Alcune risorse per R programming :

http://www.statmethods.net/

http://www.cookbook-r.com/

http://www.r-bloggers.com/

http://www.inside-r.org/blogs

Per visualizzare lo spazio di lavoro (workspace) basta eseguire il comando :

getwd() è cioè get work directory oppure setwd() per cambiare area di lavoro.

Se si vuole usare una libreria bisogna prima installarla/ scaricarla con (esempio ggplot2) :

install.packages("ggplot2")

poi per usarla bisogna fare :

library(ggplot2) per importare la libreria e usarla nel codice corrente.

vettori in R

```
v1 <- c(1,2,3,4)

v2 <- c(1:10)

v3 <- c("A", "B", "C")

v4 <- c(3,4,5,6)

v4 <- c(v1, v3)
```

R converte i numeri di v1 in stringhe per rendere compatibile il *mode* (e cioè i tipi)

Alcune funzioni statistiche

```
mean(v1)

median(v1)

sd(v1)

var(v1)

cor(v1, v4)

cov(v1, v4)
```

Di più su vettori e matrici

Strutture dati in R:

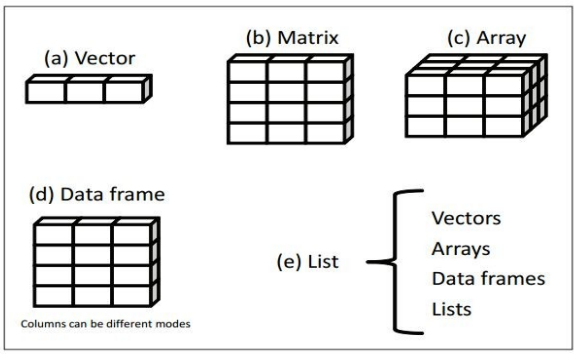


Figure 2.1 R data structures

Di più sui vectori : esempi

b <- c("one", "two", "three")

c <- c(TRUE, TRUE, TRUE, FALSE, TRUE, FALSE)

Per accedere ai vari elementi :

> a <- c(1, 2, 5, 3, 6, -2, 4)

> a[3]

[1] 5

> a[c(1, 3, 5)]

[1] 1 5 6

> a[2:6]

[1] 2 5 3 6 -2

Le matrici:

> y <- matrix(1:20, nrow=5, ncol=4) // dichiaro la matrice tramite la parola chiave matrix

> y // visualizzo la matrice

```
     [,1] [,2] [,3] [,4]
[1,]    1    6   11   16
[2,]    2    7   12   17
[3,]    3    8   13   18
```

[4,] 4 9 14 19

[5,] 5 10 15 20

Una matrice si può creare a partendo da un vettore :

```
> cells <- c(1,26,24,68)
 rnames <- c("R1", "R2")
 cnames <- c("C1", "C2")
mymatrix <- matrix(cells, nrow=2, ncol=2, byrow=TRUE,
dimnames=list(rnames, cnames))
> mymatrix  // visualizzo la matrice
```

C1 C2

R1 1 26

R2 24 68

Posso accedere alle righe o alle colonne usando i subscript come x[i,](per le righe) oppure x[,i] per le colonne

Si può inoltre prendere anche una parte della riga come nel seguente codice :

Per creare un array invece si potrebbe usare :

```
dim1 <- c("A1", "A2")
dim2 <- c("B1", "B2", "B3")
 dim3 <- c("C1", "C2", "C3", "C4")
 z <- array(1:24, c(2, 3, 4), dimnames=list(dim1, dim2, dim3))
```

IN questo caso si crea un array 3 dimensionale (2,3,4) (cioè un cubo con 3 righe , tre colonne e misura 4 di profondita)come si vede dalle dimensioni della list. Gli array in R sono l'estensione naturale delle matrix .

Data frames: è una matrice generica in quanto le righe / colonne possono contenere diversi tipi di dati (numeric, character..)

Ad esempio :

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

age <- c(25, 34, 28, 52)

diabetes <- c("Type1", "Type2", "Type1", "Type1")

 status <- c("Poor", "Improved", "Excellent", "Poor")

 patientdata <- data.frame(patientID, age, diabetes, status)
```

Quindi a partire dai vettori riesco a costruire un dataframe ocn l'istruzione data.frame. In questo modo posso rappresentare i dati come una matrice che contiene un insieme di dati di tipi diversi (numeri e char)

che visualizzato è :

```
patientID age diabetes    status

1  25    Type1     Poor

2  34    Type2  Improved

3  28    Type1 Excellent

4  52    Type1     Poor
```

Per accedere alle varie colonne bisogna usare :

```
table(patientdata$diabetes, patientdata$status)   // visualizzato è :

    Excellent  Improved  Poor

 Type1       1       0    2

 Type2       0       1    0
```

In particolare l'operatore '$' ha le stesse funzionalità che ha anche l'operatore '.' in java e cioè permette l'accesso a un attributo del oggetto in questo caso il dataframe.

Le liste o list(): è la struttura dati più complessa in R ed è una collezione orinata di oggetti, dove gli oggetti possono essere una struttura dati qualsiasi vista fin qui.

Ad esempio:

```
g <- "My First List"

h <- c(25, 26, 18, 39)

 j <- matrix(1:10, nrow=5)

 k <- c("one", "two", "three")
```

mylist <- list(title=g, ages=h, j, k)

che visualizza :

$title

[1] "My First List"

$ages

[1] 25 26 18 39

[[3]]

     [,1] [,2]

[1,]   1   6

[2,]   2   7

[3,]   3   8

[4,]   4   9

[5,]   5   10

[[4]]

[1] "one"   "two"   "three"

Le dataframe sono molto commode in quanto è possibile usare la funzione split() e dividere il dataframe secondo una certa Colonna. Ad esempio avendo un dataset come il seguente che si chiama airquality allora

 Ozone Solar.R Wind Temp Month Day

    41   190 7.4   67   5   1

    36   118 8.0   72   5   2

    12   149 12.6   74   5   3

    NA   286 8.6   78   6   1

    NA   287 9.7   74   6   2

   135   269 4.1   84   7   1

    49   248 9.2   85   7   2

    39   83 6.9   81   8   1

     9   24 13.8   81   8   2

Usando la funzione split() ho la possibilità di poter splitare i dati secondo una certa colonna in modo tale da poter ragruppare un po i dati.

s<- split(airquality, airquality$Month)

e poi chiamando la funzione str(s) posso visualizzare i dati come segue in base al mese:

List of 5

$ 5:'data.frame':	31 obs. of 6 variables:

..$ Ozone : int [1:31] 41 36 12 18 NA 28 23 19 8 NA ...

..$ Solar.R: int [1:31] 190 118 149 313 NA NA 299 99 19 194 ...

..$ Wind   : num [1:31] 7.4 8 12.6 11.5 14.3 14.9 8.6 13.8 20.1 8.6 ...

..$ Temp   : int [1:31] 67 72 74 62 56 66 65 59 61 69 ...

..$ Month : int [1:31] 5 5 5 5 5 5 5 5 5 5 ...

..$ Day   : int [1:31] 1 2 3 4 5 6 7 8 9 10 ...

$ 6:'data.frame':	30 obs. of 6 variables:

..$ Ozone : int [1:30] NA NA NA NA NA NA 29 NA 71 39 ...

..$ Solar.R: int [1:30] 286 287 242 186 220 264 127 273 291 323 ...

..$ Wind   : num [1:30] 8.6 9.7 16.1 9.2 8.6 14.3 9.7 6.9 13.8 11.5 ...

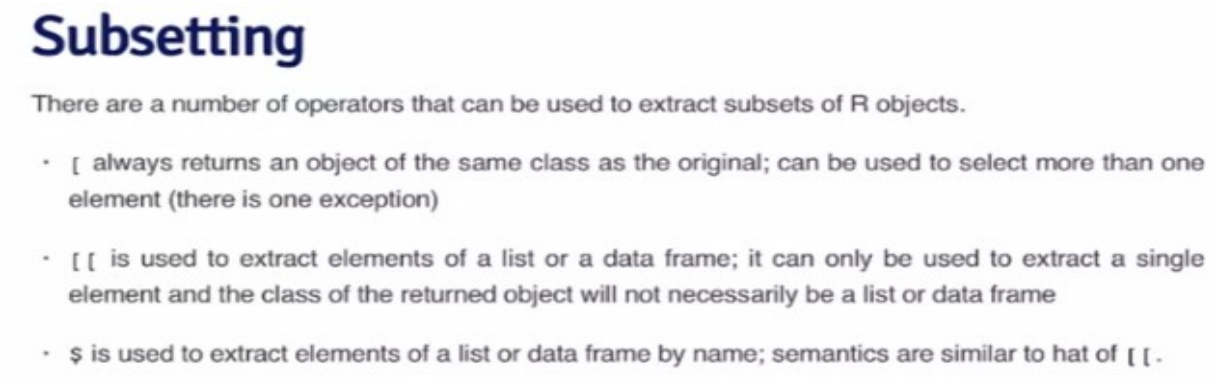
..$ Temp   : int [1:30] 78 74 67 84 85 79 82 87 90 87 ...

..$ Month : int [1:30] 6 6 6 6 6 6 6 6 6 6 ...

..$ Day   : int [1:30] 1 2 3 4 5 6 7 8 9 10 ...

Subsets in R :

A volte serve considerare solo una parte di un dataset per varie ragioni quindi si rende necessario il subsetting e cioè il sottoinsieme di un dataset che siano un sottoinsieme di righe o di colonne. Acune regole sul subsetting :



Nel caso di un vettore usando il suo indice si può fare subsetting in maniera molto facile :

```
x <- c ("a", "a", "b", "c", "d", "f", "g")
 x[1]
[1] "a"

 x[1:4]
[1] "a" "a" "b" "c"
```

Oppure usando una espressione logica:

```
x[x > "a"]
[1] "b" "c" "d" "f" "g"
```

Il subsetting di una lista come segue :

```
x <- list(a = 1:4, buz = 0.6, g = "hello")
```

avviene in diversi modi : usando l'indice oppure usando un vettore come espressione di indice e cioè :

```
x[c(1,3)]

che da come risultato il primo e il terzo elemento della lista x:
$a
[1] 1 2 3 4

$g
[1] "hello"

Inoltre posso accedere agli elementi usando l'operatore $ .. quindi:

x$g
[1] "hello"
```

```
Un'altra questione per il subsetting è quella di togliere i valori NA, Un esemp
io è dato dal seguente codice :

x <- c(1, 2, 3, NA, NA, 4)
bad <- is.na(x)
> bad
[1] FALSE FALSE FALSE  TRUE  TRUE FALSE

Dove bad rappresenta il vettore logico di valori NA e i loro indici.

Ovviamente ho che :
x[!bad]
[1] 1 2 3 4

Posso anche usare la funzione complete.cases() che mi ritorna solo i valori num
erici come nel caso sotto:

good <- complete.cases(x)
 good
[1]  TRUE  TRUE  TRUE FALSE FALSE  TRUE
 x[good]
[1] 1 2 3 4
```

Quindi partendo dalla matrice sotto posso voler considerare solo il campo subject con un valore minore di 3 cioe subject < 3.

```
data <- read.table(header=T, text='
```

```
  subject sex size
        1   M    7
        2   F    6
        3   F    9
        4   M   11
  ')
```

```
Per avere I subject < 3 eseguo il seguente codice:
```

```
subset(data, subject < 3)
```

```
La funzione subset() automaticamente mi prende le righe della matrice dove subj
ect è < 3 e ritorna il seguente risultato.
```

```
  subject sex size
1       1   M    7
2       2   F    6
```

Or if the dataframe has headers then it is possible to choose the columns based in the names of the column . This kind of subsetting is particulary usefull when the dataframe has a big number of columns as for exam ple the census dataset of Big data challenge 2015. This can be done as in the following example:
If the dataframe is census then the subset of relevant data can be :

Relevantdata = census[, c("ID_Region", "Region_name", "ID_province")]

This way we have a subset of three fields that are the ones that compare in the vector determining the colu mns of the relevant data and of course we take all the rows of the census data.

Strutture di controllo in R:

if-else

Partendo con due variabili come sotto :

```
> x <- 10
> y <- 20
```

Posso valutare con un if-else quale delle var ha il valore assoluto più grande :

```
> if(x > y){
  print(x)
 }else { print(y)
 }
```

```
Ovviamente ho :
[1] 20
```

Ciclo for : ci sono diversi modi di scrivere un ciclo for()

Ad es. partendo da un vettore `x <- c("a", "b", "c", "d")`

```
Posso scrivere :
for(letter in x){
  print(letter)
}
```

```
Un altro modo è anche :
for(i in 1:4){
   print(x[i])
}
```

```
Un altro modo ancora :
for (i in seq_along(x)){
   print(x[i])
}
```

Oppure :

```
for (i in 1:4) print(x[i])
```

In oltre si possono fare i cicli innestati come il seguente :

Partendo dalla matrice `y <- matrix(1:6, 2, 3)`

```
> y
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

```
for(i in seq_len(nrow(y))){

    for (j in seq_len(ncol(y))){

              print(y[i, j])
    }
 }
```

IN maniera analoga funziona anche il ciclo while.

```
count <- 0

while(count < 10){

     print(count)
     count <- count +1

}
```

La parola “next” in R si usa come “continue” in Java. Un esempio:

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

      if (i < 3) next

      print(i)

}
```

Che da come output :

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

Mentre la parola “break ” viene usata esatamente come in Java per uscire da un ciclo in maniera improvisa se viene verificata una certa condizione: +

```
for( i in 1:10){
+ if(i == 5) break
+ print(i)
+ }
[1] 1
[1] 2
[1] 3
[1] 4
```

Funzioni in R :

Per definire una funzione si usa la parola chiave function. In segiuto voiene riportato l’esempio di una funzione semplice che restituisce la somma di due variabili che prende in input:

```
add2 <- function(x, y){
  x + y
```

```
}
```

```
> add2(2,8)
[1] 10
```

Una funzione che legge diversi file e a partire da questi file crea un dataframe sul quale si possono calcolare summary statistics. IN questo caso si calcola la media dei valori di un certo campo.

```
> weightmean <- function(directory, day) {

    files_list <- list.files(directory, full.names = TRUE) #creates list files
     dat <- data.frame() #creates an empty data frame
     for (i in 1:5) {
        # loops through the files, rbinding them together
         dat <- rbind(dat, read.csv(files_list[i]))
     }
     #subsets the rows that match the 'day' argument
     dat_subset <- dat[which(dat[, "Day"] == day), ]

     mean(dat_subset[, "Weight"], na.rm = TRUE) #identifies the median weight
     # while stripping out the NAs
 }
```

Delle funzioni molto utili sono le apply che si chaimano loop functions. Queste funzioni permettono di scorrere un intervallo di numeri ed applicare la funzione a ogniuno dei numeri.

Alcune di queste funzioni sono lapply(), sapply(), apply(), tapply(), mapply(). Un'altra funzione utile è anche split() che permette di splittare gli oggetti in sotto pezzi.

Come funzionano queste funzioni:

lapply(): partendo dalla lista seguente posso applicare l'apply per calcolare la media dei due array che compongono la lista.

```
x <- list(a = 1:5, b = rnorm(10))
lapply(x, mean)
```

```
che mi da come risultato :
```

```
$a
[1] 3

$b
[1] -0.1454672
```

Un altro utilizzo della lapply() è anche quello con le funzioni anonime come segue:

```
x <- list(a = matrix(1:4, 2,2), b = matrix(1:6, 3,2))
lapply(x, function(elt) elt[,1])
```

```
che mi da come risultato :
```

```
$a
[1] 1 2

$b
[1] 1 2 3
```

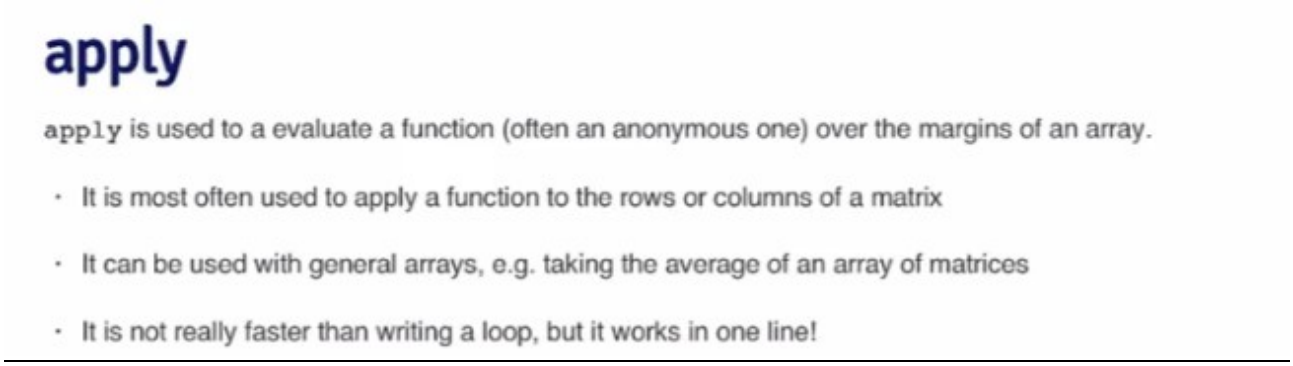
Il tipo di ritorno di una funzione lapply() è una lista dove probabilmente l sta per lista.

La funzione sapply() invece permette di avere come tipo di ritorno una matrice come nell'esempio seguente partendo da una lista è applicando la funzione sapply() con la media è possibile avere il risultato organizzato in una matrice.

```
x <- list(a = 1:5, b = rnorm(10), c = rnorm(20))
sapply(x, mean)
```

```
        a         b         c
3.0000000 0.4348445 0.2647948
```

La funzione apply() : un breve riassunto



Analogamente agli esempi precedenti partendo con una matrice come la seguente , 20 X 10 ho che :

```
x <- matrix (rnorm(200), 20, 10)

apply(x, 2, mean)
[1] -0.304457290  0.154454266  0.002727834  0.347782047 -0.036280071  0.0937104
[7] -0.077051814 -0.009154117 -0.027041981 -0.227033395

Quello che succede in questo caso è che la funzione apply calcola la media sull
a seconda dimensione della matrice e cioè sulle colonne (10 valori in tutto) e
ritorna per ogniuna la media della colonna stessa.

Con la seguente riga invece :

apply(x, 1, mean)

ottengo la media di tutte le righe sulle colonne. Quindi 20 valori in tutto.


[1] -0.33993241  0.26756958 -0.45141646  0.25874458  0.17029130 -0.30187436
[7]  0.15597552  0.12558340  0.37624312 -0.24176120  0.05922954 -0.05844885
[13]  0.74411702 -0.49464954 -0.06924623 -0.54846669  0.24205192  0.21549374
[19] -0.11785095 -0.15634113
```

Andando avanti con le dimensioni ho che per una matrice cubica di 2X2X10 (da immaginare come una pila di custodie per cd) per la quale voglio tenere la prima e la seconda dimensione ma voglio collassare la terza sulla quale voglio calcolare la media allora :

```
a <- array(rnorm(2*2*10), c(2,2,10))
```

Allora ho che la funzione apply() mi da :

```
apply(a, c(1,2), mean)
            [,1]       [,2]
[1,] -0.072240602 0.38098285
[2,] -0.001049388 0.07944459
```

```
Ho ottenuto quindi la media su ciascuna delle colonne della terza dimensione.
```

La funzione mapply() : è una funzione per dati multivariati dove mapply() prende come argomenti liste multiple e calcola per ogniuna un determinato risultato.

La funzione split() .. breve riassunto delle sue caratteristiche

split

`split` takes a vector or other objects and splits it into groups determined by a factor or list of factors.

```
> str(split)
function (x, f, drop = FALSE, ...)
```

- `x` is a vector (or list) or data frame
- `f` is a factor (or coerced to one) or a list of factors
- `drop` indicates whether empty factors levels should be dropped

Un'aspetto molto importante in R sono le cosidette scoping rules : che hanno a che fare con la visibilità delle funzioni e l'inclusione delle librerie nel environment . Alcune delle regole che valgono ogni volta sono riportate nella schermata sotto:

Scoping Rules

The scoping rules for R are the main feature that make it di↩erent from the original S language.

- The scoping rules determine how a value is associated with a free variable in a function
- R uses *lexical scoping* or *static scoping*. A common alternative is *dynamic scoping*.
- Related to the scoping rules is how R uses the search *list* to bind a value to a symbol
- Lexical scoping turns out to be particularly useful for simplifying statistical computations

Lexical Scoping

Lexical scoping in R means that

the values of free variables are searched for in the environment in which the function was defined.

What is an environment?

- An *environment* is a collection of (symbol, value) pairs, i.e. `x` is a symbol and `3.14` might be its value.
- Every environment has a parent environment; it is possible for an environment to have multiple "children"
- the only environment without a parent is the empty environment
- A function + an environment = a *closure* or *function closure*.

Lexical Scoping

Searching for the value for a free variable:

- If the value of a symbol is not found in the environment in which a function was defined, then the search is continued in the *parent environment*.
- The search continues down the sequence of parent environments until we hit the *top-level environment*; this usually the global environment (workspace) or the namespace of a package.
- After the top-level environment, the search continues down the search list until we hit the *empty environment*. If a value for a given symbol cannot be found once the empty environment is arrived at, then an error is thrown.

Alcune regole (best practises) per scrivere codice in R:

Coding Standards for R

1. Always use text files / text editor
2. Indent your code
3. Limit the width of your code (80 columns?)
4. Limit the length of individual functions

Le date in R : come viene rappresentato il tempo in R. Usando la funzione as.Date si possono rapp. delle date a partire dal 01-01-1970.

```
x <- as.Date("1970-01-01")
x
[1] "1970-01-01"
> unclass(x)
[1] 0
```

```
La funzione inclass(x) da 0 perche quello è il promo giorno da quando è iniziat
a il conteggio dei giorni.
```

Visualizzazione dati e tipi di grafici :

Grafico scatterplot 0.

Dato il dataset builtin di R

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

6 9 10

7 10 18

8 10 26

9 10 34

10 11 17

11 11 28

12 12 14

13 12 20

14 12 24

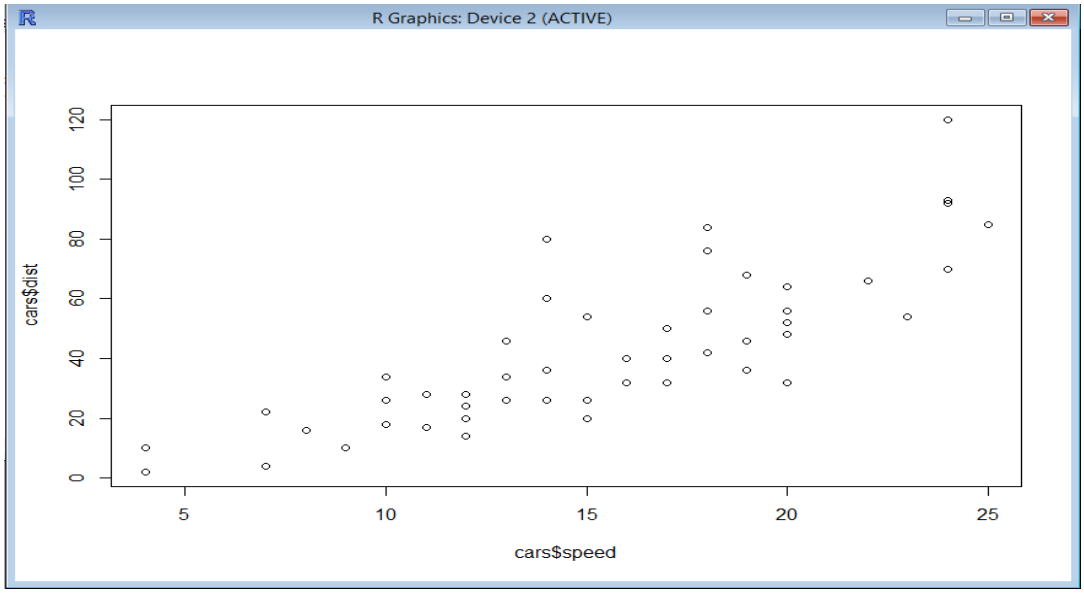
15 12 28

16 13 26

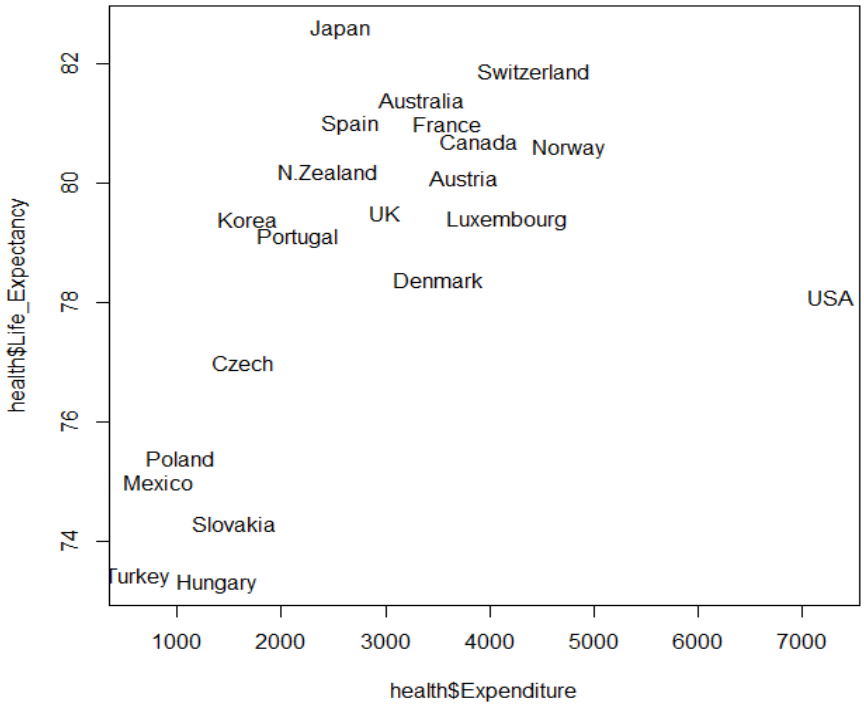
30 17 40

31 17 50

Con il comando ; *plot(cars$dist ~ cars$speed)* si può ottenere lo scatterplot seguente :



Un altro tipo di scatterplot con delle etichette potrebbe essere il seguente :



Si fa a partire dal dataset seguente:

"Country","Expenditure","Life_Expectancy","Doctor_Visits"

"Australia",3357,81.4,6.3

"Austria",3763,80.1,6.7

"Canada",3895,80.7,5.8

"Czech",1626,77,12.6

"Denmark",3512,78.4,7.5

"France",3601,81,6.3

"Hungary",1388,73.3,10.8

"Japan",2581,82.6,13.6

"Korea",1688,79.4,11.8

"Luxembourg",4162,79.4,6.1

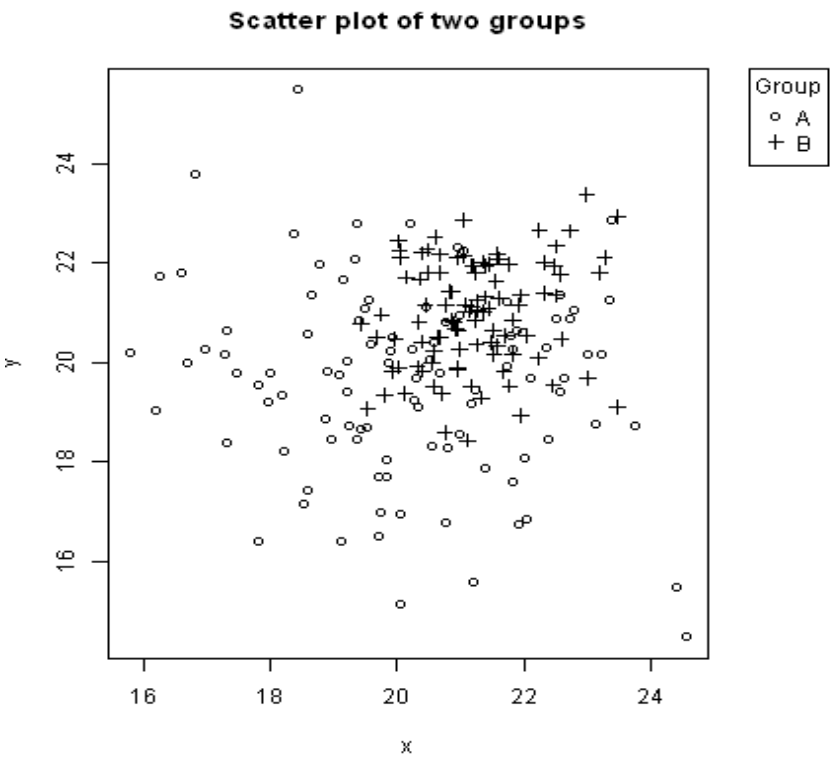
Con i comandi seguenti :

```
health<-read.csv("HealthExpenditure.csv",header=TRUE)
plot(health$Expenditure,health$Life_Expectancy,type="n")
text(health$Expenditure,health$Life_Expectancy,health$Country)
```

Per mettere la leggenda fuori dallo spazio di plotting invece si potrebbe procedere cosi:

```
A <- data.frame(x=rnorm(100, 20, 2), y=rnorm(100, 20, 2))
B <- data.frame(x=rnorm(100, 21, 1), y=rnorm(100, 21, 1))

# Add extra space to right of plot area; change clipping to figure
par(mar=c(5.1, 4.1, 4.1, 8.1), xpd=TRUE)

# Plot both groups
plot(y ~ x, A, ylim=range(c(A$y, B$y)), xlim=range(c(A$x, B$x)), pch=1,
               main="Scatter plot of two groups")
points(y ~ x, B, pch=3)

# Add legend to top right, outside plot region
legend("topright", inset=c(-0.2,0), legend=c("A","B"), pch=c(1,3), title="Group")
```

per avere come risultato la figura seguente:



Fare scatterplot tridimesionali:

Partendo dal seguente dataset mtcars di R :

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4

Datsun 710 22.8 4 108.0 93 3.85 2.320 18.61 1 1 4 1

Hornet 4 Drive 21.4 6 258.0 110 3.08 3.215 19.44 1 0 3 1

Hornet Sportabout 18.7 8 360.0 175 3.15 3.440 17.02 0 0 3 2

Valiant 18.1 6 225.0 105 2.76 3.460 20.22 1 0 3 1

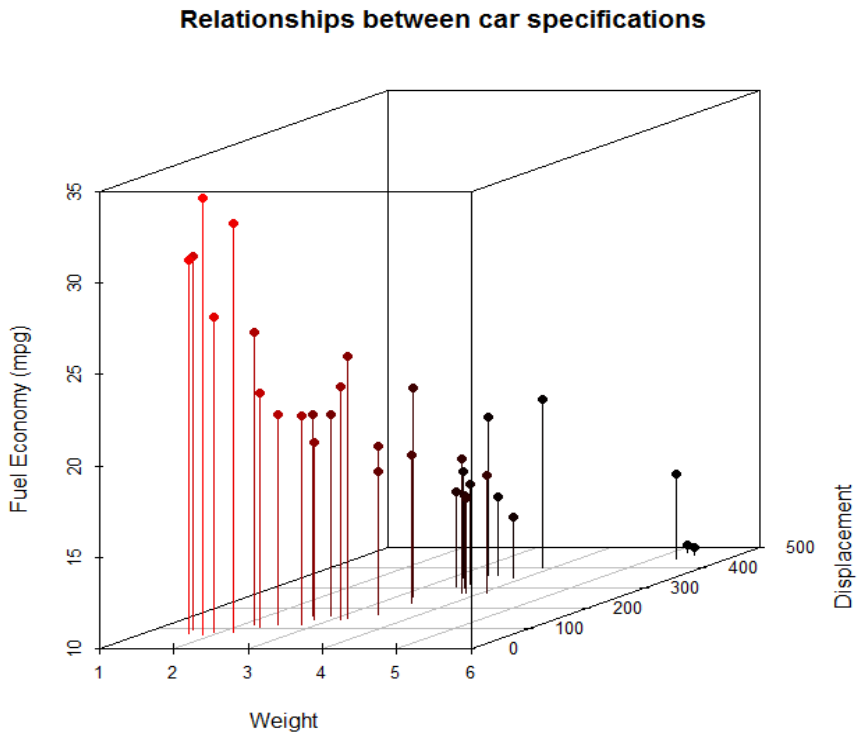
Duster 360 14.3 8 360.0 245 3.21 3.570 15.84 0 0 3 4

Merc 240D 24.4 4 146.7 62 3.69 3.190 20.00 1 0 4 2

Posso visualizzare i dati in un cubo con anche riferimento all 'asse z. Questo mi permette di capire meglio come sono disposti i valori e il codice seguente permette di fare questo :

```
scatterplot3d(mtcars$wt,mtcars$disp,mtcars$mpg,
pch=16, highlight.3d=TRUE, angle=20,
xlab="Weight",ylab="Displacement",zlab="Fuel Economy (mpg)",
type="h", main="Relationships between car specifications")
```

Il risultato e dato nella figura seguente ;



Prima però bisogna installare il package scatterplot3d con : install.packages("scatterplot3d").

Grafico a linee 1.1: avendo il seguente dataset:

date,units
01/01/2010,5063.782

02/01/2010,6115.308
03/01/2010,5305.093
04/01/2010,3184.974
05/01/2010,4181.691
.....

25/01/2010,6960.612
26/01/2010,4715.019
27/01/2010,4429.967
28/01/2010,5538.701
29/01/2010,5130.118
30/01/2010,5369.892
31/01/2010,4879.031

Posso caricare i dati con la seguente linea di codice R :

```
sales<-read.csv("dailysales.csv", header=TRUE)
```

Carico quindi in un ogetto sales il contenuto del dataset con il metodo read.csv() . Poi eseguendo il codice :

```
plot(sales$units~as.Date(sales$date,"%d/%m/%y"),
type="l", #Specify type of plot as l for line
main="Unit Sales in the month of January 2010",
xlab="Date",
ylab="Number of units sold",
col="blue")
```

Il risultato è il segunete

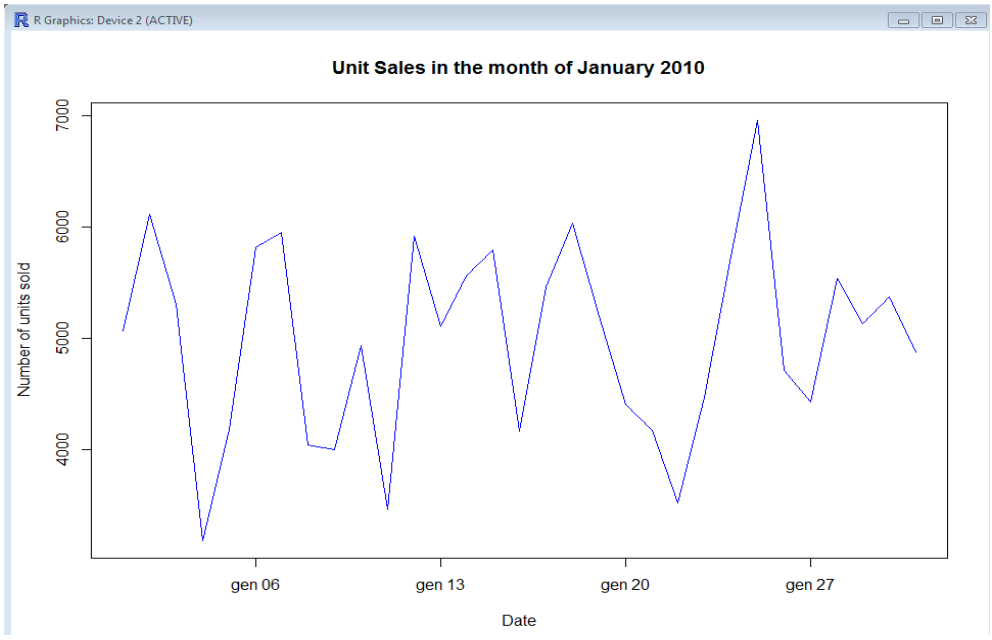


Grafico a barre 1.2 :

DataSet

City,ProductA,ProductB,ProductC

Seattle,23,11,12

London,89,6,56

Tokyo,24,7,13

Berlin,36,34,44

Mumbai,3,78,14

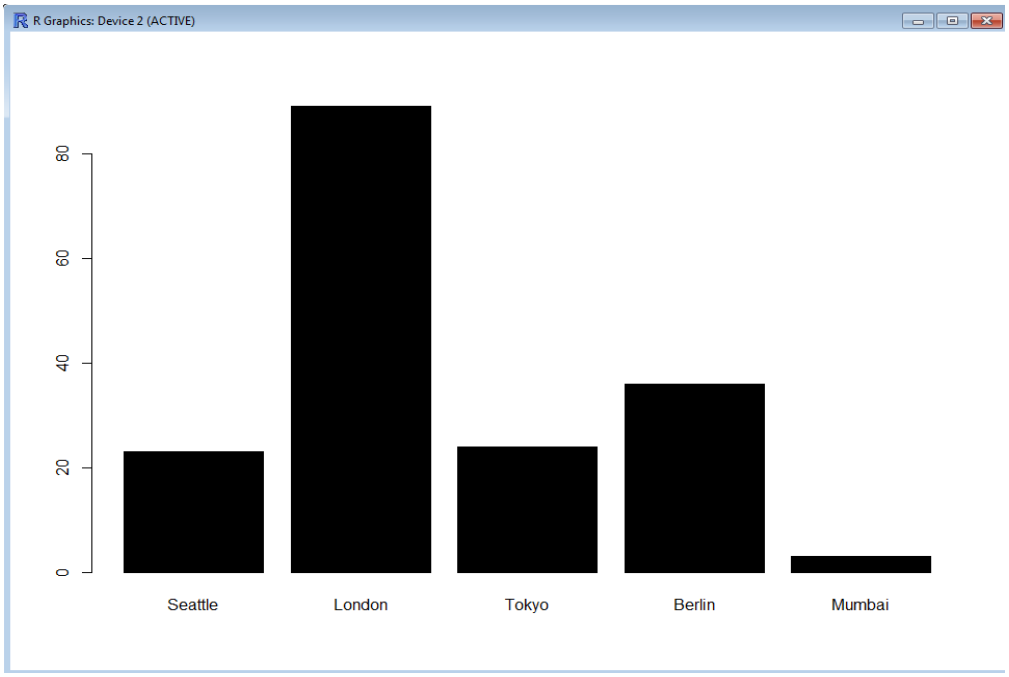
Partendo dal dataset sopra posso caricare di nuovo i dati in un oggetto

```
sales2<-read.csv("citysales.csv",header=TRUE)
```

Poi eseguendo il seguente comando :

```
barplot(sales2$ProductA,
names.arg= sales2$City,
 col="black")
```

ho come risultato il seguente grafico :



L'orientamento delle barre è per default verticale. Per visualizzare il grafico con le barre orizzontali invece si può eseguire il seguente comando :

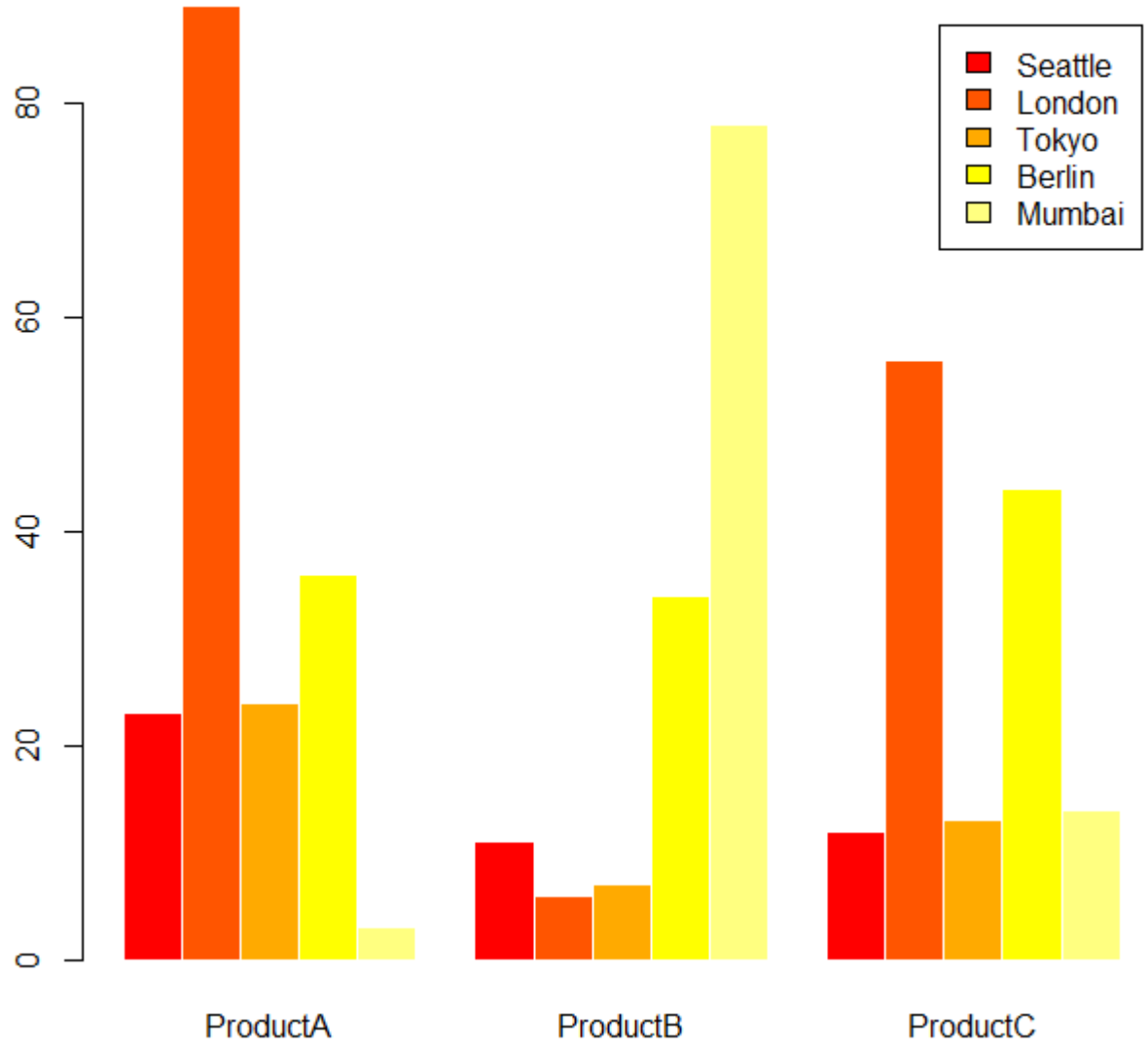
```
barplot(sales$ProductA,
names.arg= sales$City,
horiz=TRUE,
col="black")
```

Dove l'attributo horiz = TRUE è quello che realizza il diagramma orizontale

Il diagramma a barre può essere visualizzato anche in maniera raggruppata per eveidenziare le differenze tra le varie grandezze.

Con il codice seguente è possibile realizzare il diagramma :

```
barplot(as.matrix(sales2[,2:4]), beside=TRUE,
legend=sales2$City,
col=heat.colors(5),
 border="white")
```



IN particolare la notazione as.matrix(sales[,2:4]) permette di riferirsi a tutte le righe delle colonna 2 fino a 4. Cosi se ad esempio se si volesse riferire a tutte le righe della colonna 2 allora avremo sales[,2] e dall'altra parte per tutte le colonne della riga 3 avremmo sales[3,]

Creare diagrammi a barre sovrapposte a pila: dopo aver installato le seguenti librerie :

```
install.packages("RColorBrewer")
library(RColorBrewer)
```

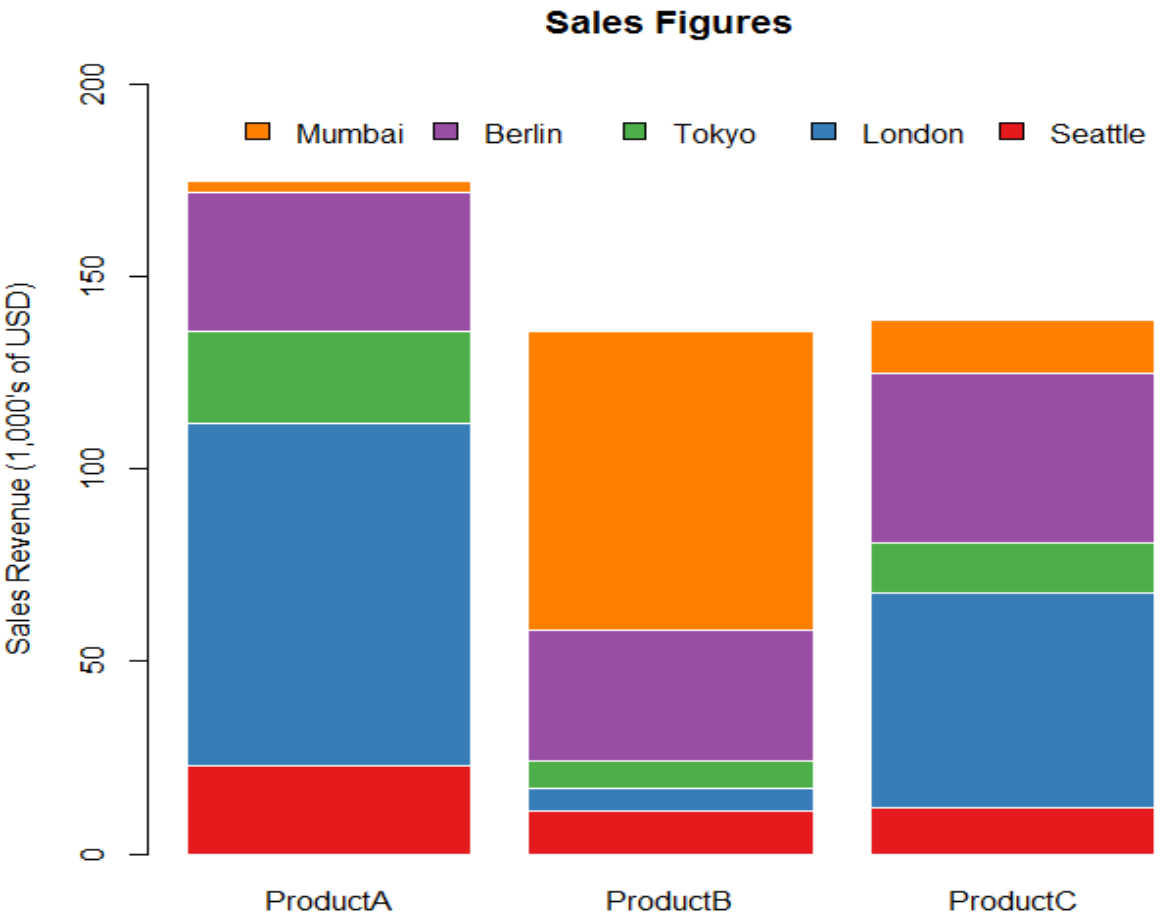
si possono inserire le seguenti righe di codice :

```
citysales<-read.csv("citysales.csv")
barplot(as.matrix(citysales[,2:4]),
legend.text=citysales$City,
args.legend=list(bty="n",horiz=TRUE),
col=brewer.pal(5,"Set1"),border="white",
ylim=c(0,200),ylab="Sales Revenue (1,000's of USD)",
main="Sales Figures")
```

che caricano i dati del file :

```
City,ProductA,ProductB,ProductC
Seattle,23,11,12
London,89,6,56
Tokyo,24,7,13
Berlin,36,34,44
Mumbai,3,78,14
```

E visualizzano il risultato mostrato nella figura seguente :



Creare diagrammi a torta: si userà il seguente dataset;

Browser	Share
IE8	13.50
IE7	12.80
IE6	10.90
Firefox	46.40
Chrome	9.80
Safari	3.60
Opera	2.30

Le istruzioni in R sono :

Per avere come risultato :

browsers<-read.table("browsers.txt",header=TRUE)

```
browsers<-browsers[order(browsers[,2]),]
pielabels <- sprintf("%s = %3.1f%s", browsers[,1],
100*browsers[,2]/sum(browsers[,2]), "%")
pie(browsers[,2],
labels=pielabels,
clockwise=TRUE,
radius=1,
col=brewer.pal(7,"Set1"),
border="white",
cex=0.8,
main="Percentage Share of Internet Browser usage")
```

Per avere il risultato seguente;

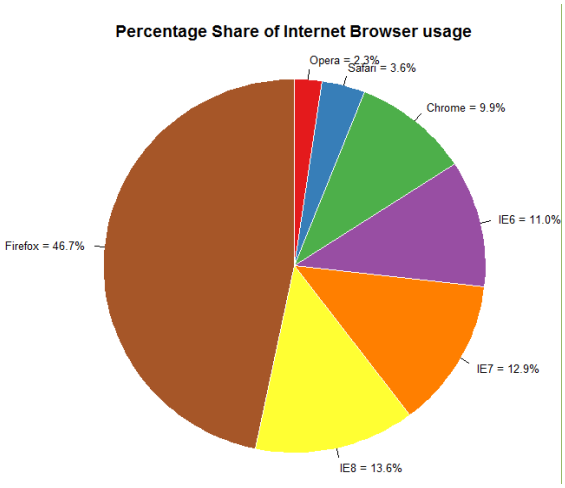


Grafico a punti 1.3 :

Si può anche fare un grafico di punti (x, y) come quello seguente che viene genarato a partire da due vettori X, Y.

```
x <-c(1,2,3,4,5,6,7,8,9,10)

> y <- c(10, 100, 1000, 10000, 20000, 100000, 1000000, 2000000, 3000000, 5000000)

> data <-c(x,y)

> plot(data)
```

che da come risultato :

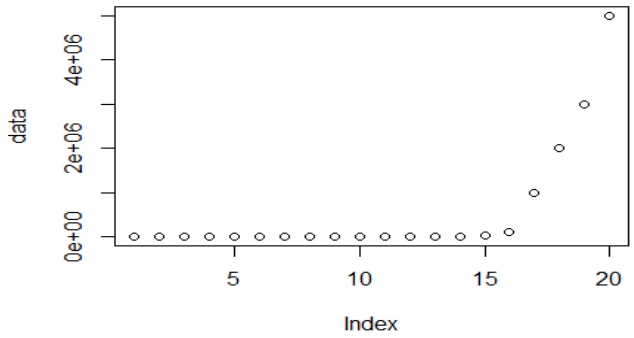
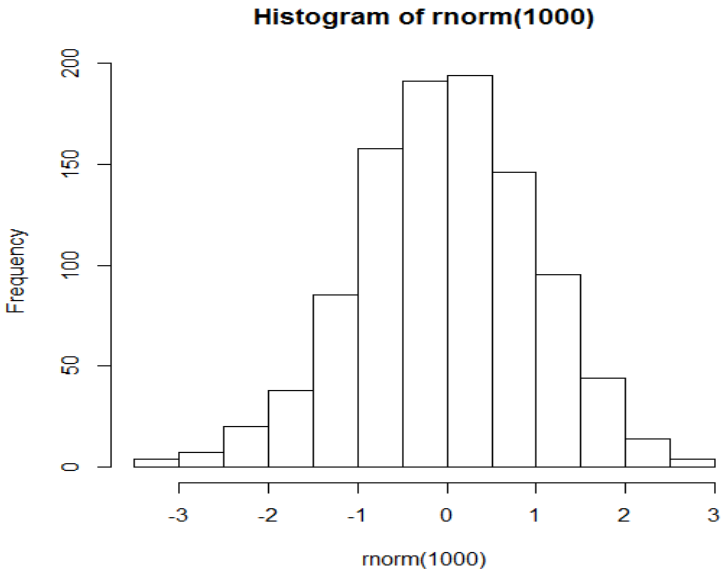


Grafico histogramma 1.4

Con il seguente comando viene visualizzato un istogramma che rappresenta i valori delle distribuzione normale.

hist(rnorm(1000))

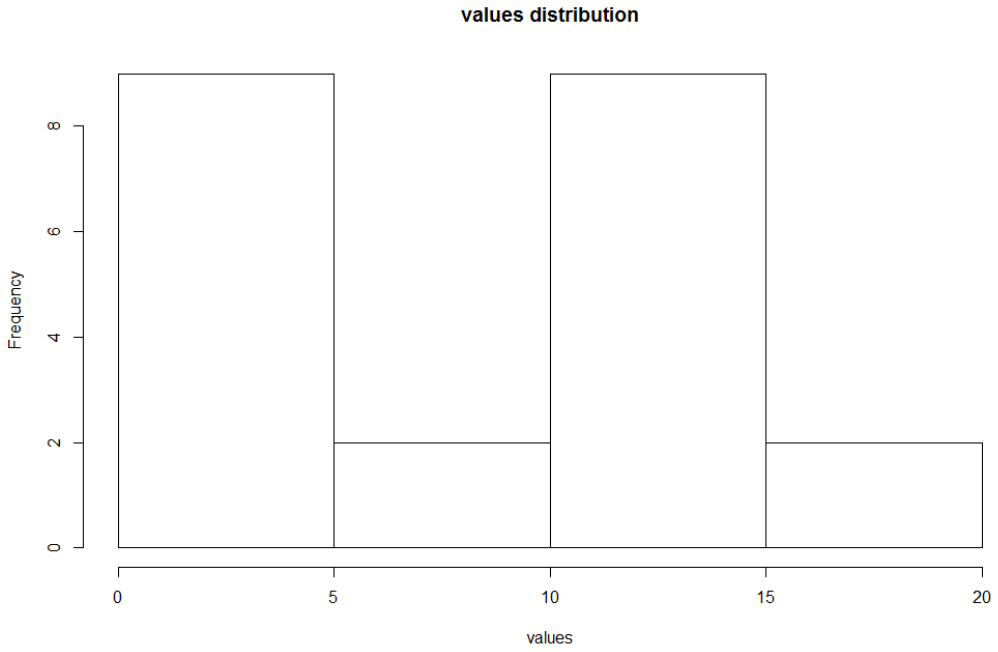


Un altro esempio un pò più semplificato di un istogramma è il seguente ;

```
data4hist2 <- c(1,1,2,2,2,2,4,4,4,10,10,20,20,12,12,12,12,12,13,13,13,13)

hist(data4hist2, xlab = "values", main = "values distribution")
```

per avere poi un grafico come il seguente :

```
data4hist2 <- c(1,1,2,2,2,2,4,4,4,10,10,20,20,12,12,12,12,12,13,13,13,13)

 hist(data4hist2, xlab = "values", main = "values distribution")
```



Che vuol dire che ci sono 9 valori compresi nell'intervallo (0:5), oppure che ci sono due valori copmresi nell'intervallo (5:10).

Agiungendo l'attributo (col = "blue") si potrebbe avere un istogramma colorato.

Grafico BoxPlot 1.5:

Dato il seguente dataset che rappresenta la presenza di un certo elemento chimico in un sito allora:

Source,As,Ba,Cd,Cr,Cu,Mn,Mo,Ni,Pb,Sb,Sr,V,Zn

Site1,0.701728639,76.1784901,0.081646023,-3.470121296,19.69988853,3.082432845,0.7452889,-1.862277092,7.283459886,4.69856669,0.742519071,-8.716315033,52.88063002

Site2,-1.919269099,41.009008,-0.00344909,30.68559639,10.64262703,4.512707246,0.938550197,7.227271387,4.803263316,2.502242791,0.638923435,24.41249015,34.02637507

Site1,0.739368011,71.91438326,0.654752517,51.51144589,33.79154657,14.81578476,3.466994798,17.95716527,50.66618538,7.755198992,2.201367325,47.1806681,88.258508

Site2,-2.596413594,41.87574546,0.082761457,22.84303936,5.353111489,4.546102165,0.572534428,5.640279732,4.722584549,1.352985708,0.648626049,17.08458493,44.91797478

Site3,-1.984350877,49.22437596,0.111075531,26.10995766,16.44035005,8.423136507,1.42188533,7.962349927,5.763325223,2.86121309,0.997384501,23.04081277,62.09072039

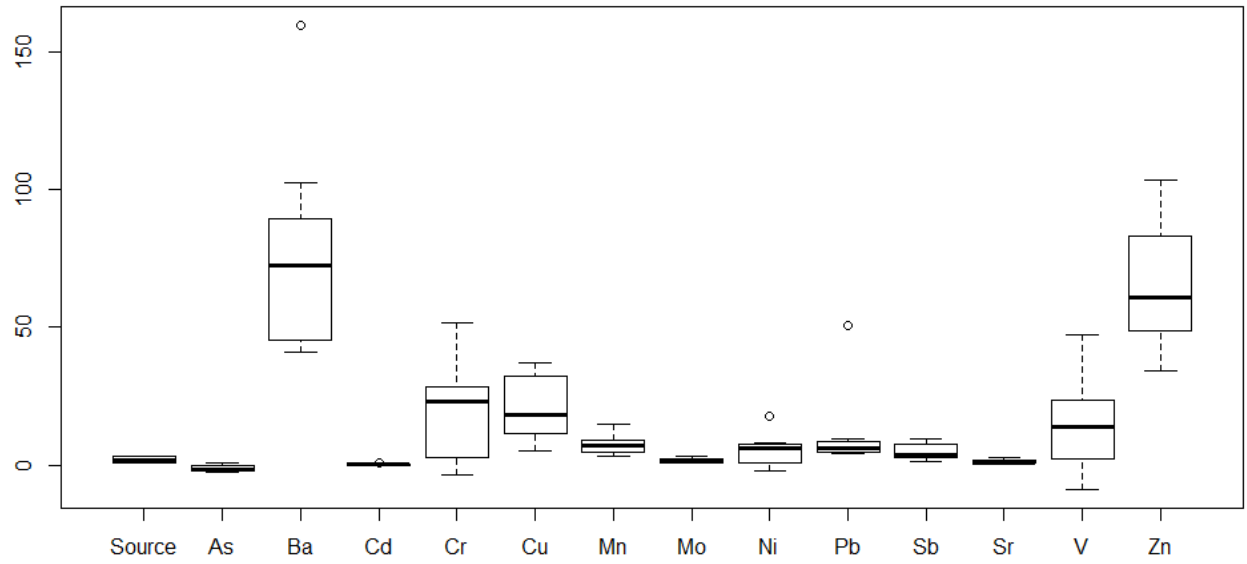
Site1,-0.974415802,73.21346682,0.066013499,23.45430786,12.72923324,5.84559729,1.197829328,6.72884248,4.347979114,2.97124821,1.011529389,10.83609464,78.09816877

Site3,-2.519845713,102.6626292,0.230340797,5.546926802,36.90032728,8.901423424,2.385792945,-0.369218342,9.564525784,9.516278135,1.812854556,-0.859273949,59.73824924

Site3,-0.673901264,159.5702943,0.165666992,-0.225998726,30.7134094,9.08540307,1.967770568,2.038494867,6.931350723,6.967700885,2.99032634,5.372554185,103.538705

Posso costruire il boxplot della distribuzione degli elementi

```
boxplotdiag <-read.csv("metals2.csv", TRUE, sep = ",")
boxplot(boxplotdiag)
```



Un'altra versione di un box plot o comunque un diagramma abbastanza simile è il digramma d'errore.

Partendo dal seguente dataset :

points,mean,std,percentile25,percentile75

1,565.9783050847458,636.3455115834935,112.0,818.0

2,437.2801899592944,518.172933412794,80.75,616.25

3,353.3324269204621,442.3187990911216,56.0,491.0

4,306.5212038303697,399.0969827898025,44.0,421.0

5,266.48280605226927,362.24819630494767,32.0,359.0

Posso costruire il diagramma d'errore come segue:

```
datasts <- read.csv("statsd4d.txt", TRUE)
 x <- datasts$points
 y <- datasts$mean
```

devstd<-datasts$std

Poi uso la funzione :

errbar(x, y, y + devstd, y - devstd, xlab = "number of points in comon")

della libreria **library(Hmisc).**

Il risultato è mostrato nella fig seguente :

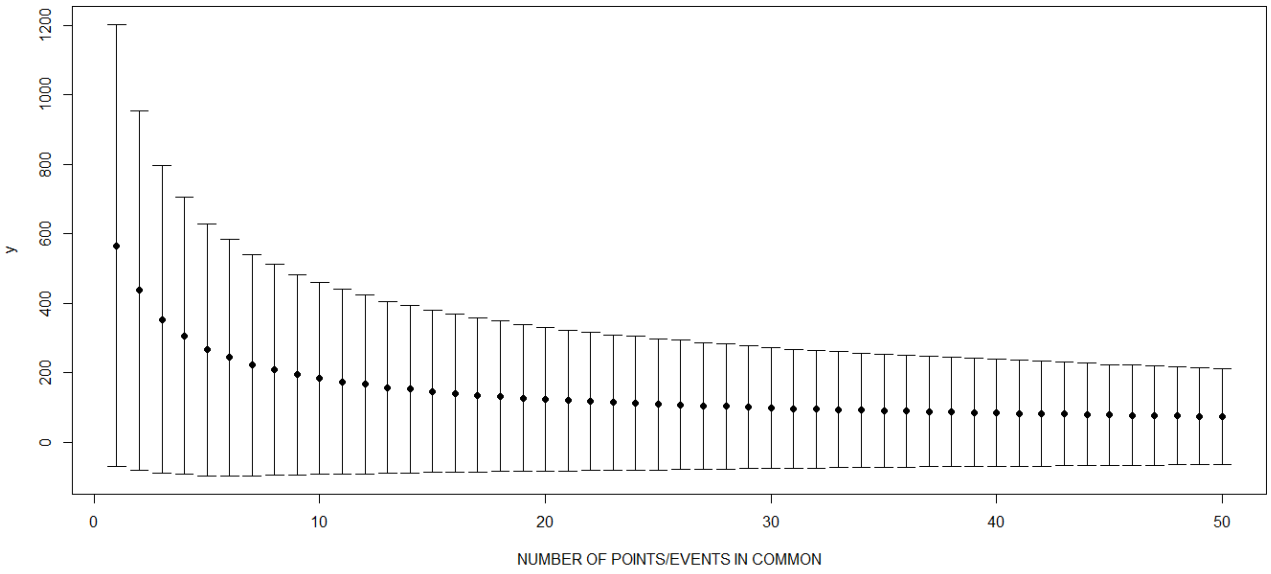


Grafico 1.6: Creare mappe di calore con R:

Usando il dataset "cars" built-in di R che si ottiene semplicemente scrivendo mtcars nella console di R.

Tale dataset è fatto in questo modo :

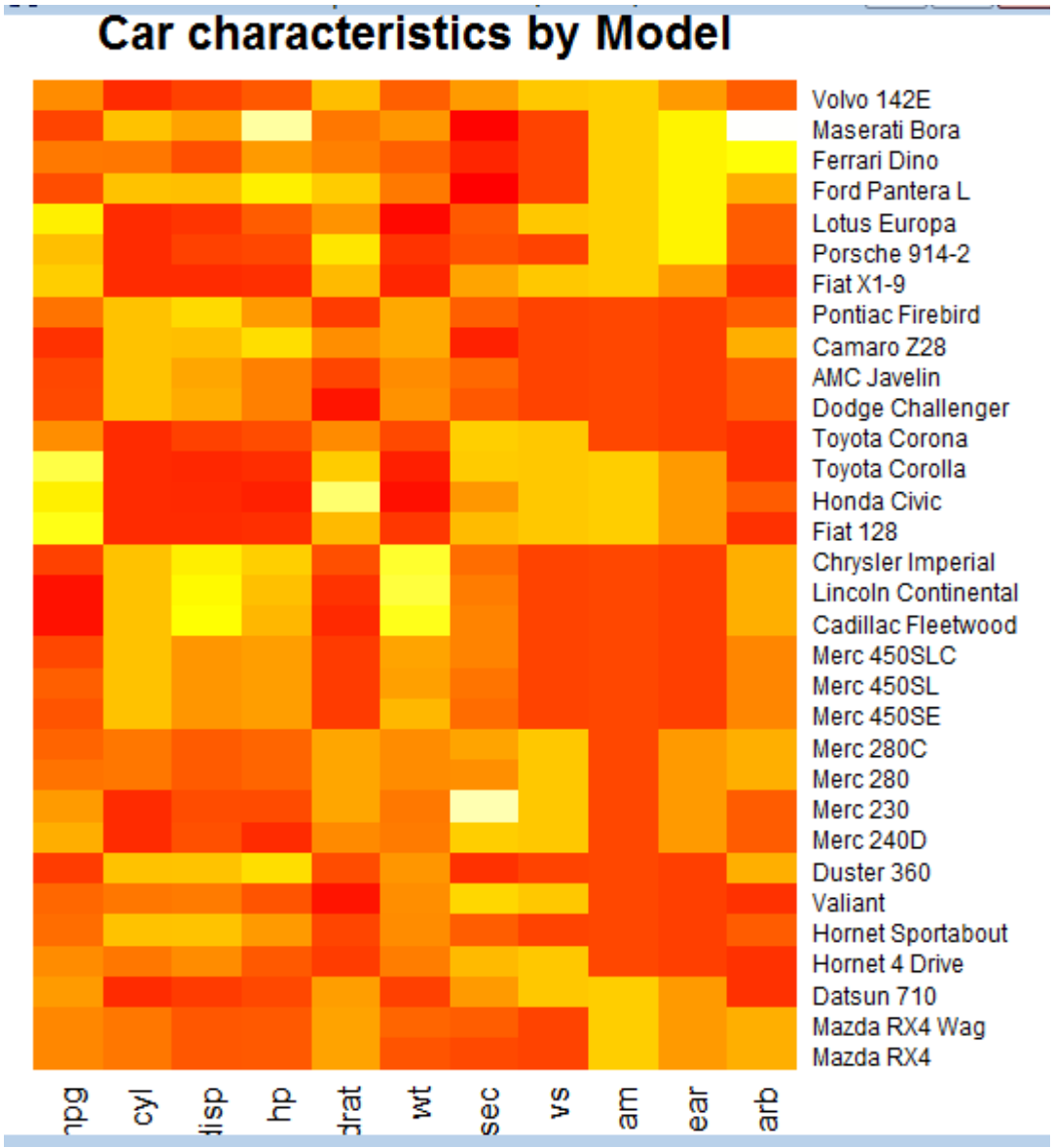
	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1

……..

Con il seguente commando :

```
        heatmap(as.matrix(mtcars),
        Rowv=NA,
        Colv=NA,
        col = heat.colors(256),
        scale="column",
        margins=c(2,8),
main = "Car characteristics by Model")
```

posso creare la mappa di calore seguente :



Un altro tipo di heatmap è dato nell'esempio seguente:

Partendo dal dataset seguente :

	Month	London	NewYork	Tokyo	Paris
1	Jan	5064	3388	7074	8701
2	Feb	6115	4459	4603	8249

3 Mar 5305 5091 4787 8560

4 Apr 3185 4015 6214 7144

5 May 4182 4864 4700 8645

6 Jun 5816 4333 4592 10172

7 Jul 5947 4895 5719 5337

8 Aug 4049 4520 4219 11076

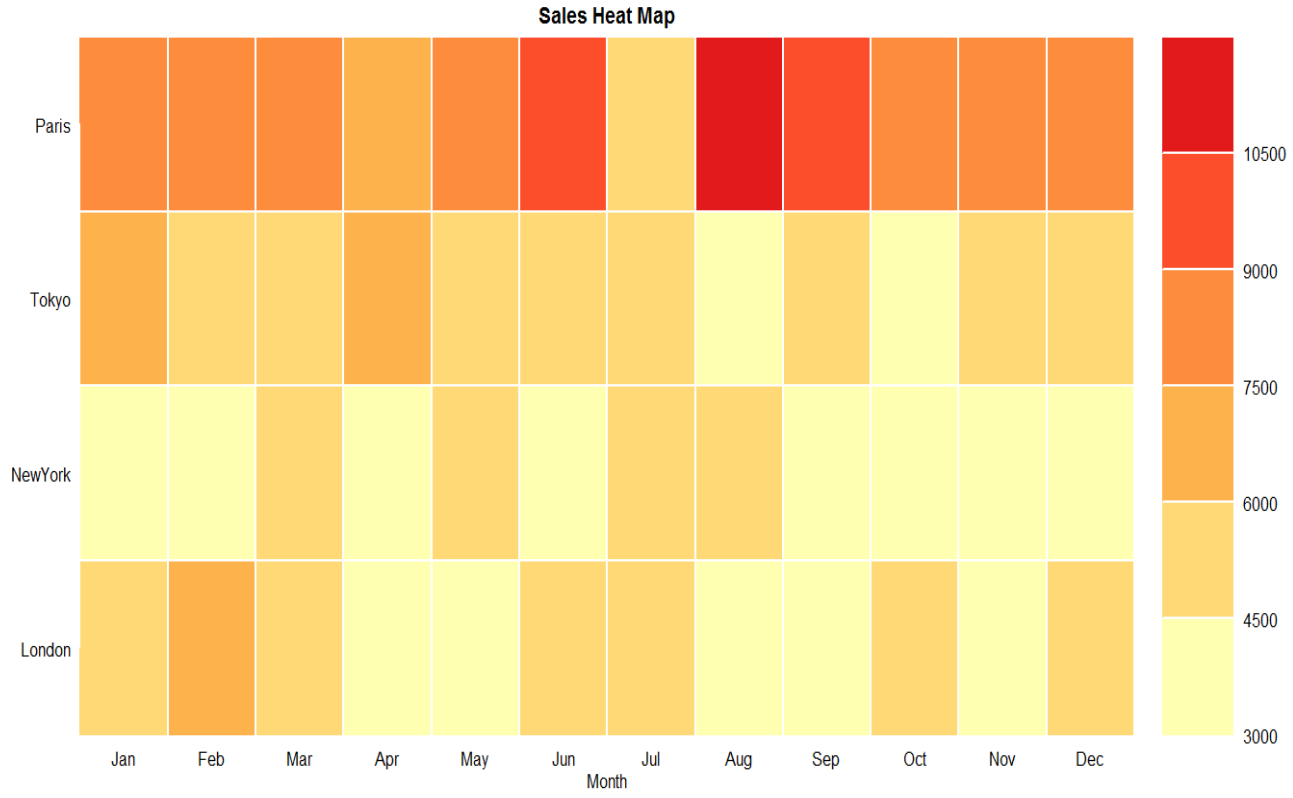
9 Sep 4003 3649 5079 10026

10 Oct 4937 3986 4499 7556

11 Nov 3470 3551 4540 8539

12 Dec 5915 3514 5658 7812

Posso visualizzare i dati in maniera tale da avere una figura del tipo :

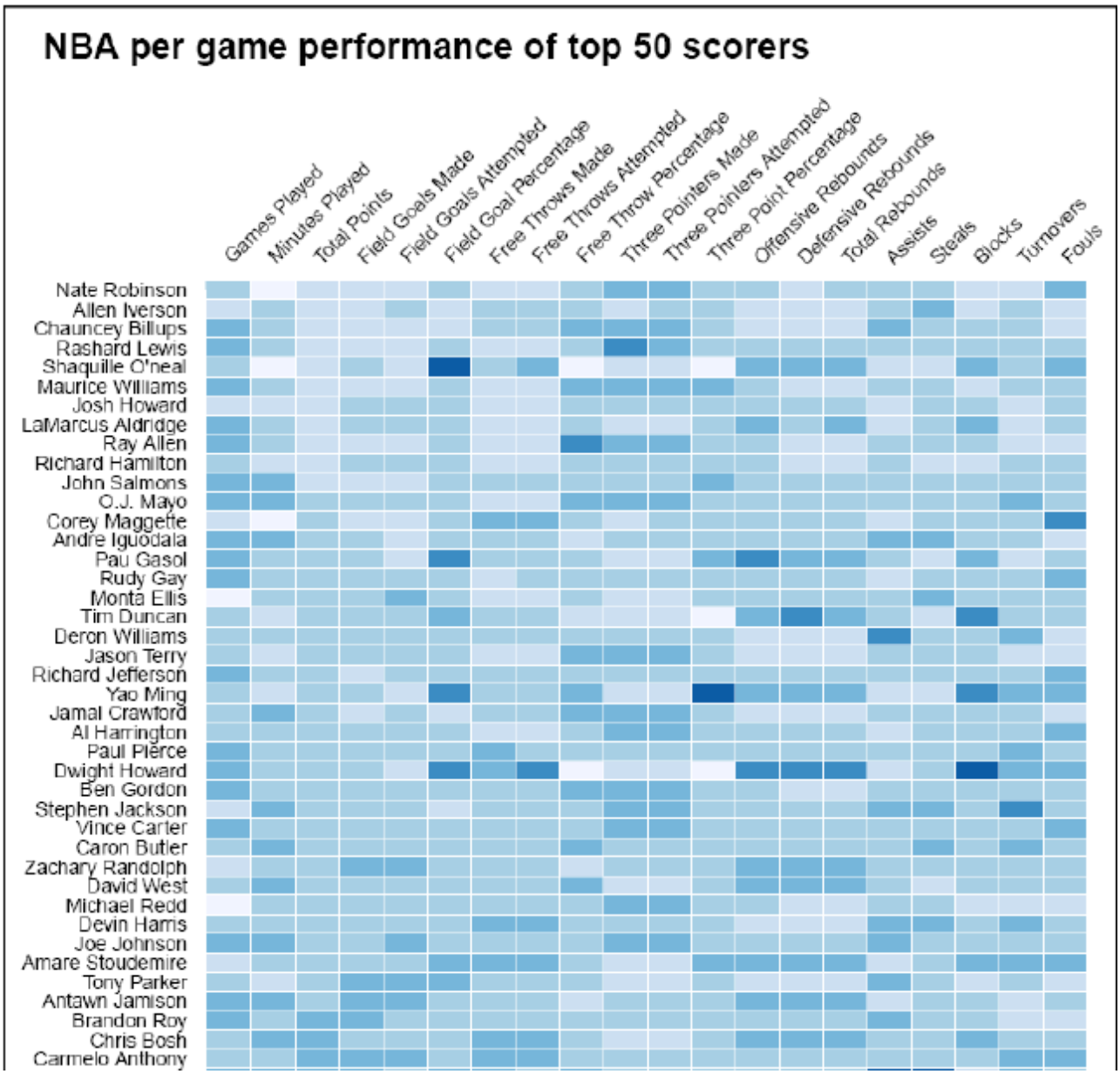


Per fare questo basta inserire le seguenti righe di codice:

```
sales<-read.csv("sales.csv")
 library(RColorBrewer)
 rownames(sales)<-sales[,1]
```

```
row.names(sales)<-sales[,1]

sales<-sales[,-1]

data_matrix<-data.matrix(sales)

pal=brewer.pal(7,"YlOrRd")

breaks<-seq(3000,12000,1500)

layout(matrix(data=c(1,2), nrow=1, ncol=2), widths=c(8,1),  heights=c(1,1))

par(mar = c(5,10,4,2),oma=c(0.2,0.2,0.2,0.2),mex=0.5)

image(x=1:nrow(data_matrix),y=1:ncol(data_matrix),  z=data_matrix,axes=FALSE,xlab="Month",

ylab="",col=pal[1:(length(breaks)-1)],

breaks=breaks,main="Sales Heat Map")

axis(1,at=1:nrow(data_matrix),labels=rownames(data_matrix), col="white",las=1)

axis(2,at=1:ncol(data_matrix),labels=colnames(data_matrix), col="white",las=1)

abline(h=c(1:ncol(data_matrix))+0.5, v=c(1:nrow(data_matrix))+0.5, col="white",lwd=2,xpd=FALSE)

breaks2<-breaks[-length(breaks)]

par(mar = c(5,1,4,7))

image(x=1, y=0:length(breaks2),z=t(matrix(breaks2))*1.001,  col=pal[1:length(breaks)-
1],axes=FALSE,breaks=breaks,  xlab="", ylab="",xaxt="n")

axis(4,at=0:(length(breaks2)-1), labels=breaks2, col="white",  las=1)

abline(h=c(1:length(breaks2)),col="white",lwd=2,xpd=F)
```

In maniera del tutto analoga si potrebbe fare anche la seguente fig :



Visualizzare un TreeMap in R.

Se i dati sono gerarchici (cioè innestati)allora si possono visualizzare con una TreeMap come :

id,views,comments,category

5019,148896,28,Artistic Visualization

1416,81374,26,Visualization

1416,81374,26,Featured

3485,80819,37,Featured

3485,80819,37,Mapping

3485,80819,37,Data Sources

500,76495,10,Statistical Visualization

I dati sopra riportati sono stati presi dal sito flowing data e rappresentano l'attività degli utenti del sito e cioè commenti fatti su post , pagine visitate etc.. dove tali attività sono divisi per categoria anche in base al tipo di post. ("post id, number of views, number of comments, and category").

Le seguenti line di codice dovrebbero visualizzare un treemap.

```
data <- read.csv("http://datasets.flowingdata.com/post-data.txt")
install.packages("portfolio")
library(portfolio)
map.market(id=data$id, area=data$views, group=data$category,
color=data$comments, main="FlowingData Map")
```

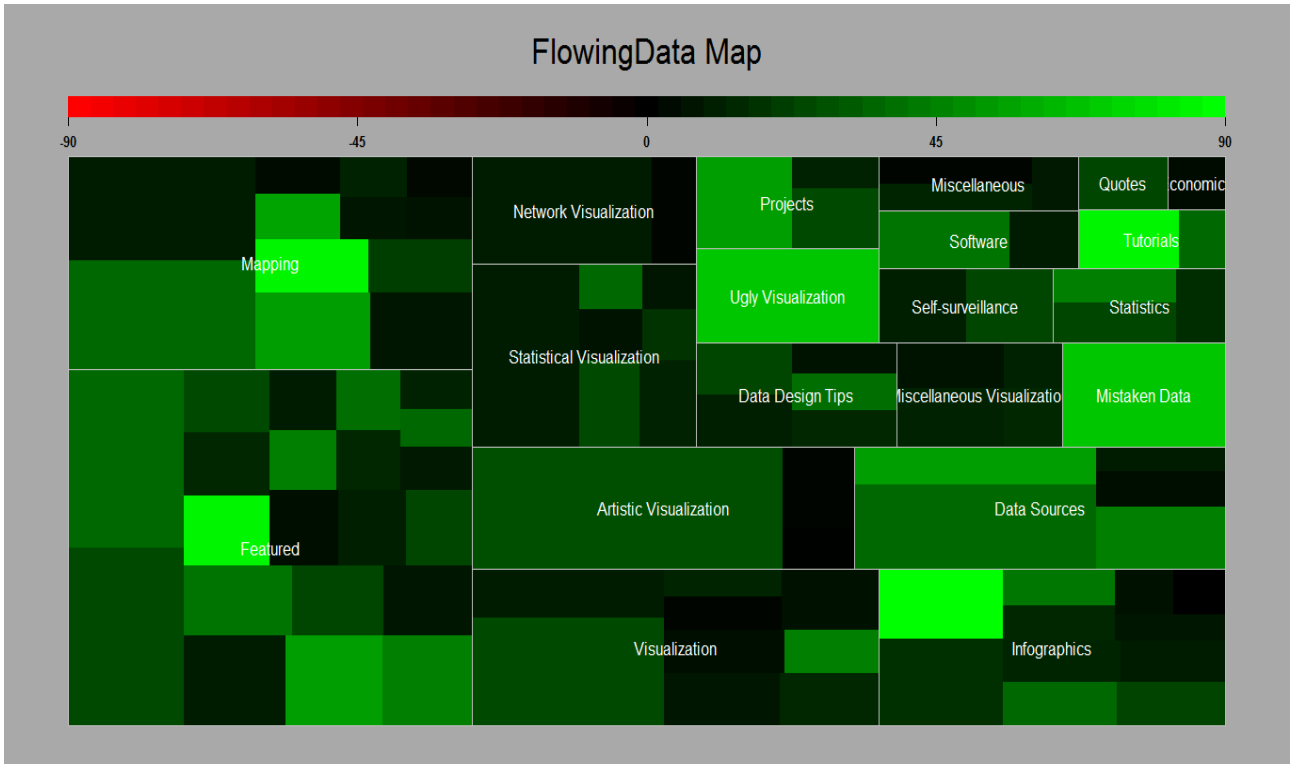


Grafico 1.7 Matrice di correlazione :

Avendo il seguente datatset di geni :

Gene1,Gene2,Gene3,Gene4,Gene5,Gene6,Gene7,Gene8,Gene9,Gene10,Gene11,Gene12,Gene13,Gene14,
Gene15,Gene16,Gene17,Gene18,Gene19,Gene20

1,0.24237,0.39215,0.015243,0.18112,0.39281,0.16023,0.25127,0.03247,0.14787,0.01422,0.51053,0.02811
9,0.28345,0.10544,0.10342,0.49834,0.10497,0.32569,0.0017114

0.24237,1,0.1542,0.05582,0.066869,0.035527,0.27955,0.38314,0.016948,0.2265,0.1551,0.30355,0.036471,
0.12503,0.48079,0.10244,0.35106,0.50826,0.19603,0.58481

0.39215,0.1542,1,0.022688,0.076589,0.32241,0.025886,0.1805,0.10122,0.12738,0.51726,0.04382,0.33166,
0.47378,0.024432,0.16155,0.35307,0.071801,0.42308,0.19031

0.015243,0.05582,0.022688,1,0.16963,0.026578,0.0069915,0.29756,0.2823,0.37105,0.22205,0.0054919,0.
078182,0.011737,0.25502,0.043142,0.30392,0.18233,0.17097,0.20298

0.18112,0.066869,0.076589,0.16963,1,0.22767,0.16113,0.15246,0.2492,0.15011,0.18528,0.080533,0.2140
7,0.15107,0.1438,0.13135,0.033155,0.32642,0.051211,0.04879

0.39281,0.035527,0.32241,0.026578,0.22767,1,0.042642,0.53499,0.0044696,0.33426,0.035081,0.052119,0
.14607,0.58051,0.25088,0.057096,0.57839,0.058623,0.54404,0.028476

0.16023,0.27955,0.025886,0.0069915,0.16113,0.042642,1,0.36557,0.7272,0.34073,0.065367,0.29044,0.37
157,0.37219,0.68096,0.11805,0.017893,0.057846,0.1871,0.68869

0.25127,0.38314,0.1805,0.29756,0.15246,0.53499,0.36557,1,0.26761,0.14574,0.20591,0.20873,0.20266,0.
36728,0.64924,0.20132,0.2397,0.039921,0.3777,0.55309

0.03247,0.016948,0.10122,0.2823,0.2492,0.0044696,0.7272,0.26761,1,0.30417,0.13571,0.025796,0.22511,
0.27504,0.55997,0.12808,0.27374,0.21675,0.030912,0.49057

0.14787,0.2265,0.12738,0.37105,0.15011,0.33426,0.34073,0.14574,0.30417,1,0.14016,0.13779,0.099958,0
.070591,0.33813,0.11654,0.20092,0.24108,0.25977,0.34695

0.01422,0.1551,0.51726,0.22205,0.18528,0.035081,0.065367,0.20591,0.13571,0.14016,1,0.088146,0.3502
5,0.19515,0.19167,0.36018,0.11883,0.15694,0.27238,0.21226

……

Posso caricare il tutto in un ogetto R:

```
genes<-read.csv("genes.csv",header=T)
```

per poi visualizzare la matrice di correlazione tra i geni con il seguente codice :

```
rownames(genes)<-colnames(genes)
image(x=1:ncol(genes),
y=1:nrow(genes),
z=t(as.matrix(genes)),
axes=FALSE,
xlab="",
```

```
ylab="" ,
main="Gene Correlation Matrix")
axis(1,at=1:ncol(genes),labels=colnames(genes),col="white",
las=2,cex.axis=0.8)
axis(2,at=1:nrow(genes),labels=rownames(genes),col="white",
las=1,cex.axis=0.8)
```

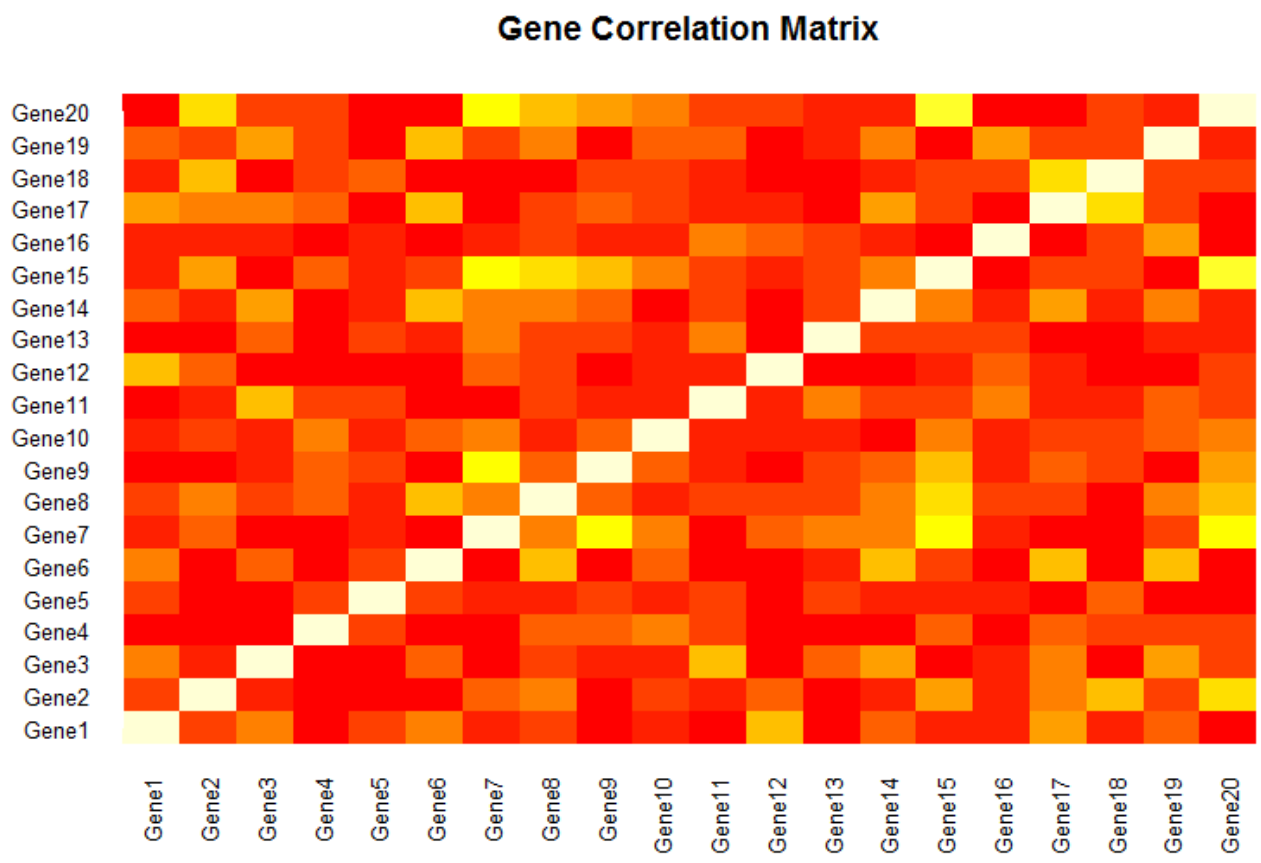


Grafico 1.8 Diagramma a linee multiple:

Partendo dal seguente dataset che mostra i mm di pioggia nelle varie città;

Month,Tokyo,NewYork,London,Berlin

Jan,49.9,83.6,48.9,42.4

Feb,71.5,78.8,38.8,33.2

Mar,106.4,98.5,39.3,34.5

Apr,129.2,93.4,42.4,39.7

May,144,106,47,52.6

……..

Posso mostrare l'andamento nelle varie città con un diagramma a linee:

Carico i dati

```
rain<-read.csv("cityrain.csv",header=TRUE)
```

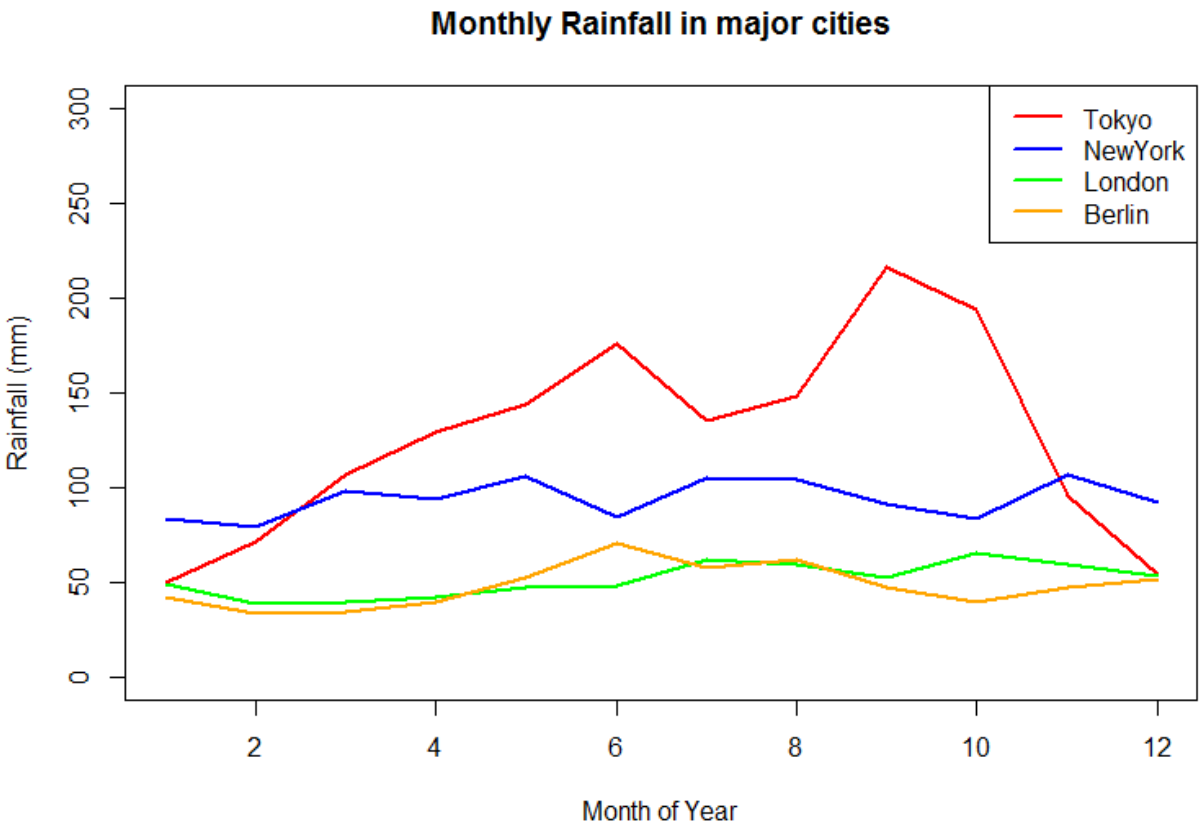
Con questo codice creo il grafico

```
    plot(rain$Tokyo,type="l",col="red",
    ylim=c(0,300),
    main="Monthly Rainfall in major cities",
    xlab="Month of Year",
    ylab="Rainfall (mm)",
    lwd=2)
    lines(rain$NewYork,type="l",col="blue",lwd=2)
    lines(rain$London,type="l",col="green",lwd=2)
lines(rain$Berlin,type="l",col="orange",lwd=2)
```

E poi aggiungo la legenda:

```
    legend("topright",
    legend=c("Tokyo","NewYork","London","Berlin"),
    col=c("red","blue","green","orange"),
lty=1,lwd=2)
```

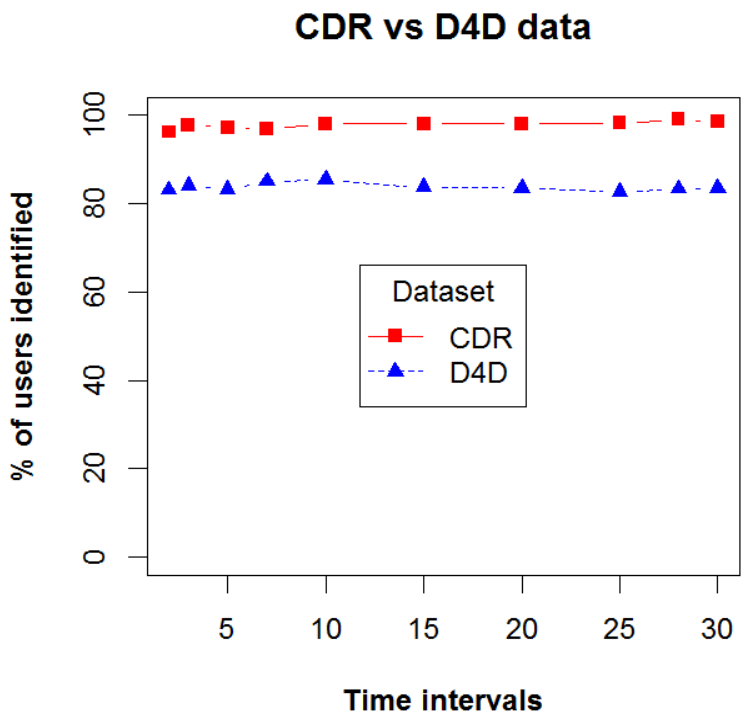
Per ottenere poi un grafico cosi :



Un altro tipo di grafico a linee è il seguente che si realizza usano righe di codice come riportato sotto :

```
tempo <- c(2,3,5,7,10,15,20,25,28,30)

 percent <- c(96.08,97.64,97.14,96.88,97.91,97.91,97.91,98.17,98.95,98.43)

 nrpoints <-c(3.38,3.72,4.33,4.92,5.58,6.33,6.91,7.3,7.38,7.42)

 percentd4d <-c(83.02,83.91,83.19,84.91,85.34,83.62,83.45,82.56,83.29,83.43)

 nrpointsd4d <-c(3.66,4.24,5.21,5.93,6.72,7.58,8.07,8.3,8.18,8.15)

 opar <- par(no.readonly = TRUE)

 par(lw = 2, cex = 1.5, font.lab = 2)

plot(tempo,percent, type = "b", pch = 15, lty = 1, col = "red", ylim = c(0,100), main = "CDR vs D4D data",
xlab = "Time intervals", ylab = "% of users identified" )

 lines(tempo, percentd4d, type = "b", pch = 17, lty = 2, col = "blue")

 legend ("center", inset = .05, title = "Dataset",c("CDR", "D4D"), lty = c(1,2), pch = c(15, 17), col = c("red",
"blue") )
```

Il risultato è il grafico mostrato in figura :



Altro tipo di grafico a multilinee è il seguente grafico : Partendo dal seguente dataset:

k	**precision**	**recall**
1.0	0.05	1.0

1.0	0.1	1.0
1.0	0.13	1.0
1.0	0.15	1.0
1.0	0.16	1.0
1.0	0.19	1.0
1.0	0.22	1.0
1.0	0.24	1.0
1.0	0.26	1.0
1.0	0.28	1.0
1.0	0.29	1.0
1.0	0.31	1.0
1.0	0.34	0.991
1.0	0.35	0.971
1.0	0.38	0.951
1.0	0.42	0.931
1.0	0.43	0.911
1.0	0.43	0.871
1.0	0.44	0.861
1.0	0.49	0.851
1.0	0.52	0.851
1.0	0.52	0.791
1.0	0.52	0.771
1.0	0.53	0.751
1.0	0.55	0.751
1.0	0.6	0.731
1.0	0.61	0.691
2	0.03	0.951
2	0.05	0.941
2	0.08	0.921

2	0.11	0.901
2	0.13	0.871
2	0.16	0.851
2	0.18	0.801
2	0.21	0.761
2	0.24	0.721
2	0.29	0.701
2	0.33	0.651
2	0.4	0.631
2	0.48	0.631
2	0.57	0.591
2	0.63	0.551
2	0.67	0.531
2	0.71	0.521
2	0.78	0.501
2	0.91	0.481
2	0.91	0.481
3	0.06	0.951
3	0.09	0.951
3	0.11	0.951
3	0.13	0.941
3	0.16	0.931
3	0.19	0.921
3	0.21	0.911
3	0.24	0.901
3	0.28	0.891
3	0.3	0.881
3	0.34	0.871
3	0.37	0.841

3	0.4	0.811
3	0.41	0.791
3	0.43	0.761
3	0.46	0.731
3	0.46	0.691
3	0.47	0.651
3	0.47	0.621
3	0.5	0.581
3	0.55	0.581
3	0.56	0.561
3	0.59	0.541
3	0.61	0.531
3	0.62	0.521
3	0.7	0.511
3	0.72	0.491
3	0.78	0.481
3	0.78	0.421
3	0.85	0.411
3	0.85	0.411
3	0.88	0.401
3	0.88	0.371
3	0.9	0.371
3	0.95	0.371
3	0.95	0.371
3	0.95	0.371
4	0.09	0.961
4	0.13	0.961
4	0.16	0.961
4	0.18	0.961

4	0.2	0.951
4	0.22	0.951
4	0.25	0.941
4	0.27	0.941
4	0.3	0.941
4	0.33	0.941
4	0.35	0.931
4	0.38	0.931
4	0.41	0.931
4	0.42	0.911
4	0.46	0.901
4	0.51	0.901
4	0.52	0.881
4	0.52	0.861
4	0.54	0.851
4	0.58	0.831
4	0.61	0.821
4	0.61	0.791
4	0.62	0.771
4	0.68	0.751
4	0.72	0.741
4	0.77	0.741
4	0.77	0.701
4	0.78	0.671
4	0.78	0.651
4	0.81	0.631
4	0.84	0.611
4	0.85	0.611
4	0.93	0.601

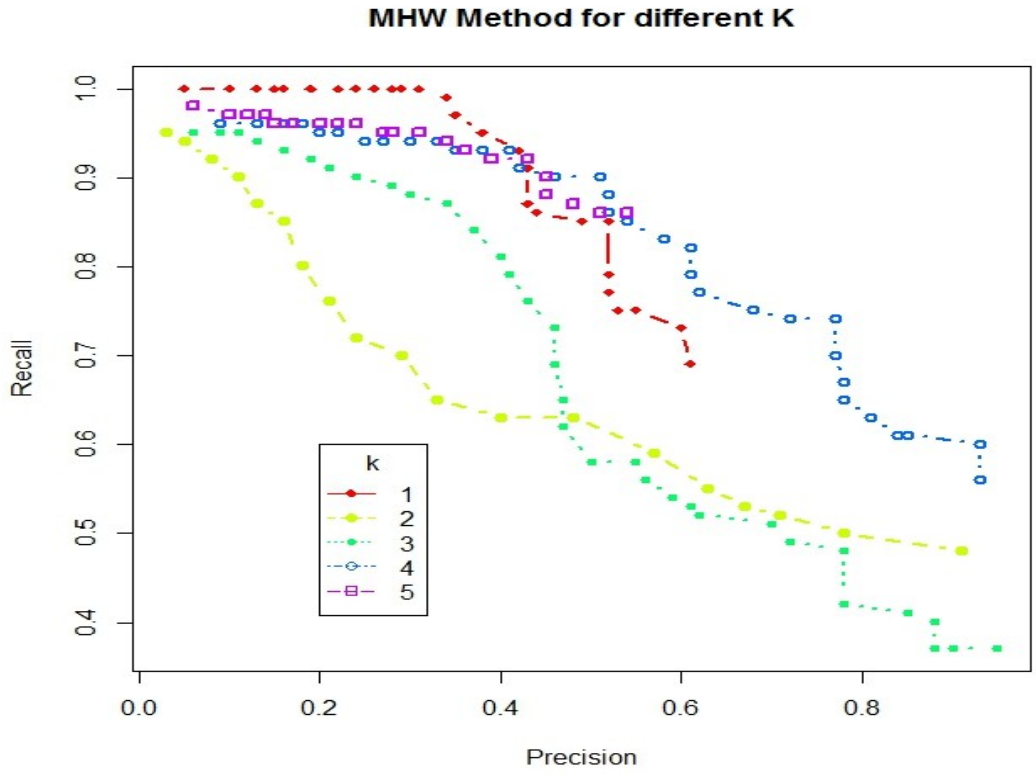
4	0.93	0.561
5.0	0.06	0.981
5.0	0.1	0.971
5.0	0.12	0.971
5.0	0.14	0.971
5.0	0.15	0.961
5.0	0.17	0.961
5.0	0.2	0.961
5.0	0.22	0.961
5.0	0.24	0.961
5.0	0.27	0.951
5.0	0.28	0.951
5.0	0.31	0.951
5.0	0.34	0.941
5.0	0.36	0.931
5.0	0.39	0.921
5.0	0.43	0.921
5.0	0.45	0.901
5.0	0.45	0.881
5.0	0.48	0.871
5.0	0.51	0.861
5.0	0.54	0.861

Posso visualizare i dati perogni tipo di k con il seguente codice .

```
dtf$k <- as.numeric(dtf$k)
nk <- max(dtf$k)
xrange <- range(dtf$precision)
yrange <- range(dtf$recall)
plot(xrange, yrange,
```

```
 type="n",
 xlab="Precision",
 ylab="Recall"
 )
colors <- rainbow(nk)
#colors <- c ( "#FF0000FF" "#FFFF00FF" "#00FF00FF" "#00FFFFFF" "#0000FFFF" "#FF00FFFF")
linetype <- c(1:nk)
plotchar <- seq(18, 18+nk, 1)
for (i in 1:nk) {
 Ki <- subset(dtf, k==i)
 lines(Ki$precision, Ki$recall,
 type="b",
 lwd=2,
 lty=linetype[i],
 col=colors[i],
 pch=plotchar[i]
 )
}
title("MHW Method for different K")
legend(0.2,0.6,
 1:nk,
 cex=1.0,
 col=colors,
inset=c(0.2,0),
 pch=plotchar,
 lty=linetype,
 title="k"
)
```

Che permette di ottenere il seguente grafico:

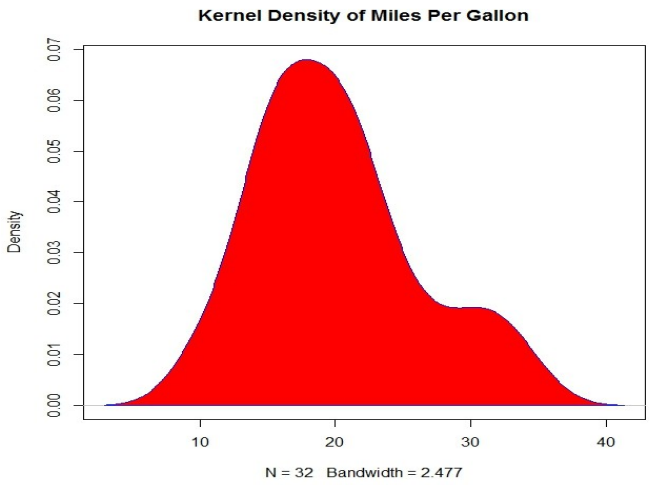


Dove perogni tipo di k ho una linea che mi descrive l'andamento della precision e recall che sono le mie due variabili (dipendente e indipendente) per quel tipo di k.

Per fare la visualizzazione dei valori di una distribuzione secondo la density kernel function (la funzione di densità di probabilita) allora posso seguire il seguenet prodedimento :

```
d <- density(mtcars$mpg)
plot(d, main="Kernel Density of Miles Per Gallon")
polygon(d, col="red", border="blue")
```

Partendo quindi dal dataset di R mtcars$mpg posso fare una rappresentazione della funzione di densità

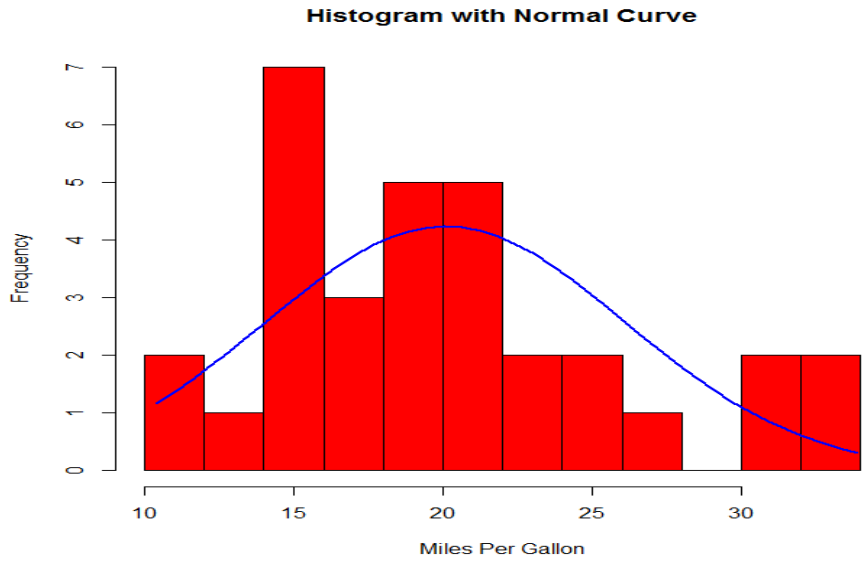


Per fare il fit alla normale di una distribuzione posso seguire il procedimento che si trova in :

http://www.statmethods.net/graphs/density.html

Con le seguenti linee di codice posso fare :

```
x <- mtcars$mpg
h<-hist(x, breaks=10, col="red", xlab="Miles Per Gallon",
   main="Histogram with Normal Curve")
xfit<-seq(min(x),max(x),length=40)
yfit<-dnorm(xfit,mean=mean(x),sd=sd(x))
yfit <- yfit*diff(h$mids[1:2])*length(x)
lines(xfit, yfit, col="blue", lwd=2)
```



Visualizzazione serie Temporali

Avendo un datatset fatto come segue :

date,ws,wd,nox,no2,o3,pm10,so2,co,pm25

01/01/1998 00:00,0.6,280,285,39,1,29,4.7225,3.3725,

01/01/1998 01:00,2.16,230,,,,37,,,

01/01/1998 02:00,2.76,190,,,3,34,6.83,9.6025,

01/01/1998 03:00,2.16,170,493,52,3,35,7.6625,10.2175,

01/01/1998 04:00,2.4,180,468,78,2,34,8.07,8.9125,

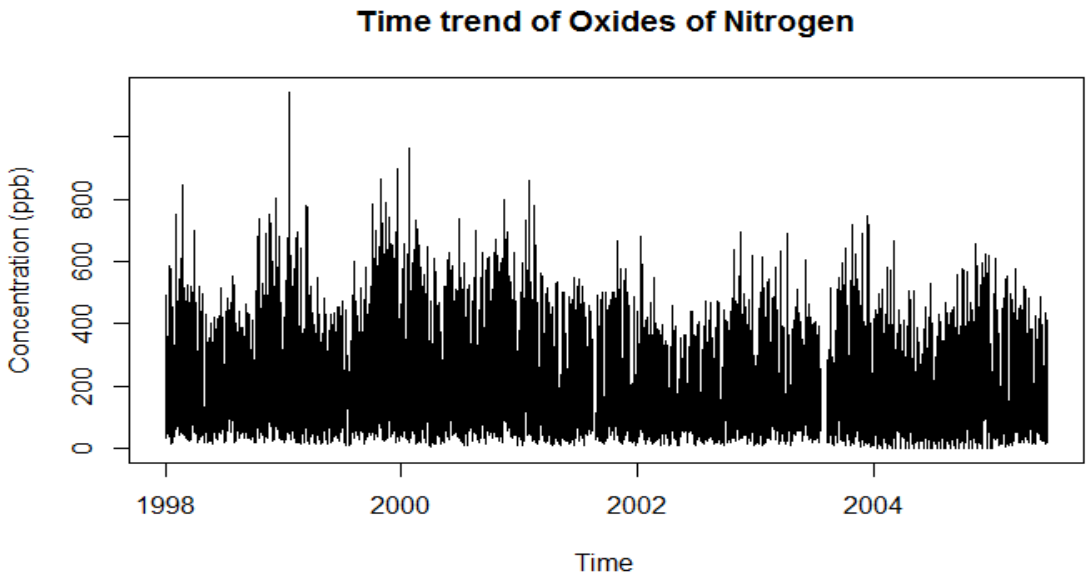
Si possono visualizzare i dati come una serie temporale con i seguenti comandi :

air<-read.csv("openair.csv")

plot(air$nox~as.Date(air$date,"%d/%m/%Y %H:%M"),type="l",

xlab="Time", ylab="Concentration (ppb)",

main="Time trend of Oxides of Nitrogen")



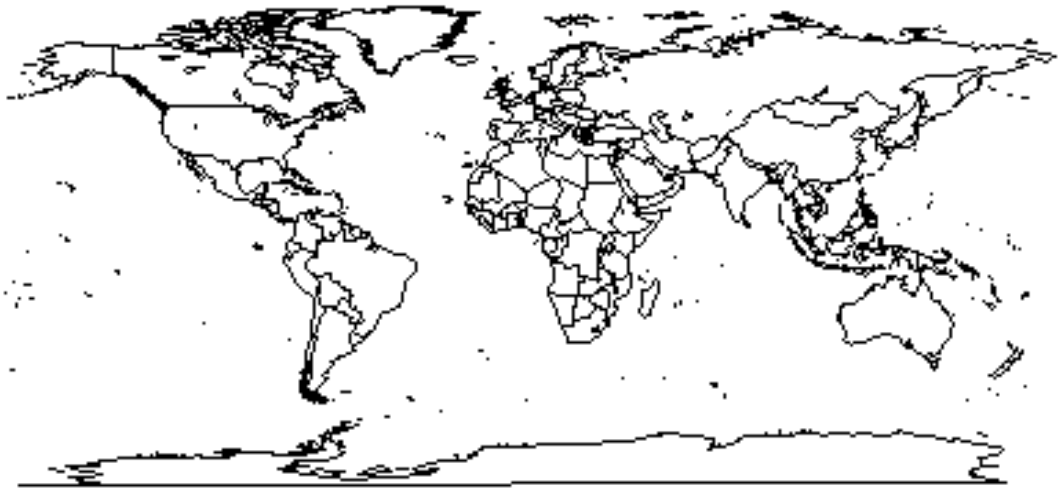
In R si possono anche visualizzare le mappe.

Bisogna prima installare il package giusto per questo :

install.packages("maps")

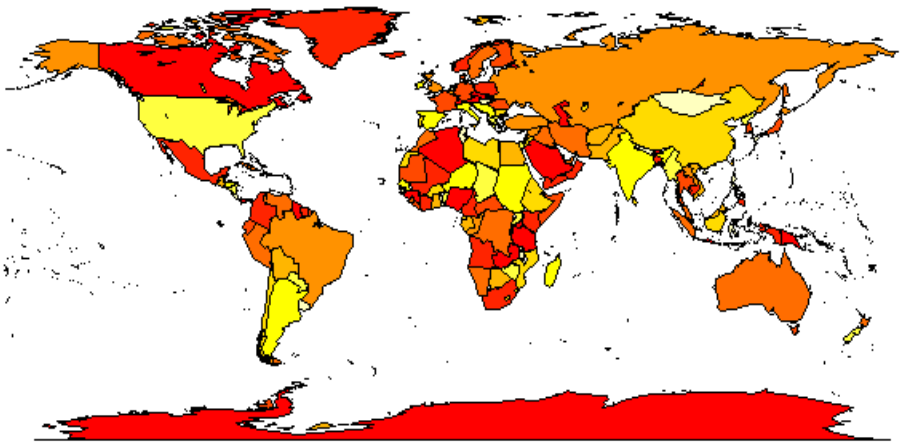
e usare il package con la riga ; *library (maps)*

La funzione map() visualizza la seguente fig:



Si possono aggiungere i colori con la seguente riga :

map('world', fill = TRUE,col=heat.colors(10))



Per visualizzare la mappa di un singolo paese invece bisogna importare il contorno di quel paese dopo aver installato la seguente libreria :

install.packages("sp")

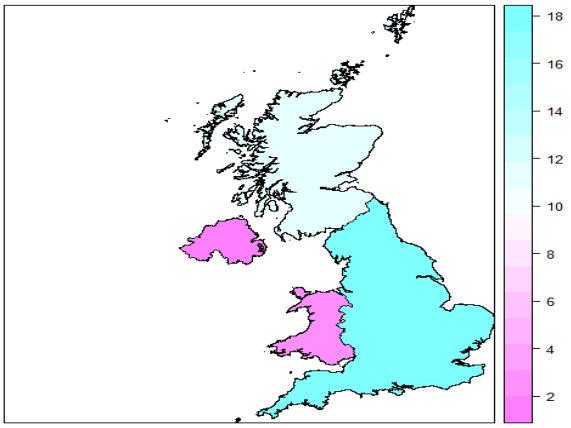
library(sp)

e utilizzato la libreria stessa si possono chiamare le seguenti funzioni che in questo caso visualizzano i dati delle regioni amministrative del regno unito :

load(url("http://gadm.org/data/rda/GBR_adm1.RData"))

spplot(gadm,"Shape_Area")

la seguente figura mostra come :



Si possono anche visualizzare le regioni amministrative di un paese ad esempio Italia :

library("maps")

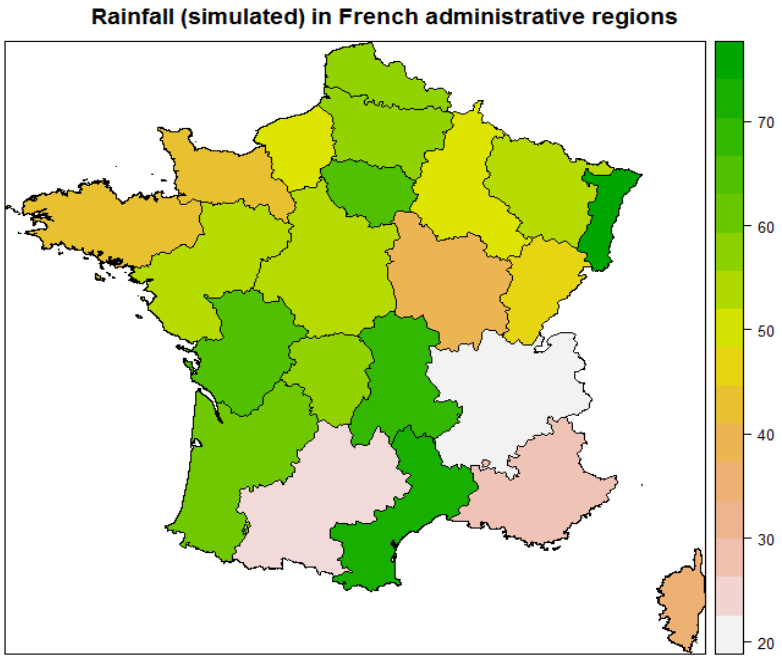
library(RColorBrewer)

map('italy', fill = TRUE, col = brewer.pal(7,"Set1"))

Il risultato è dato dalla figura seguente:

Si possono simulare le precipitazioni nelle regioni francesi (con il codice riportato sotto) ad esempio come nella figura seguente :

```
library(sp)

> load(url("http://gadm.org/data/rda/FRA_adm1.RData"))

> gadm$rainfall<-rnorm(length(gadm$NAME_1),mean=50,sd=15)

> spplot(gadm,"rainfall",

+ col.regions = rev(terrain.colors(gadm$rainfall)),

+ main="Rainfall (simulated) in French administrative regions")
```

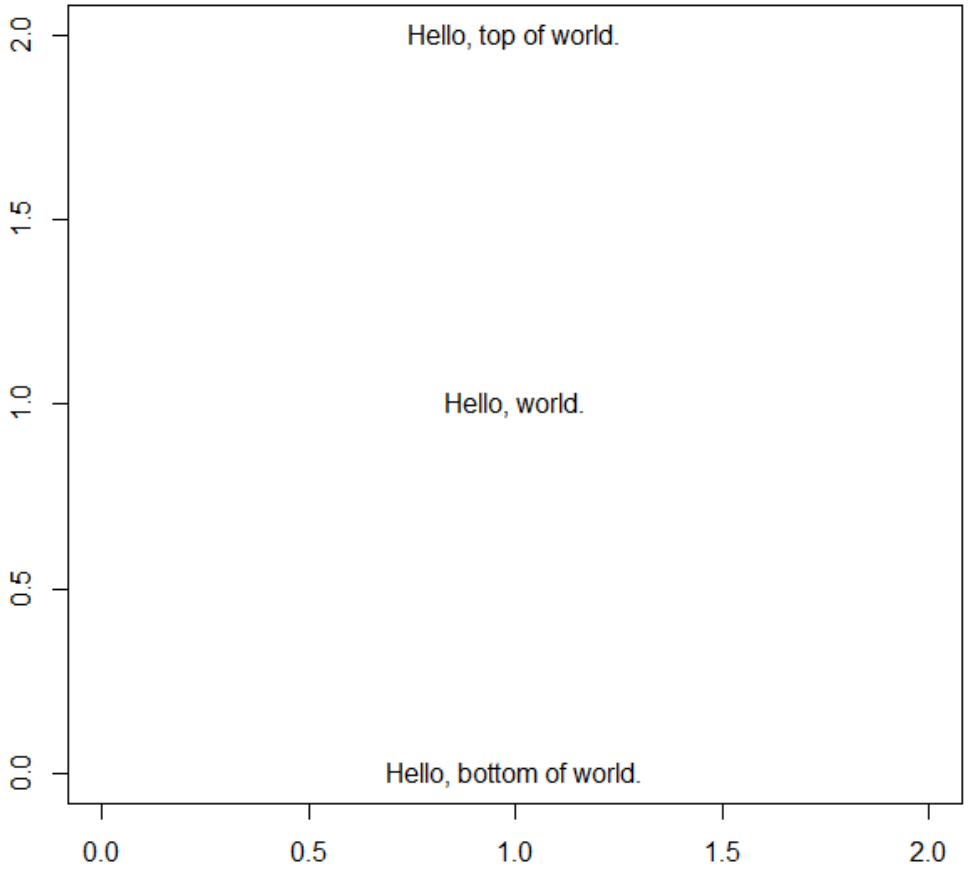


Visualizzare il testo in R

Con R è possibile visualizzare del testo. Il testo può essere posizionato in una determinata area del grafico con le righe di codice seguenti :

```
plot(0, 0, type="n", xlim=c(0, 2), ylim=c(0, 2), xlab="", ylab="")

text(1, 1, 'Hello, world.')

plot(0, 0, type="n", xlim=c(0, 2), ylim=c(0, 2), xlab="", ylab="")

text(1, 1, 'Hello, world.')# Middle

text(1, 2, 'Hello, top of world.') # Top

text(1, 0, 'Hello, bottom of world.')# Bottom
```

La figura seguente mostra I risultati



The R library ggvis is a great library for visualizing data even as a web page. This library (ggvis) is analogus to shiny and can be used in a more simple way.

Some examples bellow show how it is possible to visualize data with very little code and export the result as a web page. The graphics is also interactive as it can expand by simple dragging.

Make a scatterplot in ggvis by adding a third dimension(coloring points by a third variable).

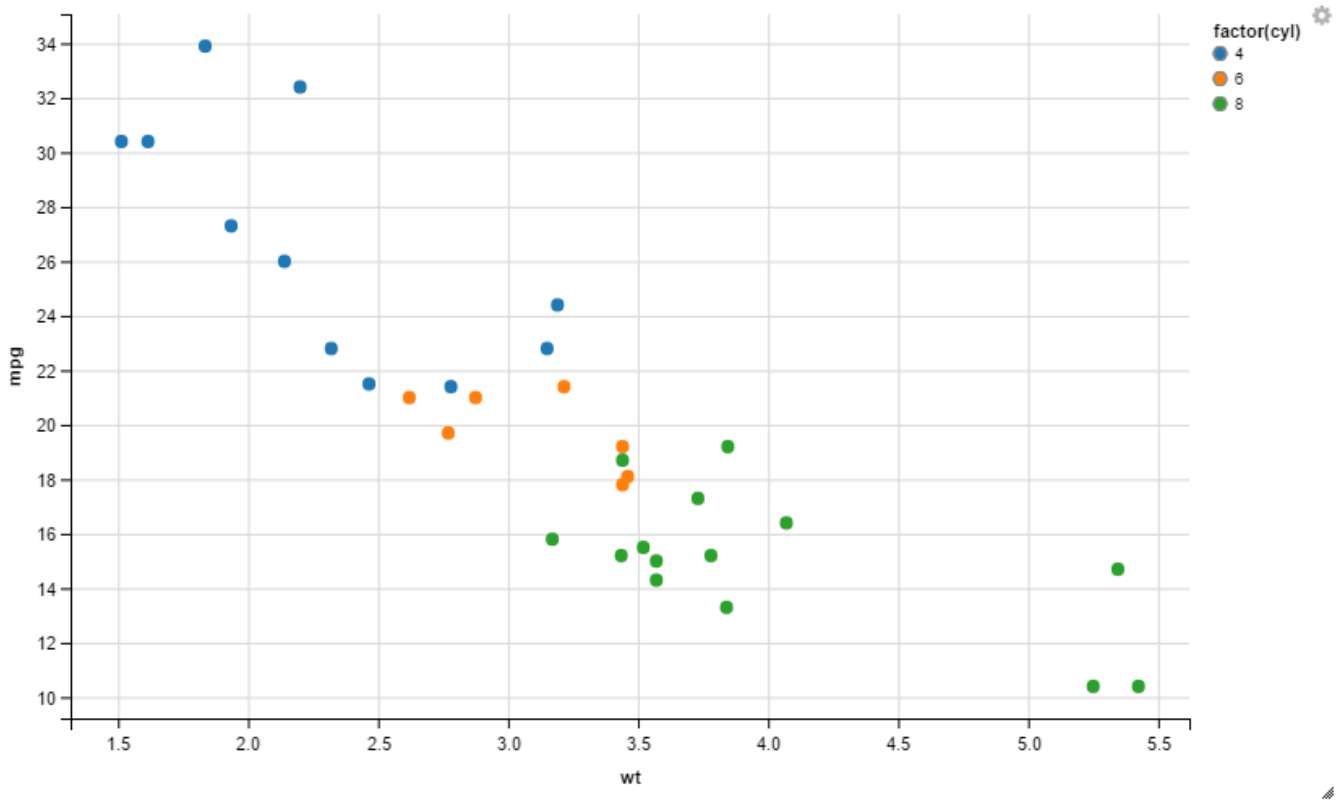
By using the mtcars dataset:

```
                   mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Merc 230          22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Merc 280C         17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
Merc 450SE        16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
Merc 450SL        17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
Merc 450SLC       15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
Cadillac Fleetwood 10.4  8 472.0 205 2.93 5.250 17.98  0  0    3    4
```

```
Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

By using the following code we have :

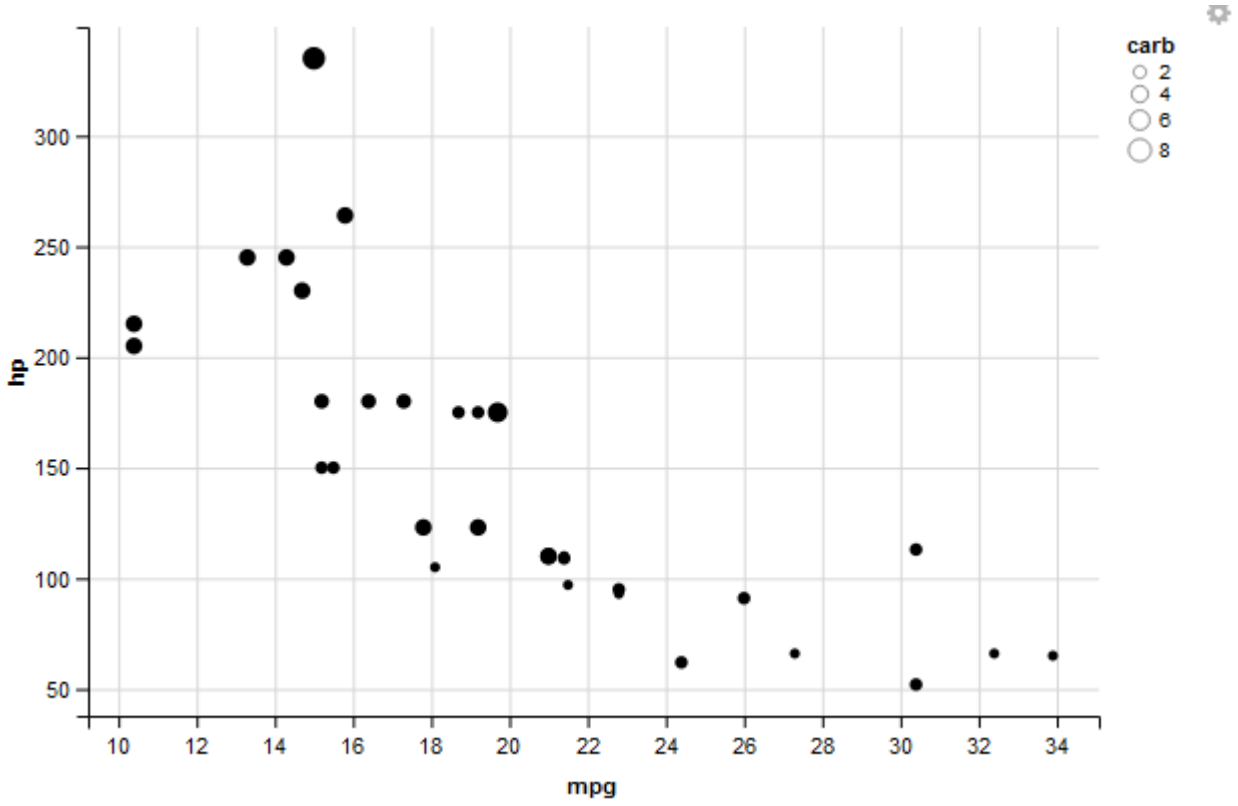
```
mtcars %>%

  ggvis(~wt, ~mpg) %>%

  layer_points(fill = ~factor(cyl))
```



In a very similar way it is possible to make also the following visualization that relates cars by horse power and mpg. The size of each dot represents the number or carburatores:

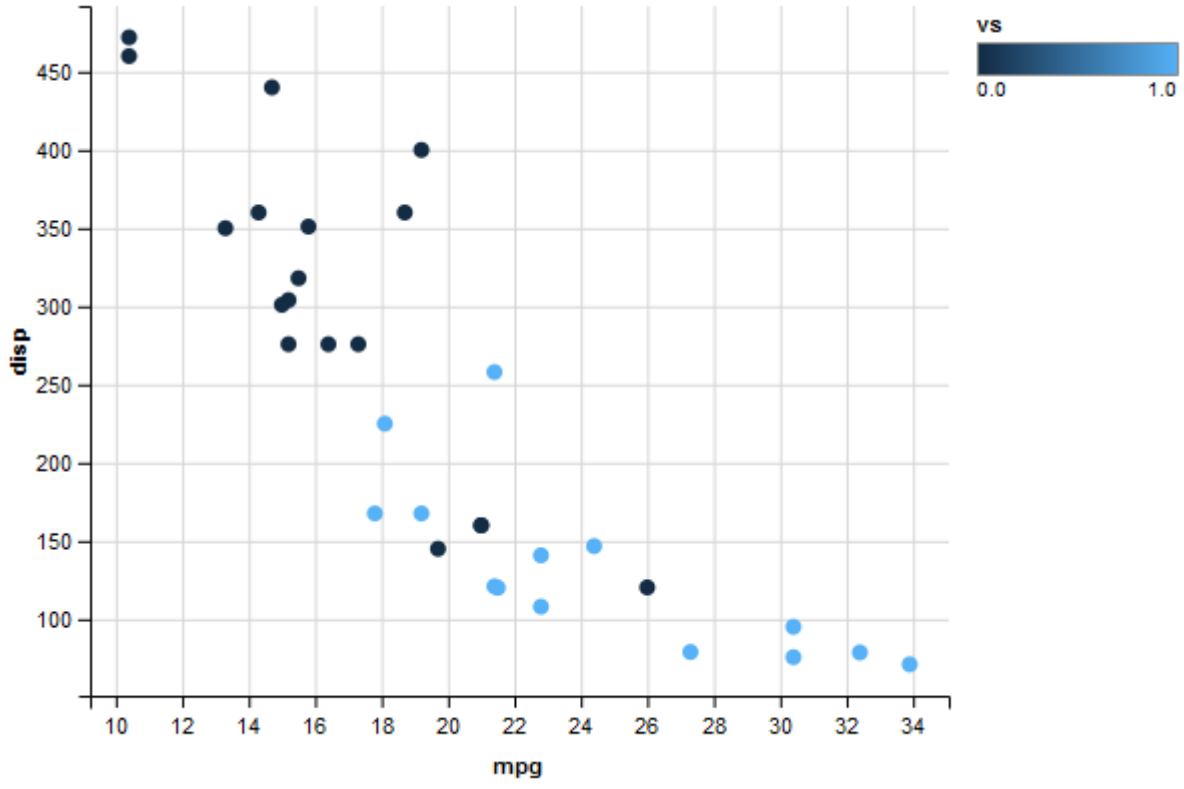
```
mtcars %>% ggvis(~mpg, ~hp, size = ~carb) %>% layer_points()
```

The following picture represents how it looks:



In the same way by using a color in a continuum :

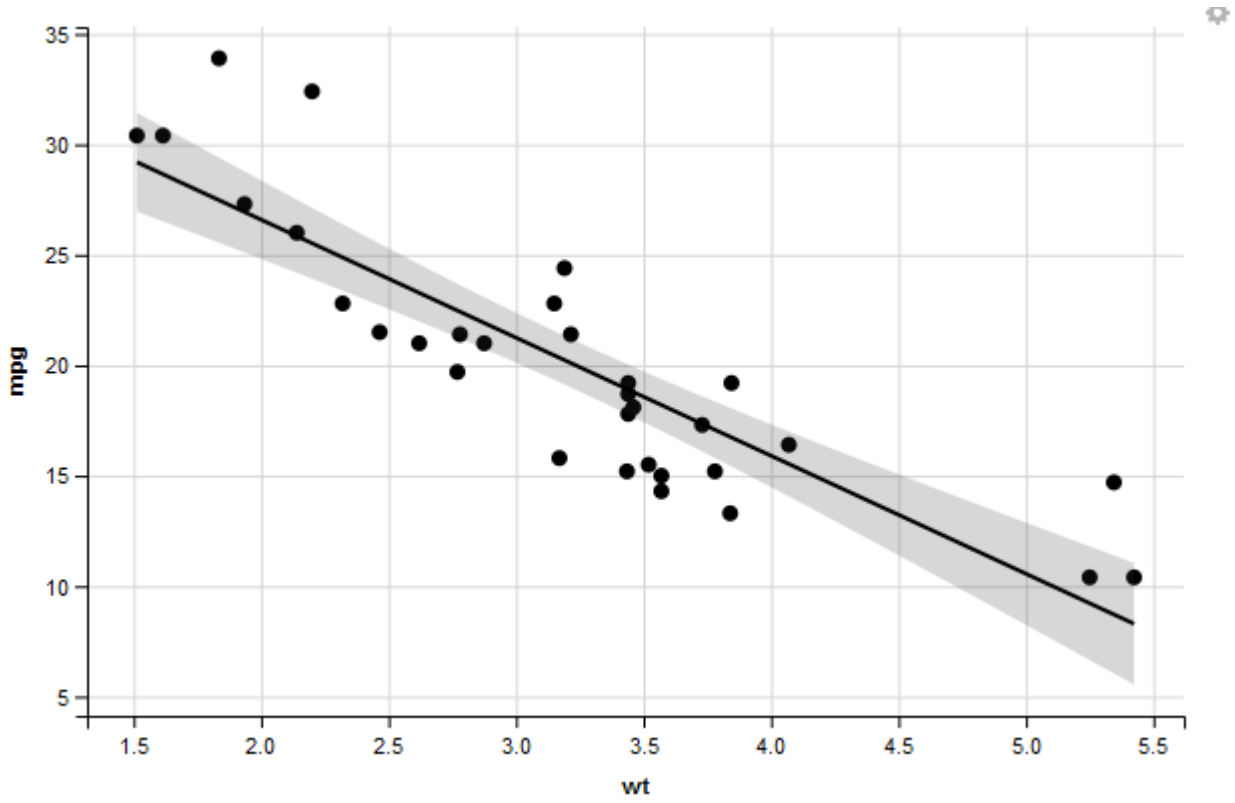
```
mtcars %>% ggvis(~mpg, ~disp, fill = ~vs) %>% layer_points()
```



For making a representation of a linear regression model and an interval of confidence for that model then it is possible to procede in the following way:

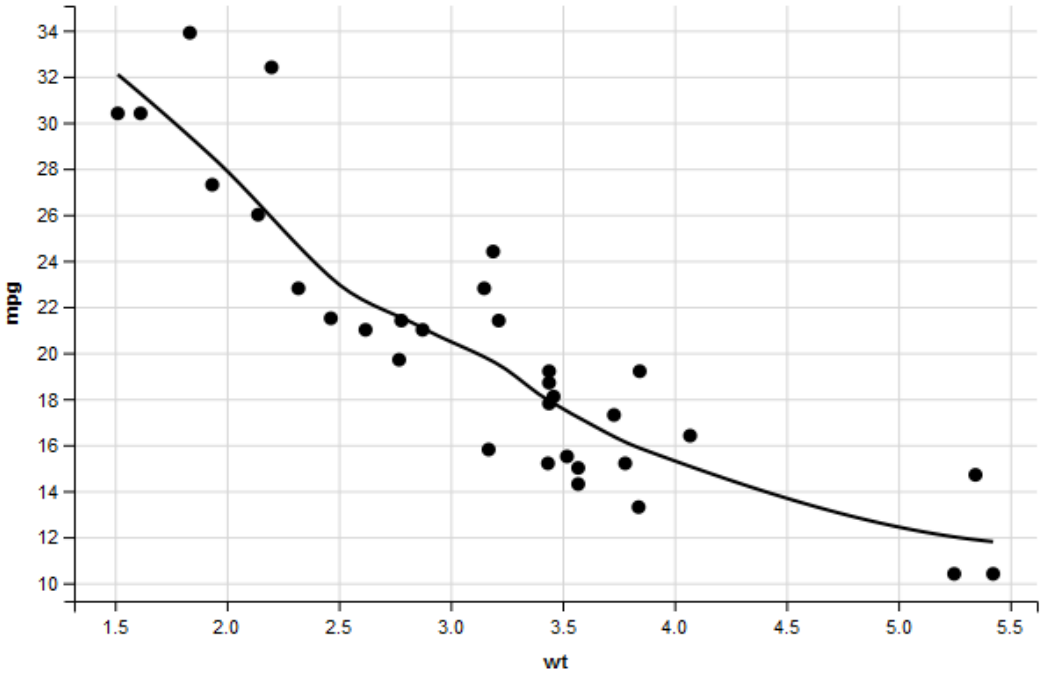
With a linear model, and 95% confidence interval for the model:

```
mtcars %>%
  ggvis(~wt, ~mpg) %>%
  layer_points() %>%
  layer_model_predictions(model = "lm", se = TRUE)
```



It is also possible to add a smooth line that represents the way scatterplot points are distributed :

```
mtcars %>%
  ggvis(~wt, ~mpg) %>%
  layer_points() %>%
  layer_smooths()
```



How to make an age pyramid :

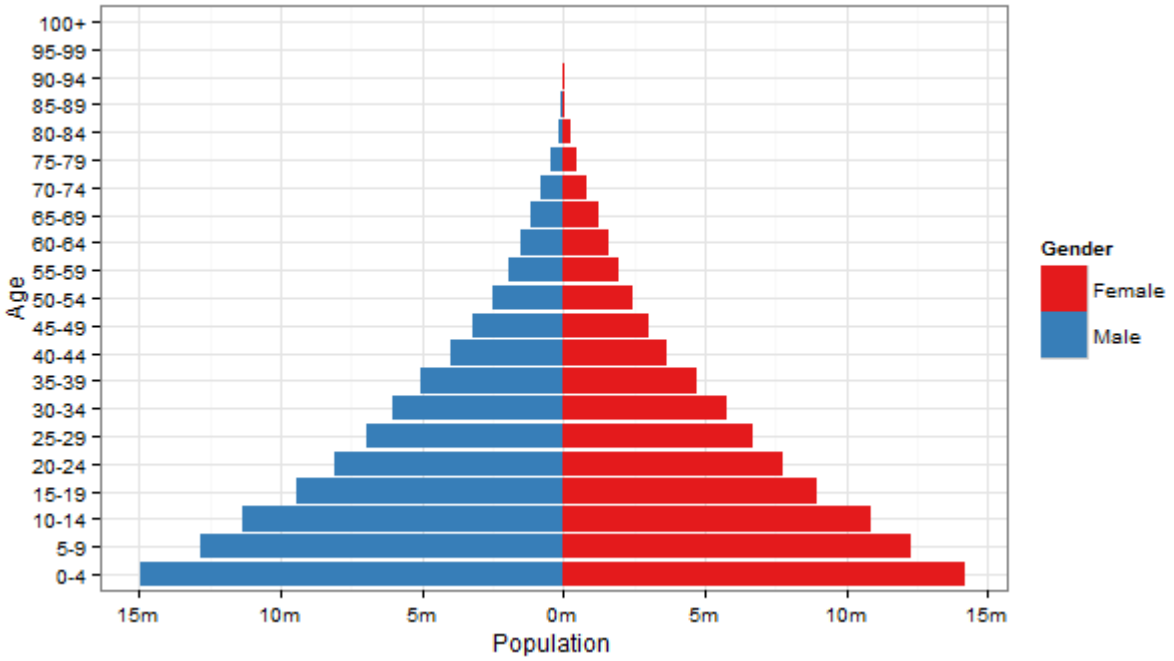
```
library(XML)

library(reshape2)

library(ggplot2)

library(plyr)

get_data <- function(country, year) {

  c1 <- "http://www.census.gov/population/international/data/idb/region.php?N=%20Results%
20&T=10&A=separate&RT=0&Y="

  c2 <- "&R=-1&C="

  url <- paste0(c1, year, c2, country)

  df <- data.frame(readHTMLTable(url))

  keep <- c(2, 4, 5)

  df <- df[,keep]

  names(df) <- c("Age", "Male", "Female")

  cols <- 2:3

  df[,cols] <- apply(df[,cols], 2, function(x) as.numeric(as.character(gsub(",", "", x)))
)

  df <- df[df$Age != 'Total', ]

  df$Male <- -1 * df$Male

  df$Age <- factor(df$Age, levels = df$Age, labels = df$Age)
```

```
  df.melt <- melt(df,
                  value.name='Population',
                  variable.name = 'Gender',
                  id.vars='Age' )
  return(df.melt)
}
```

Now, I can call the `get_data` function and create a population pyramid. My code was basically adapted from this StackOverflow post so the credit is due to Didzis Elferts for his excellent answer. Here's a pyramid for a fast-growing country, Nigeria, in 2014:

```
nigeria <- get_data("NI", 2014)


n1 <- ggplot(nigeria, aes(x = Age, y = Population, fill = Gender)) +
  geom_bar(subset = .(Gender == "Female"), stat = "identity") +
  geom_bar(subset = .(Gender == "Male"), stat = "identity") +
  scale_y_continuous(breaks = seq(-15000000, 15000000, 5000000),
                     labels = paste0(as.character(c(seq(15, 0, -5), seq(5, 15, 5))), "m")
) +
  coord_flip() +
  scale_fill_brewer(palette = "Set1") +
  theme_bw()


n1
```

That will visualize a graphic like this :



Social Network Analysis in R.

Ci sono vari package in R che possono aiutare con questo compito. Uno di questi è igraph.

Per mostrare passo passo come caricare i dati della matrice di adiacenza in R e visualizzare il grafo si segue il procedimento :

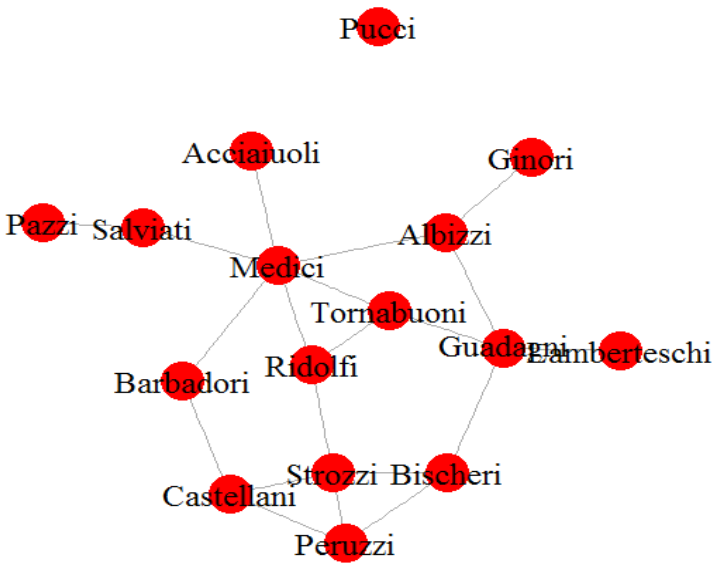
1- Carico i dati come una matrice
florence <- as.matrix(read.csv("firenze.csv"))
che mi da come risultato l amatrice sottostante :

	Acciaiuoli	Albizzi	Barbadori	Bischeri	Castellani	Ginori	Guadagni	Lamberteschi	Medici
Acciaiuoli	0	0	0	0	0	0	0	1	0	0	0	0	0
Albizzi	0	0	0	0	0	1	1	0	1	0	0	0	0	0
Barbadori	0	0	0	0	1	0	0	0	1	0	0	0	0	0
Bischeri	0	0	0	0	0	0	1	0	0	0	1	0	0	0
Castellani	0	0	1	0	0	0	0	0	0	0	1	0	0	0

2- Creo l'oggetto mariage che virtualmente rappresenta i legami di nozze tra le famiglie fiorentine del epoca .
marriage <- graph.adjacency(florence,mode="undirected", diag=FALSE)

3- POsso quindi visualizzare il grafo a partire da questo :
set.seed(1)
plot(marriage,layout=layout.fruchterman.reingold,vertex.label=V(marriage)$name,vertex.color="red",vertex.label.color="black",vertex.frame.color=0,vertex.label.cex=1.5)

come nel grafico sottostante :



Alcune delle misure importanti in un social network sono anche il "node degree" è cioè il grado di connettività di un certo nodo e la sua "betweeness", un'altra misura importante.

Per visualizzare il grado di connettività posso trattare i dati come un dataframe in cui perogni famiglia nodo si mostra anche il grado di connettività : in questo caso la famiglia Medici è quella più importante.

data.frame(V(marriage)$name,degree(marriage))

	V.marriage..name	degree.marriage.
Acciaiuoli	Acciaiuoli	1
Albizzi	Albizzi	3
Barbadori	Barbadori	2
Bischeri	Bischeri	3
Castellani	Castellani	3
Ginori	Ginori	1
Guadagni	Guadagni	4

Lamberteschi	Lamberteschi	1
Medici	Medici	6
Pazzi	Pazzi	1
Peruzzi	Peruzzi	3
Pucci	Pucci	0
Ridolfi	Ridolfi	3
Salviati	Salviati	2
Strozzi	Strozzi	4
Tornabuoni	Tornabuoni	3

La “betweeness” che è una misura di centralità di un nodo si calcola nel modo seguente :

Implementazione di algoritmi e metodi statistici in R.

In questa parte sono implementati alcune procedure e algoritmi (anche con relativa visualizzazione dei risultati) di analisi dati. Il background teorico di tali metodi non viene trattato per andare dritto al punto con l’implementazione degli algoritmi stessi in R.

Clustering : k-means.

Partendo dal seguente dataset dove per ogni paese europeo viene mostrato da quale cibo arriva la quantità di proteine che assume l’abitante medio di quel paese :

Country, RedMeat, WhiteMeat, Eggs, Milk, Fish, Cereals, Starch, Nuts, Fr&Veg

Albania, 10.1, 1.4, 0.5, 8.9, 0.2, 42.3, 0.6, 5.5, 1.7

Austria, 8.9, 14.0, 4.3, 19.9, 2.1, 28.0, 3.6, 1.3, 4.3

Belgium, 13.5, 9.3, 4.1, 17.5, 4.5, 26.6, 5.7, 2.1, 4.0

Bulgaria, 7.8, 6.0, 1.6, 8.3, 1.2, 56.7, 1.1, 3.7, 4.2

Czechoslovakia, 9.7, 11.4, 2.8, 12.5, 2.0, 34.3, 5.0, 1.1, 4.0

Denmark, 10.6, 10.8, 3.7, 25.0, 9.9, 21.9, 4.8, 0.7, 2.4

In base alle colonne quindi alla tipologia di cibo consumato (e quindi fonte di proteine) si può effettuare uno clustering dei paesi basandosi appunto sulla dieta dei suoi abitanti.

1- Carico i dati
   food <- read.csv("data/protein.csv")

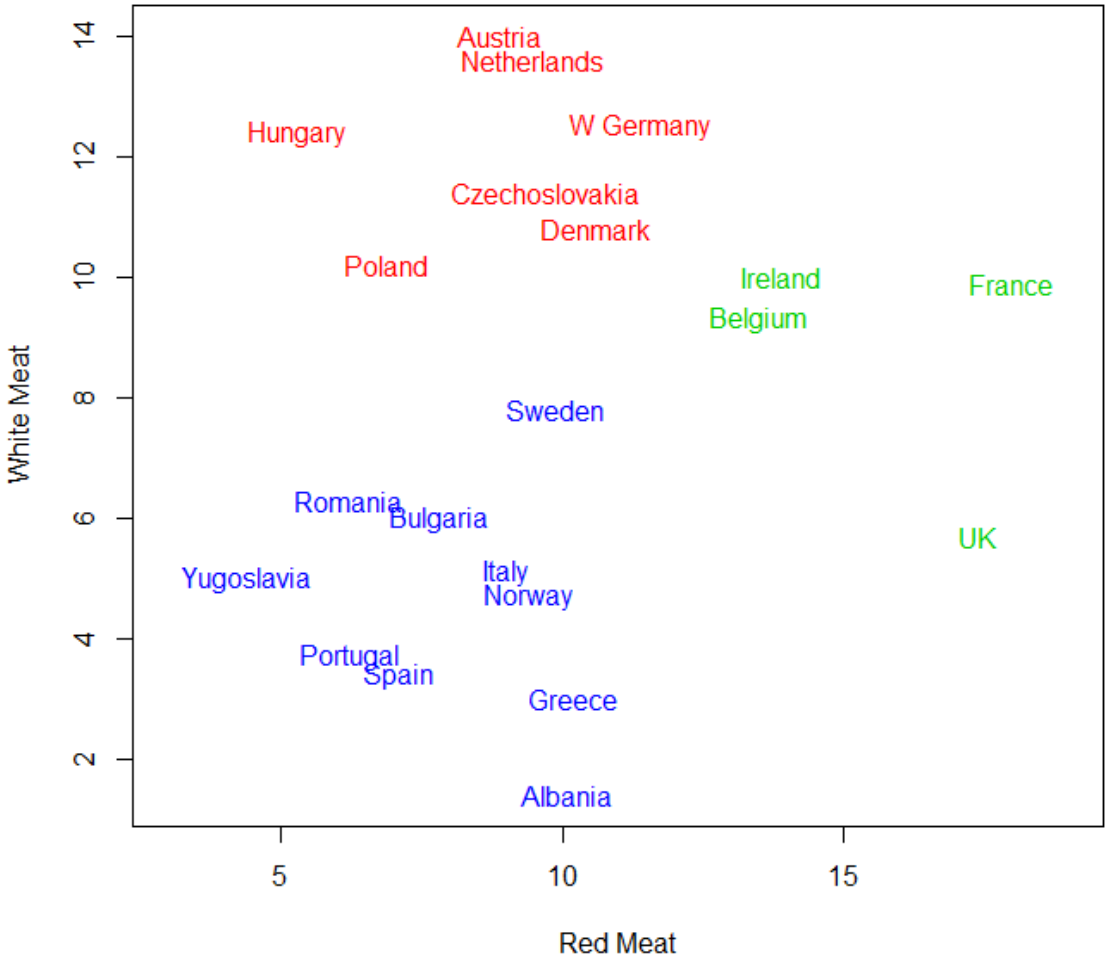
2- Per fissare la funzione random degli cluster uso
   set.seed(1)

3- Assegno a un oggetto grpMeat il risultato della funzione cluster dove vado a clusterizzare i paese solo rispetto alle due prime colonne.
   grpMeat <- kmeans(food[,c("WhiteMeat", "RedMeat")], centers = 3, nstart = 10)
4- Ordino i paesi in base al cluster a cui appartengono e cioè faccio il modo che vengano ordinate in maniera decrescente.
   o <- order(grpMeat$cluster)
5- In base a questo costruisco un data.frame.
   data.frame(food$Country[o], grpMeat$cluster[o])
   il quale ha il seguente aspetto :

	food.Country.o.	grpMeat.cluster.o.
1	Austria	1
2	Czechoslovakia	1
3	Denmark	1
4	Hungary	1
5	Netherlands	1
6	Poland	1
7	W Germany	1
8	Belgium	2
9	France	2

6- Faccio il plotting dei paesi con la seguente riga :
   plot(food$Red, food$White, type = "n", xlim = c(3,19), xlab = "Red Meat", ylab = "White Meat")
7- Visualizzo il tutto andando a visualizzare ciascun paese come una stringa di testo
   text (x = food$Red, y = food$White, labels = food$Country, col = grpMeat$cluster+1)

   Il risultato è dato dalla fig dove i cluster si possono distinguere oltre che dalla distanza anche dai colori dei paesi facenti parte ai cluster stessi:



La regressione lineare in R.

In questo esempio vengono messe in confronto(per come sono correlate) delle variabili per poi visualizzare la loro retta di regressione.

Ad esempio si cerca di capire come sono correlate il gdp per capita di un certo paese e il tasso di software "piratato" installo nei computer di quel paese. In questo modo dato il gdp per capita di un paese posso prevedere il suo tasso di sw pirata presente nel dato paese.

Il mio dataset è composto da due file gdp.csv e piracy.csv

gdp.csv

Rank,Country,GDP

1,Liechtenstein,141100

2,Australia,121199

3,Luxembourg,101000

4,Qatar,100300

5,Brunei,100000

5,China,77000

6,Bermuda,61900

7,Bangladesh,40000

piracy.csv

Country,Piracy

Australia,23

Bangladesh,90

Brunei,67

China,77

Liechtenstein,10

Qatar,12

Luxembourg,9

Bermuda,90

I passi da eseguire sono elencati come segue

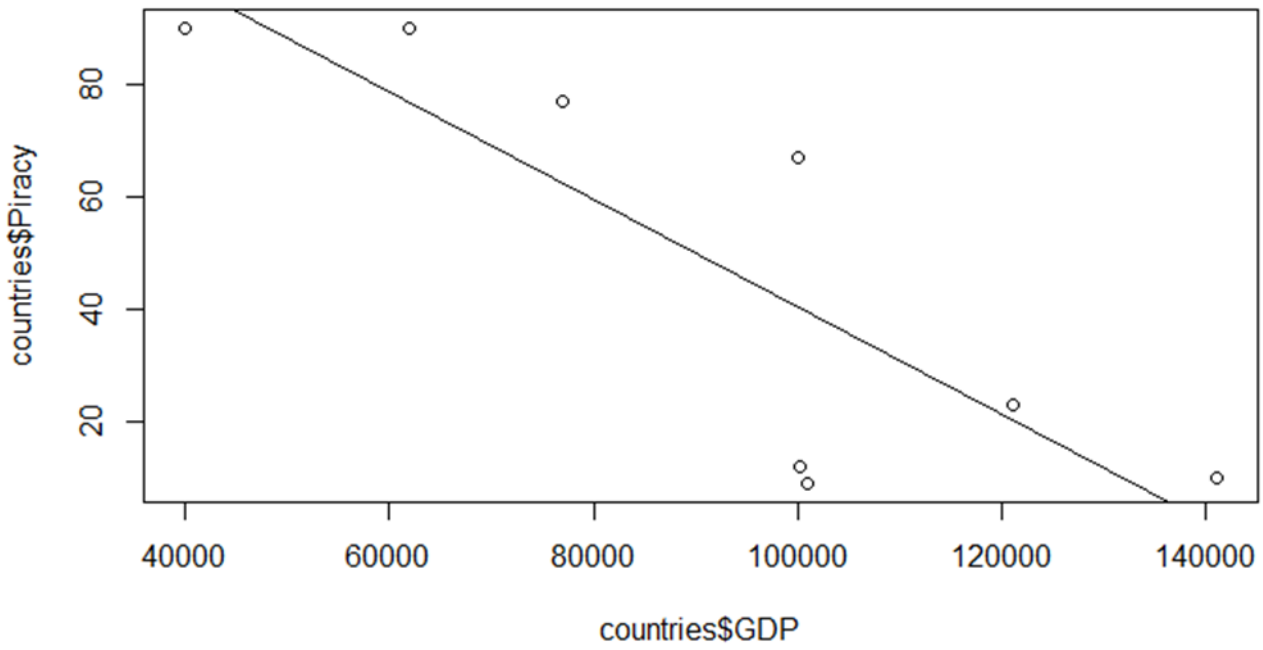
1-Carico i due dataset nelle variabili piaracy e gdp

```
piracy<- read.csv("C:\\Users\\Alket\\Documents\\piracy.csv")
```

```
gdp <- read.table("gdp.csv", sep=",", header=TRUE)
```

```
2- Unisco  le due dataset in uno solo con una operazione di merge().
countries <- merge(x = gdp, y = piracy)
```

```
3- Faccio il plot delle due variabili con la funzione plot di R
plot(countries$GDP, countries$Piracy)


4- trovo  la correlazione tra le due variabili gdp e piracy.
cor.test(countries$GDP, countries$Piracy)


5-traccio la linea che mi da la retta di regressione con la funzione lm() "line
ar model".
line <- lm(countries$Piracy ~ countries$GDP)

6- la visualizzo
> abline(line)
```



Cosi dato un paese che ha un gdp per capita di 80000 posso dire che il tasso di sw piratato in quel paese è circa il 60 %.

Dimension Reduction : principal components analysis :

Trovare la struttura in un dataset.

The purpose of principal component analysis is to find the best low-dimensional representation of the variation in a multivariate data set. For example, in the case of the wine data set, we have 13 chemical concentrations describing wine samples from three different cultivars. We can carry out a principal component analysis to investigate whether we can capture most of the variation between samples using a smaller number of new variables (principal components), where each of these new variables is a linear combination of all or some of the 13 chemical concentrations.
To carry out a principal component analysis (PCA) on a multivariate data set, the first step is often to standardise the variables under study using the "scale()" function (see above). This is necessary if the input variables have very different variances, which is true in this case as the concentrations of the 13 chemicals have very different variances (see above).

Lasciano da parte la teoria del PCA andiamo direttamente all' implementazione dell'algoritmo usando un dataset che descrive gli elementi chimici che si trovano in tre varietà di vini . La prima colonna (V1) indica la varietà di vino. Cosi ad esempio le prime tre righe contengono info sulla varietà 1 di vino mentre le ultime due righe indicano la varietà 2.Le altre 13 colonne (V2-V14) invece indicano gli elementi chimici (Na, Mg, Al…) che si trovano nelle varie varietà.

V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14
1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.640000	1.040	3.92	1065
1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.380000	1.050	3.40	1050
1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.680000	1.030	3.17	1185
2	11.79	2.13	2.78	28.5	92	2.13	2.24	0.58	1.76	3.000000	0.970	2.44	466
2	12.37	1.63	2.30	24.5	88	2.22	2.45	0.40	1.90	2.120000	0.890	2.78	342
2	12.04	4.30	2.38	22.0	80	2.10	1.75	0.42	1.35	2.600000	0.790	2.57	580
3	12.86	1.35	2.32	18.0	122	1.51	1.25	0.21	0.94	4.100000	0.760	1.29	630
3	12.88	2.99	2.40	20.0	104	1.30	1.22	0.24	0.83	5.400000	0.740	1.42	530

Anche in questo caso i passi da fare sono:

1- Carico il dataset:
wine <- read.table("http://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data",
+ sep=",")

2- Dato che le variabili possono variare su scale diverse allora è una buona norma eseguire una standartizazione in modo tale da avere meno distorsione possibile dagli outliers.
standardisedconcentrations <- as.data.frame(scale(wine[2:14]))

3- Eseguo il calcolo dei principal components usando la funzione prcomp().
wine.pca <- prcomp(standardisedconcentrations)

4- Faccio il summary per ogniuno dei principal components :
summary(wine.pca) che mi da come risultato le seguenti righe :

Importance of components:

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10	PC11	PC12	PC13
Standard deviation	2.169	1.580	1.202	0.95	0.92	0.80	0.74231	0.54	0.537	0.5009	0.475	0.41	0.321
Proportion of Variance	0.32	0.19	0.11	0.079	0.063	0.046	0.049	0.021	0.02	0.013	0.017	0.018	0.0075
Cumulative Proportion	0.32	0.51	0.63	0.79	0.82	0.88	0.87	0.98	0.940	0.9617	0.97907	0.99205	1.00

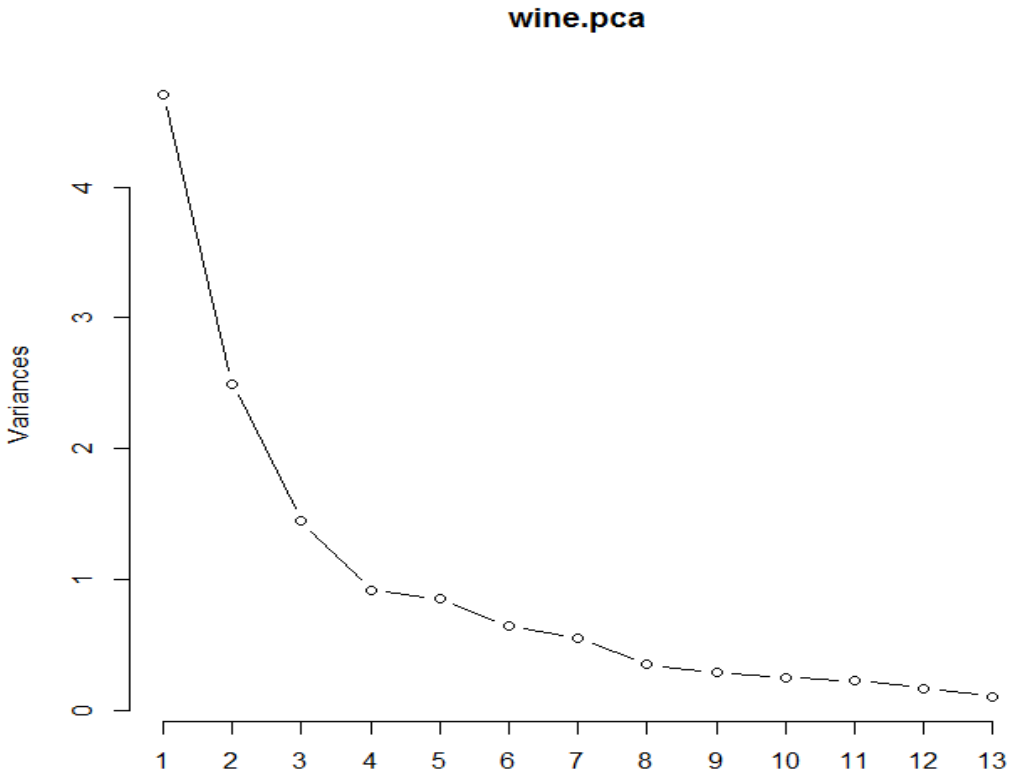
5- Visualizzo anche la dev.std degli principal components con la seguente :
wine.pca$sdev

che mi da come risultato :
2.1692972 1.5801816 1.2025273 0.9586313 0.9237035 0.8010350 0.7423128 0.5903367 0.5374755 0.5009017 0.4751722 0.4108165 0.3215244
Avendo standardizzato le variabili all 'inizio abbiamo che la varianza totale è uguale al numero delle variabili stesse in quanto la varianza di ogni variabile standartizzata è uguale a 1, il che spiega il risultato sottostante :
sum((wine.pca$sdev)^2)
[1] 13

6- Per decidere quanti PCA tenere (cioè quanti sono quelli più importanti che descrivono la maggior parte delle info del dataset) possiamo visualizzare la varianza di ciascun PCA:

screeplot(wine.pca, npcs = min(100, length(wine.pca$sdev)), type = "lines")

ottendo il seguente grafico :



Dal grafico si capisce come è possibile scartare il PCA dal quarto in poi in quanto passando dal quarto PCA al 13-esimo non si ottengono cambiamenti significativi della varianza.

Anche il criterio di Kaiser secondo il quale bisogna tenere solo gli PCA la cui dev.std è maggiore di 1 suggerisce la stessa conclusione come sopra.

Un'altra regola per fare questo (cioè decidere quanti PCA tenere) è definire una soglia ad esempio 80% della varianza totale che si vuole spiegare nel dataset. Secondo questa regola invece dovremo tenere i primi 5 PCA in quanto sono questi quelli che descrivono almeno 80% della varianza totale.

Per visualizzare in R i vari livelli di popolazione dell'Italia si può procedere secondo la seguente maniera :

Il link per il seguente tutorial si trova in questa pagina :

http://www.r-bloggers.com/mapping-with-ggplot-create-a-nice-choropleth-map-in-r

Si caricano le seguenti librerie :

```
library(ggplot2)
library(maptools)
library(rgeos)
library(Cairo)
library(ggmap)
library(scales)
library(RColorBrewer)

set.seed(8000)
```

Si carica il file .shp in un oggetto

```
ita <- readShapeSpatial("ITA/ITA_adm1.shp")
```

```
Definisco il numero dei regioni e la relativa popolazione come una distribuzion
e normale :
```

```
num.reg<-length(ita$NAME_1)
mydata<-data.frame(NAME_1=ita$NAME_1, id=ita$ID_1, prevalence=rnorm(num.reg, 55
, 20))

head(mydata)
```

```
          NAME_1   id prevalence
1        Abruzzo 1411   51.59156
2         Apulia 1412   45.76908
3     Basilicata 1413   20.47711
4       Calabria 1414   59.05009
5       Campania 1415   52.99557
6 Emilia-Romagna 1416   28.58262
```

Now we need to merge the shapefile and the dataset together. First, we **fortify()** the shapefile (a functio n of ggplot) to get it into a dataframe. We need to include a region identifier so that ggplot keeps that id when it turns it into a dataframe; otherwise it will make it up and it will not be possible to merge properly . This should be the same region identifier as what is in your prevalence data (or that data you want to map) so that later this can be merged properly.s

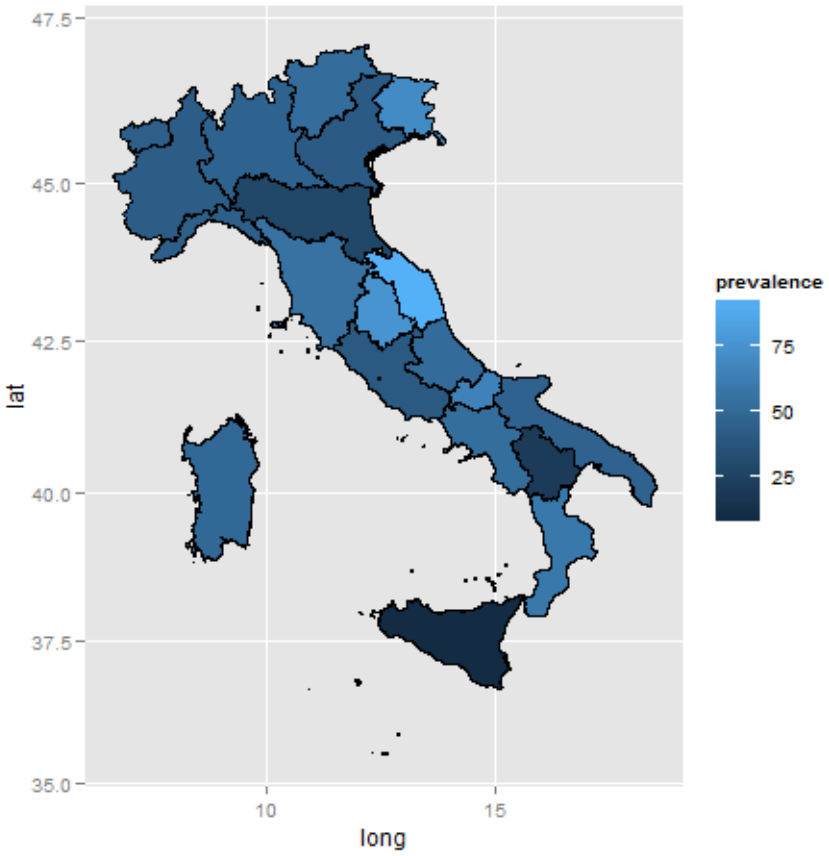
Per poter mergere I due data set in uno solo devo prima fare l'operazione di fortify :

```
ita.f <- fortify(ita, region = "ID_1")
```

```
Faccio il merging
```

```
merge.shp.coef<-merge(ita.f, mydata, by="id", all.x=TRUE)
final.plot<-merge.shp.coef[order(merge.shp.coef$order), ]
```

```
Finalmente posso visualizzare il risultato :
ggplot() + geom_polygon(data = final.plot, aes(x = long, y = lat, group = group
, fill = prevalence), color = "black", size = 0.25) + coord_map()
```



Graphics Tips

Per ruotare le etichete dei grafici bisogna mettere il parametron las = 2 come nel seguente Esempio :

```
barplot(mytable,main="Car makes",ylab="Freqency",xlab="make",las=2)
```

Association rules algorithm

One of the algorithms that is easy to implement in R is also the association rules algorithm. This algorithm tries to find a number of associations between the items of different records in a dataset.

An example is that of the records of a supermarket's dataset made of 5 records.

T1 milk, bread, beer, chips, butter

T2 milk, bread, potatos, beer,salad

T3 milk, bread, biscuits, tea, butter, jam

T4 wine, meat, potatos, cheese, salad, bread

T5 beer, potatos, cheese, salad, bread, toothpaste, tuna fish

T6 wine, meat, potatos, salad, beer, cake, tea, icecream

In this dataset there are present different items such as milk, bread, beer, potatos, meat, salad, biscuits, butter, tea, cheese, jam, chips and wine.

The association rules algorithm tries to find a rule that associates an item X with an item Y such as milk and bread.

There are three transactions in which bread and milk are present together. That means that people that buy milk will buy also bread. This problem is also called the market basket analysis problem

There are measures that indicate how strong is a certain association between two types of items. Those measures are support and confidence.

Support is the indication of how frequently an item appears in the dataset or how frequently an itemset appears in the database. Confidence instead is an indication of how often the rule(spotted by support measure) is found to be true.

Other measures are lift and conviction.

Analysis of a large georeferenced dataset. (Census population dataset of Telecom BigData challenge 2015)