola la somma di tutti gli elementi di un vettore e la funzione `diff()` che calcola la differenza di un valore da uno dei precedenti (è possibile indicare quanto prima "guardare").

```
sum(x)
```

```
## [1] 39
```

```
diff(x)
```

```
## [1]  2  2 -6  2  0  6 -1
```

```
diff(x,2)
```

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

2.7 Operatori relazionali e logici

R ci permette anche di valutare espressioni relazionali o logiche. Il loro risultato sarà un valore logico indicato con TRUE o FALSE.

```
6 > 10
```

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

```
6 <= 10
```

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

```
is_bigger <- 6> 10
is_bigger
```

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

```
as.integer(is_bigger)
```

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

R permette di valutare diverse espressioni relazionari, oltre a maggiore (uguale) o minore (uguale). Ad esempio è possibile valutare se due valori o variabili siano diversi `!=` o uguali `==`. Questo può essere particolarmente utile quando si definiscono delle funzioni proprie e si valuta se una condizione è soddisfatta.

R permette anche di valutare gli operatori logici `&` (and), `|` (or), `xor` e `!` (not).

Questi operatori possono essere valutati sia su vettori logici, che di numeri qualsiasi. Nel secondo caso, tuttavia, tutto ciò che è diverso da zero conterà come TRUE, e solo lo 0 conterà come FALSE. Nel caso in cui si definiscano dei vettori misti (dove sono presenti sia valori logici che interi/reali/complessi), verranno tutti convertiti nel formato numerico presente.

2.8 Matrici

Come abbiamo visto i vettori sono considerati in generale vettori riga, e non è possibile generare una matrice usando il comando `c()`.

Tuttavia le matrici esistono e si possono definire usando la funzione `cbind()` che unisce i vettori passati come argomento in una matrice dove i vettori argomento sono le colonne. La funzione `rbind()` ha la stessa funzione, ma i vettori passati come argomento saranno le righe della matrice. In tutti e due i casi, i vettori passati come argomento devono avere stessa dimensione (no uno riga e uno colonna). La funzione `dim()` restituisce le dimensioni dell'oggetto passato come argomento. La funzione `length()` (che avevamo visto con i vettori) ci restituisce il *prodotto* delle dimensioni.

```
a <- cbind(c(1, 2, 3), c(4, 5, 6))
dim(a)
```

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

```
b <- rbind(c(1, 2, 3), c(4, 5, 6))
dim(b)
```

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

```
a
```

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

```
b
```

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

E' possibile variare la dimensione di un oggetto usando gli stessi comandi `length` e `dim` come assegnazioni.

```
dim(a) <- c(2,3)
a
```

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

Il comando precedente cambia la forma della matrice, secondo la forma indicata. Tali comandi vanno comunque usati con attenzione, visto che non si ha controllo su come il contenuto verrà ridistribuito.

Altra utile funzione per creare matrici è la funzione `array()` che chiede di specificare gli elementi della matrice e le dimensioni. Risultato analogo si può otenere con la funzione `matrix(elementi, num righe, num colonne)`. Si noti che con `array()` è possibile definire strutture con più di due dimensioni.

Gli elementi una matrice si accedono usando le parentesi quadre, indicando la posizione dell'elemento.

```
a[2,3]
```

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

Se si intende selezionare un'intera riga o colonna, basta lasciare vuoto tale campo. Per esempio `a[1,]` restitusce l'intera prima riga.

Alcune funzioni come `contour(), persp(), image()` permettono di ottenere dei grafici, partendo da matrici.

2.9 Stringhe (vettori di caratteri)

R permette di manipolare anche vettori di caratteri, o stringhe, che possono essere salvati anche come vettori. Le stringhe vengono delimitate da doppie virgolette `" "` (o anche semplici virgolette `' '`).

```
nomi <- c("Francesco", "Sofia", "Alessandro")
nomi[1]
```

```
## [1] "Francesco"
```

```
nomi_e_numeri <- c("Francesco", "Sofia", "Alessandro", 45)
```

In R non possono convivere nello stesso array caratteri e numeri. Se ad esempio nell'assegnazione indichiamo nomi e numeri, questi ultimi verranno convertiti in caratteri.

Una funzione molto utile per maneggiare stringhe, ma anche altri risultati, è la funzione `paste()` che concatena dei vettori dopo averli trasformati in stringhe.

```
paste(1:12)
```

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

```
(nth <- paste0(1:12, c("st", "nd", "rd", rep("th", 9))))
```

```
##  [1] "1st"  "2nd"  "3rd"  "4th"  "5th"  "6th"  "7th"  "8th"  "9th"  "10th"
## [11] "11th" "12th"
```

Questa funzione può essere molto utile, ad esempio, se si devono creare vettori di nomi per delle variabili.

Per capire il tipo di dati contenuto in un vettore possiamo usare il comando `typeof()` o `class()`.

2.10 Liste e data frame

Possiamo pensare ad una matrice come ad un metodo efficace per immagazzinare informazioni numeriche, ed è così! Però se il tipo di informazioni che dobbiamo immagazzinare è misto, ad esempio contiene sia numeri che caratteri, la loro efficienza viene meno e si possono preferire altri metodi.

Una lista è un insieme ordinato di oggetti. Si possono definire liste usando il comando `list()`.

```
c <- list(destinazione = c("London", "Madrid"), compagnia = c("Ryanair", "EasyJet"), costo = c(60, 80), va
```

Si può accedere al contenuto di una lista sia per posizione con le doppie parentesi `c[[2]]` oppure per nome `c[["compagnia"]]` o `c$compagnia`. Tutti i precedenti comandi restituiscono il contenuto della lista definito da *compagnia*. Se vogliamo accedere direttamente ad un elemento possiamo usare indifferentemente i comandi `c[[2]][2]`, `c[["compagnia"]][2]` o `c$compagnia[2]`.

2.10.1 Data frame

Un'altra struttura dati che permette di immagazzinate dati di tipo misto sono i *data frame*. Questa struttura è di gran lunga la più usata per leggere e manipolare dati in R.

I *data frame* sono liste di tipo "data.frame" contenenti variabili con lo stesso numero di righe, il cui identificativo è univoco.

```
L3 <- LETTERS[1:3]
fac <- sample(L3, 10, replace = TRUE)
(d <- data.frame(x = 1, y = 1:10, fac = fac))
```

```
##    x  y fac
## 1  1  1   B
## 2  1  2   B
## 3  1  3   C
## 4  1  4   C
## 5  1  5   C
## 6  1  6   C
## 7  1  7   B
## 8  1  8   A
## 9  1  9   B
## 10 1 10   A
```

```
data.frame(1, 1:10, fac)
```

```
##    X1 X1.10 fac
## 1   1     1   B
## 2   1     2   B
## 3   1     3   C
## 4   1     4   C
## 5   1     5   C
## 6   1     6   C
## 7   1     7   B
## 8   1     8   A
## 9   1     9   B
## 10  1    10   A
```

```
## The "same" with automatic column names:
data.frame(rep(1,10), 1:10, fac)
```

```
##    rep.1..10. X1.10 fac
## 1           1     1   B
## 2           1     2   B
## 3           1     3   C
## 4           1     4   C
## 5           1     5   C
## 6           1     6   C
## 7           1     7   B
## 8           1     8   A
## 9           1     9   B
## 10          1    10   A
```

```
data.frame(1, 1:10, sample(L3, 10, replace = TRUE))
```

```
##    X1 X1.10 sample.L3..10..replace...TRUE.
## 1   1     1                               C
## 2   1     2                               B
## 3   1     3                               C
## 4   1     4                               B
## 5   1     5                               A
## 6   1     6                               C
## 7   1     7                               B
## 8   1     8                               A
## 9   1     9                               A
## 10  1    10                               C
```

2.10.2 Tidy data, please!

> A huge amount of effort is spent cleaning data to get it ready for analysis, but there has been little research on how to make data cleaning as easy and effective as possible. This paper tackles a small, but important, component of data cleaning: data tidying. **Tidy datasets** are easy to manipulate, model and visualize, and **have a specific structure: each variable is a column, each observation is a row, and each type of observational unit is a table**. This framework makes it easy to tidy messy datasets because only a small set of tools are needed to deal with a wide range of un-tidy datasets. This structure also makes it easier to develop tidy tools for data analysis, tools that both input and output tidy datasets. The advantages of a consistent data structure and matching tools are demonstrated with a case study free from mundane data manipulation chores.
>
> Tidy data - Hadley Wickham

Visto l'importanza dei dati nel mondo moderno e l'innegabile importanza del tempo nella nostra società, evitiamo di perdere troppo tempo a risistemare i nostri dati e cerchiamo di uniformarci ai *tidy data* da quando creiamo un dataset in poi. Questo ci farà risparmiare molto tempo e migliorerà la ripetibilità delle nostre analisi!

2.11 Programmazione in R

R permette di eseguire comandi che permetto l'iterazione, la valutazione di espressioni condizionali e la definizione di funzioni.

2.11.1 Condizioni: if

```
if (condizione) comando1 else comando2
#Oppure
ifelse(condizione,comando1,comando2)
```

La prima espressione verifica una *condizione* **solo** su singolo elemento. Al contrario, il comando `ifelse` permette di vettorializzare il controllo. Se la *condizione* è va valutata su un vettore, il comando `ifelse` la valuta su tutte le entrate e applica *comando1* se soddisfatta per quell'entrata e *comando2* in caso contrario. Qualora venisse passato un vettore all'espressione `if`, questo valuterebbe la *condizione* solo rispetto al primo elemento del vettore ed eseguirebbe il comando opportuno.

Si possono raggruppare più comandi usando le parentesi graffe e il punto e virgola. Ad esempio, nell'`if` è possibile in questo modo fare più operazioni per ogni caso.

2.11.2 Iterazioni: for - while - repeat

Si possono iterare dei comandi usando i comandi `for`, `while`o `repeat`.

```
for (i in sequenza) comando1
```

`for` permette di iterare comando1 al variare di una variabile, in questo caso *i*. Questa espressione è molto utile quando si visitano dei vettori e alle diverse entrate viene applicato comando1. Usando le parentesi graffe è possibile indicare più comandi.

```
while (condizione) comando1
```

`while` ripete comando1 finche la condizione è vera. L'uso di `while` può essere rischioso qualora la condizione sia sempre soddisfatta e si rimanga bloccati in un *loop infinito*.

`repeat` ripete semplicemente un comando. `break` permette di interrompere qualsiasi iterazione ed è l'unico modo per fermare un ciclo `repeat`.

2.11.3 Definire funzioni

R permette all'utente di definire funzioni, attraverso il comando `function`.

```
nomeFunzione <- function( lista_argomenti ) comando1
return(valore)
```

La sintassi precedente permette di definire una funzione chiamata *nomeFunzione* che valuterà comando1 e restituirà *valore*.

L'uso di funzioni *user-defined* permette di richiamare nei passaggi successivi la funzione che è stata definita passando diversi argomenti. Usando le parentesi graffe è possibile specificare più comandi.

```
my_fun <- function( a , b , c) {
return(a*b + c)
}
```

```
my_fun2 <- function( a , b , c) {
    y <- a**b
return(y + a*b + c)
}
```

Si possono salvare funzioni e script (usando l'estensione .R), che possono essere poi eseguiti attraverso il comando `source("nome_della_funzione.R")`. Per richiamare tali funzioni è essenziale che siano nella cartella di lavoro, altrimenti va indicato il path per raggiungere il file.

2.12 Esercizi

2.12.1 Esercizio 1

Si trovino 2 modi differenti per calcolare la somma dai cavalli vapore (hp) di quelle auto contenute in mtcars che hanno più di 100 cavalli (hp). *Suggerimento:* In un caso si definisca una funzione apposita (senza usare funzioni di libreria) e nell'altro si usino i comandi visti, così da ottenere il risultato con una sola riga di comandi.

2.12.2 Esercizio 2

Si definisca una funzione chiamata my_max che senza usare funzioni di libreria restituisce il massimo di un vettore.

2.12.3 Esercizio 3

Si definisca una funzione chiamata funzione my_mean che senza usare funzioni di libreria restituisce la media di un vettore.

2.12.4 Esercizio 4

Si definisca una funzione chiamata my_cov che verifica se una matrice può essere la matrice di covarianza di due variabili aleatorie.

2.13 Link utili

- Esistono molti corsi online gratuiti o a pagamento che insegnano le basi di R. Tra i tanti segnalo quello di Data Camp che è gratuito (in inglese). Come questo ne esistono innumerevoli, sia in inglese che in italiano.

Chapter 3

Lavorare con i dati

Before Starting (again)!

Abbiamo già visto come spostare la *working directory* in R Studio. Usando i comandi `getwd()` e `setwd()` ricordiamoci di selezionare la cartella relativa a questo corso. Per la lezione di oggi vediamo di organizzare la cartella

del corso nel seguente modo. La cartella, chiamata *R*, deve contenere a sua volta due cartelle: *data* e *source*. Nella prima salveremo i dataset che ci interessano, nella seconda lo script R della lezione.

3.1 Pacchetti

R, come molti altri linguaggi di programmazione, ha un core di funzioni già scritte che sono disponibili all'utente, come quelle che abbiamo visto nelle lezioni precedenti. Tuttavia, con il passare del tempo, sono state sviluppate sempre più funzioni che risolvono nuovi problemi e aumentano le potenzialità di R. Tuttavia non è possibile, ne desiderabile, aggiungere tutte queste funzioni al pacchetto base di R: questo lo renderebbe più lento, occuperebbe più memoria e sarebbe pieno di funzioni che l'utente medio non userebbe!

Questa filosofia è applicata a molti altri linguaggi di programmazione che propongono un set fondamentale di funzioni accessibile a tutti (*base*), e la possibilità di istallare o caricare delle estensioni per svolgere ulteriori compiti particolari.

In R queste estensioni sono i *pacchetti* (package). Si può accedere alla lista dei pacchetti installati usando il comando `library()`, senza indicare argomento. Se si indica come argomento il nome di un pacchetto, R caricherà il pacchetto ed il suo contenuto sarà disponibile per l'utente.

Ad esempio, possiamo caricare il pacchetto *MASS*, che ci sarà utile in seguito, con il comando `library(MASS)`.

Non tutti i pacchetti sono già installati in R, ad esempio il pacchetto *hflights* non lo è.

```
library(hflights)
```

Il comando da errore in quanto il pacchetto non è installato e lo possiamo installare usando il comando:

```
install.packages('hflights')
library(hflights)
# ora il pacchetto viene caricato senza errori!
```

Alcuni pacchetti potrebbero dipendere da altri pacchetti non installati per funzionare. Per indicare a R di installare anche le dipendenze, basta inserire nel comando, dopo il nome del pacchetto l'istruzione `install.packages( nome_pacchetto , dependencies = TRUE)`.

Il comando `search()` elenca i pacchetti caricati nella sessione.

I pacchetti possono essere prodotti da qualsiasi utente e solitamente sono reperibili usando il comando indicato sopra. Di default R Studio si appoggia al sito cran per reperire i pacchetti.

Qualora il pacchetto desiderato non fosse presente, è possibile indicare un diverso server. Ad esempio, il comando seguente permette di installare un pacchetto presente nei server di Bioconductor.

```
source("https://bioconductor.org/biocLite.R")
biocLite("RnaSeqTutorial")
```

Analogamente, R Studio permette di installare i pacchetti manualmente, dal menù **Tools | Install Packages... | Install from: | Package Archive File**.

L'uso del nome del pacchetto seguito da `::`, permette di accedere le funzioni e i dataset contenuti nel pacchetto. Questo è molto utile nel caso di omonimia tra funzioni. Specificando il pacchetto si può essere sicuri di usare la funzione desiderata.

3.2 Manipolare dati

R fornisce alcuni dataset già disponibili all'uso, mentre altri se ne possono aggiungere usando specifici pacchetti.

Ad esempio, nella lezione precedente abbiamo lavorato con il dataset *mtcars* che è già disponibile. Usando i comandi precedenti abbiamo in realtà richiamato il pacchetto MASS che contiene diversi dataset e installato *hflights* che contiene un altro dataset che ci tornerà utile.

3.2.1 Leggere e scrivere file

Quando si parla di dati o dataset, è possibile acquisirne in diversi modi, non solo attraverso i pacchetti. Uno dei modi più semplici e intuitivi è la lettura da file. Esistono molti formati di che possono contenere dati. I più semplici e diffusi sono i file *.csv* (comma sepatated values) o i *.tsv* (tab separated values).

Uno dei comandi più usati per leggere file è il comando `read.table()`. Questo permette di leggere un file e di salvarlo come data frame. Facciamo un esempio pratico.

Consideriamo il dataset body temperature che contiene i dati relativi alla temperatura corporea di alcuni pazienti. Salviamo il dataset nella cartella *data* e leggiamolo usando la funzione `read.table()`.

```
BodyTemperature <- read.table(file = "./data/BodyTemperature.txt", header = TRUE, sep = " ")
```

La funzione read.csv accetta anche *url* come argomento. In questo modo R scaricherà da solo il dataset e lo salverà nella variabile indicata.

```
BodyTemperature <- read.csv(url('http://extras.springer.com/2012/978-1-4614-1301-1/BodyTemperature.txt'), s
head(BodyTemperature)
```

```
##   Gender Age HeartRate Temperature
## 1      M  33        69        97.0
## 2      M  32        72        98.8
## 3      M  42        68        96.2
## 4      F  33        75        97.8
## 5      F  26        68        98.8
## 6      M  37        79       101.3
```

Gli argomenti del comando, oltre a specificare il percorso in cui si trova il file da leggere, indicano che sono presenti degli *header*, cioè intestazioni che riportano il nome delle variabili, e che il separatore usato nel file per indicare i diversi elementi delle righe è lo spazio. Lo spazio è il separatore di default di `read.table()` ma è possibile indicarne di diversi. Se il file fosse stato un csv (*comma separated values*) avremmo indicato `sep=","`. Allo stesso modo avremmo potuto usare la funzione `read.csv()` che considera la virgola come separatore di default. Se i valori fossero stati separati da punto e virgola avremmo potuto usare `read.csv2()` e `read.delim()` se separate da tab ().

In modo del tutto simile, la funzione `write.table()` permette di scrivere un data frame (o una variabile) in un file. Ad esempio, il comando seguente scrive il dataset scaricato da internet nella cartella data come file csv.

```
write.csv(BodyTemperature, "./data/BodyTemperature.txt")
```

3.2.2 Esplorare i file

Una volta letto o caricato il dataset che ci interessa è ora di iniziare ad analizzare i dati che contiene. Spesso i dataset sono accompagnati da delle descrizioni che ne spiegano il contenuto ed è buona regola leggerle.

Possiamo visualizzare la testa del dataset o la coda, usando i comandi `head()` e `tail()` o anche visualizzare l'intero dataset in una finestra di R Studio usando il comando `View()`. Possiamo conoscerne le dimensioni con la funzione `dim()` e possiamo conoscere i nomi delle colonne con la funzione `names()`. `str()` restituisce invece informazioni sulla struttura del dataset. In R Studio, l'equivalente di questo comando è accessibile nella finesta **Environment**. L'uso di queste funzioni è sempre utile per iniziare a capire i dati con cui si ha a che fare.

Una funzione utilissima è `summary()`. Questa restituisce diverse informazioni per le *variabili numeriche*: minimo, massimo, media e i quartili. Va specificato *numeriche* in quanto un data frame potrebbe anche contenere delle variabili non numeriche. Nel caso di BodyTemperature ad esempio, la prima colonna è la variabile categorica del sesso del paziente.

Altre funzione utili sono le funzioni come `mean()` e `sd()` che ci restituiscono media e standard deviation dell'argomento passato. Le funzioni `IQR()` e `range()` restituiscono la distanza interquartile ed il range.

Un utile comando per applicare funzioni pensate per elementi scalari agli elementi di un array o una matrice è la funzione `apply()`. Tale funzione è utile anche per applicare funzioni lungo una sola dimensione. Le varianti

`lapply()` e `tapply()` funzionano in modo simile. Un altro comando per la manipolazione di dati è `round()` che arrotonda un valore o un vettore all'intero più vicino, o indicando il numero di cifre decimali, arrotonda fino alle cifre decimali indicati.

Altre funzioni utili sono `sort()`, che ordina gli elementi di un vettore, `unique()` che rimuove le entrate ripetute da array o data frame. Le funzioni come `any()` e `which()` verificano se una condizione è soddisfatta. La prima restituisce `TRUE` se almeno un elemento che verifica la condizione, mentre `which()` che riporta l'indice degli elementi che la soddisfano.

Vediamo alcuni esempi con il dataset `BodyTemperature`.

3.2.3 Factors

A volte alcune variabili categoriche sono salvate come variabili numeriche. Esploriamo ad esempio il dataset *birthwt* (birth weight) contenuto nel pacchetto MASS. Possiamo notare che alcune variabili, sebbene siano numeriche, rappresentano delle categorie: ad esempio fumatori o no, o ipertesi o no. Tuttavia, se usiamo il comando `summary()`, vediamo che vengono trattate come variabili numeriche. Il comando `as.factor()` permette di convertire variabili numeriche in variabili categoriche.

```
library(MASS)
birthwt1 <- birthwt
birthwt1$race <- as.factor(birthwt1$race)
```

Dalla descrizione delle variabili fornita da R Studio nella finestra **Environment**, possiamo vedere che dopo l'assegnazione, race è indicato come factor, non più integer.

La funzione `table()` costruisce una contingency table tra le combinazioni di factors mentre `levels()` restituisce gli attributi di livello di una variabile. Usando `levels()` è anche possibile assegnare o variare gli attributi di livello.

3.3 Missing data

Spesso i dati forniti contengono valori non affidabili, oppure ci sono errori in qualche riga o ancora mancano dei valori. Questo è un grande problema, sia perché, se non riconosciuti in anticipo, questi valori possono influenzare i nostri risultati, sia perché non esistono protocolli standard per affrontare questi casi.

Alcuni elementi posso non essere stati inseriti ed essi solitamente vengono segnalati con NA (not available). R ha una funzione dedicata per trovare questi elementi: `is.na()`. Tale funzione restituisce valori booleani per indicare se il contenuto è NA o no. Analogamente, `na.omit()` restituisce l'argomento privato delle righe contenenti NA.

In modo del tutto analogo funziona il comando `is.nan()` che però verifica se il contenuto è NaN (not a number). Esistono anche i controlli `is.infinite()` e `is.finite()` che verificano se ci sono valori `Inf/-Inf`.

3.4 Manipolare i dati

Abbiamo ampiamente visto come lavorare e manipolare i dati. Tuttavia un importante funzione che permette di raggruppare i dati non è stata spiegata in classe: si tratta di `aggregate()`. Tale comando, permette di applicare una funzione ad un sottoinsieme dei dati, seguendo uno schema preciso. Vediamo alcuni esempi.

Altro utile comando è `merge()` che permette di unire dataframe indicando il campo da usare per confrontare le entrate. In modo più semplice, e dopo essersi assicurati che le osservazioni siano "allineate", è possibile usare anche i comandi `rbin()` e `cbind()` che sono già stati trattati.

3.5 Distribuzioni di probabilità

R ci permette di accedere a molte distribuzioni di probabilità attraverso delle apposite funzioni che sono contenuto nel pacchetto *stast*, che va quindi caricato.

Le distribuzioni sono accessibili attraverso delle funzioni che richiamano il nome della distribuzione, ad esempio con *norm*, *binom* e *gamma*, solo per citarne alcune, si accede alla distribuzione normale, binomiale e gamma. A tale nome si aggiunge un *prefisso*, che serve a specificare se si è interessati alla funzione densità (d), alla distribuzione (p), alla funzione che riporta i quantili (q) o se si vogliono generare dei numeri usando tale distribuzione (r).

Cercando *distribution* nell'help è possibile accedere all'elenco di tutte le distribuzioni di R. Il fatto che R permetta di accedere facilmente alle distribuzioni tornerà particolarmente utile quando lavoreremo con gli intervalli di confidenza, e potremmo interrogare R, piuttosto che lavorare con la versione tabulare delle stesse.

Va segnalata, almeno per la distribuzione normale, che esiste la funzione `qqnorm` che permette di confrontare graficamente i quantili dei propri dati con quelli della distribuzione normale. Questo è un modo per verificare graficamente (e quindi non formalmente) se i propri dati seguono una distribuzione normale. Insieme alla richiesta che le misurazioni siano indipendenti, queste due sono delle ipotesi che abbiamo visto molto spesso, specialmente nell'uso di teoremi limite.

Chapter 4

Visualizzare i dati

This John Snow knew something!

La visualizzazione dei dati permettere di individuare trend, connessioni e carpire informazioni dai dati che in forma tabulare non sono ovvie.

I comandi che seguono aiutano in tali indagini. Tutti questi comandi possono essere personalizzati usando diversi colori, personalizzando le label degli assi o affiancando le immagini.

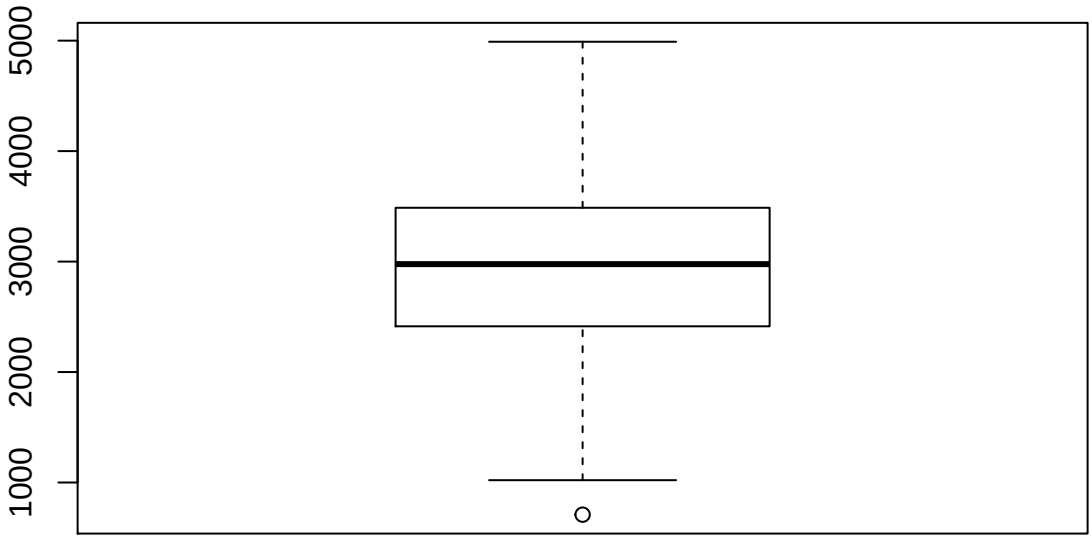
Un pacchetto che permette non solo di produrre immagini (un po' più belle), ma anche di utilizzare funzioni avanzate è ggplot2 che però non tratteremo in questo corso.

Oltre ad introdurre alcune funzioni che permettono di ottenere dei grafici, esploreremo anche alcuni dataset.

4.1 Box plot

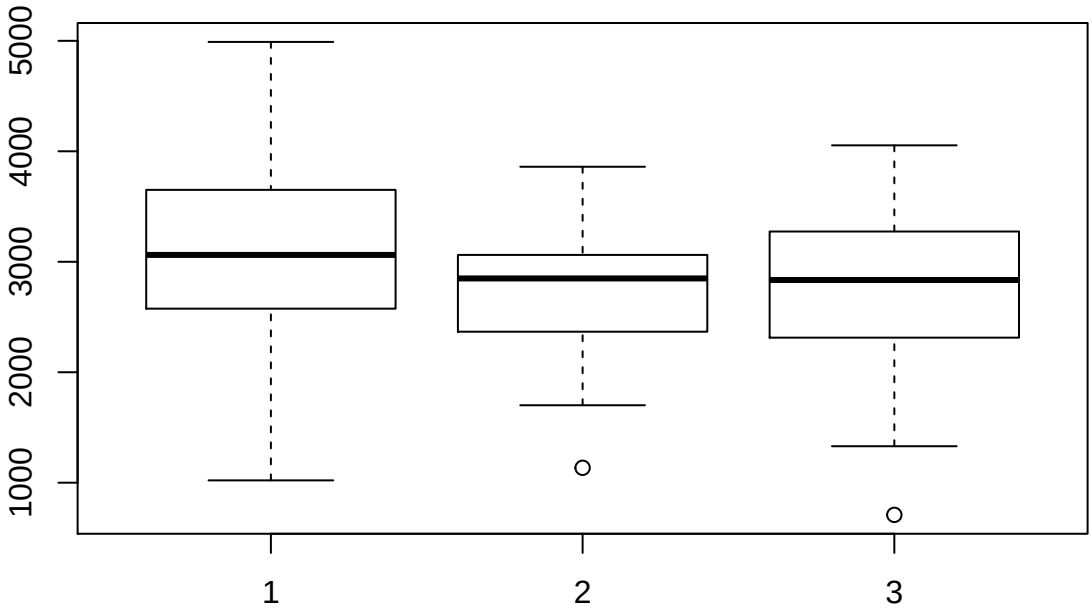
A lezione abbiamo già incontrato i boxplot. Usando R è molto semplice produrre dei boxplot usando gli appositi comandi. Vediamoli usando il dataset *birthwt*, contenuto in MASS.

```
library(MASS)
boxplot(birthwt$bwt)
```



La seguente notazione permette di produrre più boxplot relativi a bwt, divisi rispetto alla variabile race.

```
boxplot(bwt ~ race, data = birthwt )
```



Esercizio Si investighi la variabile lwt rispetto a race e smoke.

Esercizio Si noti che il pacchetto *car* contiene una funzione chiamata `Boxplot()`, con la maiuscola, che vicino ai potenziali outliers indica l'indice dell'elemento. Si replichino i precedenti boxplot usando tale funzione.

4.2 Istogrammi e scatter plost

Usando il comando `hist()` è possibile ottenere degli istogrammi per le variabili, passando gli argomenti in modo analogo ai boxplot.

Analogamente, con i comandi `plot(x,y)` or `scatterplot(x,y)` è possibile rappresentare i dati come punti.

Vedremo che questo può essere molto utile per ottenere degli indizi sulle relazioni tra variabili.

4.3 Analizzare il dataset Body Temperature

BodyTemperature è un dataset che contiene la misurazione di quattro variabili (sesso, età, frequenza cardiaca e temperatura corporea ascellare) di 100 pazienti. Possiamo leggere importare il file specificando invece che il percorso, l'indirizzo url.

```
BodyTemperature <- read.csv(url('http://extras.springer.com/2012/978-1-4614-1301-1/BodyTemperature.txt'))
```

Se però visualizziamo questo dataset qualosa non torna. Il comando giusto specifica quale separatore viene considerato: in questo caso lo spazio, che è diverso da quello di default di `read.csv`. In modo analogo avremmo potuto leggere i dati usando `read.table()`. Tuttavia, controllaimo bene l'output per capire se effettivamente il dataset è stato letto nel modo giusto!

```
BodyTemperature <- read.table(url('http://extras.springer.com/2012/978-1-4614-1301-1/BodyTemperature.txt'))
```

```
BodyTemperature <- read.csv(url('http://extras.springer.com/2012/978-1-4614-1301-1/BodyTemperature.txt'), s
```

Esploriamo un po' il dataset per capirne il contenuto e le variabili

```
str(BodyTemperature)
```

```
## 'data.frame':    100 obs. of  4 variables:
##  $ Gender     : Factor w/ 2 levels "F","M": 2 2 2 1 1 2 1 1 1 2 ...
##  $ Age        : int  33 32 42 33 26 37 32 45 31 49 ...
##  $ HeartRate  : int  69 72 68 75 68 79 71 73 77 81 ...
##  $ Temperature: num  97 98.8 96.2 97.8 98.8 ...
```

```
head(BodyTemperature)
```

```
##   Gender Age HeartRate Temperature
## 1      M  33        69        97.0
## 2      M  32        72        98.8
## 3      M  42        68        96.2
## 4      F  33        75        97.8
## 5      F  26        68        98.8
## 6      M  37        79       101.3
```

```
names(BodyTemperature)
```

```
## [1] "Gender"      "Age"         "HeartRate"   "Temperature"
```

Solo a scopo didattico, vediamo l'applicazione di uno dei comandi visti: `levels()`:

```
BodyTemperature$GenderLong <- BodyTemperature$Gender
levels(BodyTemperature$GenderLong) <- c("Female", "Male")
```

```
summary(BodyTemperature)
```

```
##  Gender      Age          HeartRate      Temperature
##  F:51   Min.   :21.00   Min.   :61.00   Min.   : 96.20
##  M:49   1st Qu.:33.75   1st Qu.:69.00   1st Qu.: 97.70
##         Median :37.00   Median :73.00   Median : 98.30
```

```
##         Mean   :37.62   Mean   :73.66   Mean   : 98.33
##         3rd Qu.:42.00   3rd Qu.:78.00   3rd Qu.: 98.90
##         Max.   :50.00   Max.   :87.00   Max.   :101.30
```

```
range(BodyTemperature[,-1])
```

```
## [1]  21.0 101.3
```

In modo analogo possiamo cercare di ottenere la distanza interquartile per i gli elementi del dataset con `IQR(BodyTemperature[,2:4])`, tuttavia questo ci da errore.

Possiamo aggirare questa limitazione usando il comando `apply()`. La stessa funzione la possiamo usare per individuare, ad esempio, il valore minimo rispetto alle variabili (numeriche) e quale indice lo assume.

```
apply(BodyTemperature[,-1], 2, IQR )
```

```
##         Age   HeartRate Temperature
##        8.25        9.00        1.20
```

```
apply(BodyTemperature[,-1], 2, min )
```

```
##         Age   HeartRate Temperature
##        21.0        61.0        96.2
```

```
apply(BodyTemperature[,-1], 2, which.min )
```

```
##         Age   HeartRate Temperature
##          37          28           3
```

Possiamo notare che la temperatura non è espressa in gradi Celsius (°C), bensì in Fahrenheit (F). Usando delle semplici operazioni di base possiamo trasformare le temperature in gradi Celsius e salvare questi nuovi dati nella colonna che chiamiamo *TempC*.

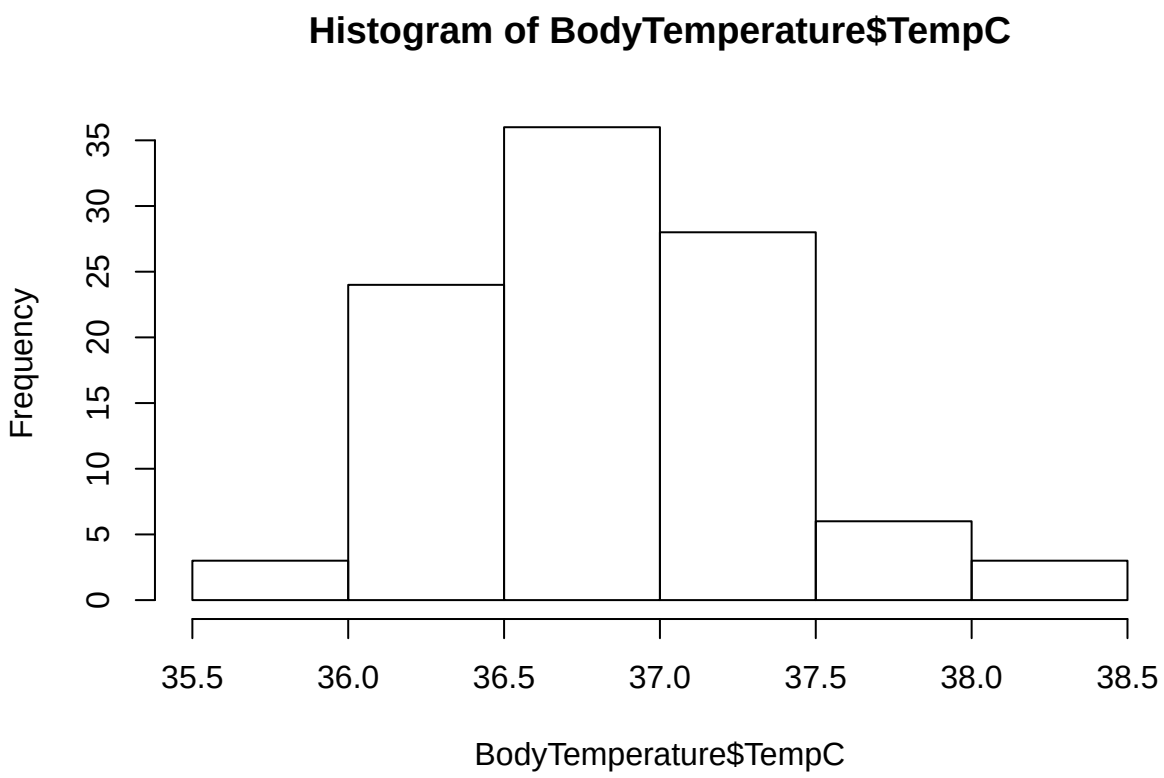
```
fromFtoC <- function(dataF) {
return((dataF - 32)*(5/9))
}
BodyTemperature$TempC <- (BodyTemperature$Temperature -32)*5/9

head(BodyTemperature)
```

```
##   Gender Age HeartRate Temperature    TempC
## 1      M  33        69        97.0 36.11111
## 2      M  32        72        98.8 37.11111
## 3      M  42        68        96.2 35.66667
## 4      F  33        75        97.8 36.55556
## 5      F  26        68        98.8 37.11111
## 6      M  37        79       101.3 38.50000
```

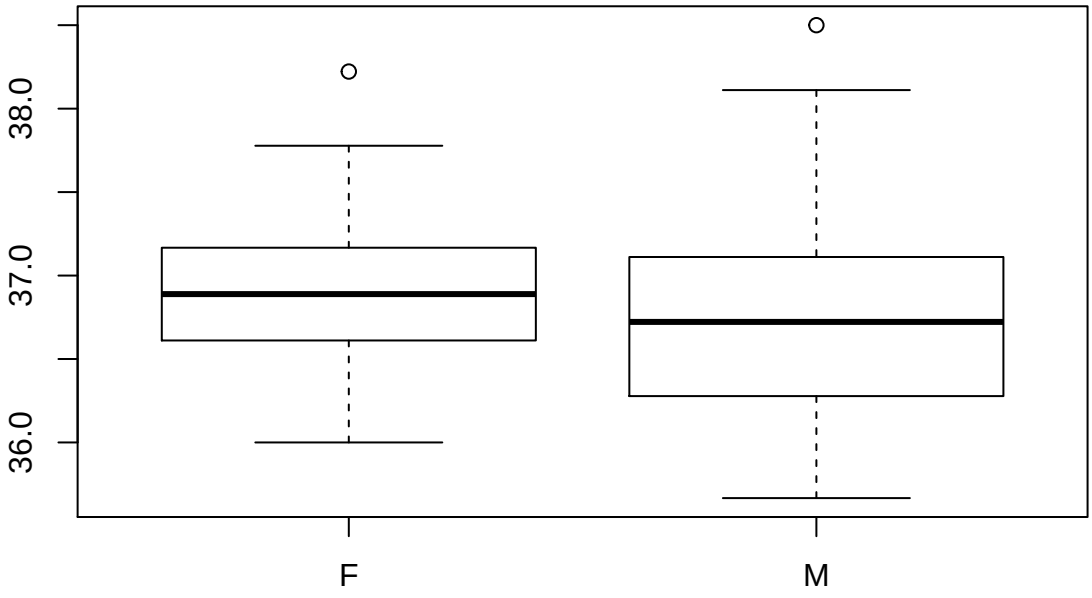
4.4 Visualizzare il dataset BodyTemperature

Passiamo ora a visualizzare il contenuto del dateset.

```
hist(BodyTemperature$TempC )
```

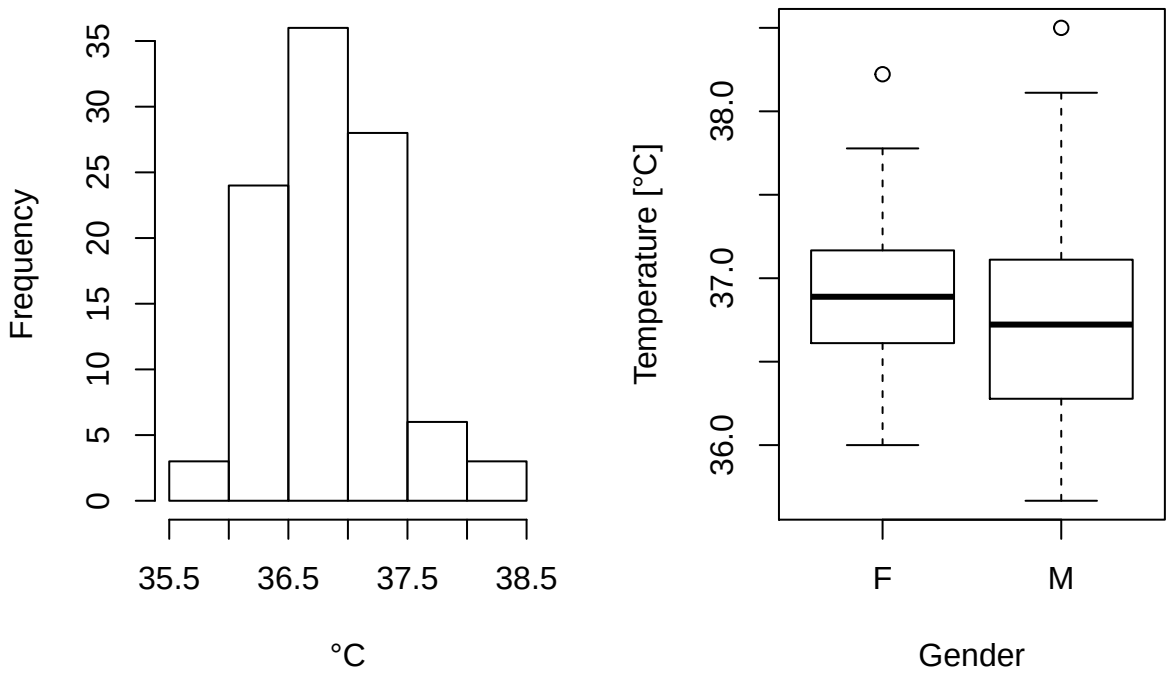



```
boxplot(TempC ~ Gender, data = BodyTemperature)
```

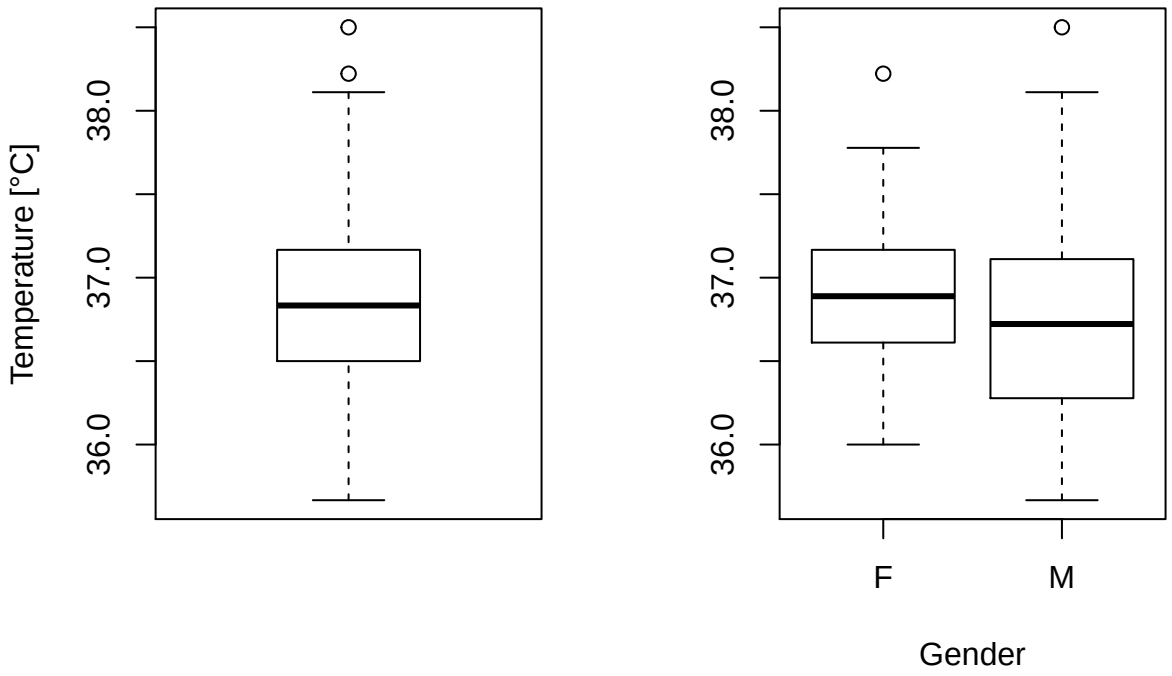


Vediamo ora come affiancare diversi plot e usiamo alcuni dei parametri delle funzioni di plot.

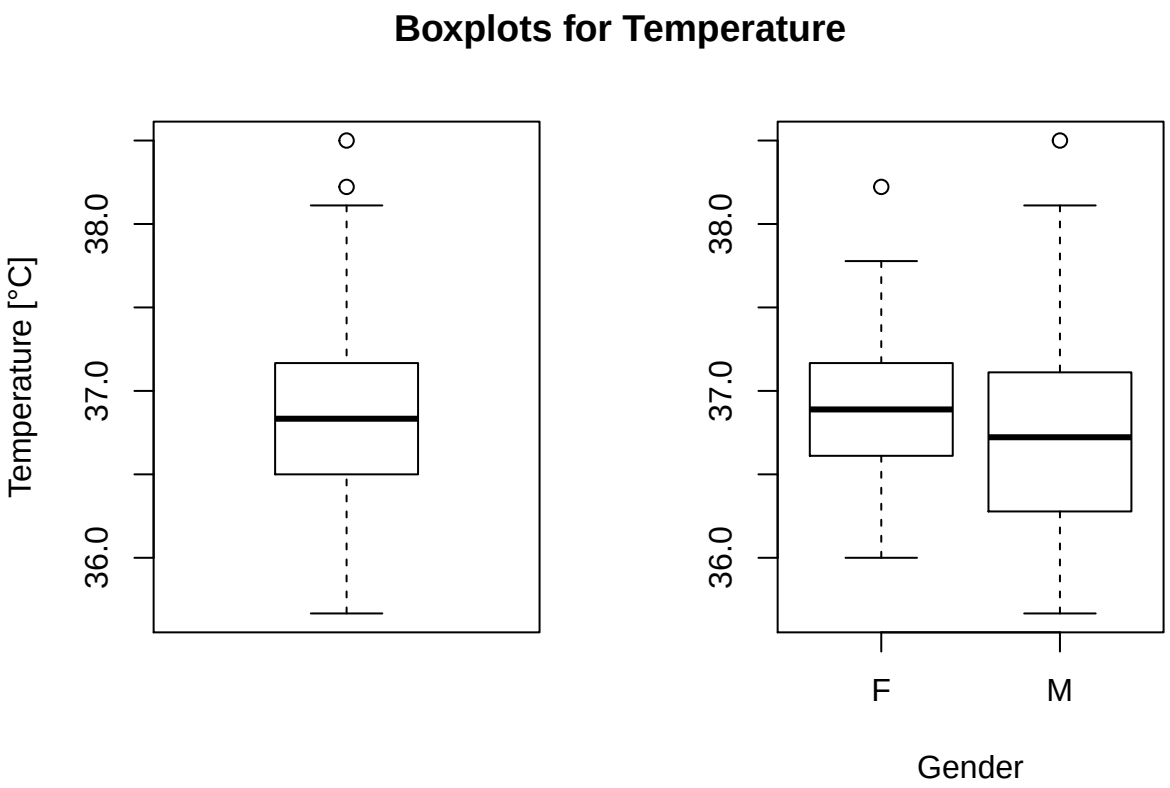
```
par(mfrow=c(1, 2))
hist(BodyTemperature$TempC, main = "", xlab = "°C")
boxplot(TempC ~ Gender, data = BodyTemperature, xlab = "Gender", ylab = "Temperature [°C]")
```



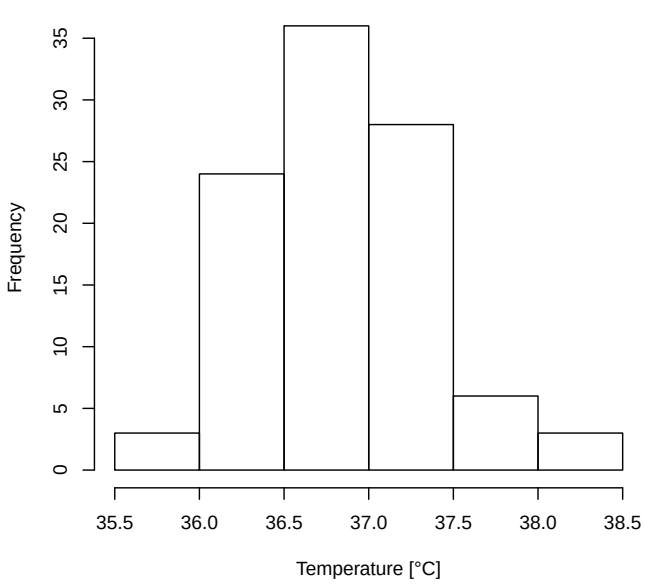

```
par(mfrow=c(1, 2))
boxplot(BodyTemperature$TempC , ylab = "Temperature [°C]")
boxplot(TempC ~ Gender, data = BodyTemperature, xlab = "Gender")
```

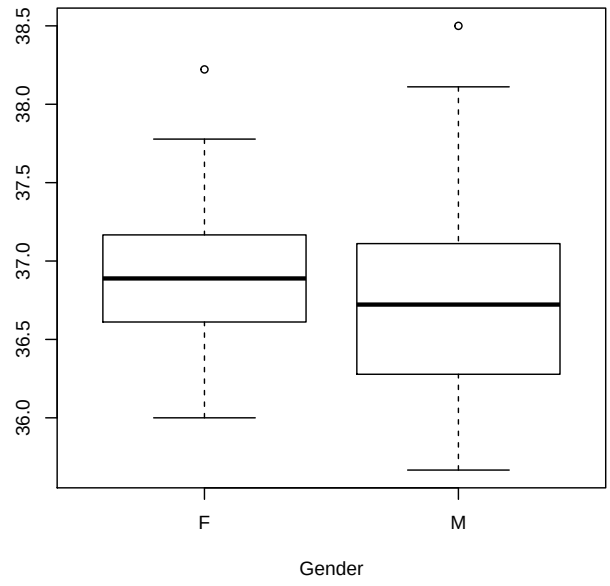


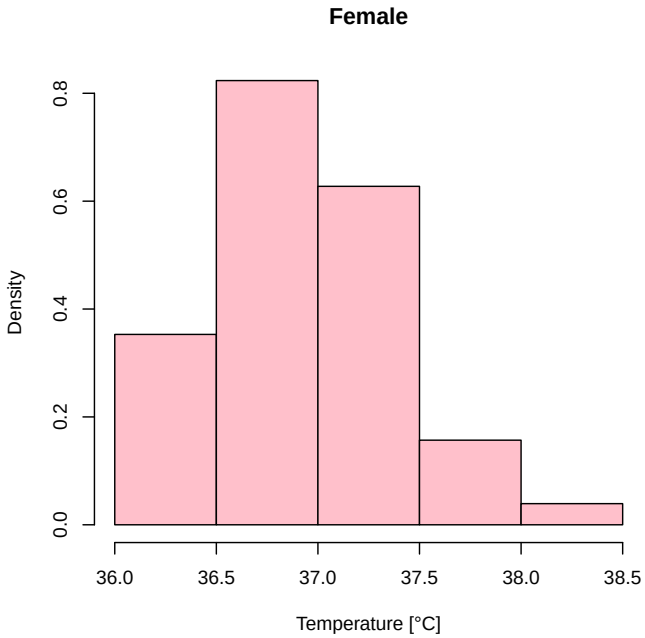

```
par(mfrow=c(1, 2))
boxplot(BodyTemperature$TempC , ylab = "Temperature [°C]")
boxplot(TempC ~ Gender, data = BodyTemperature, xlab = "Gender")
title("Boxplots for Temperature", outer = T, line = -2)
```

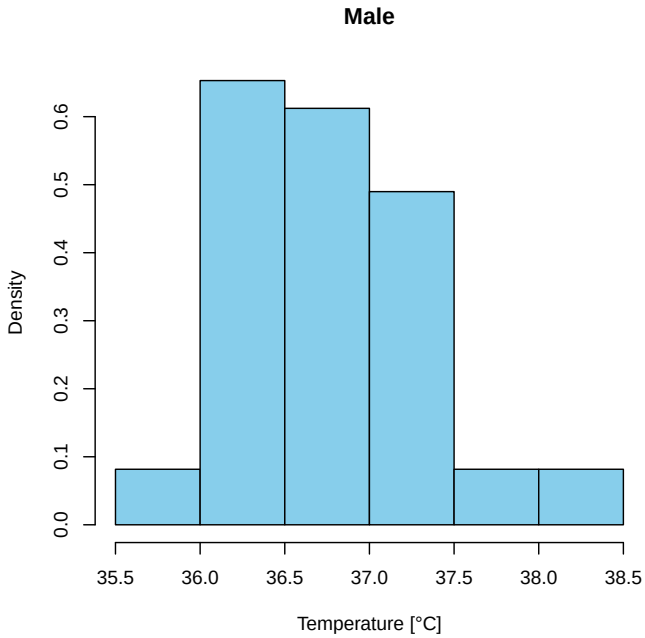



```
par(mfrow=c(2, 2))
hist((BodyTemperature$TempC) , xlab = "Temperature [°C]", main = "")
boxplot(TempC ~ Gender, data = BodyTemperature, xlab = "Gender")
hist(BodyTemperature$TempC[BodyTemperature$Gender == "F"],freq = FALSE ,main = "Female", col = "pink", xla
hist(BodyTemperature$TempC[BodyTemperature$Gender == "M"],freq = FALSE ,main = "Male", col = "skyblue" , x
```









Esistono altri comandi che si possono abbinare a quelli di visualizzazione visti fin qui, ad esempio `abline()`, `text()` e `lines()`.

Esercizio Usare l'help per capire le funzionalità dei comandi precedenti e si usino per aggiungere informazioni ai grafici precedenti.

4.5 Analizzare il dataset Pima.tr2

```
library(MASS) # necessario per accedere al dataset
str(Pima.tr2)
```

```
## 'data.frame':    300 obs. of  8 variables:
##  $ npreg: int  5 7 5 0 0 5 3 1 3 2 ...
##  $ glu  : int  86 195 77 165 107 97 83 193 142 128 ...
##  $ bp   : int  68 70 82 76 60 76 58 50 80 78 ...
```

```
##  $ skin : int  28 33 41 43 25 27 31 16 15 37 ...
##  $ bmi  : num  30.2 25.1 35.8 47.9 26.4 35.6 34.3 25.9 32.4 43.3 ...
##  $ ped  : num  0.364 0.163 0.156 0.259 0.133 ...
##  $ age  : int  24 55 35 26 23 52 25 24 63 31 ...
##  $ type : Factor w/ 2 levels "No","Yes": 1 2 1 1 1 2 1 1 1 2 ...
```

```
dim(Pima.tr2)
```

```
## [1] 300   8
```

```
names(Pima.tr2)
```

```
## [1] "npreg" "glu"   "bp"    "skin"  "bmi"   "ped"   "age"   "type"
```

```
?Pima.tr2
head(Pima.tr2)
```

```
##   npreg glu bp skin  bmi   ped age type
## 1     5  86 68   28 30.2 0.364  24   No
## 2     7 195 70   33 25.1 0.163  55  Yes
## 3     5  77 82   41 35.8 0.156  35   No
## 4     0 165 76   43 47.9 0.259  26   No
## 5     0 107 60   25 26.4 0.133  23   No
## 6     5  97 76   27 35.6 0.378  52  Yes
```

```
summary(Pima.tr2)
```

```
##      npreg              glu              bp              skin
##  Min.   : 0.000   Min.   : 56.0   Min.   : 38.00   Min.   : 7.00
##  1st Qu.: 1.000   1st Qu.:101.0   1st Qu.: 64.00   1st Qu.:21.00
##  Median : 3.000   Median :121.0   Median : 72.00   Median :29.00
##  Mean   : 3.787   Mean   :123.7   Mean   : 72.32   Mean   :29.15
##  3rd Qu.: 6.000   3rd Qu.:142.0   3rd Qu.: 80.00   3rd Qu.:36.00
##  Max.   :14.000   Max.   :199.0   Max.   :114.00   Max.   :99.00
##                                   NA's   :13       NA's   :98
##       bmi             ped              age        type
##  Min.   :18.20   Min.   :0.0780   Min.   :21.0   No :194
##  1st Qu.:27.10   1st Qu.:0.2367   1st Qu.:24.0   Yes:106
##  Median :32.00   Median :0.3360   Median :29.0
##  Mean   :32.05   Mean   :0.4357   Mean   :33.1
##  3rd Qu.:36.50   3rd Qu.:0.5867   3rd Qu.:40.0
##  Max.   :52.90   Max.   :2.2880   Max.   :72.0
##  NA's   :3
```

```
which(is.na(Pima.tr2))
```

```
##   [1]  804  834  836  853  854  863  871  885  887  888  889  898  899 1101
##  [15] 1102 1103 1104 1105 1106 1107 1108 1109 1110 1111 1112 1113 1114 1115
##  [29] 1116 1117 1118 1119 1120 1121 1122 1123 1124 1125 1126 1127 1128 1129
##  [43] 1131 1132 1133 1134 1135 1136 1137 1138 1139 1140 1141 1142 1143 1144
##  [57] 1145 1146 1147 1148 1149 1150 1151 1152 1153 1154 1155 1156 1157 1158
##  [71] 1159 1160 1161 1162 1163 1164 1165 1166 1167 1168 1169 1170 1172 1173
##  [85] 1174 1175 1176 1177 1178 1179 1180 1181 1182 1183 1184 1185 1186 1187
##  [99] 1188 1189 1190 1191 1192 1193 1194 1195 1196 1197 1198 1199 1200 1413
## [113] 1430 1468
```

```
# amico di apply
lapply(lapply(Pima.tr2 , is.na),which)
```

```
## $npreg
## integer(0)
```

```
##
## $glu
## integer(0)
##
## $bp
##  [1] 204 234 236 253 254 263 271 285 287 288 289 298 299
##
## $skin
##  [1] 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217
## [18] 218 219 220 221 222 223 224 225 226 227 228 229 231 232 233 234 235
## [35] 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252
## [52] 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269
## [69] 270 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287
## [86] 288 289 290 291 292 293 294 295 296 297 298 299 300
##
## $bmi
## [1] 213 230 268
##
## $ped
## integer(0)
##
## $age
## integer(0)
##
## $type
## integer(0)
```

Cosa possiamo fare con i diversi NA?

Un'idea ovviamente è quella di rimuovere le righe dove compare un NA

```
data_no_Na <- na.omit(Pima.tr2)

dim(data_no_Na)
```

```
## [1] 200   8
```

Abbiamo rimosso un terzo del dataset…..

Rimuovere così tanti dati impoverisce significativamente il dataset, quindi magari si possono trovare soluzioni meno radicali.

Ad esempio, possiamo notare come gran parte dei NA siano nella variabile *skin*… se questa non ci interessa, possiamo evitare di rimuovere quelle righe!

```
data_no_Na <- na.omit(Pima.tr2[,-4])

dim(data_no_Na)
```

```
## [1] 284   7
```

In questo modo abbiamo rimosso molte meno righe. Si noti che alcune funzioni (vedi mean) hanno specifici argomenti per gestire i NA e si può quindi evitare di rimuoverli dal dataset ma gestirli i vari casi dalle funzioni

In alcuni casi, si preferisce non rimuovere alcun dato, ma piuttosto si sostituiscono i valori mancanti con informazioni prese dai dati correnti.

Ad esempio, si possono sostituire con la media, la mediana o si possono anche definire modelli più complessi per definire i sostituti.

4.6 Visualizzare il dataset Pima.tr2

Usando gli strumenti di visualizzazione visti, si esplorino le relazioni tra le variabili del dataset.

Si faccia attenzione agli NA e si confrontino i risultati nei casi in cui:

- vengono rimosse tutte le righe contenenti NA;
- vengono solo le righe contenenti NA nelle variabili di interesse;
- vengono sostituiti gli NA con la media o la mediana della variabile;

4.7 Esercizi

4.7.1 Esercizio 1

Si scarichi il dataset AstmaLOS.txt (descrizione). Dopo averlo analizzato ed aver individuato eventuali errori nei dati relativi ad age e owner.type, si analizzi e visualizzi la variabile age. Si provino almeno due tecniche di rimozione degli errori dai dati e si confrontino i risultati.

4.7.2 Esercizio 2

Si analizzi il dataset *hflights* e si scelga una destinazione. Si analizzino e visualizzino i dati relativi a lunghezza del volo e ritardi per tale destinazione. Inoltre si determini:

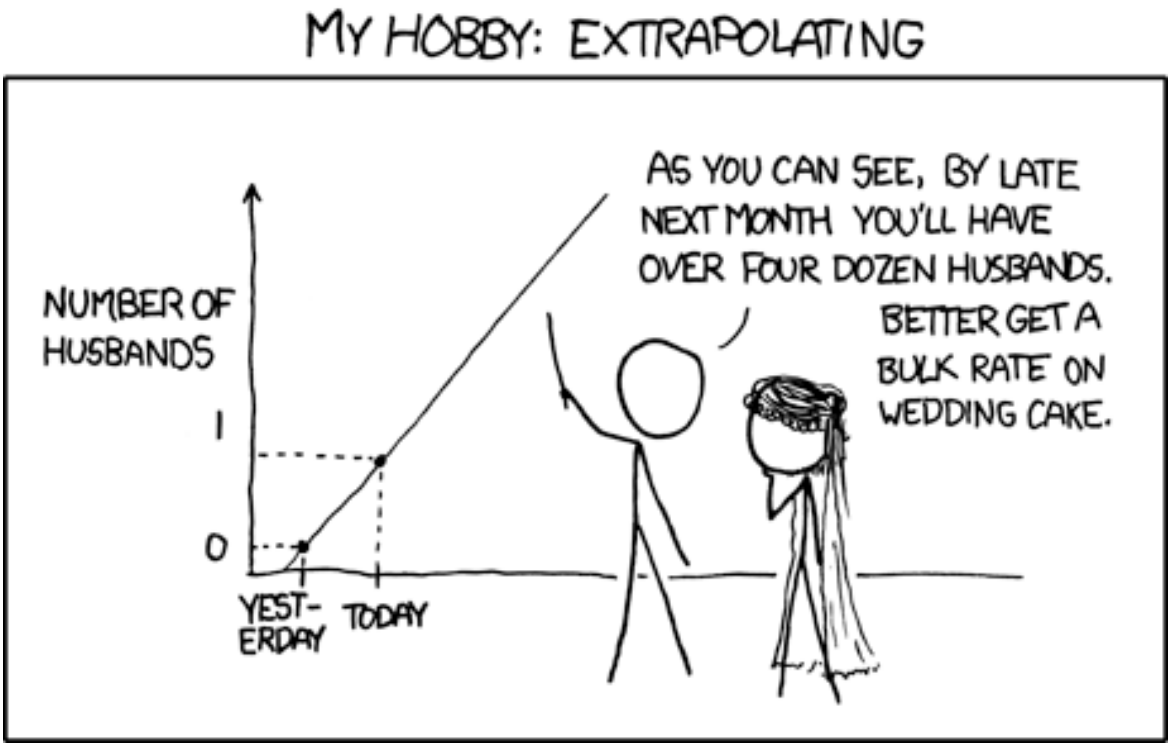
- il giorno in cui è possibile volare con minore probabilità di subire ritardo.
- il giorno in cui è possibile volare con maggiore probabilità di subire la cancellazione del volo.

4.8 Link utili

- Questo sito presenta diversi datasets che possono essere interessanti da esaminare.
- A questo link è possibile trovare gli appunti di un ottimo corso di data visualization (in inglese).

Chapter 5

Fare pratica con i dati



Originally posted here.

5.1 Inquinamento a San Andreas

Iniziamo scaricando il dataset dal link dove ho isolato (e pulito) i dati raccolti dall'agenzia americana per la tutela dell'ambiente, relativi alle rilevazioni orarie della concentrazione delle PM10 nell'aria nella città di San Andreas, CA.

Questo dataset è una parte di un dataset molto più grande che si può trovare al link. Queste analisi sono ispirate a (Peng, 2015).

Una volta salvato il file nella nostra cartella *data* passiamo a leggere il suo contenuto.

```
PM10dataSA <- read.csv("./data/PM10dataSanAndreas.csv")
knitr::kable(
head(PM10dataSA),
```

```
booktabs = TRUE
)
```

X.2	X.1	X	State.Code	County.Code	Site.Num	Parameter.Code	POC	Latitude	Longitude	Datum	P
1	1	308088	6	9	1	81102	3	38.20185	-120.6816	WGS84	P
2	2	308089	6	9	1	81102	3	38.20185	-120.6816	WGS84	P
3	3	308090	6	9	1	81102	3	38.20185	-120.6816	WGS84	P
4	4	308091	6	9	1	81102	3	38.20185	-120.6816	WGS84	P
5	5	308092	6	9	1	81102	3	38.20185	-120.6816	WGS84	P
6	6	308093	6	9	1	81102	3	38.20185	-120.6816	WGS84	P

Possiamo iniziare a prendere confidenza con il dataset leggendo alcune informazioni essenziali, come la dimensione, il nome delle variabili, oppure ne possiamo esplorare la struttura, visualizzare le prime o le ultime righe.

```
dim(PM10dataSA)
```

```
## [1] 2581   28
```

Vediamo subito che il dataset contiene più di 2mila osservazioni per 26 variabili. Accediamo i nomi delle variabili per vedere se riusciamo a capirne il significato.

```
names(PM10dataSA)
```

```
##  [1] "X.2"                "X.1"                "X"
##  [4] "State.Code"         "County.Code"        "Site.Num"
##  [7] "Parameter.Code"     "POC"                "Latitude"
## [10] "Longitude"          "Datum"              "Parameter.Name"
## [13] "Date.Local"         "Time.Local"         "Date.GMT"
## [16] "Time.GMT"           "Sample.Measurement" "Units.of.Measure"
## [19] "MDL"                "Uncertainty"        "Qualifier"
## [22] "Method.Type"        "Method.Code"        "Method.Name"
## [25] "State.Name"         "County.Name"        "Date.of.Last.Change"
## [28] "DateTime.Local"
```

Proseguiamo visualizzando la struttura ed alcune righe dei dati.

```
str(PM10dataSA)
```

```
## 'data.frame':    2581 obs. of  28 variables:
##  $ X.2                : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ X.1                : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ X                  : int  308088 308089 308090 308091 308092 308093 308094 308095 308096 308097 ...
##  $ State.Code         : int  6 6 6 6 6 6 6 6 6 6 ...
##  $ County.Code        : int  9 9 9 9 9 9 9 9 9 9 ...
##  $ Site.Num           : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ Parameter.Code     : int  81102 81102 81102 81102 81102 81102 81102 81102 81102 81102 ...
##  $ POC                : int  3 3 3 3 3 3 3 3 3 3 ...
##  $ Latitude           : num  38.2 38.2 38.2 38.2 38.2 ...
##  $ Longitude          : num  -121 -121 -121 -121 -121 ...
##  $ Datum              : Factor w/ 1 level "WGS84": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Parameter.Name     : Factor w/ 1 level "PM10 Total 0-10um STP": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Date.Local         : Factor w/ 120 levels "2016-01-01","2016-01-02",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ Time.Local         : Factor w/ 24 levels "00:00","01:00",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ Date.GMT           : Factor w/ 122 levels "2016-01-01","2016-01-02",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ Time.GMT           : Factor w/ 24 levels "00:00","01:00",..: 9 10 11 12 13 14 15 16 17 18 ...
##  $ Sample.Measurement : int  25 48 26 14 7 14 11 16 17 11 ...
##  $ Units.of.Measure   : Factor w/ 1 level "Micrograms/cubic meter (25 C)": 1 1 1 1 1 1 1 1 1 1 ...
##  $ MDL                : int  4 4 4 4 4 4 4 4 4 4 ...
##  $ Uncertainty        : logi  NA NA NA NA NA NA ...
```

```
##  $ Qualifier          : logi  NA NA NA NA NA NA ...
##  $ Method.Type        : Factor w/ 1 level "FEM": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Method.Code        : int  122 122 122 122 122 122 122 122 122 122 ...
##  $ Method.Name        : Factor w/ 1 level "INSTRUMENT MET ONE 4 MODELS - BETA ATTENUATION": 1 1 1 1 1 1
##  $ State.Name         : Factor w/ 1 level "California": 1 1 1 1 1 1 1 1 1 1 ...
##  $ County.Name        : Factor w/ 1 level "Calaveras": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Date.of.Last.Change: Factor w/ 4 levels "2016-05-06","2016-05-12",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ DateTime.Local     : Factor w/ 2580 levels "2016-01-01 00:00:00",..: 1 2 3 4 5 6 7 8 9 10 ...
```

```
knitr::kable(
head(PM10dataSA),
booktabs = TRUE
)
```

X.2	X.1	X	State.Code	County.Code	Site.Num	Parameter.Code	POC	Latitude	Longitude	Datum	P
1	1	308088	6	9	1	81102	3	38.20185	-120.6816	WGS84	P
2	2	308089	6	9	1	81102	3	38.20185	-120.6816	WGS84	P
3	3	308090	6	9	1	81102	3	38.20185	-120.6816	WGS84	P
4	4	308091	6	9	1	81102	3	38.20185	-120.6816	WGS84	P
5	5	308092	6	9	1	81102	3	38.20185	-120.6816	WGS84	P
6	6	308093	6	9	1	81102	3	38.20185	-120.6816	WGS84	P

```
knitr::kable(
tail(PM10dataSA),
booktabs = TRUE
)
```

	X.2	X.1	X	State.Code	County.Code	Site.Num	Parameter.Code	POC	Latitude	Longitude	Da
2576	2576	2862	310949	6	9	1	81102	3	38.20185	-120.6816	W
2577	2577	2863	310950	6	9	1	81102	3	38.20185	-120.6816	W
2578	2578	2864	310951	6	9	1	81102	3	38.20185	-120.6816	W
2579	2579	2865	310952	6	9	1	81102	3	38.20185	-120.6816	W
2580	2580	2866	310953	6	9	1	81102	3	38.20185	-120.6816	W
2581	2581	2867	310954	6	9	1	81102	3	38.20185	-120.6816	W

Vediamo che i dati contengono indicazioni come coordinate geografiche e lo stato dove sono stati raccolti i dati. Nel caso del dataset originale, queste informazioni sono essenziali per individuare il luogo di raccolta dati, mentre nel nostro caso potrebbero essere omesse, **dopo** aver verificato che siano consistenti in tutto il dataset.

Siamo interessati a capire l'andamento delle PM10 nel nostro dataset durante il tempo in cui sono stati raccolti i dati. Per prima cosa convertiamo i dati riguardanti le date nel giusto formato, cioè in *Date*. Usiamo per ora solo le informazioni relative al fuso orario locale.

```
PM10dataSA$Date.Local <- as.Date(PM10dataSA$Date.Local)
```

Ora che i dati sono in formato *Date*, R capisce che sono numeri salvati con un particolare formato e non stringhe.

In questo modo possiamo, ad esempio, vedere il periodo temporale che coprono usano i comandi min e max.

```
paste(min(PM10dataSA$Date.Local), which.min(PM10dataSA$Date.Local))
```

```
## [1] "2016-01-01 1"
```

```
paste(max(PM10dataSA$Date.Local), which.max(PM10dataSA$Date.Local))
```

```
## [1] "2016-04-30 2559"
```

Possiamo notare che il dataset non supera Aprile e che il max non è assunto nell'ultima osservazione, che sarebbe la 2867. Usando il comando `tail()` vediamo di capire perché.

```
knitr::kable(
tail(PM10dataSA),
booktabs = TRUE
)
```

	X.2	X.1	X	State.Code	County.Code	Site.Num	Parameter.Code	POC	Latitude	Longitude	Da
2576	2576	2862	310949	6	9	1	81102	3	38.20185	-120.6816	W
2577	2577	2863	310950	6	9	1	81102	3	38.20185	-120.6816	W
2578	2578	2864	310951	6	9	1	81102	3	38.20185	-120.6816	W
2579	2579	2865	310952	6	9	1	81102	3	38.20185	-120.6816	W
2580	2580	2866	310953	6	9	1	81102	3	38.20185	-120.6816	W
2581	2581	2867	310954	6	9	1	81102	3	38.20185	-120.6816	W

```
knitr::kable(
tail(PM10dataSA$Date.Local, 25),
booktabs = TRUE
)
```

x
2016-04-29
2016-04-29
2016-04-30
2016-04-30
2016-04-30
2016-04-30
2016-04-30
2016-04-30
2016-04-30
2016-04-30
2016-04-30
2016-04-30
2016-04-30
2016-04-30
2016-04-30
2016-04-30
2016-04-30
2016-04-30
2016-04-30
2016-04-30
2016-04-30
2016-04-30
2016-04-30
2016-04-30
2016-04-30

Fortunatamente vediamo che tutte le ultime 24 rilevazioni sono state fatte durante lo stesso giorno, pertanto il risultato di `which.min()` non è un errore. Ce lo potevamo aspettare, visto che le misurazioni sono ogni ora, ma è sempre meglio controllare.

Usando la funzione `summary()` estrapoliamo altre informazioni dal dataset.

```
summary(PM10dataSA)
```

```
##       X.2            X.1             X            State.Code  County.Code
##  Min.   :   1   Min.   :   1   Min.   :308088   Min.   :6    Min.   :9
##  1st Qu.: 646   1st Qu.: 771   1st Qu.:308858   1st Qu.:6    1st Qu.:9
```

```
##  Median :1291   Median :1442   Median :309529   Median :6    Median :9
##  Mean   :1291   Mean   :1464   Mean   :309551   Mean   :6    Mean   :9
##  3rd Qu.:1936   3rd Qu.:2192   3rd Qu.:310279   3rd Qu.:6    3rd Qu.:9
##  Max.   :2581   Max.   :2867   Max.   :310954   Max.   :6    Max.   :9
##
##     Site.Num Parameter.Code       POC       Latitude      Longitude
##  Min.   :1   Min.   :81102   Min.   :3   Min.   :38.2   Min.   :-120.7
##  1st Qu.:1   1st Qu.:81102   1st Qu.:3   1st Qu.:38.2   1st Qu.:-120.7
##  Median :1   Median :81102   Median :3   Median :38.2   Median :-120.7
##  Mean   :1   Mean   :81102   Mean   :3   Mean   :38.2   Mean   :-120.7
##  3rd Qu.:1   3rd Qu.:81102   3rd Qu.:3   3rd Qu.:38.2   3rd Qu.:-120.7
##  Max.   :1   Max.   :81102   Max.   :3   Max.   :38.2   Max.   :-120.7
##
##    Datum                     Parameter.Name   Date.Local
##  WGS84:2581   PM10 Total 0-10um STP:2581   Min.   :2016-01-01
##                                            1st Qu.:2016-02-02
##                                            Median :2016-03-01
##                                            Mean   :2016-03-02
##                                            3rd Qu.:2016-04-02
##                                            Max.   :2016-04-30
##
##    Time.Local         Date.GMT       Time.GMT    Sample.Measurement
##  19:00  : 116   2016-01-02:  24   03:00  : 116   Min.   : 1.00
##  20:00  : 116   2016-01-03:  24   04:00  : 116   1st Qu.: 4.00
##  06:00  : 113   2016-01-04:  24   02:00  : 113   Median : 7.00
##  07:00  : 113   2016-01-26:  24   05:00  : 113   Mean   : 8.24
##  18:00  : 113   2016-01-28:  24   14:00  : 113   3rd Qu.:11.00
##  21:00  : 113   2016-01-29:  24   15:00  : 113   Max.   :60.00
##  (Other):1897   (Other)   :2437   (Other):1897
##                       Units.of.Measure      MDL    Uncertainty
##  Micrograms/cubic meter (25 C):2581    Min.   :4   Mode:logical
##                                        1st Qu.:4   NA's:2581
##                                        Median :4
##                                        Mean   :4
##                                        3rd Qu.:4
##                                        Max.   :4
##
##  Qualifier      Method.Type  Method.Code
##  Mode:logical   FEM:2581    Min.   :122
##  NA's:2581                  1st Qu.:122
##                             Median :122
##                             Mean   :122
##                             3rd Qu.:122
##                             Max.   :122
##
##                                          Method.Name        State.Name
##  INSTRUMENT MET ONE 4 MODELS - BETA ATTENUATION:2581   California:2581
##
##
##
##
##
##
##
##     County.Name   Date.of.Last.Change             DateTime.Local
##  Calaveras:2581   2016-05-06:619      2016-01-01 00:00:00:   1
##                   2016-05-12:656      2016-01-01 01:00:00:   1
```

```
##                   2016-06-22:618      2016-01-01 02:00:00:   1
##                   2016-11-08:688      2016-01-01 03:00:00:   1
##                                       2016-01-01 04:00:00:   1
##                                       (Other)            :2575
##                                       NA's               :   1
```

Possiamo notare che alcune delle variabili sono trattate come numeriche anche se dovrebbero essere di tipo Factor. Possiamo o convertirle, o semplicemente tenerlo a mente qualora dovessimo lavorarci. Inoltre potremmo eliminare le colonne che non ci interessano usano l'assegnazione `<- NULL`. Ad esempio, **dopo** aver verificato che latitudine e longitudine sono le stesse in tutto il dataset, e corrispondono alla città di San Andreas, potremmo eliminarle. Va prima verificato che il dataset non contenga dati estranei, altrimenti eliminando una variabile, potremmo non essere più in grado di capirlo!

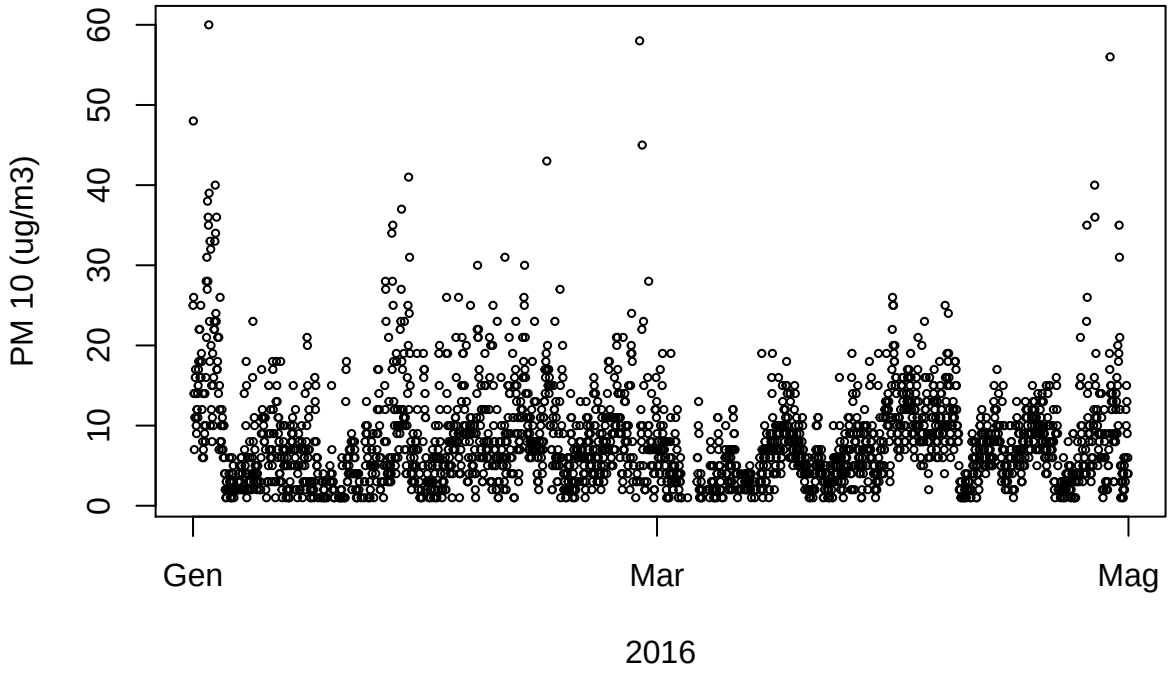
Tale verifica si può fare (in questo caso, ma può dipende dai dati) o usando il comando `unique()`, o anche leggendo i risultati di `summary()`.

Volendo visualizzare i dati, che sono raccolti giornalmente, potrebbe fare comodo una colonna che riporta data e ora, in un formato che R riconosce. Possiamo ottenere tutto ciò con un solo comando. Va segnalato che il datset che è stato fornito, già contiene tale colonna, che è stata creata con il comando:

```
Sys.setenv(TZ='GMT')
#Sys.setlocale("LC_TIME","it_IT")
PM10dataSA$DateTime.Local <- as.POSIXct(paste(PM10dataSA[,c("Date.Local")] , PM10dataSA[,c("Time.Local")]
```

Ora possiamo visualizzare i dati usando la funzione `plot()` aggiungendo delle opportune label lungo gli assi per migliorare la leggibilità.

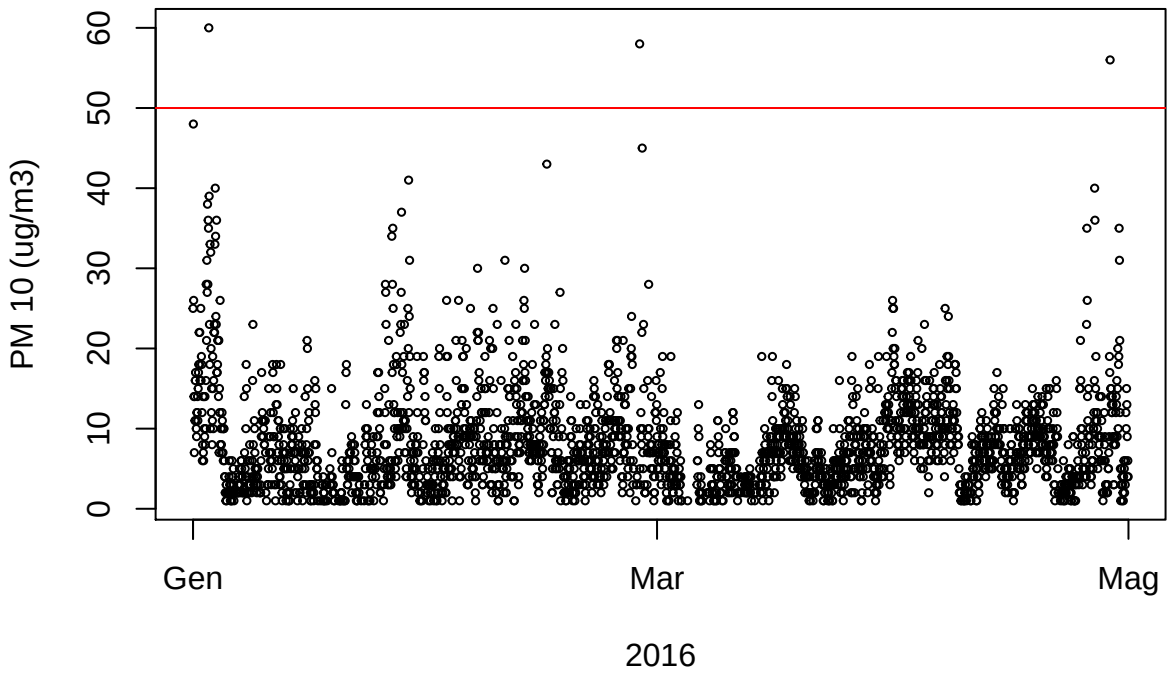
```
plot(PM10dataSA$DateTime.Local, PM10dataSA$Sample.Measurement, xlab = "2016", ylab= "PM 10 (ug/m3)", cex =
```



Poiché il grafico contiene un punto per ogni misurazione oraria, il grafico contiene moltissime informazioni.

Aggiungiamo una linea che indica il limite massimo giornaliero di PM 10 consentito (in Italia).

```
plot(PM10dataSA$DateTime.Local, PM10dataSA$Sample.Measurement, xlab = "2016", ylab= "PM 10 (ug/m3)", cex =
abline(h = 50, col = "red")
```




```
# Per salviare l'immagine
#png(filename="../plot/DailyPM10.png")
```

Si può notare che (fortunatamente) poche misurazioni superano i limiti consentiti.

5.1.1 Dati giornalieri

Vediamo di ridurle il numero di dati, aggregandoli per avere misurazioni giornaliere. Possiamo farlo usando la funzione `aggregate()` che ci permette di applicare una funzione ad un dataset indicando. Possiamo, ad esempio, decidere di salvare in un nuovo dataset le misure medie e massime per i dati di SA.

```
DailyVal <- aggregate(PM10dataSA$Sample.Measurement ~ Date.Local, data = PM10dataSA, FUN = mean)
DailyVal[,3] <- aggregate(PM10dataSA$Sample.Measurement ~ Date.Local, data = PM10dataSA, FUN = max)[2]
head(DailyVal)
```

```
##   Date.Local PM10dataSA$Sample.Measurement PM10dataSA$Sample.Measurement.1
## 1 2016-01-01                     16.70833                                48
## 2 2016-01-02                     17.54167                                38
## 3 2016-01-03                     23.91667                                60
## 4 2016-01-04                     14.37500                                36
## 5 2016-01-05                      3.05000                                10
## 6 2016-01-06                      2.87500                                 6
```

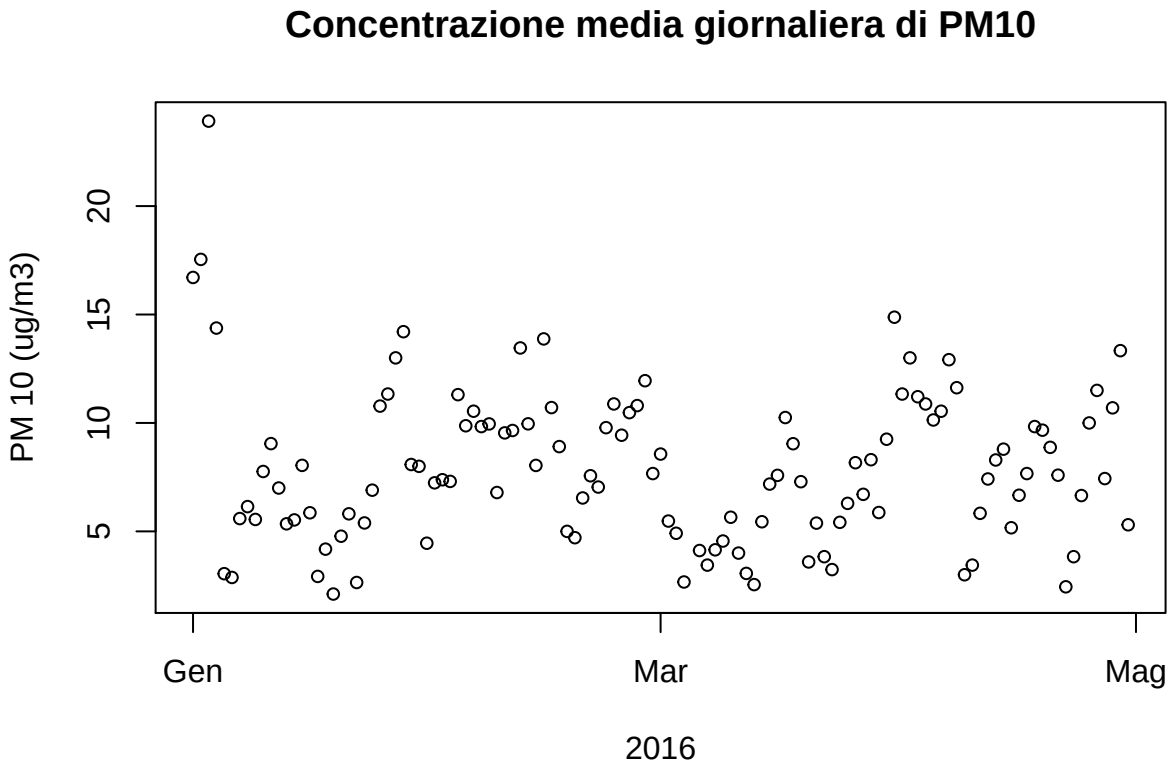
Possiamo migliorare la leggibilità del dataset cambiando i nomi delle variabili.

```
names(DailyVal)[2:3] <- c("mean daily PM10", "max daily PM 10")
head(DailyVal)
```

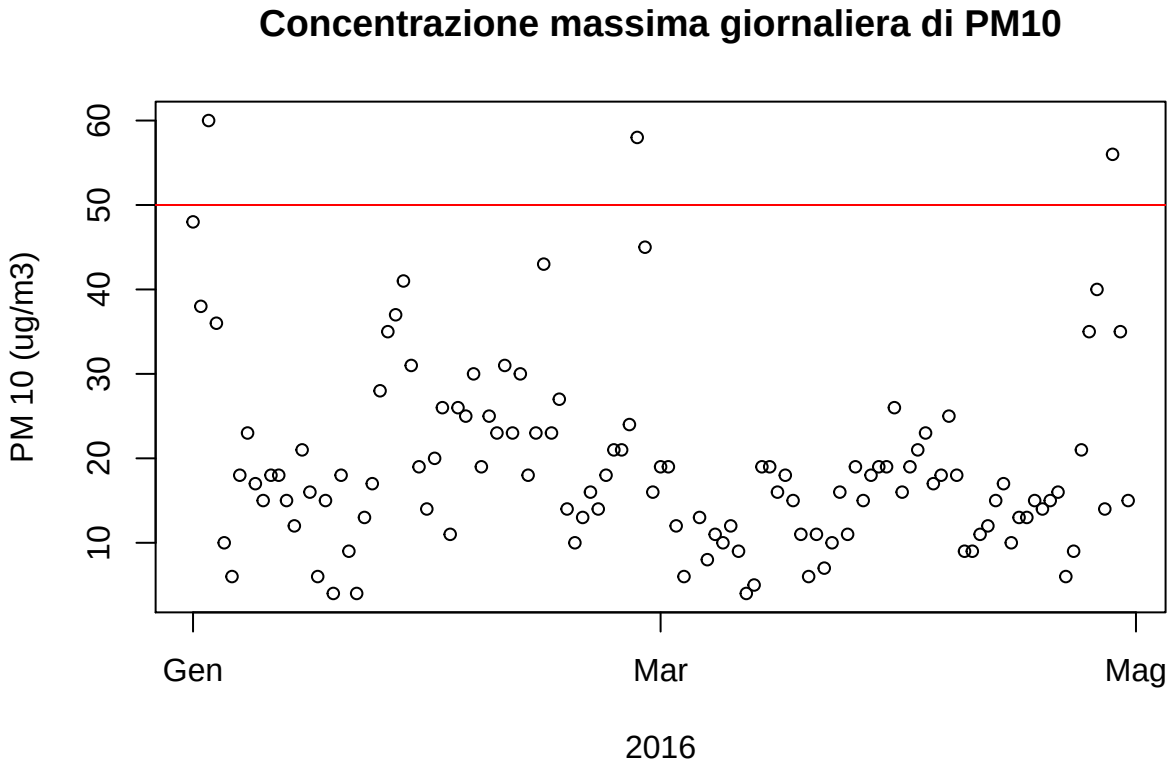
```
##   Date.Local mean daily PM10 max daily PM 10
## 1 2016-01-01        16.70833              48
## 2 2016-01-02        17.54167              38
## 3 2016-01-03        23.91667              60
## 4 2016-01-04        14.37500              36
## 5 2016-01-05         3.05000              10
## 6 2016-01-06         2.87500               6
```

Visualizziamo ora i dati medi e massimi giornalieri:

```
plot(DailyVal$Date.Local, DailyVal$`mean daily PM10`, xlab = "2016", ylab= "PM 10 (ug/m3)", cex = .8, main
```




```
plot(DailyVal$Date.Local, DailyVal$`max daily PM 10`, xlab = "2016", ylab= "PM 10 (ug/m3)", cex = .8, main
abline(h = 50, col = "red")
```



Potrebbe essere interessante vedere se il giorno della settimana influenza la concentrazione di PM10, ad esempio in giorni con più traffico potremmo aspettarci più inquinamento.

```
typeof(DailyVal$Date.Local)
```

```
## [1] "double"
```

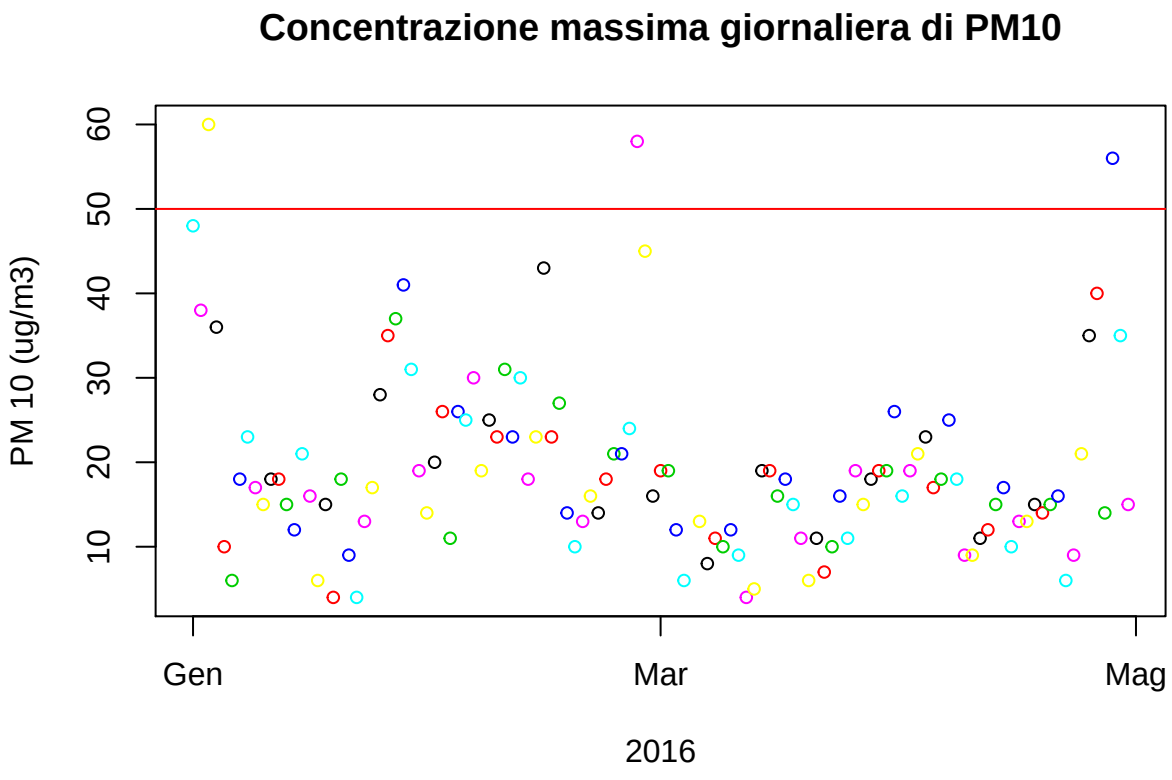
```
#Sys.setlocale("LC_TIME","C") #Nomi
DailyVal$DayOfTheWeek <- weekdays(DailyVal$Date.Local, abbreviate = TRUE)
head(DailyVal)
```

```
##   Date.Local mean daily PM10 max daily PM 10 DayOfTheWeek
## 1 2016-01-01        16.70833              48          Ven
## 2 2016-01-02        17.54167              38          Sab
## 3 2016-01-03        23.91667              60          Dom
## 4 2016-01-04        14.37500              36          Lun
## 5 2016-01-05         3.05000              10          Mar
## 6 2016-01-06         2.87500               6          Mer
```

```
#ordiniamo i giorni in modo che la settimana inizi di lunedì
# ci sarà utile dopo
DailyVal$DayOfTheWeek <- ordered(DailyVal$DayOfTheWeek, levels=c( "Lun" , "Mar", "Mer" , "Gio", "Ven" ,  "
```

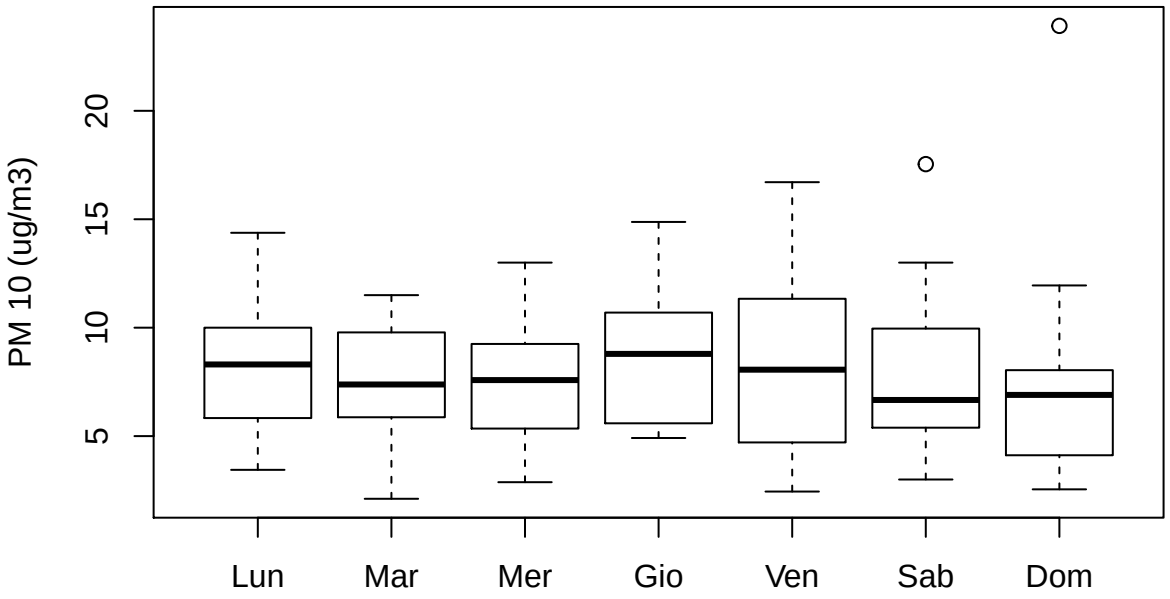
Vediamo di colorare il grafico precedente usando un colore diverso per ogni giorno della settimana.

```
plot(DailyVal$Date.Local, DailyVal$`max daily PM 10`, xlab = "2016", ylab= "PM 10 (ug/m3)", cex = .8, main
abline(h = 50, col = "red")
```



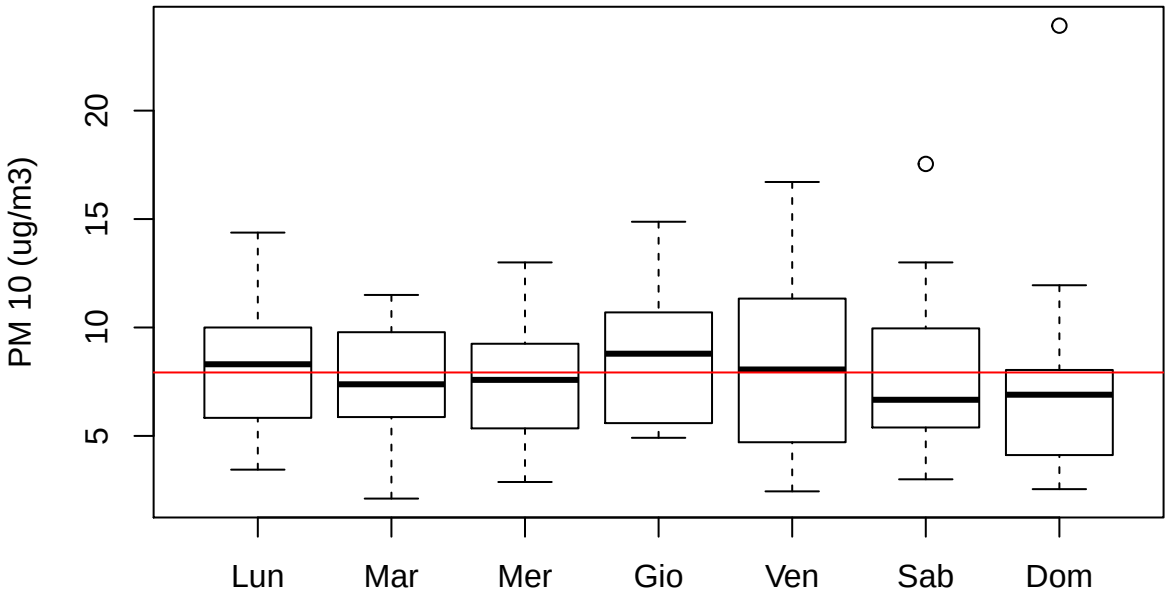
Vediamo che non sembra che i picchi siano raggiunti negli stessi giorni. Comunque una legenda andrebbe aggiunta.
Vediamo con dei boxplot come la quantità media di polveri PM10 è distribuita rispetto ai giorni della settimana.

```
boxplot(`mean daily PM10` ~ DayOfTheWeek, data = DailyVal, ylab= "PM 10 (ug/m3)")
```



Aggiungere ad esempio una linea che indica le media delle misurazioni, aiuta a fare un confronto tra i dati.

```
boxplot( `mean daily PM10` ~ DayOfTheWeek, data = DailyVal, ylab= "PM 10 (ug/m3)")
abline(h = mean(DailyVal$`mean daily PM10`), col = "red")
```



Vediamo la data in cui è stato assunto il valore massimo:

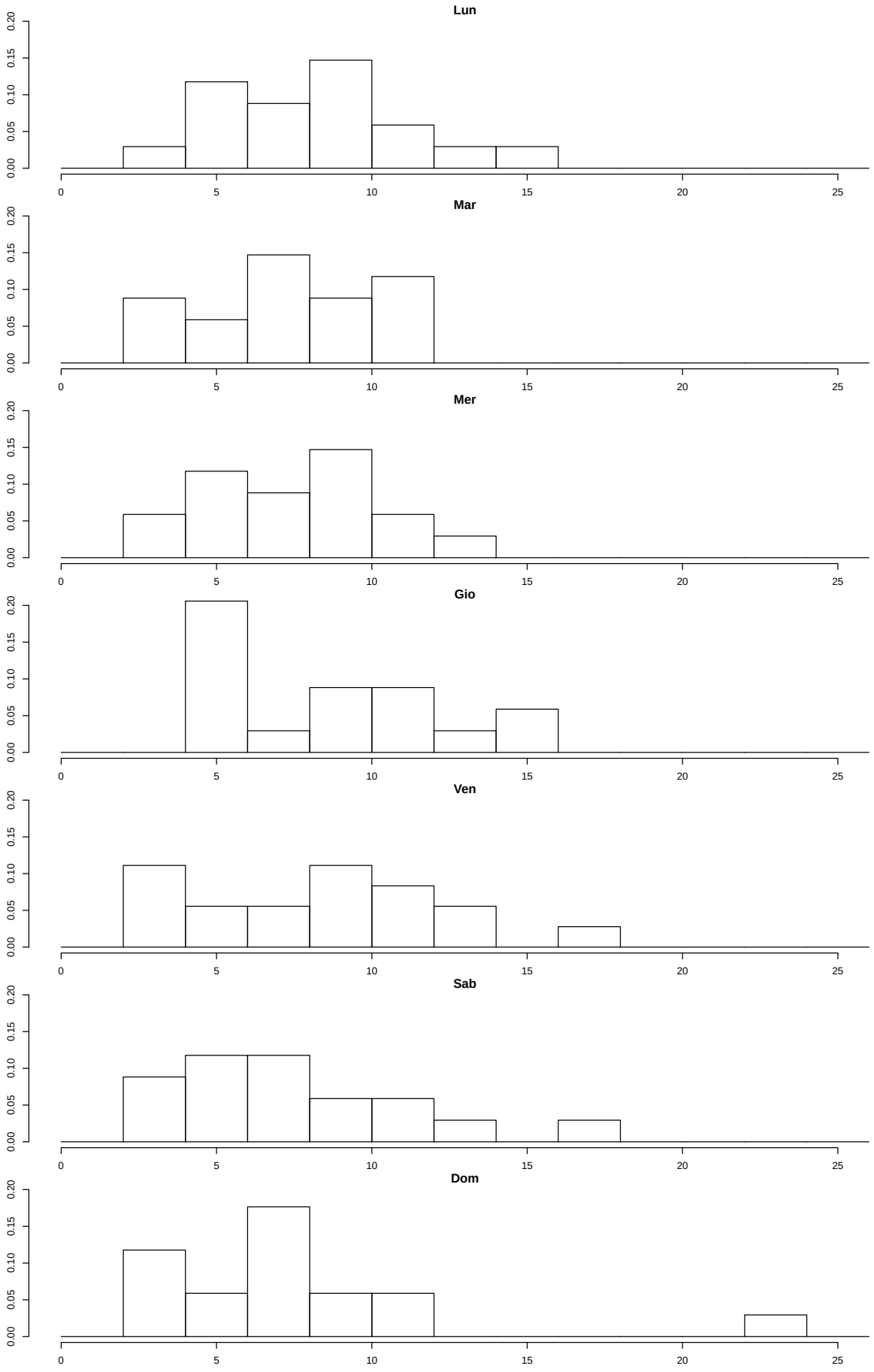
```
DailyVal[DailyVal$`mean daily PM10` > 20,]
```

```
##   Date.Local mean daily PM10 max daily PM 10 DayOfTheWeek
## 3 2016-01-03        23.91667              60          Dom
```

Che sarà successo quel giorno? Potrebbe essere un errore di lettura?

Se volessimo visualizzare la distribuzione dei dati rispetto ai giorni, possiamo usare la funzione `hist()`. Purtroppo tale funzione non supporta la notazione ~, perciò dovremo specificare un grafico per giorno della settimana.

```
par(mfrow = c(7,1), mar = c(2,2,1,1))
for (i in levels(DailyVal$DayOfTheWeek) ) {
  hist( DailyVal[DailyVal$DayOfTheWeek == i, "mean daily PM10"] , freq = FALSE, main = i, ylim = c(0, 0.2)
}
```


Lun
Mar
Mer
Gio
Ven
Sab
Dom
0.00
0.05
0.10
0.15
0.20
0
5
10
15
20
25

Dal grafico possiamo capire come si distribuiscono le misurazioni rispetto ai giorni della settimana. Si usi l'help per comprendere i parametri che sono stati usati.

5.2 Inquinamento in California

Decidiamo ora di concentrare la nostra attenzione sull'inquinamento in California e dopo aver letto il dataset che contiene i valori **medi giornalieri** di PM10, salviamo un sottoinsieme di dati che contiene solo dati relativi alla California.

```
PM10data <- read.csv("./data/daily_81102_2016.csv")
dim(PM10data)
```

```
## [1] 118885     29
```

Vediamo che il file è molto grande, contiene molte righe e molte variabili. R impiega diverso tempo a leggerlo.

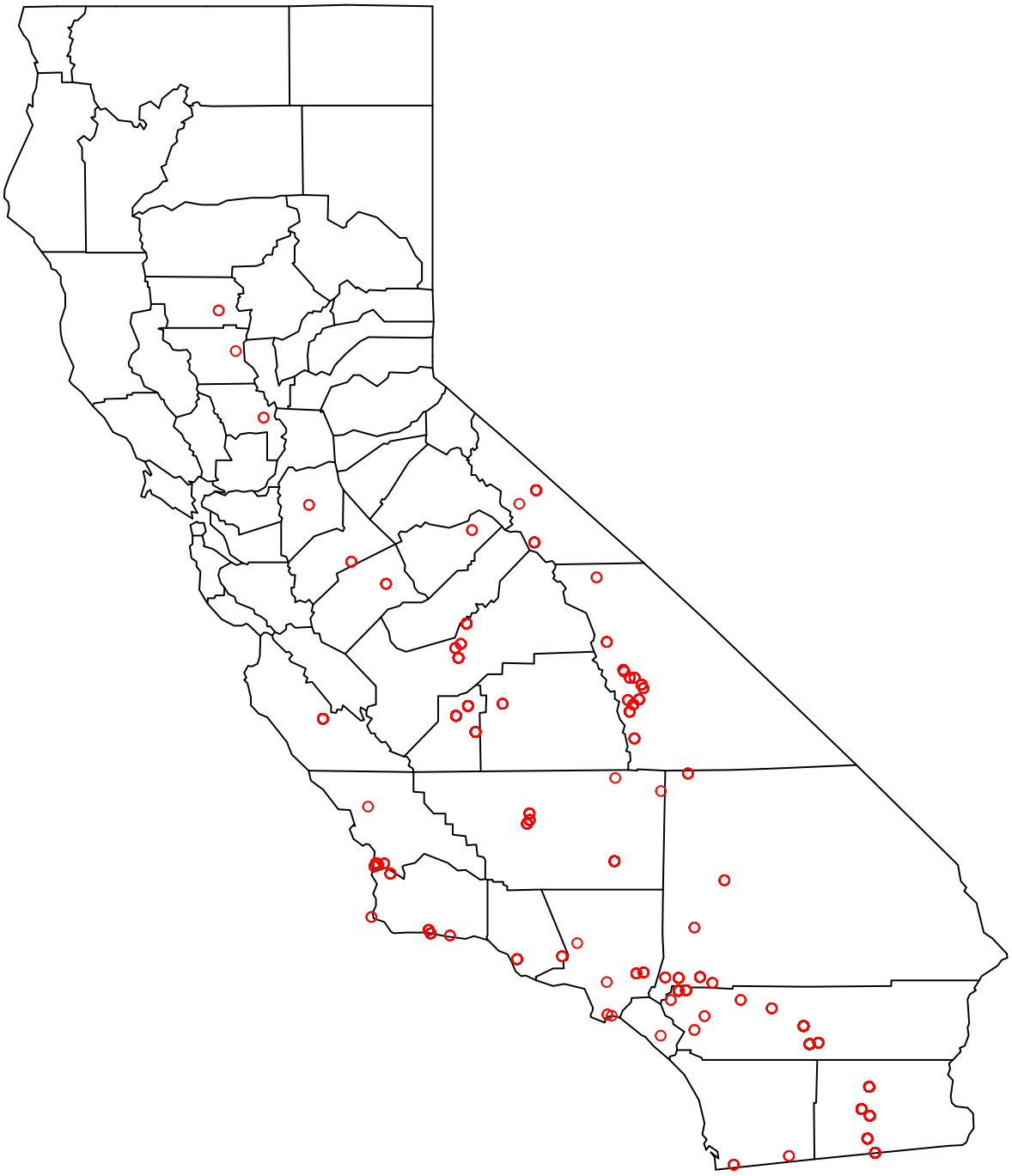
```
PM10dataCAL <- PM10data[PM10data$State.Name == "California",]
knitr::kable(
head(PM10dataCAL),
booktabs = TRUE
)
```

	State.Code	County.Code	Site.Num	Parameter.Code	POC	Latitude	Longitude	Datum	Parameter.Nam
15472	6	7	8	81102	3	39.76154	-121.8416	WGS84	PM10 Total 0-
15473	6	7	8	81102	3	39.76154	-121.8416	WGS84	PM10 Total 0-
15474	6	7	8	81102	3	39.76154	-121.8416	WGS84	PM10 Total 0-
15475	6	7	8	81102	3	39.76154	-121.8416	WGS84	PM10 Total 0-
15476	6	7	8	81102	3	39.76154	-121.8416	WGS84	PM10 Total 0-
15477	6	7	8	81102	3	39.76154	-121.8416	WGS84	PM10 Total 0-

In questo caso il nome della variabile che ci interessa è *Arithmetic.Mean*.

Ora vogliamo visualizzare su una mappa quali sono lo città che superano la soglia di inquinamento consentita durante le osservazioni. In R, questa operazione è molto semplice usando i pacchetti `maps` o `RgoogleMaps`.

```
#install.packages("maps")
library(maps)
map("county", "california", xlim=c(-125,-114), ylim=c(32,43))
points( PM10dataCAL[PM10dataCAL$Arithmetic.Mean > 50,c("Longitude", "Latitude")] ,cex = .8, col = "red")
```



5.3 Inquinamento a Los Angeles

Usiamo lo stesso dataset contenente i dati giornalieri relativi alle PM10 negli Stati Uniti e concentriamoci sulla sola città di Los Angeles.

```
PM10dataLA <- PM10data[PM10data$City.Name== "Los Angeles",]
dim(PM10dataLA)
```

```
## [1] 90 29
```

Vediamo che sono state raccolte 90 osservazioni relative alla sola città di LA, cerchiamo di capire quando e con che frequenza sono state raccolte.

```
typeof(PM10dataLA)
```

```
## [1] "list"
```

```
PM10dataLA$Date.Local[1:10]
```

```
##  [1] 2016-01-01 2016-01-07 2016-01-13 2016-01-19 2016-01-25 2016-01-31
##  [7] 2016-02-06 2016-02-12 2016-02-18 2016-02-24
## 336 Levels: 2016-01-01 2016-01-02 2016-01-03 2016-01-04 ... 2016-12-02
```

Vediamo che le date sono state salvate come liste. In realtà, come già visto, R prevede il formato *Date* che potrebbe essere utile e più maneggevole di una lista per fare "operazioni aritmetiche". Convertiamo quindi la colonna in *Date*.

```
PM10dataLA$Date.Local <- as.Date(PM10dataLA$Date.Local)
typeof(PM10dataLA$Date.Local)
```

```
## [1] "double"
```

La variabile ora risulta di tipo *Date*. Questo ci permette, ad esempio, di vedere quanti giorni passano tra una rilevazione e l'altra.

```
diff(PM10dataLA$Date.Local)
```

```
## Time differences in days
##  [1]   6   6   6   6   6   6   6   6   6   6   6   6   6   6
## [15]   6   6   6   6   6   6   7   5   6   6   6   6   6   6
## [29]   6   6   6   6   6   6   6   6   6   6   6   6   6   6
## [43]   6  12 -270   6   6   6   6   6   6   6   6   6   6   6
## [57]   6   6  12   6   6   6   6   6   6   6   6   6   6   6
## [71]   6   6   6   6   6   6   6   6   6   6   6   6   6   6
## [85]   6   6   6   6   6
```

Salta all'occhio che la frequenza delle registrazioni è circa ogni 6 giorni, ma compare un dato inatteso. Investighiamo meglio cosa è successo.

```
unique(PM10dataLA$Date.Local)
```

```
##  [1] "2016-01-01" "2016-01-07" "2016-01-13" "2016-01-19" "2016-01-25"
##  [6] "2016-01-31" "2016-02-06" "2016-02-12" "2016-02-18" "2016-02-24"
## [11] "2016-03-01" "2016-03-07" "2016-03-13" "2016-03-19" "2016-03-25"
## [16] "2016-03-31" "2016-04-06" "2016-04-12" "2016-04-18" "2016-04-24"
## [21] "2016-04-30" "2016-05-07" "2016-05-12" "2016-05-18" "2016-05-24"
## [26] "2016-05-30" "2016-06-05" "2016-06-11" "2016-06-17" "2016-06-23"
## [31] "2016-06-29" "2016-07-05" "2016-07-11" "2016-07-17" "2016-07-23"
## [36] "2016-07-29" "2016-08-04" "2016-08-10" "2016-08-16" "2016-08-22"
## [41] "2016-08-28" "2016-09-03" "2016-09-09" "2016-09-15" "2016-09-27"
## [46] "2016-05-06" "2016-09-21"
```

```
length(unique(PM10dataLA$Date.Local))
```

```
## [1] 47
```

Capiamo che sebbene le osservazioni siano 90, non tutte si riferiscono a giorni differenti, quindi bisogna capire come gestire le misurazioni ripetute durante lo stesso giorno. Vediamo, ad esempio, cosa caratterizza le misurazioni ripetute del primo giorno dell'anno.

```
PM10dataLA[PM10dataLA$Date.Local== "2016-01-01",]
```

```
##       State.Code County.Code Site.Num Parameter.Code POC Latitude
## 23477          6          37     1103          81102   2 34.06659
## 23613          6          37     5005          81102   1 33.95080
##       Longitude Datum        Parameter.Name Sample.Duration
## 23477 -118.2269 WGS84 PM10 Total 0-10um STP         24 HOUR
```

```
## 23613 -118.4304 WGS84 PM10 Total 0-10um STP         24 HOUR
##       Pollutant.Standard Date.Local              Units.of.Measure
## 23477  PM10 24-hour 2006 2016-01-01 Micrograms/cubic meter (25 C)
## 23613  PM10 24-hour 2006 2016-01-01 Micrograms/cubic meter (25 C)
##       Event.Type Observation.Count Observation.Percent Arithmetic.Mean
## 23477       None                1                 100              17
## 23613       None                1                 100              12
##       X1st.Max.Value X1st.Max.Hour AQI Method.Code
## 23477             17             0  16          63
## 23613             12             0  11          63
##                            Method.Name               Local.Site.Name
## 23477 HI-VOL SA/GMW-1200 - GRAVIMETRIC Los Angeles-North Main Street
## 23613 HI-VOL SA/GMW-1200 - GRAVIMETRIC                   LAX Hastings
##                           Address State.Name County.Name   City.Name
## 23477 1630 N MAIN ST, LOS ANGELES California Los Angeles Los Angeles
## 23613 7201 W. WESTCHESTER PARKWAY California Los Angeles Los Angeles
##                                CBSA.Name Date.of.Last.Change
## 23477 Los Angeles-Long Beach-Anaheim, CA          2016-12-20
## 23613 Los Angeles-Long Beach-Anaheim, CA          2016-12-20
```

Possiamo osservare che ci sono almeno due siti diversi dove vengono raccolti i dati. Questa osservazione segue sia la comparsa di due diversi *Site.Num* per la stessa data, sia le differenze nei valori di *Latitude* e *Longitude*. Vediamo se esistono solo due siti o più.

```
paste(unique(PM10dataLA$Site.Num) , unique(PM10dataLA$Latitude), unique(PM10dataLA$Longitude))
```

```
## [1] "1103 34.06659 -118.22688" "5005 33.9508 -118.43043"
```

Abbiamo appurato che esistono due (e solo due) siti individuati da diverse coordinate geografiche.

Estrapoliamo ora alcune informazioni per le variabili numeriche usando il comando `summary()`.

```
summary(PM10dataLA)
```

```
##    State.Code  County.Code    Site.Num    Parameter.Code       POC
##  Min.   :6    Min.   :37   Min.   :1103   Min.   :81102   Min.   :1.0
##  1st Qu.:6    1st Qu.:37   1st Qu.:1103   1st Qu.:81102   1st Qu.:1.0
##  Median :6    Median :37   Median :3054   Median :81102   Median :1.5
##  Mean   :6    Mean   :37   Mean   :3054   Mean   :81102   Mean   :1.5
##  3rd Qu.:6    3rd Qu.:37   3rd Qu.:5005   3rd Qu.:81102   3rd Qu.:2.0
##  Max.   :6    Max.   :37   Max.   :5005   Max.   :81102   Max.   :2.0
##
##     Latitude       Longitude        Datum                  Parameter.Name
##  Min.   :33.95   Min.   :-118.4   NAD83: 0   PM10 Total 0-10um STP:90
##  1st Qu.:33.95   1st Qu.:-118.4   WGS84:90
##  Median :34.01   Median :-118.3
##  Mean   :34.01   Mean   :-118.3
##  3rd Qu.:34.07   3rd Qu.:-118.2
##  Max.   :34.07   Max.   :-118.2
##
##       Sample.Duration         Pollutant.Standard   Date.Local
##  24 HOUR      :90     PM10 24-hour 2006:90       Min.   :2016-01-01
##  24-HR BLK AVG: 0                                1st Qu.:2016-03-07
##                                                  Median :2016-05-15
##                                                  Mean   :2016-05-14
##                                                  3rd Qu.:2016-07-21
##                                                  Max.   :2016-09-27
##
##                      Units.of.Measure    Event.Type Observation.Count
```

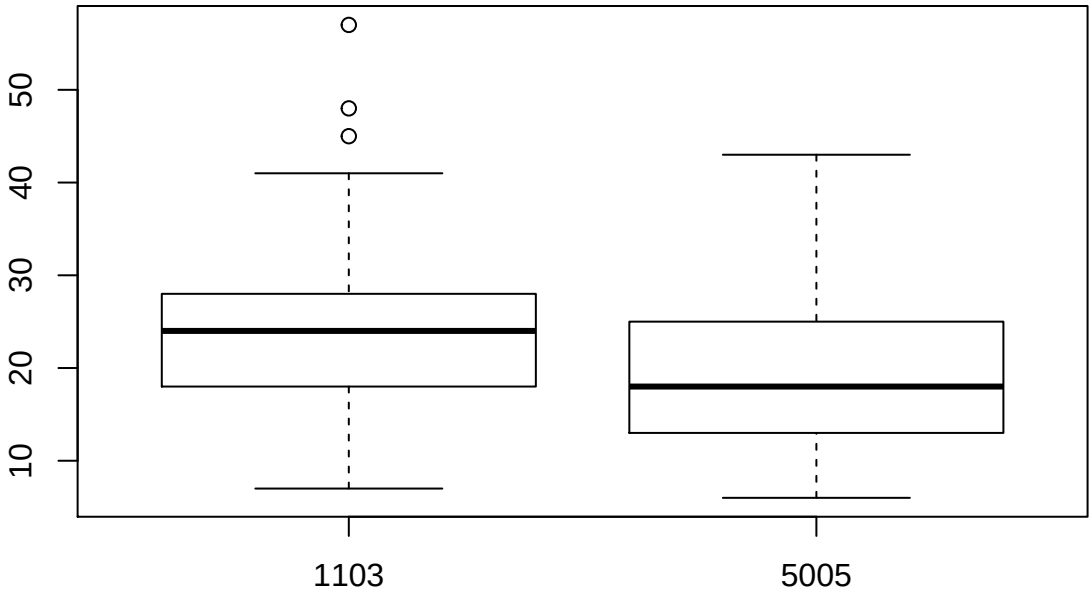
```
##  Micrograms/cubic meter (25 C):90      Excluded: 0   Min.   :1
##                                        Included: 0   1st Qu.:1
##                                        None    :90   Median :1
##                                                      Mean   :1
##                                                      3rd Qu.:1
##                                                      Max.   :1
##
##  Observation.Percent Arithmetic.Mean X1st.Max.Value  X1st.Max.Hour
##  Min.   :100         Min.   : 6.00   Min.   : 6.00   Min.   :0
##  1st Qu.:100         1st Qu.:16.00   1st Qu.:16.00   1st Qu.:0
##  Median :100         Median :20.50   Median :20.50   Median :0
##  Mean   :100         Mean   :22.18   Mean   :22.18   Mean   :0
##  3rd Qu.:100         3rd Qu.:28.00   3rd Qu.:28.00   3rd Qu.:0
##  Max.   :100         Max.   :57.00   Max.   :57.00   Max.   :0
##
##       AQI         Method.Code
##  Min.   : 6.00   Min.   :63
##  1st Qu.:15.00   1st Qu.:63
##  Median :19.00   Median :63
##  Mean   :20.54   Mean   :63
##  3rd Qu.:26.00   3rd Qu.:63
##  Max.   :52.00   Max.   :63
##
##                                        Method.Name
##  HI-VOL SA/GMW-1200 - GRAVIMETRIC           :90
##   -                                         : 0
##  BGI Inc. Model PQ200 PM10 - Gravimetric    : 0
##  Hi Vol SSI Ecotech Model 3000 - Gravimetric: 0
##  HI-VOL-SA/GMW-321-B - GRAVIMETRIC          : 0
##  HI-VOL-WEDDING-INLET - GRAVIMETRIC         : 0
##  (Other)                                    : 0
##                                          Local.Site.Name
##  LAX Hastings                                    :45
##  Los Angeles-North Main Street                   :45
##                                                  : 0
##  1 Site (Met Station and Hi-Vols BA1 and BA1-B)   : 0
##  10 METER TOWER, AT NW CORNER, U OF I RESEARCH CENTER: 0
##  19th & Burt                                     : 0
##  (Other)                                         : 0
##                                    Address        State.Name
##  1630 N MAIN ST, LOS ANGELES           :45   California:90
##  7201 W. WESTCHESTER PARKWAY           :45   Alabama   : 0
##   151 NO SUNRISE BLVD, ROSEVILLE, CA   : 0   Alaska    : 0
##   201 ASHVILLE ROAD                    : 0   Arizona   : 0
##   2220 NORTH STREET, ANDERSON, CA 96007: 0   Arkansas  : 0
##   3337 Sandy Way, South Lake Tahoe, CA : 0   Colorado  : 0
##  (Other)                               : 0   (Other)   : 0
##       County.Name           City.Name
##  Los Angeles:90   Los Angeles    :90
##  Ada         : 0   Aberdeen       : 0
##  Adair       : 0   Ajo            : 0
##  Adams       : 0   Ak-Chin Village: 0
##  Alamosa     : 0   Alamosa        : 0
##  Albany      : 0   Albuquerque    : 0
##  (Other)     : 0   (Other)        : 0
##                               CBSA.Name  Date.of.Last.Change
```

```
##  Los Angeles-Long Beach-Anaheim, CA:90   2016-12-20:90
##                                     : 0   2016-04-11: 0
##  Aberdeen, SD                       : 0   2016-04-15: 0
##  Albuquerque, NM                    : 0   2016-05-11: 0
##  Allentown-Bethlehem-Easton, PA-NJ  : 0   2016-05-13: 0
##  Altoona, PA                        : 0   2016-05-18: 0
##  (Other)                            : 0   (Other)   : 0
```

Possiamo notare che, anche in questo caso, alcune delle variabili sono trattate come numeriche anche se dovrebbero essere di tipo Factor. Possiamo o convertirle, o semplicemente tenerlo a mente qualora dovessimo lavorarci. Possiamo altrimenti rimuoverle come visto prima.

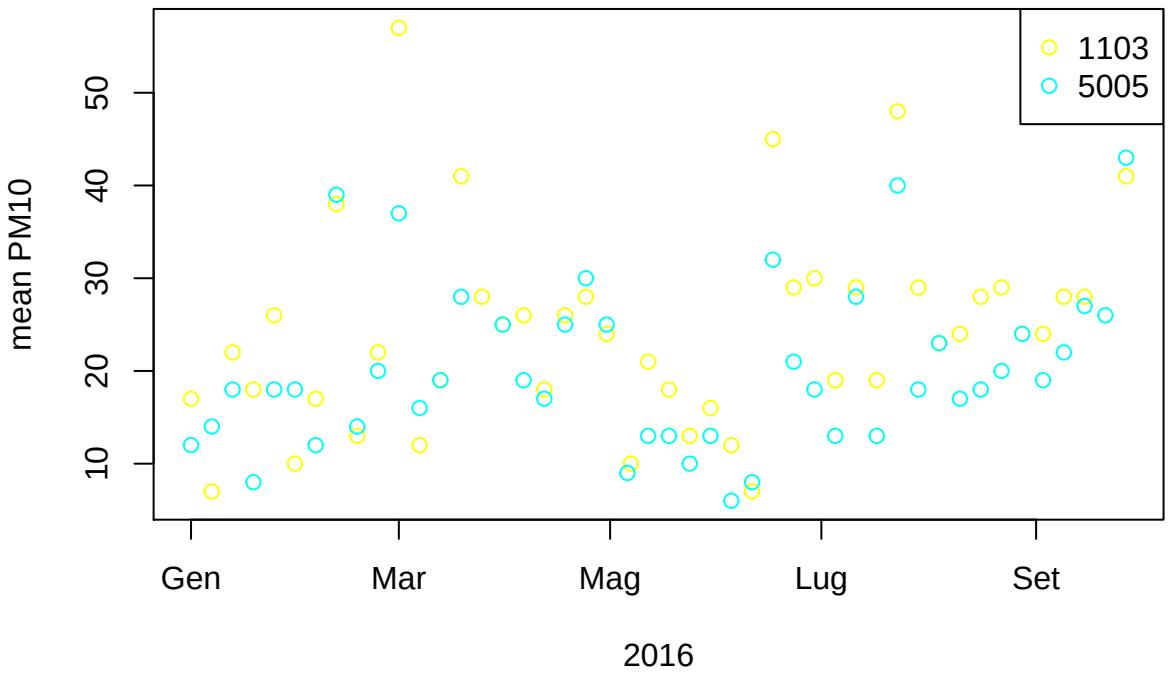
Per prima cosa, definiamo un campo di interesse. Se vogliamo, ad esempio, vedere l'andamento delle misurazioni medie di PM10 nei due siti, molte delle variabili non ci interessano. Possiamo investigare più nel dettaglio le variabili legate alla nostra indagine e non considerare le altre:

```
boxplot(Arithmetic.Mean ~ Site.Num, data = PM10dataLA)
```



Possiamo notare che uno dei due siti presenta delle misurazioni che in generale sono più alte.

```
plot(PM10dataLA$Date.Local , PM10dataLA$Arithmetic.Mean, col= PM10dataLA$Site.Num, xlab = "2016", ylab = "
legend( "topright", legend = c("1103","5005" ), col= unique(PM10dataLA$Site.Num), pch = c(1,1) )
```



5.4 Esercizi

5.4.1 Esercizio 1

Si trovi un modo adeguato per importare i dati relativi al reddito nazionale lordo pro capite e alla percentuale di strade asfaltate in R. Dopo aver analizzato e preparato i dataset, si usino i dati per investigare le due variabili rispetto ad un paese del G7, un paese in via di sviluppo ed un paese del terzo mondo a scelta, nel periodo dal 1990 al 2009. Si analizzino i dati e si visualizzino.

5.4.2 Esercizio 2

Si svolga il primo capitolo del corso online di DataCamp che riguarda i modelli di rischio nel credito. Il primo capitolo del corso (in inglese) è gratuito.

5.4.3 Esercizio 3

Si importi il dataset contenente i dati relativi alla aspettativa di vita, popolazione e prodotto interno lordo pro capite di 142 paesi del mondo. Si aggreghino i dati per continenti e si visualizzi l'andamento dell'aspettativa di vita rispetto alle variabili tempo, reddito pro capite e popolazione. Si investighi una possibile trasformazione dei dati per rendere più informativa l'analisi e si ripetano le analisi sui dati trasformati.

5.4.4 Esercizio 4

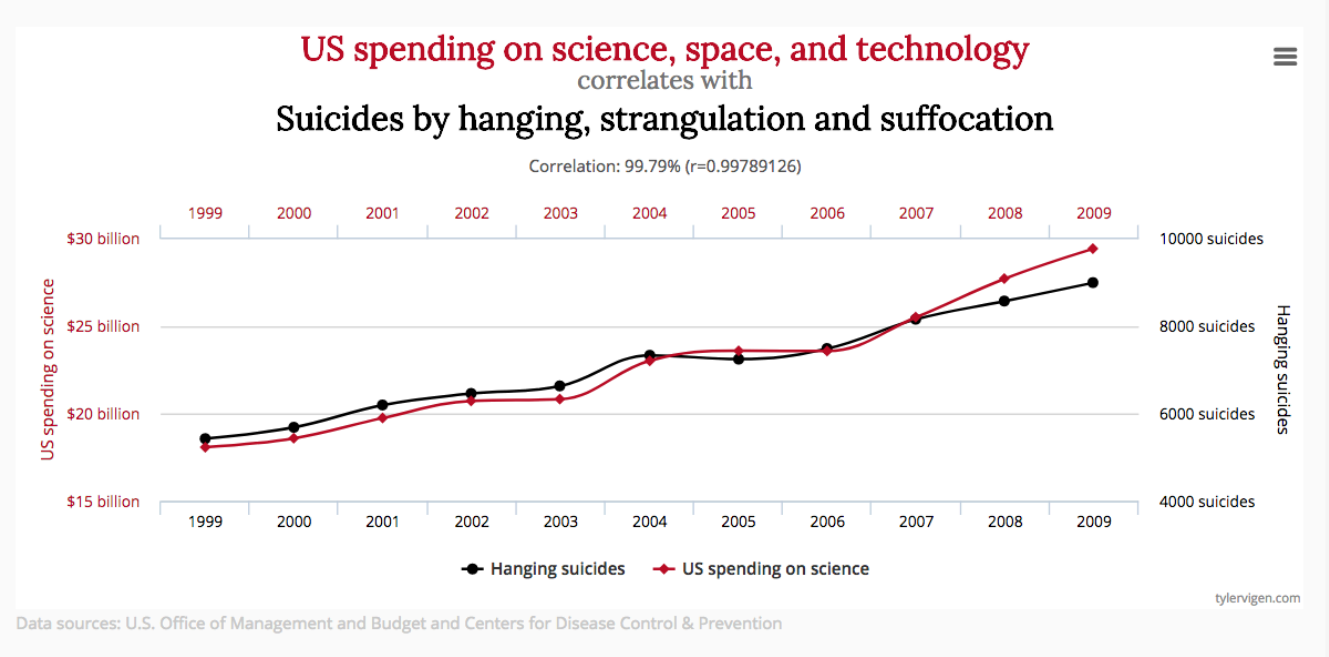
Si consideri il dataset che riporta 14 parametri di 303 pazienti con problemi di cuore. Si importi il dataset e si visualizzino con degli istogrammi le distribuzioni delle variabili. Inoltre usando la funzione `aggregate`, si determini al variare dell'età, il valore medio e massimo delle variabili `chol` (colesterolo) e `thalach` (frequenza cardiaca massima).

5.4.5 Esercizio 5

Dopo aver svolto l'Esercizio 4 del capitolo 4, si consulti il notebook R al link. Quali sono gli obbiettivi dell'autore? Quale tecniche utilizza? Quale di queste non sono ancora state trattate nel nostro corso?

Chapter 6

Regressione



Originally posted here.

Il contenuto di questo capitolo si basa sull'omonimo capitolo del libro Calcolo delle probabilità e statistica di Paolo Baldi (Baldi, 1998).

Si consideri il problema, piuttosto comune, di voler esprimere una variabile, ad esempio y, in funzione di altre variabili, ad esempio $x_1, \ldots, x_n$, più delle perturbazioni aleatorie.

6.1 Regressione lineare

Prendiamo in considerazione il caso in cui tale funzione sia lineare. Parleremo di *regressione lineare*. Ciò significa che assumiamo che la variabile y, detta *dipendente*, si possa esprimere come

$$y = \beta_0 + \beta_1 \cdot x + \omega$$

dove β_0, β_1 sono parametri da determinare e ω è una perturbazione stocastica con distribuzione normale di media 0 e varianza σ^2. I parametri β_0, β_1 vengono solitamente determinati in base a diversi valori di x e y.

Se abbiamo più osservazioni per la variabile y, ottenute rispetto a diversi valori di x, indicheremo con y_i e x_i tali valori.

Se l'assunzione che la dipendenza sia lineare è plausibile, ci aspettiamo che per le varie osservazioni valga:

$$y_i = \beta_0 + \beta_1 x_i + \omega_i, \quad i = 1, \ldots, n$$

con ω_i indipendenti e tutti con distribuzione $N(0, \sigma^2)$, con σ^2 che non dipende da i.

6.2 Stimare β_0 e β_1

Il problema di stimare β_0, β_1 viene risolto andando a trovare quei valori per i parametri che minimizzano la distanza tra i dati osservati (y_i) e i valori prodotti dal modello:

$$\hat{y_i} = \beta_0 + \beta_1 x_i$$

La formulazione diviene quindi:

$$\min_{\beta_0, \beta_1} S = S(\beta_0, \beta_1) = \sum_{i=1}^{n} (y_i - \beta_0 + \beta_1 x_i)^2$$

Questo problema di minimizzazione quadratica ammette soluzione (unica) e si può ottenere andando ad imporre che il gradiente si annulli. Questo porta alla soluzione (b_0, b_1) tale che:

$$b_0 = \bar{y} - b_1 \bar{x}$$

$$b_1 = \frac{\sum_i (y_i - \bar{y}) x_i}{\sum_i x_i (x_i - \bar{x})} = \frac{\bar{\sigma}_{xy}}{\bar{\sigma_x^2}}$$

con

$$\bar{x} = \frac{1}{n} \sum_i x_i \text{ e } \bar{y} = \frac{1}{n} \sum_i y_i$$

I valori di b_0 e b_1 così ottenuti sono *stimatori* per i parametri β_0 e β_1. In particolare è possibile dimostrare che sono degli stimatori **non distorti**.

Inoltre c'è tutta una serie di risultati (che non dimostreremo) che permettono di provare, usando anche le ipotesi sugli ω_i, che:

$$b_0 \sim N\left(\beta_0, \sigma^2 (\frac{1}{n} + \frac{\bar{x}^2}{\bar{\sigma_x^2}})\right), \; b_1 \sim N\left(\beta_1, \frac{\sigma^2}{\bar{\sigma_x^2}})\right)$$

Ovviamente è possibile associare degli intervalli di confidenza ai valori ottenuti, così come eseguire altri test statistici, ma noi non tratteremo questa parte.

Se indichiamo con $r_i = y_i - \hat{y_i}$, i così detti **residui**, possiamo definire:

$$s^2 = \frac{1}{n-2} \sum_i r_i^2$$

che è uno stimatore non distorto di σ^2, la varianza (sconosciuta) delle perturbazioni aleatorie. Inoltre vale che:

$$\frac{s^2}{\sigma^2}(n-2) \sim \chi^2(n-2)$$

dove χ^2 indica la distribuzione Chi quadrato. Usando R con il comando `lm()` è possibile non solo definire un modello di regressione lineare, ma anche accedere a molte di queste informazioni.

6.3 Valutare il modello: R^2

Un utile valore per valutare la correttezza del modello, è il valore di R^2, definito come:

$$R^2 = \frac{\sum_i (\hat{y}_i - \bar{y})^2}{\sum_i (y_i - \bar{y})^2} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$

Questo valore, che assume valori tra 0 e 1, quantifica la proporzione di varianza dei dati che è spiegata dal modello. Più tale valore si avvicina a 1, migliore è il modello. Se invece tale valore è piccolo, o il modello è inadeguato (relazione non lineare o ipotesi non soddisfatte), o σ^2 potrebbe avere un valore elevato.

6.4 Regressione lineare in R

Ricordiamo che per applicare la regressione lineare dobbiamo essere sicuri che le ipotesi siano soddisfatte. In particolare dovremo assicurarci che i residui seguano una distribuzione normale centrata in 0 e siano indipendenti.

Alcune delle funzioni già viste possono aiutarci in questo compito.

Definire e richiamare un modello lineare in R è molto semplice. Basta infatti utilizzare la funzione `lm()`, dove va specificata la *variabile dipendente* e il *predittore* ed i dati da usare per definire il modello.

Sebbene i modelli di *regressione multipla* siano concettualmente e praticamente più complessi, R li supporta usando la stessa funzione e la stessa notazione. In tal caso, invece che indicare un predittore, se ne indicano più usando il simbolo `+` per elencarli.

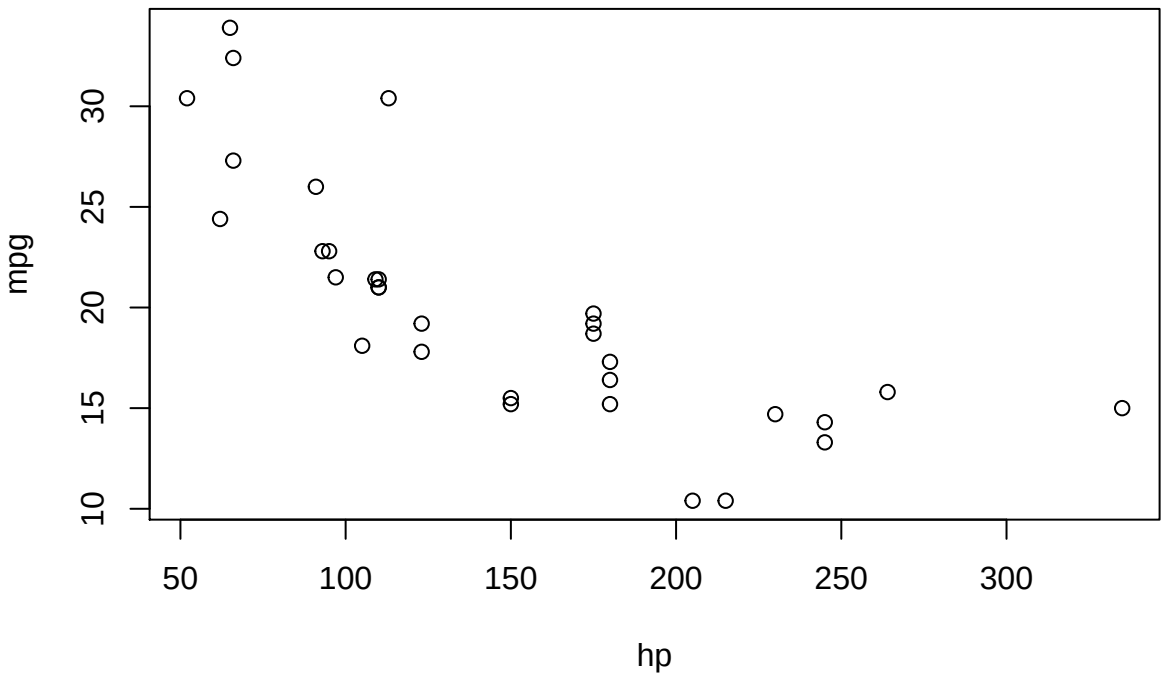
6.4.1 Mtcars

Vediamo alcuni esempi usando il dataset `mtcars`.

Supponiamo di voler investigare se il consumo di carburante e il numero di cavalli seguono una relazione lineare.

Un ottimo metodo per intuire se la relazione possa essere in qualche modo lineare è quello di disegnare la variabile dipendente in funzione del predittore.

```
plot(mpg ~ hp, data = mtcars)
```



Secondo i risultati della nostra analisi preliminare sembra plausibile che la relazione sia lineare. Continuiamo la nostra investigazione definendo il modello con la funzione `lm()` e visualizzando il risultato usando la ben nota funzione `summary()`.

```
mtcars_reg <- lm(mpg ~ hp, data = mtcars)
summary(mtcars_reg)
```

```
##
## Call:
## lm(formula = mpg ~ hp, data = mtcars)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.7121 -2.1122 -0.8854  1.5819  8.2360
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 30.09886    1.63392  18.421  < 2e-16 ***
## hp          -0.06823    0.01012  -6.742 1.79e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.863 on 30 degrees of freedom
## Multiple R-squared:  0.6024, Adjusted R-squared:  0.5892
## F-statistic: 45.46 on 1 and 30 DF,  p-value: 1.788e-07
```

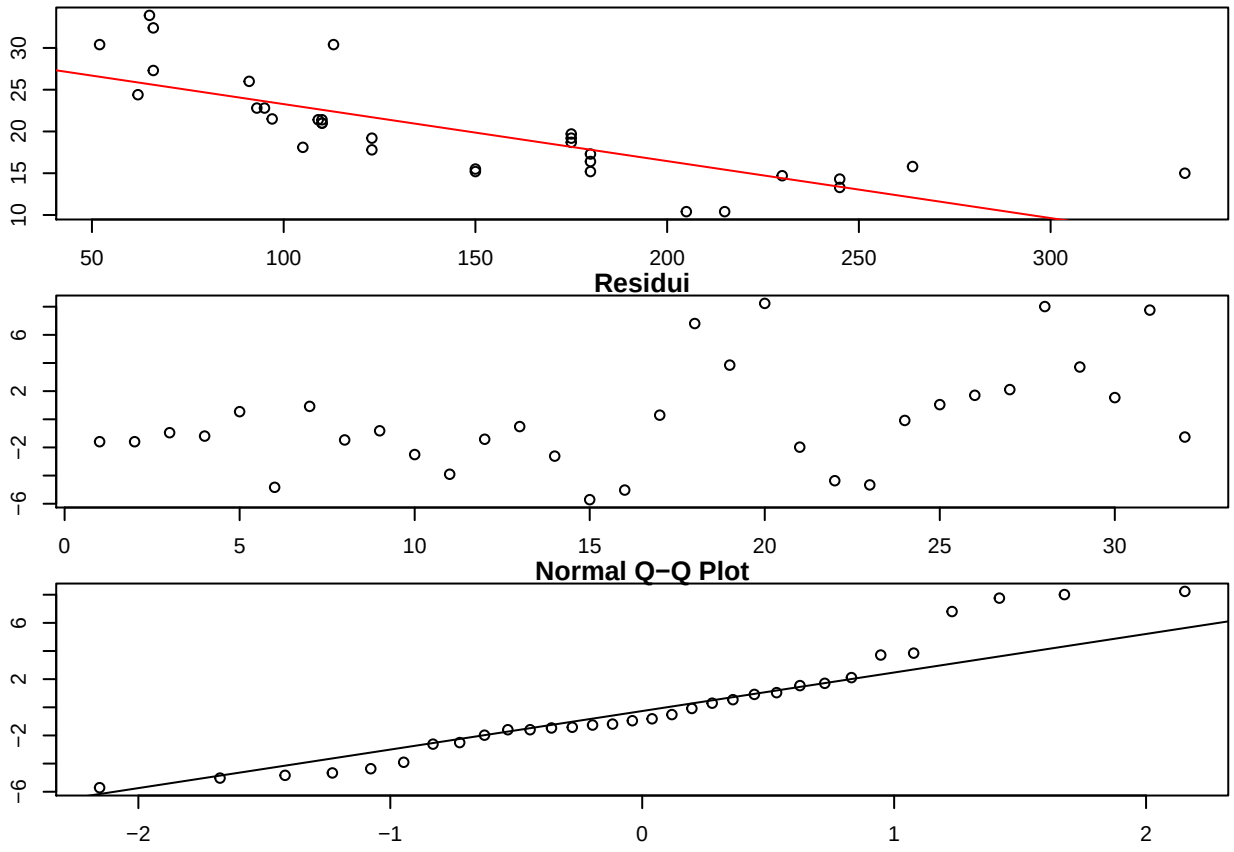
Si può accedere ai vari elementi elencati da `summary()` usando la notazione `$`.

Tra le informazioni che fornisce summary, è presente anche il valore **R-squared** (R quadro), che è la **proporzione di varianza della variabile dipendente che viene spiegata dal predittore**. Tale valore è compreso tra 0 e 1. Valori vicino a 1 indicano un buon modello, mentre valori bassi possono indicare o che il modello non spieghi completamente/correttamente i dati, o che la varianza dei residui è molto alta.

Una volta definito il modello, è importante visualizzare i risultati per valutarne la bontà e per capire il comportamento dei residui. In questo modo è infatti possibile verificare **quantitativamente** (non formalmente) se i residui soddisfano le ipotesi di indipendenza (non si devono osservare dei pattern) e che la distribuzione dei quantili sia confrontabile con quella di una normale.

I seguenti comandi permettono di visualizzare la retta di regressione, i quantili e di confrontarli con la distribuzione in quantili di una normale.

```
par(mfrow = c(3,1), mar = c(2,2,1,1))
# Retta di regressione
plot(mpg ~ hp, data = mtcars)
abline(mtcars_reg$coefficients, col = "red")
# Pattern nei residui
plot(mtcars_reg$residuals, main = "Residui")
# Distribuzione in quantili
qqnorm(mtcars_reg$residuals)
qqline(mtcars_reg$residuals)
```



Esercizio A

In modo analogo a quanto fato sopra, si investighi la relazione tra consumo e peso delle auto in esame.

Esercizio B

Si discuta un modello lineare per le variabili **x** e **y** di **Esercizo 1**.

6.5 Regressione multipla

Come anticipato, se volessimo considerare una regressione multipla, e per esempio considerare come predittori sia la potenza (cavalli) che il peso, possiamo implementare il modello in modo simile in R.

```
mtcars_reg_hp_wt <- lm(mpg ~ hp + wt, data = mtcars)
summary(mtcars_reg_hp_wt)
```

```
##
## Call:
## lm(formula = mpg ~ hp + wt, data = mtcars)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -3.941 -1.600 -0.182  1.050  5.854
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 37.22727    1.59879  23.285  < 2e-16 ***
```

```
## hp          -0.03177    0.00903  -3.519  0.00145 **
## wt          -3.87783    0.63273  -6.129 1.12e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.593 on 29 degrees of freedom
## Multiple R-squared:  0.8268, Adjusted R-squared:  0.8148
## F-statistic: 69.21 on 2 and 29 DF,  p-value: 9.109e-12
```

Chapter 7

Esercitazione 1

Si scarichi il dataset al link e si eseguano le operazioni:

1. Si importi il dataset.
2. Si crei un nuovo dataset contenente solo le righe corrispondenti alla variabile **Idataset** uguale all'ultimo numero del proprio numero di matricola. Dopo aver fatto ciò, si rimuova la variabile. Le seguenti analisi si intendono su questo dataset.
3. Si analizzi la struttura del dataset, si verifichi il numero di righe e colonne contenute. Il dataset è stato letto bene? E' tidy?
4. Si verifichi la presenza di eventuali errori nei dati e si sostituiscano con la mediana.
5. Si calcolino media, mediana, minimo, massimo e quartili e deviazione standard.
6. Si visualizzi la distribuzione delle variabili **x** e **y** e i loro box plot.
7. Si visualizzino i dati mediante uno scatterplot.

Il dataset completo e una discussione riguardo i dati usati per l'esercizio sono disponibili qui.

7.1 Esercizi

7.1.1 Esercizio 1

Si importi il dataset (descrizione) e si analizzi la relazione tra le due variabili usando la regressione lineare. Si analizzino i risultati e si visualizzino i residui.

7.1.2 Esercizio 2

Si trovi un modo adeguato per importare i dati relativi al reddito nazionale lordo pro capite e alla percentuale di strade asfaltate in R. Dopo aver analizzato e preparato i dataset, si usino i dati per investigare le due variabili rispetto ad un paese del G7, un paese in via di sviluppo ed un paese del terzo mondo a scelta, nel periodo dal 1990 al 2009. Si analizzino i dati e si visualizzino. Inoltre si usi la regressione lineare per analizzare la relazione tra le variabili. Si commentino e discutano i risultati.

7.1.3 Esercizio 3

Usando il dataset ottenuto nell'Esercizio 3 del capitolo 5, si investighi usando la regressione lineare la relazione tra prodotto interno lordo e aspettativa di vita per ognuno dei continenti. Si usi la funzione `predict()` per valutare il modello su dati diversi da quelli usati per definirlo. In particolare, si valuti il modello ottenuto per l'America sui dati europei e viceversa. Si visualizzino e si discutano i risultati ottenuti, dandone anche un interpretazione.

7.1.4 Esercizio 4

Si considerino i modelli ottenuto nell'Esercizio 3. Si valuti separatamente il modello ottenuto per l'America e quello l'Europa usando i dati dei paesi *Canada*, *Italy*, *Argentina* e *Sierra Leone* contentuti nel dataset originale. Si valutino i valori di R quadro ottenuti e si discutano i risultati.

Suggerimento si usi la formulazione equivalente di R quadro:

Bibliography

Baldi, P. (1998). *Calcolo delle probabilià e statistica*. McGraw-Hill Education, 2nd edition. ISBN 978-8838607370.

Peng, R. D. (2015). *Exploratory Data Analysis with R*. Leanpub.

Xie, Y. (2015). *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition. ISBN 978-1498716963.

Xie, Y. (2019). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.11.