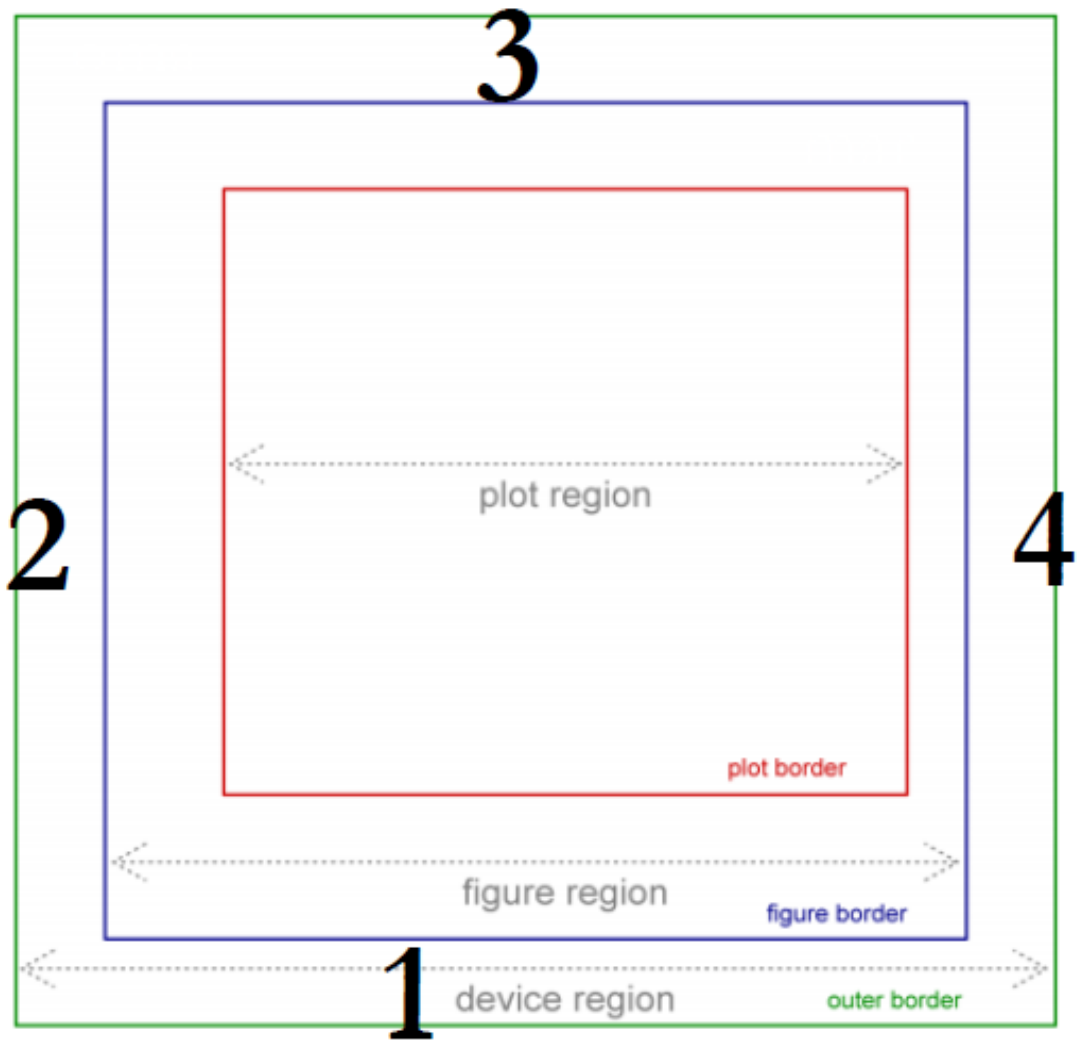
R GRAPHICS

Camilla Avanzi
Phd Student in Biologia Evoluzionistica ed Ecologia
camilla.avanzi@gmail.com

LA FINESTRA GRAFICA DI R



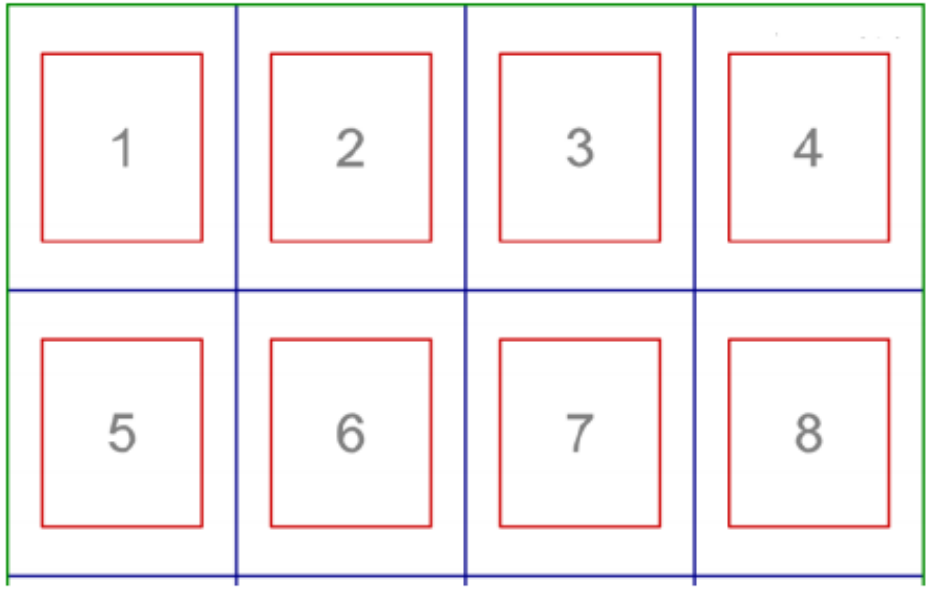

```
par()$mar
[1] 5.1 4.1 4.1 2.1
par()$oma
[1] 0 0 0 0
```

```
par(mar=c(x1,x2,x3,x4))
par(oma=c(x1,x2,x3,x4))
```

```
dev.off()
```

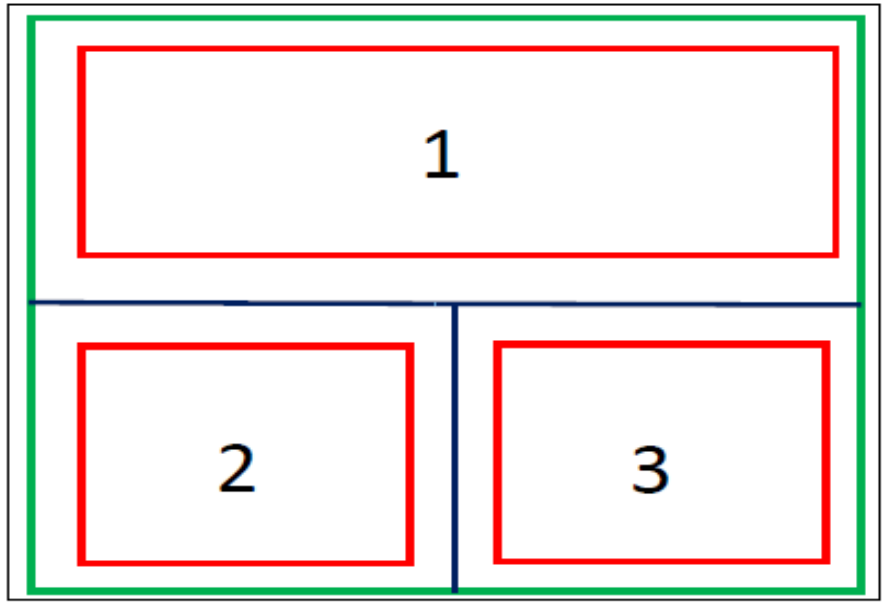
per chiudere la finestra grafica (vengono ripristinate le impostazioni di default)

ORGANIZZAZIONE DELLA FINESTRA GRAFICA




```
par()$mfrow
[1] 1 1
```

```
par(mfrow=c(2,4))
```

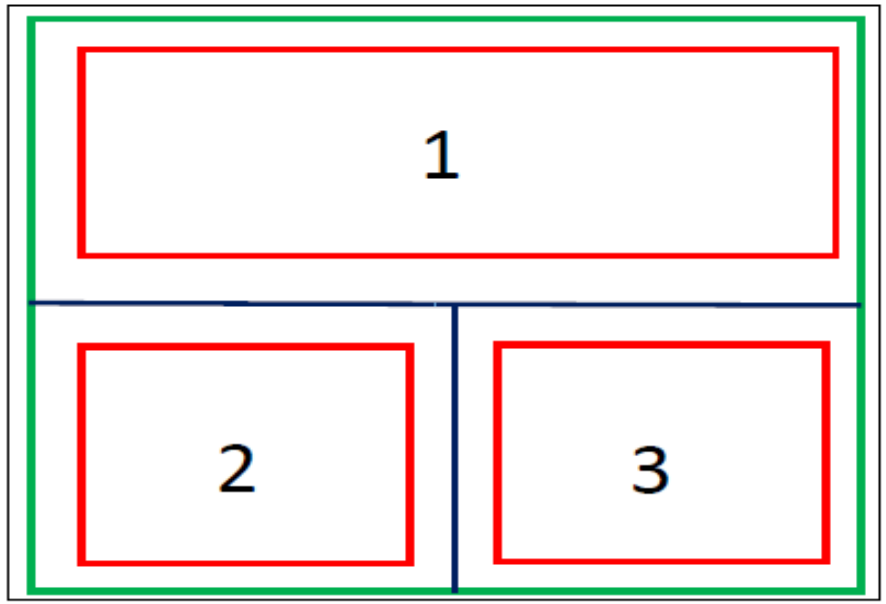


La funzione `layout()` permette di suddividere lo spazio grafico in modo più articolato e complesso.

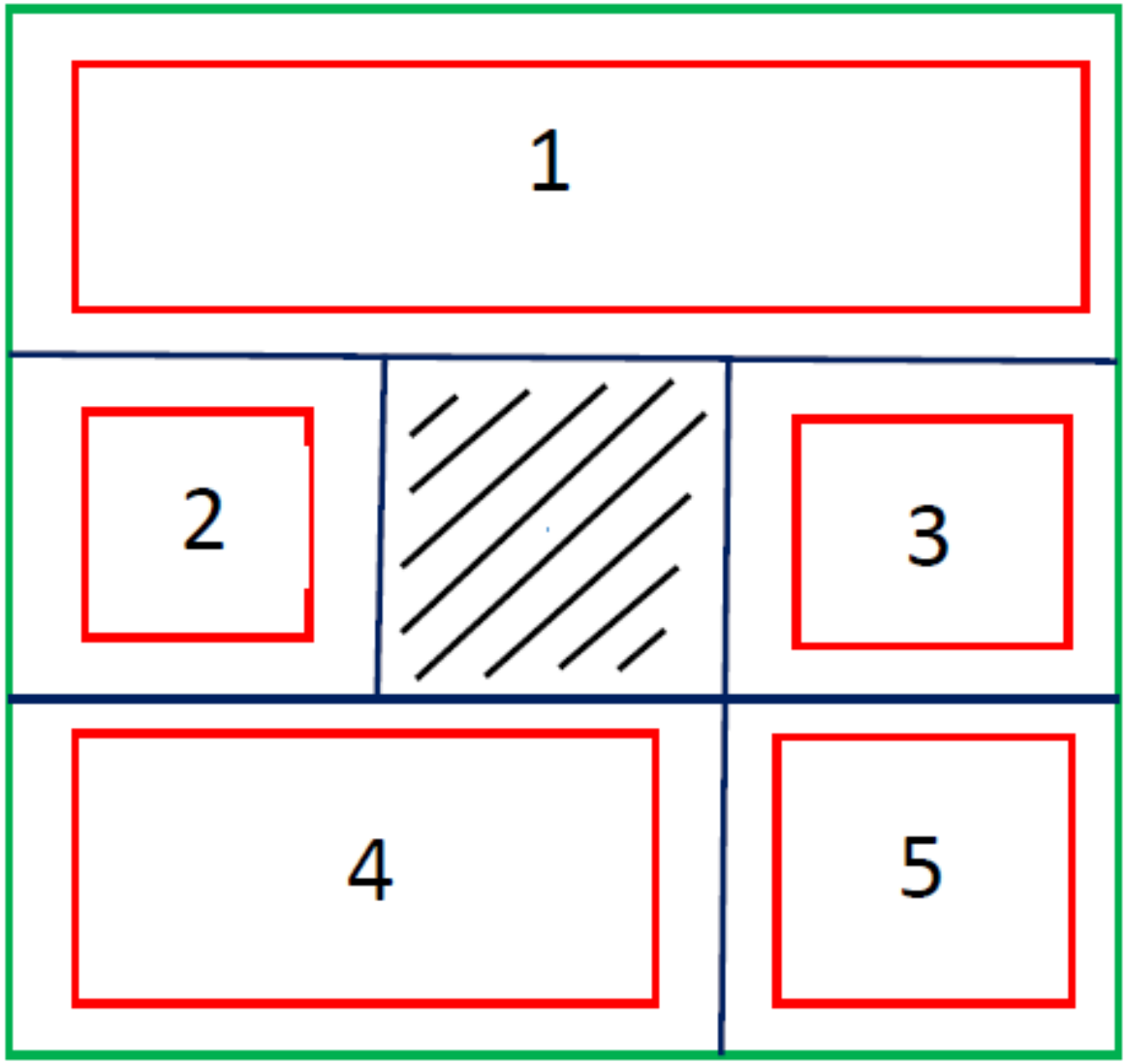
LA FUNZIONE LAYOUT()

```
layout(mat, widths=rep.int(1,ncol(mat)), heights=rep.int(1, nrow(mat)))
```

- ✓ `Mat` definisce una matrice sulla base della quale lo spazio grafico viene diviso in righe e colonne.
- ✓ I valori della matrice indicano da quale figura è occupata ciascuna cella. Se il valore è <0> la cella viene lasciata vuota.




```
elle <- layout(mat=matrix(c(1,1,2,3),
ncol=2, nrow=2, byrow=T))

layout.show(elle)
# per visualizzare la suddivisione
```

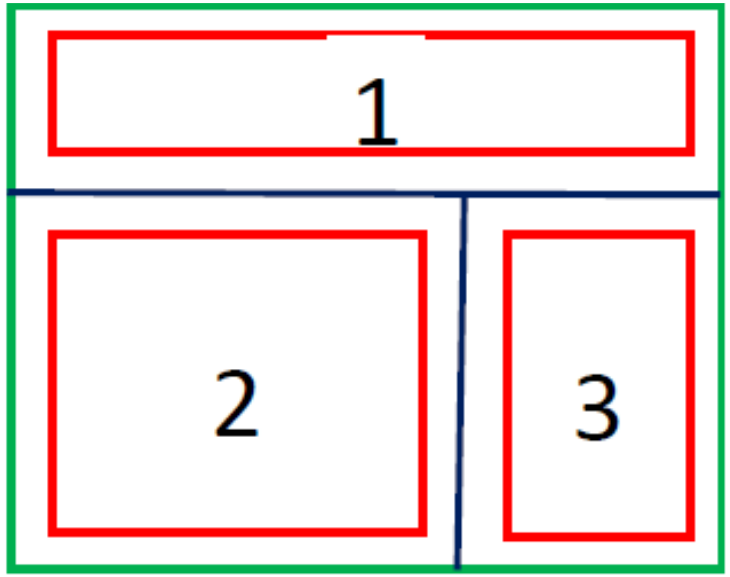


Qual è la sintassi di `layout()` per ottenere la seguente suddivisione della finestra grafica?

LA FUNZIONE LAYOUT() - 2

```
layout(mat, widths=rep.int(1,ncol(mat)), heights=rep.int(1, nrow(mat)))
```

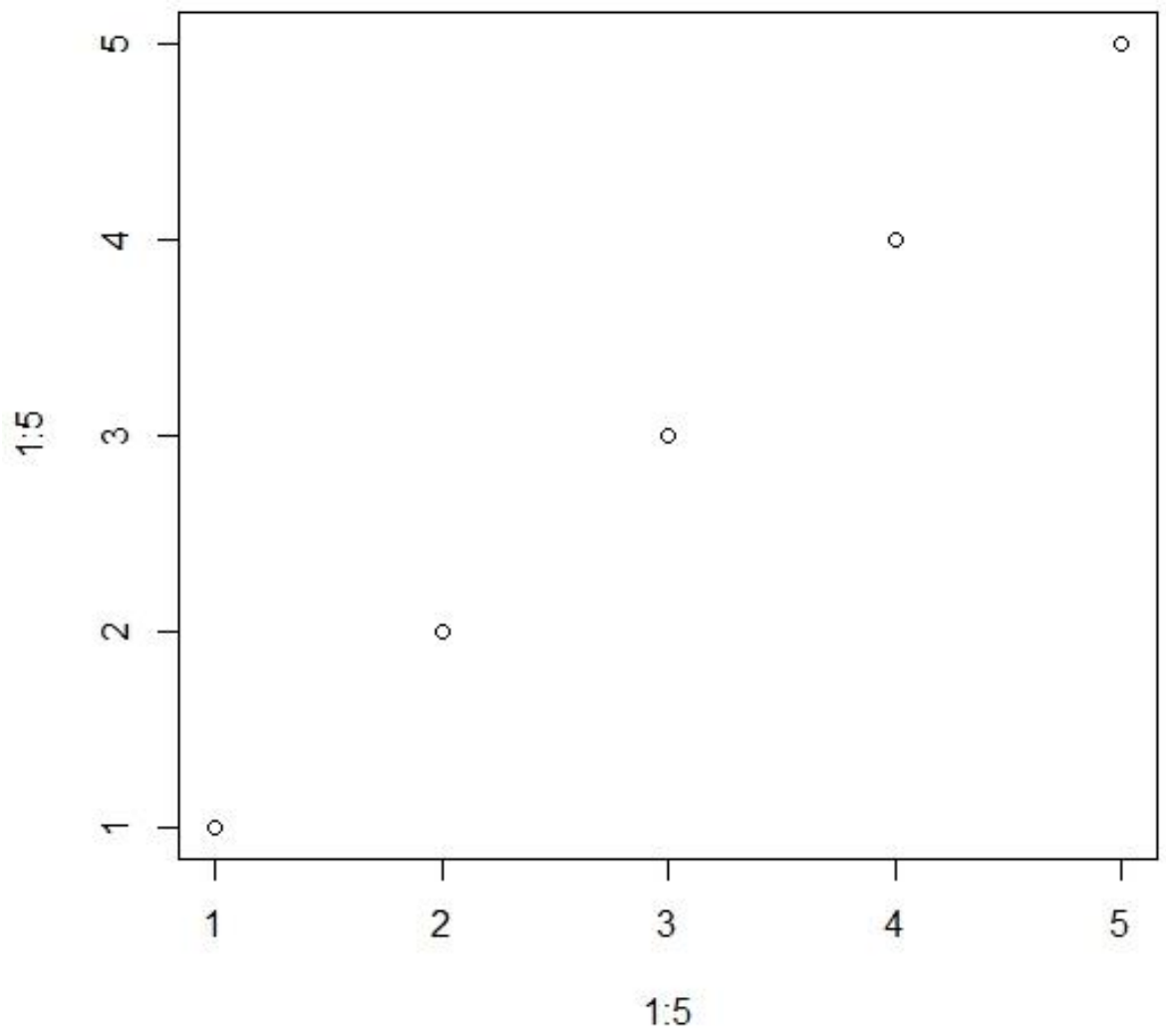
- ✓ `Widths` e `heights` sono due vettori che specificano, rispettivamente, l'ampiezza delle colonne e l'altezza delle righe.
- ✓ I due vettori sono formati da tanti elementi quante sono, rispettivamente le colonne e le righe della matrice.




```
elle <- layout(mat=matrix(c(1,1,2,3),
ncol=2, nrow=2, byrow=T),
widths=c(2,1), heights=c(1,2))

layout.show(elle)
```

LA FUNZIONE PLOT() E I SUOI ARGOMENTI

`plot(x,y,…)` oppure `plot(y~x,…)`



Alcuni dei possibili argomenti:

```
xlim   # range di ampiezza dell'asse x
ylim   # range di ampiezza dell'asse y
xlab   # titolo dell'asse x
ylab   # titolo dell'asse y
las    # orientamento dei label rispetto agli
         assi; valori possibili: {0;1;2;3}
main   # titolo del grafico
type   # tipo di grafico da disegnare ("p" per i
         punti, "l" per le linee, "b" per
         entrambi, "n" per un grafico «vuoto»)
pch    # tipo di punto
lty    # tipo di linea
cex    # dimensione del punto
col    # colore del punto
```

… e molti altri (vedi help di `par` o di `plot`)

✓ `pch=16`

`col` # definisce il contorno
`bg` # definisce l'interno

✓ `lty=2` oppure `lty="dashed"`

0=blank, 1=solid (default), 2=dashed, 3=dotted, 4=dotdash, 5=longdash, 6=twodash

✓ `col="red"`

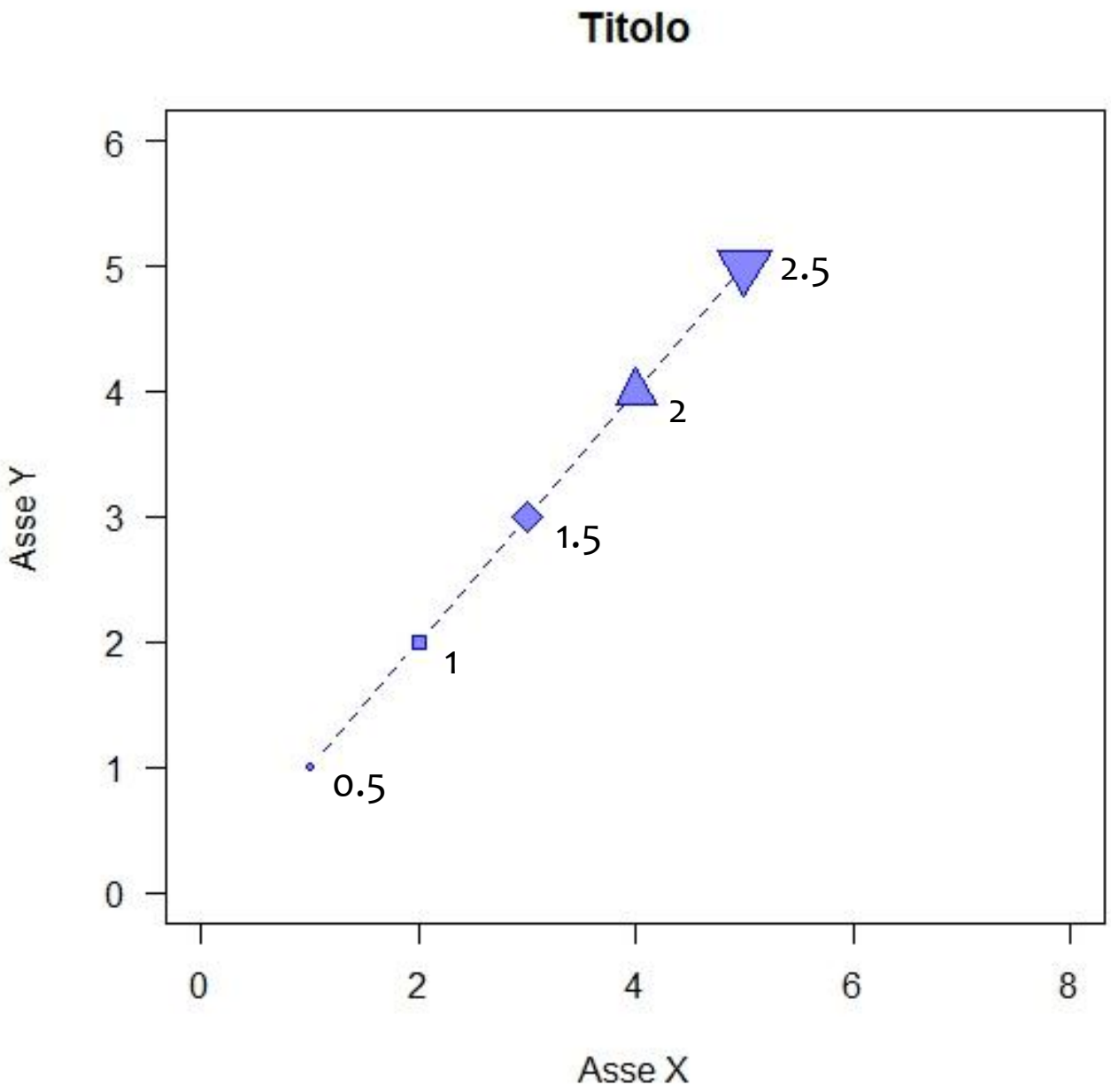
```
col2rgb("red")  # traduce il colore di un vettore RGB
      [,1]
red    255
green    0
blue     0
```

```
col=rgb(red=255,green=0,blue=0,
alpha=120,max=255)
```

`alpha` regola la trasparenza e come tutti gli altri argomenti può variare tra [0:255]

```
col=rainbow(5)
```

[Altre palette: `terrain.colors()`, `topo.colors()`,...]

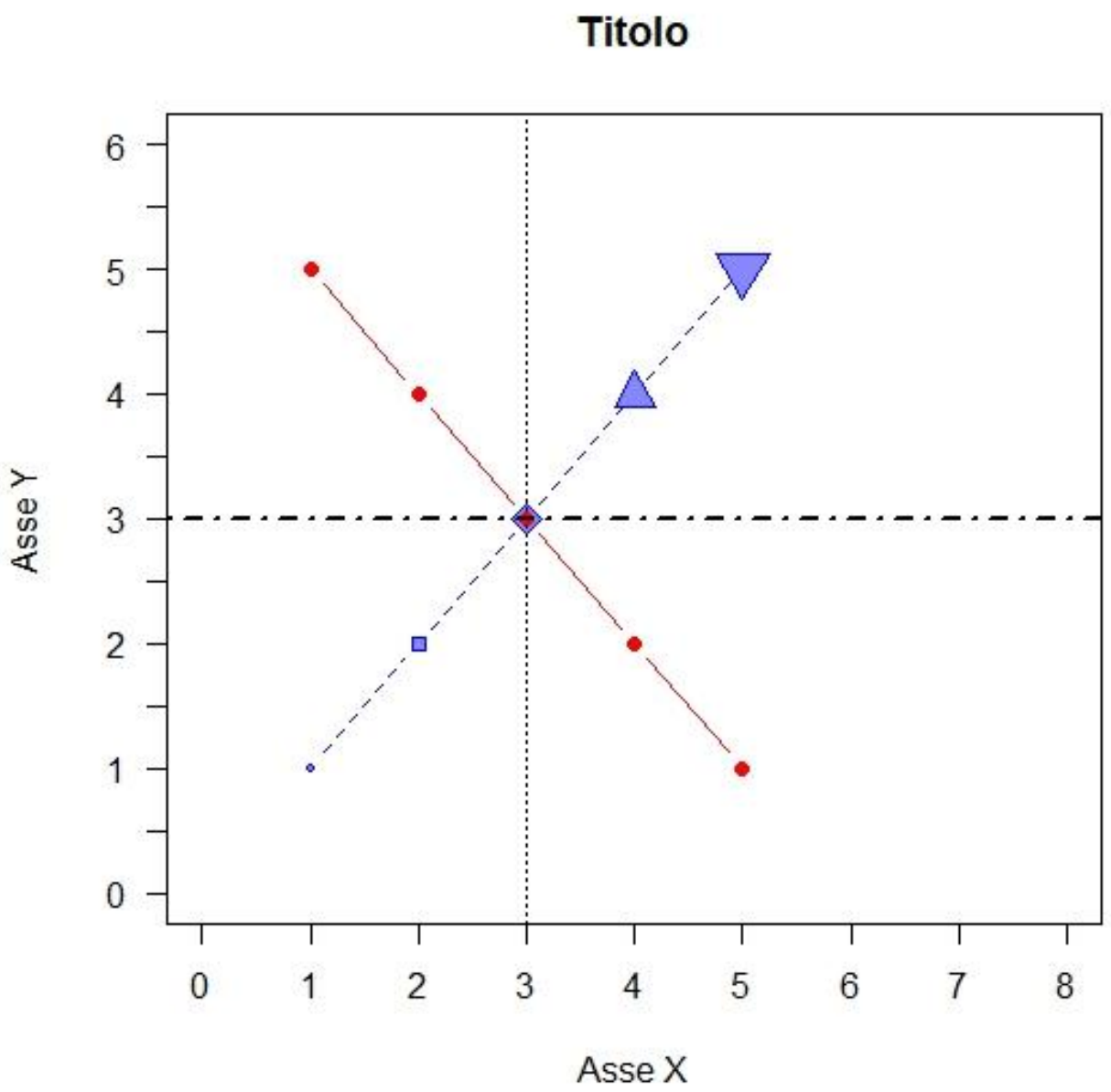


Qual è la sintassi di `plot()` per ottenere il seguente grafico?

- ✓ x=1:5, y=1:5
- ✓ Il colore è il blu, con trasparenza 120
- ✓ I valori di `cex` sono indicati accanto a ciascun punto

ALCUNE UTILI FUNZIONI LOW-LEVEL

```
points(x, y,…)        # aggiunge punti ad un grafico già esistente

lines(x, y, …)        # aggiunge una linea ad un grafico già esistente

abline()              # aggiunge una retta
  abline(lm(y~x))                 # a partire da un oggetto <lm>
  abline(a=intercept, b=slope)    # specificando i valori di intercetta e pendenza
  abline(v=x1)                    # retta verticale
  abline(h=y1)                    # retta orizzontale

axis(1,…)             # aggiunge un asse x
axis(2,…)             # aggiunge un asse y
```



Qualche esempio:

```
points(1:5,5:1, pch=16, col="red",
type="b")
```

```
abline(v=3, lty=3)
```

```
abline(h=3, lty=4, lwd=2)
```

```
axis(1, at=c(1,3,5,7),
labels=seq(1,7,2))
```

```
axis(2, at=seq(0.5,5.5,1),
labels=F)
```

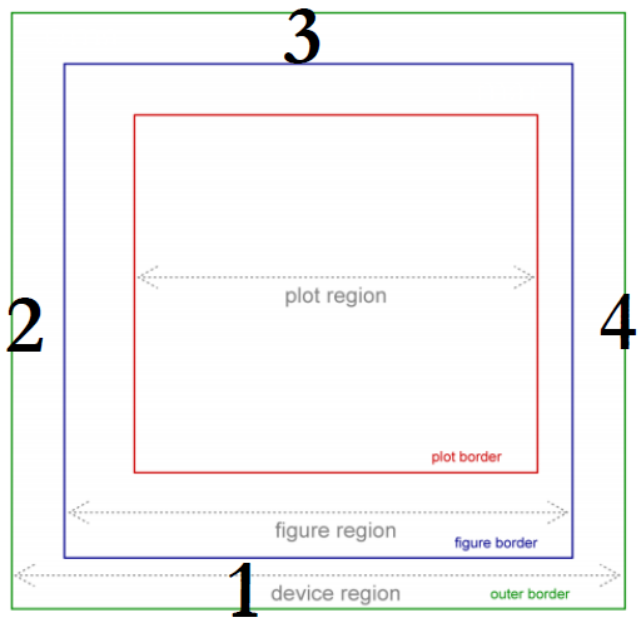
`lwd` definisce lo spessore della retta
`at` e `labels` definiscono, rispettivamente, i valori di x alla cui altezza apporre i «trattini» e le corrispondenti etichette

COME INSERIRE STRINGHE DI TESTO

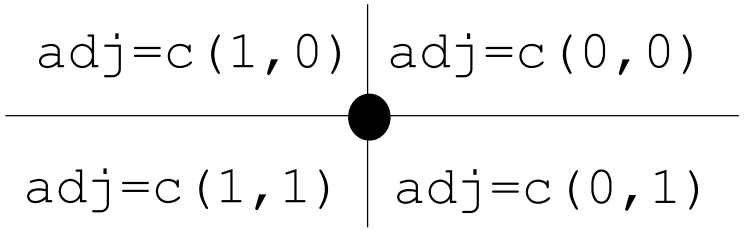
** Le funzioni `text()` e `mtext()` **

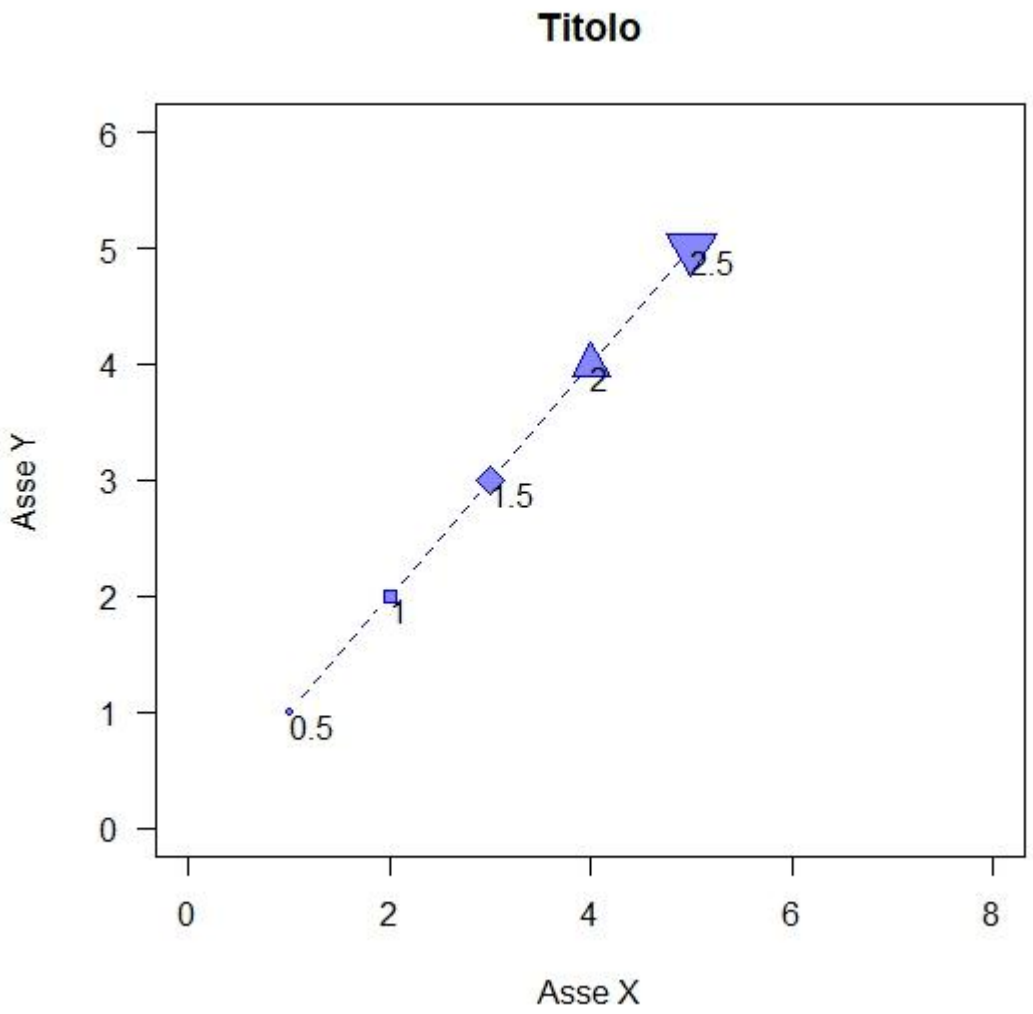
✓ `text(x, y, labels="ciao",…)`
- scrive di default nella plot region
- con `xpd=T` scrive nella figure region,
- con `xpd=NA` nella device region
- `srt` permette di modificare orientamento della stringa

✓ `mtext(side, line, text="ciao",…)`
- scrive di default nella figure region
- con `outer=T` nella device region

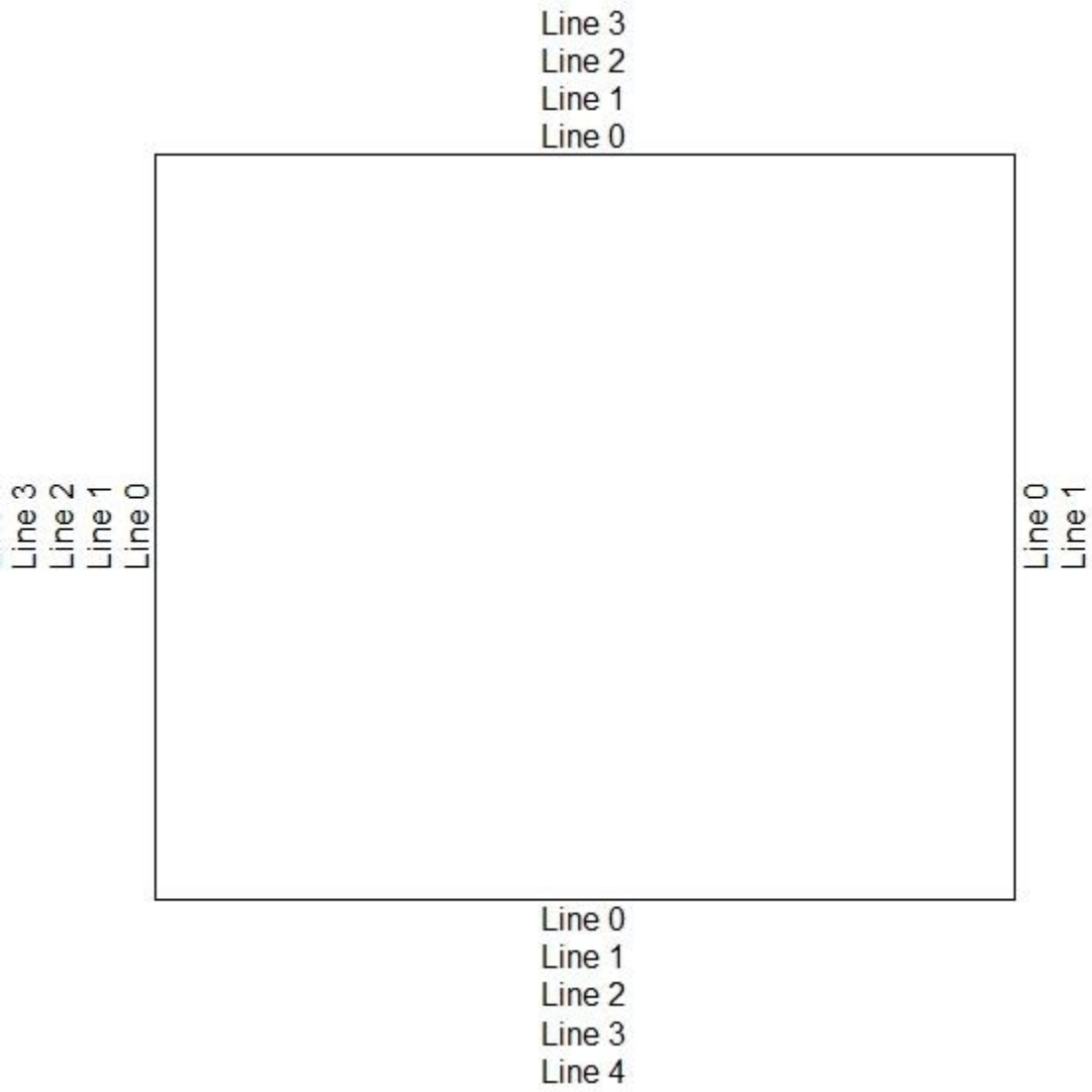


✓ `adj` è un argomento di entrambe le funzioni e determina la giustificazione della stringa: 0 (giustificato a sinistra), 0.5 (centrato, è il default), 1 (giustificato a destra).
In `text()` si può avere anche `adj=c(x,y)` per aggiustamenti sia sulle x che sulle y.

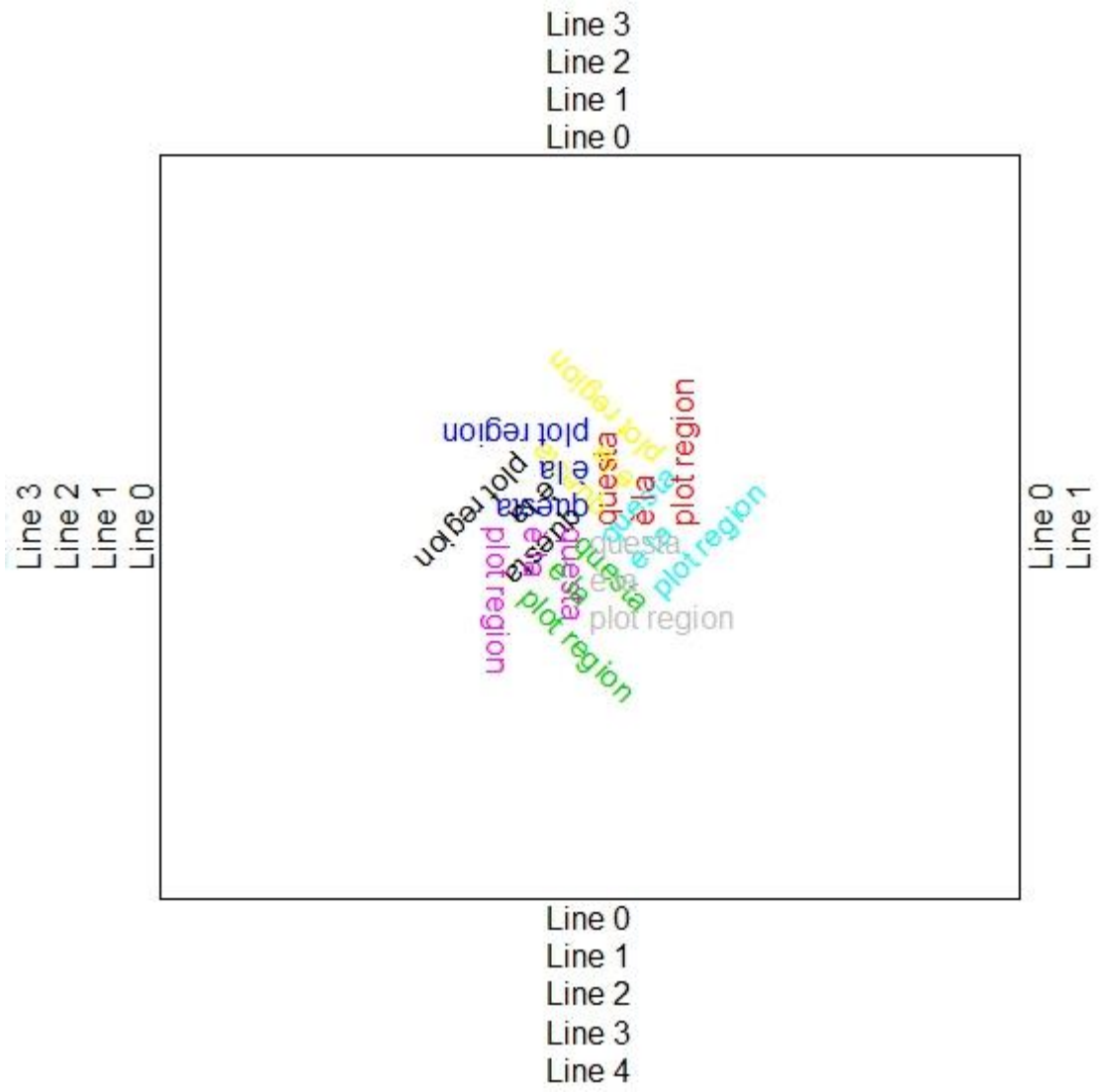





```
text(1:5,1:5, labels=seq(0.5,2.5,0.5),
adj=c(0,1))
```




```
# definiamo una finestra grafica vuota
plot(1, xlab="", ylab="", xaxt="n",
yaxt="n", type="n")
# xaxt="n" e yaxt="n" sopprimono gli assi


for(i in 0:4)
        mtext(side=1:4, line=i,
        text=paste("Line", i, sep=" "))


# definiamo dei margini esterni
par(oma=c(2,2,2,2))
```




```
for(i in seq(0,360,45))
      text(1,1,labels="questa\nè la\nplot
      region", adj=c(0,1), srt=i, col=i)
```

COME TROVARE LE COORDINATE DI UN PUNTO

** La funzione `locator()` **

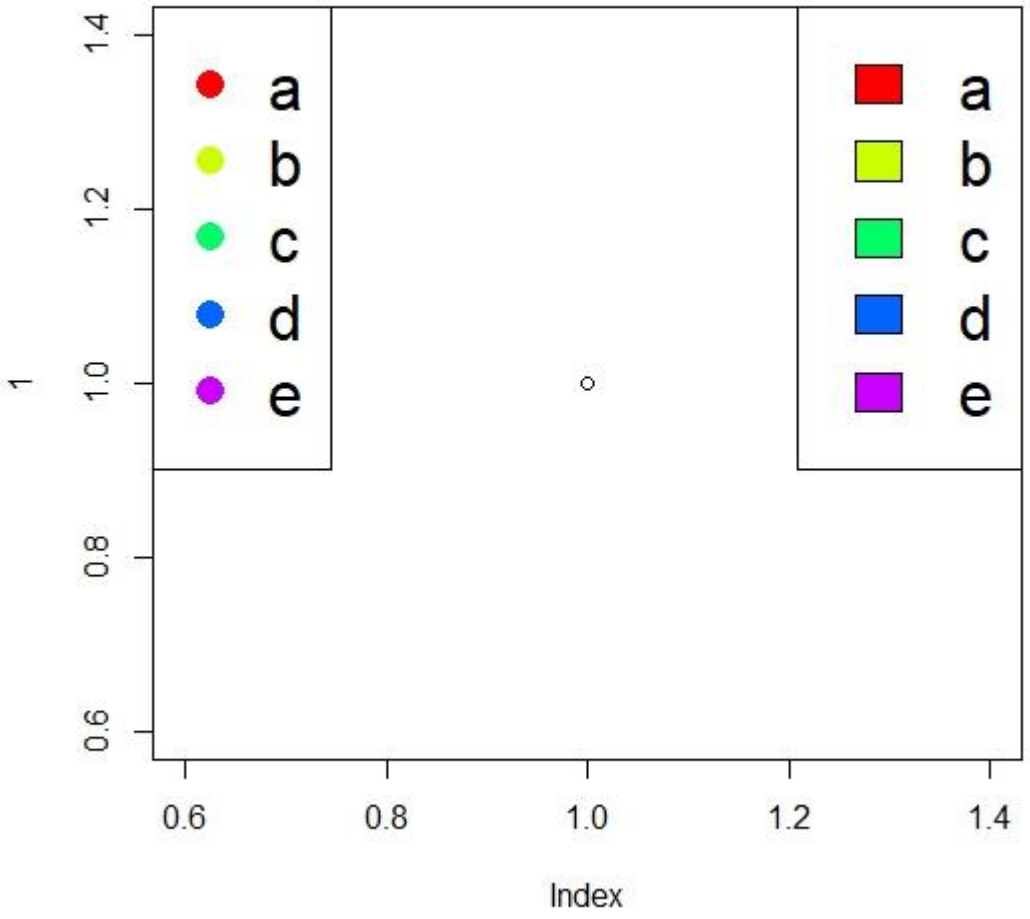
```
locator()

punto <- locator()  # è possibile assegnare le coordinate per utilizzarle successivamente

punto  # l'oggetto <punto> è una lista, stampandolo si ottengono le coordinate
$x
[1] 5.917301

$y
[1] 1.125192

text(punto$x, punto$y, labels="ciao")

punti <- locator()
punti
$x
[1] 0.7401774 0.8898448 1.1273171 1.3129047 1.3129047 1.1213304 1.1073614

$y
[1] 1.1574425 1.2357033 1.2771355 1.0055243 1.0055243 0.7408184 0.7523274

for(i in 1:7) text(punti$x[i], punti$y[i], "ciao", col=i)
```

COME INSERIRE LEGENDA

** La funzione `legend()` **

```
legend(x, y, legend, col=par("col"), fill=NULL, pch, lty, lwd, bty, cex,
       pt.cex,…)
```

- ✓ `x, y` # le coordinate alle quali posizionare la legenda, oppure valori di default quali `"topright"`, `"topleft"`, `"bottomright"`, `"bottomleft"`
- ✓ `legend` # testo della legenda
- ✓ `col` # colore dei punti o delle linee che compaiono nella legenda
- ✓ `fill` # specifica un vettore di colori di cui colorare i box della legenda (di default è nullo)
- ✓ `bty` # il tipo di box intorno alla legenda; i valori possibili sono `"o"` (default) oppure `"n"`
- ✓ `cex` # dimensione della legenda
- ✓ `pt.cex` # dimensione dei simboli della legenda



un modo per creare una legenda:

```
plot(1)
```

- `legend("topleft", legend=letters[1:5], col=rainbow(5), pch=16, cex=2)`

oppure creiamo un vettore di colori

```
mycolor <- rainbow(5)
```

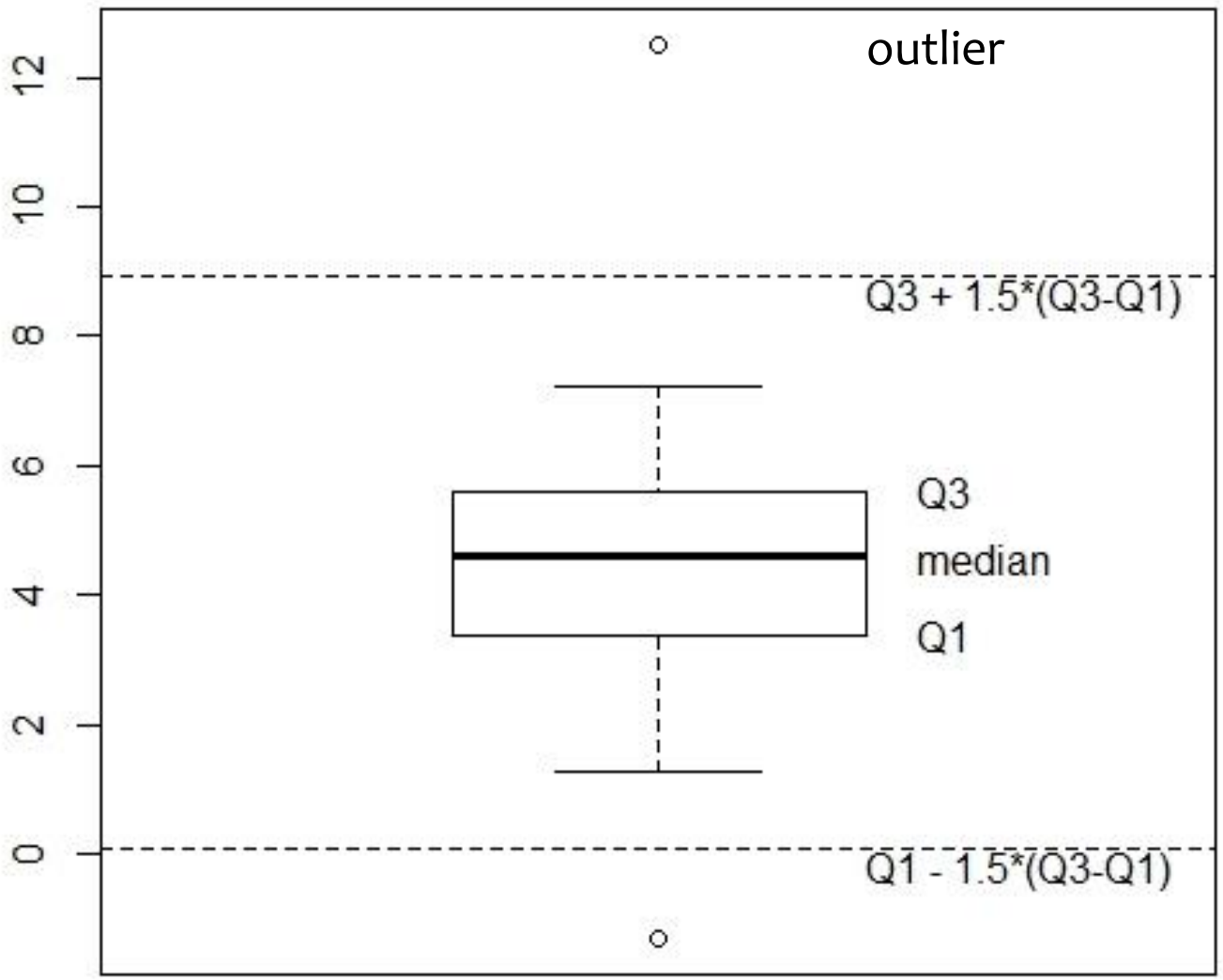
indicizziamo questo vettore con delle stringhe di testo

```
names(mycolor) <- letters[1:5]
mycolor
```

- `legend("topright", legend=names(mycolor), fill=mycolor, cex=2)`

ALTRI TIPI DI GRAFICI

```
boxplot(y ~ group, …)     # boxplot
boxplot(x, …)

barplot(x)                # istogrammi

pie(x,…)                  # grafico a torta

hist(x, …)                # distribuzione di frequenza
```



IN BASE A QUALEREGOLA SONO DISEGNATI I BAFFI DEL BOXPLOT?

```
# aggiungiamo i punti
points(rep(1,length(x)), x, pch=16,
col=rgb(255,0,0,alpha=150,max=255))

# può essere utile assegnare il boxplot
pippo <- boxplot(x)
```

```
# per aggiungere del «rumore» ad un vettore numerico
jitter(x, factor,…)

x        # il vettore su cui aggiungere il rumore
Factor   # è un numero per specificare quanto rumore
           aggiungere
```

QUALCHE GRAFICO DI ESEMPIO

dataset `iris` contiene i dati di lunghezza e larghezza di petali e sepali di tre specie di iris

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

levels(iris$Species)
[1] "setosa"     "versicolor" "virginica"

summary(iris)

boxplot(iris$Sepal.Length~iris$Species)
```

RAPPRESENTARE LA LUNGHEZZA MEDIA DEI PETALI DELLE 3 SPECIE DI IRIS CON BARRA D'ERRORE

```
# calcolare medie e deviazioni standard
mPet <- tapply(iris$Petal.Length, iris$Species, mean)
mPet
    setosa versicolor  virginica
     1.462      4.260      5.552
dsPet <- tapply(iris$Petal.Length, iris$Species, sd)
dsPet
    setosa versicolor  virginica
 0.1736640  0.4699110  0.5518947
```

```
# vettore con i nomi delle tre specie
specie <- as.character(unique(iris$Species))
specie
[1] "setosa"     "versicolor" "virginica"
```

```
# vettore numerico per asse delle ascisse, della lunghezza opportuna
x <- 1:length(specie)
x
[1] 1 2 3
```

```
plot(mPet~x)
```

definisco una serie di argomenti per rendere il grafico più leggibile

```
plot(mPet ~ x, type="l", main="Iris - Petali",
xlim=c(0.3,3.8),ylim=c(0,8), xlab="Specie",
ylab="Lunghezza (mm)", xaxt="n")
```

creo un asse x di modo da posizionare dove voglio i ticks (at), con le etichette che voglio (label)

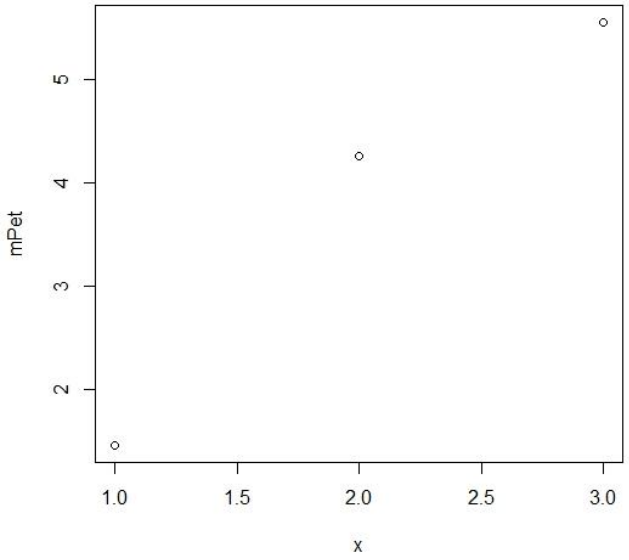
```
axis(1,at=x,labels=specie)
```

aggiungo i punti rappresentati dalla media

```
points(mPet ~ x, pch=21, col = "black", bg="red", cex=1.8)
```

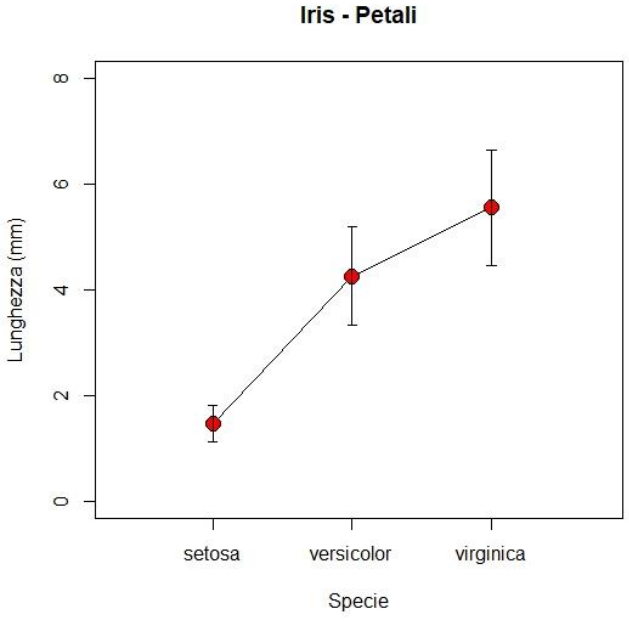
disegno le barre d'errore

```
arrows(x, mPet-2*dsPet, x, mPet+2*dsPet,
angle=90,length=0.04, code=3)
```



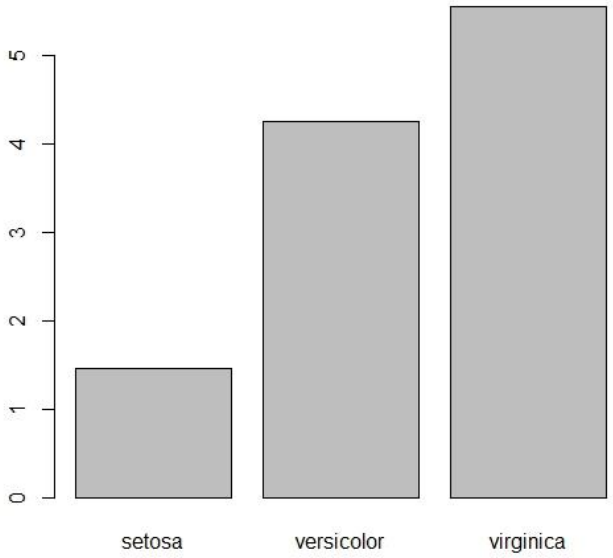

```
arrows(x0, y0, x1, y1, length, angle, code, …)
```

- ✓ `x0` e `y0` # coordinate del punto di partenza
- ✓ `x1` e `y1` # coordinate del punto di arrivo
- ✓ `angle` # angolo della freccia (90 se si vuole una «T»)
- ✓ `length` # lunghezza delle ali della freccia
- ✓ `code` # tipo di freccia da tracciare: 1 (punta su0 [x0,y0]), 2 (punta su [x1,y1]), 3 (punta da entrambi i lati



Quando sull'asse delle x è rappresentato un fattore senza un vero e proprio valore numerico, un grafico molto usato è il `barplot`.

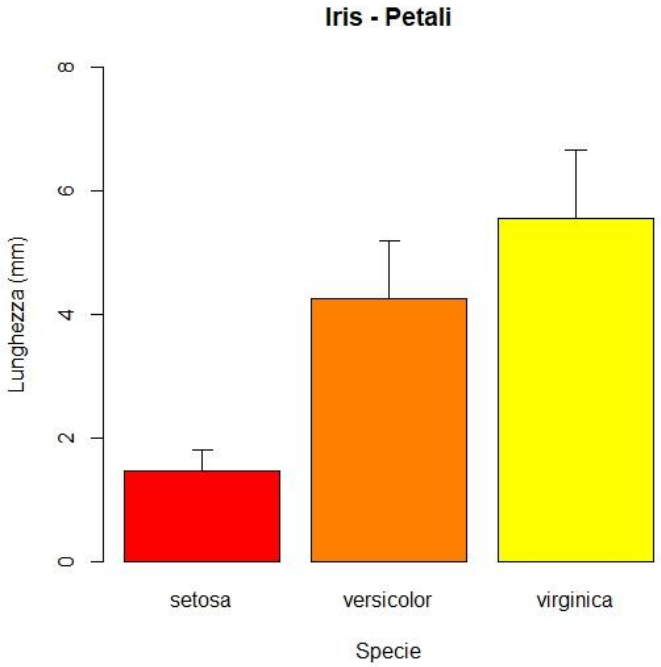
```
barplot(mPet)
```




```
xc <- barplot(mPet, main="Iris - Petali", ylim=c(0,8),
xlab="Specie", ylab="Lunghezza (mm)",
col=heat.colors(length(specie)))
```

è utile assegnare il barplot: stampando <xc> infatti si scopre che la funzione barplot restituisce un vettore con i valori in ascissa dei punti centrali di ciascuna barra.

```
arrows(xc, mPet, xc, mPet+2*dsPet, angle=90,length=0.08)
```



COMPLICHIAMO UN PO' LE COSE…

dataset `ChickWeight` contiene i dati di crescita di n pulcini sottoposti a 4 diete diverse

```
data(ChickWeight)
d.df <- ChickWeight

head(d.df)
Grouped Data: weight ~ Time | Chick
  weight Time Chick Diet
1     42    0     1    1
2     51    2     1    1
3     59    4     1    1
4     64    6     1    1
5     76    8     1    1
6     93   10     1    1

levels(d.df$Diet)
[1] "1" "2" "3" "4"
```

calcolare medie e deviazioni standard

```
m <- tapply(d.df$weight,list(d.df$Time,d.df$Diet),mean)

s <- tapply(d.df$weight,list(d.df$Time,d.df$Diet),sd)
```

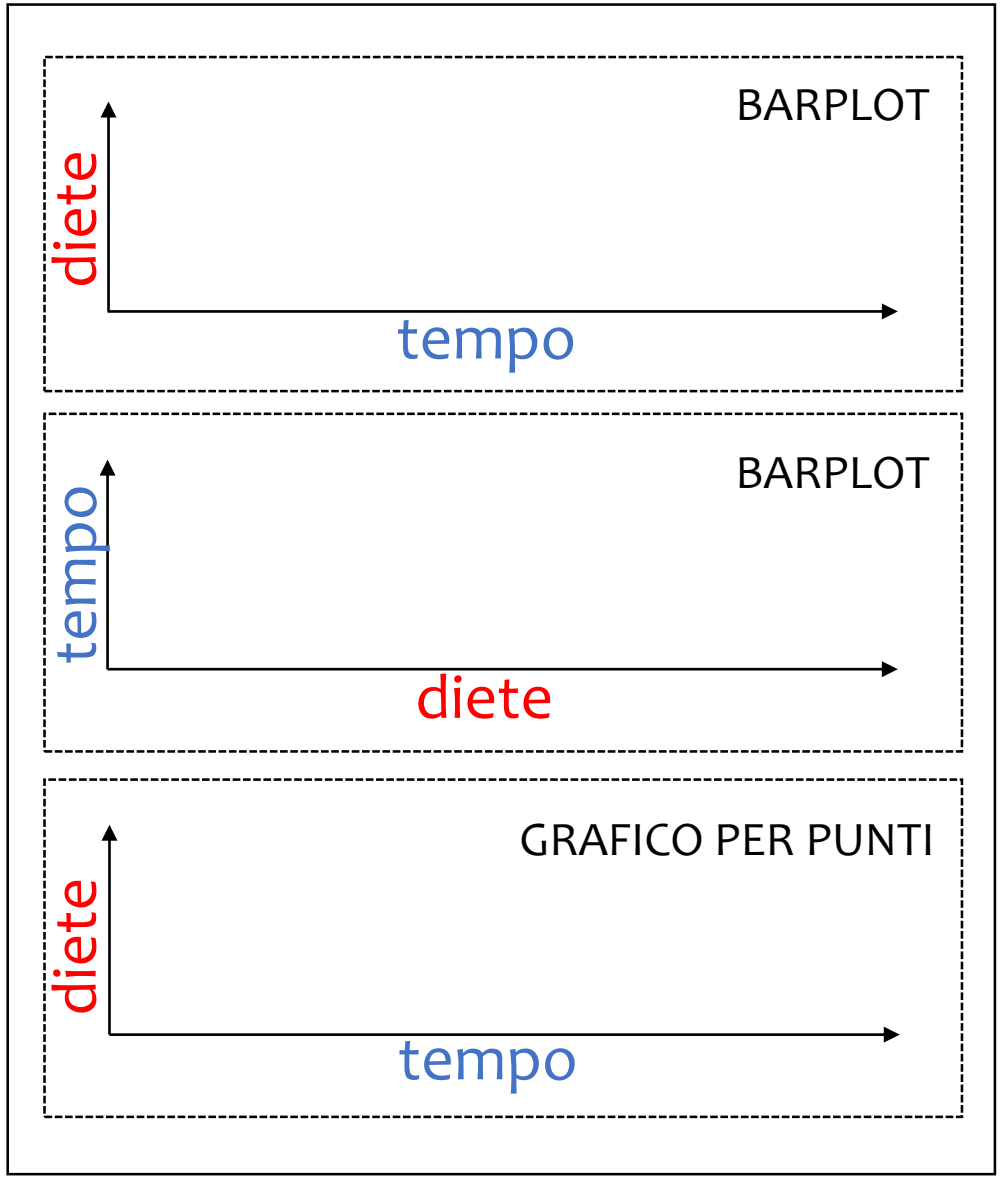
DIETE

```
m
          1      2      3        4
0   41.40000   40.7   40.8  41.0000
2   47.25000   49.4   50.4  51.8000
4   56.47368   59.8   62.2  64.5000
6   66.78947   75.4   77.9  83.9000
8   79.68421   91.7   98.4 105.6000
10  93.05263  108.5  117.1 126.0000
12 108.52632  131.3  144.4 151.4000
14 123.38889  141.9  164.5 161.8000
16 144.64706  164.7  197.4 182.0000
18 158.94118  187.7  233.1 202.9000
20 170.41176  205.6  258.9 233.8889
21 177.75000  214.7  270.3 238.5556
```

```
s
            1        2           3          4
0   0.9947229 1.494434   1.032796  1.054093
2   4.2781575 2.875181   2.412928  1.932184
4   4.1280668 2.299758   2.780887  2.549510
6   7.7572829 4.168666   5.704774  5.065131
8  13.7761978 14.802778 12.348639  9.335714
10 22.5424875 24.295633 20.190482 11.430952
12 32.6179372 37.458273 27.072742 15.276707
14 37.3837632 43.697063 34.503623 15.732486
16 45.0374028 52.824763 44.580015 25.302613
18 49.2194964 63.331667 57.587518 33.557413
20 55.4358400 70.252244 65.243901 37.568086
21 58.7020726 78.138126 71.622545 43.347754
```

TEMPO (in giorni)

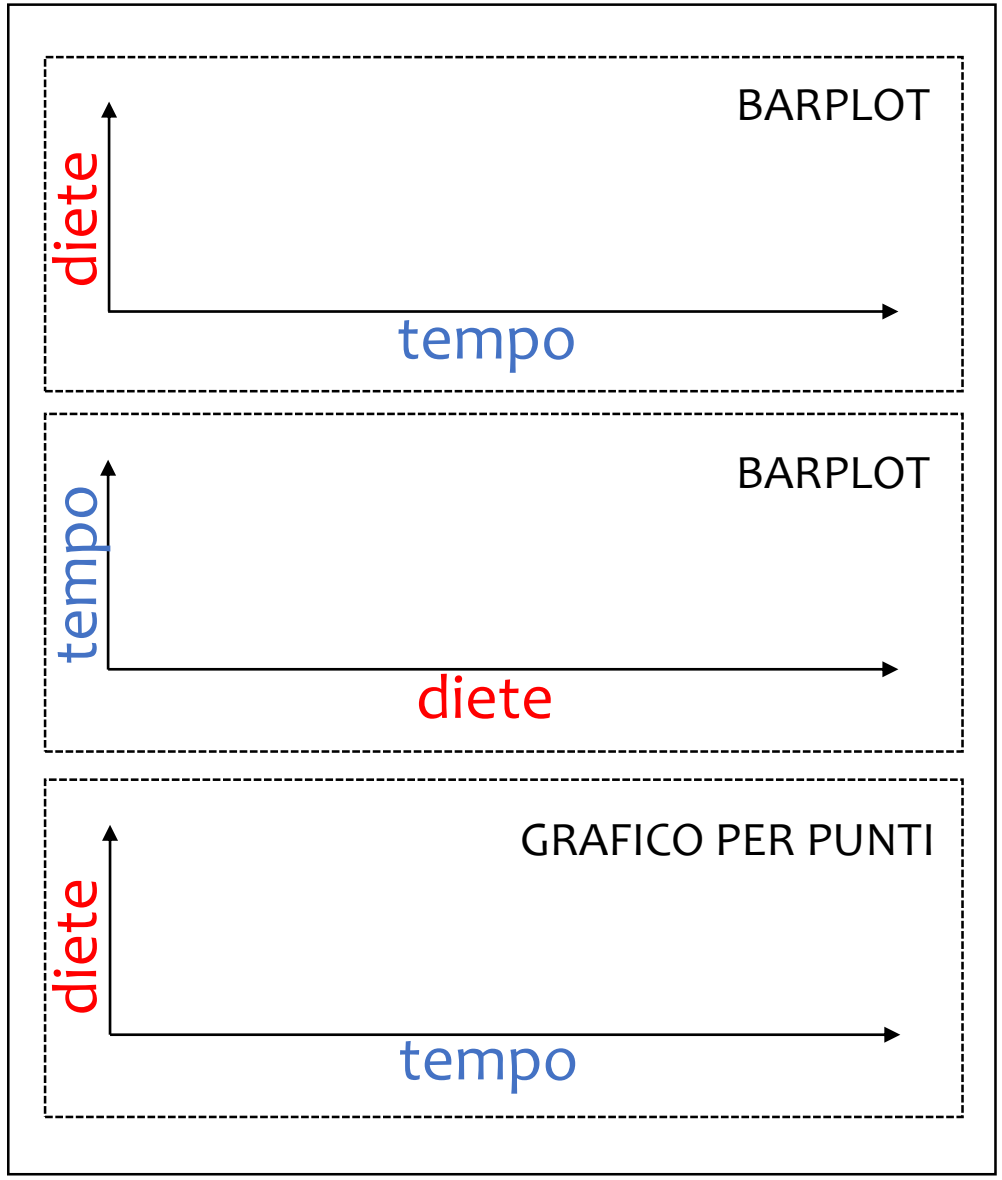
```
diets <- colnames(m)
days <- rownames(m)
```




```
# impostazione finestra grafica
par(mfrow=c(3,1))
```

impostazione finestra grafica
```
par(mfrow=c(3,1))
```

beside=T per affiancare le barre relative alle diverse diete



vettore di colori per identificare le diete
```
colori <- heat.colors(length(diets))

xc <- barplot(t(m), beside=T, col=colori, xlab="Days",
ylab="Weight (g)", ylim=c(0, 350))

arrows(xc, t(m), xc, t(m)+t(s), angle=90, length=0.02)

legend("topleft", fill=colori, legend=paste("Diet:",diets))
```

vettore di colori per identificare i giorni
```
colori2 <- topo.colors(length(days))

xc <- barplot(m, beside=T, col=colori2, xlab="Diet",
ylab="Weight (g)", ylim=c(0, 350))

arrows(xc, m, xc, m+t(s), angle=90, length=0.02)

legend("topleft", fill=colori, legend=paste("Day:",days))
```

```
days <- as.numeric(days)

## produco un grafico vuoto e senza asse delle ascisse
plot(m[,1] ~ days, type="n", xlab="Days", ylab="Weight (g)", ylim=c(0, 350),
xaxt="n")

## aggiungo l'asse x e i ticks in corrispondenza dei giorni dell'esperimento
axis(1, at=days, label=days)

## con un ciclo for sulle diete aggiungo le linee e le barre d'errore
for (d in 1:length(diets)){
       jdays <- days-0.6+d*0.2  ## sposto leggermente le x per non far sovrapporre le barre
       lines(m[,d] ~ jdays, col=colori[d], lty=d)
       arrows(jdays, m[,d]-s[,d], jdays, m[,d]+s[,d], angle=90, length=0.02, code=3)
       }

## traccio i punti per ultimi
for (d in 1:length(diets)){
       jdays <- days-0.6+d*0.2
       points(m[,d] ~ jdays, col=colori[d], pch=16, cex=1.2)
       }

## aggiungo la legenda
legend("topleft", lty=1:length(diets), col=colori, pch=16, legend=paste("Diet:",diets),cex=1.2)
```

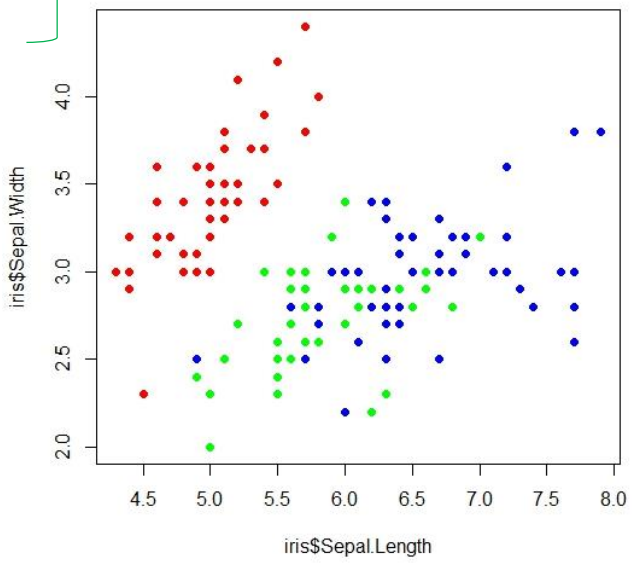
LUNGHEZZA vs LARGHEZZA DEI SEPALI

```
plot(iris$Sepal.Length, iris$Sepal.Width)
```

```
plot(iris$Sepal.Length, iris$Sepal.Width, type= "n")
points(iris$Sepal.Length[iris$Species=="setosa"],
iris$Sepal.Width[iris$Species=="setosa"],col="red", pch=16)
points(iris$Sepal.Length[iris$Species=="versicolor"],
iris$Sepal.Width[iris$Species=="versicolor"],col="green", pch=16)
points(iris$Sepal.Length[iris$Species=="virginica"],
iris$Sepal.Width[iris$Species=="virginica"],col="blue", pch=16)
```

Aggiungere i punti di una specie per volta

```
# metodo alternativo più rapido
mycol <- c("red", "green", "blue")
names(mycol) <- levels(iris$Species)
plot(iris$Sepal.Length, iris$Sepal.Width,
col=mycol[as.character(iris$Species)], pch=16)
```

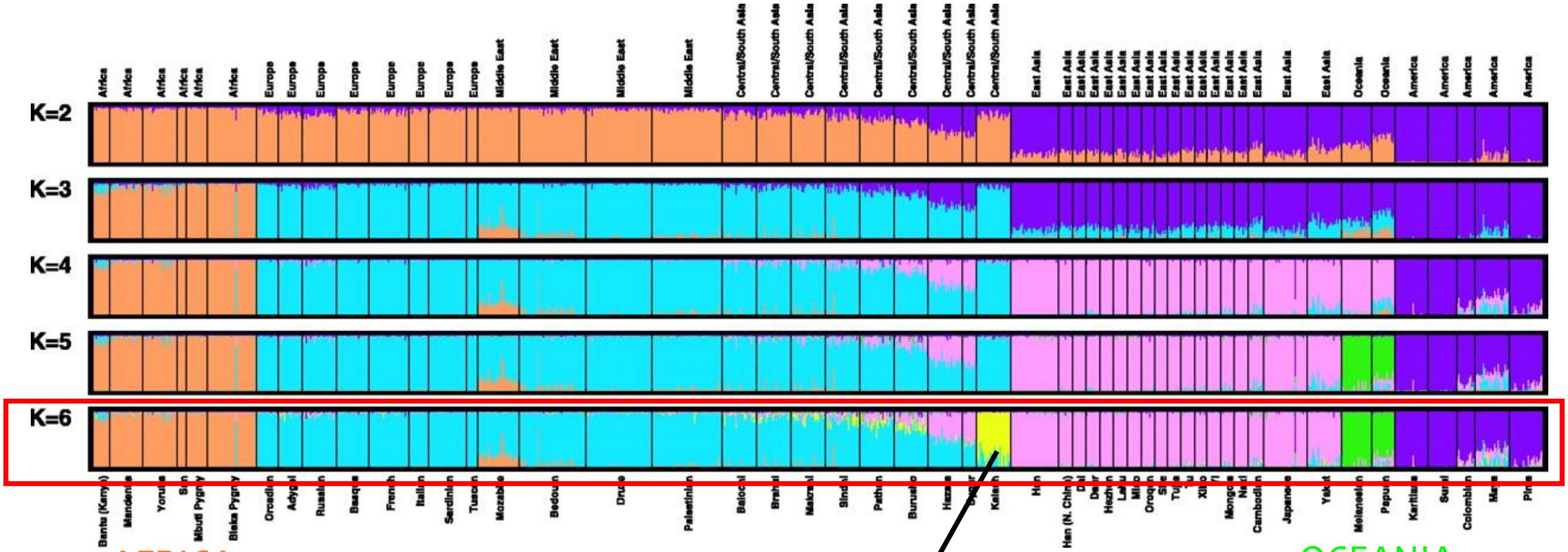


QUALCHE ALTRO GRAFICO DI ESEMPIO

** istogrammi, grafici a torta e mappe **

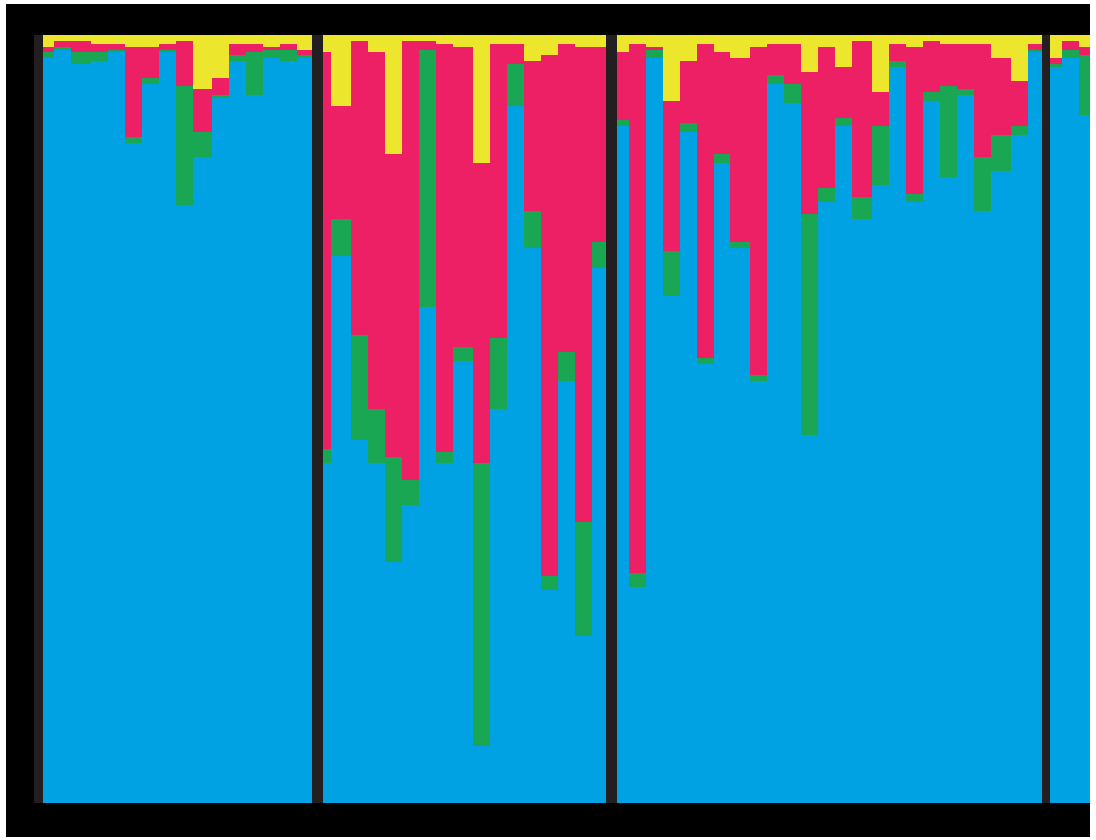
Rosenberg et al. (2002) - pubblicato su Science:

- ✓ investigano la struttura genetica di popolazioni umane per confermare se la suddivisione classica su base geografica e/o culturale abbia effettivamente un substrato e un riscontro genetico;
- ✓ 52 popolazioni, 1056 individui, 377 loci microsatelliti (multilocus genotypes);
- ✓ senza usare nessuna informazione a priori sull'origine degli individui, cercano il numero più probabile di cluster genetici (« in cui suddividere gli individui, sfruttando un algoritmo bayesiano implementato nel software STRUCTURE;
- ✓ per ogni individuo è calcolata la probabilità di appartenere a ciascun dei k cluster (q_1, …, q_k); le probabilità devono sommare a 1.



Kalash: vivono in NW Pakistan
e parlano una lingua indo-europea

Rosenberg et al. (2002)

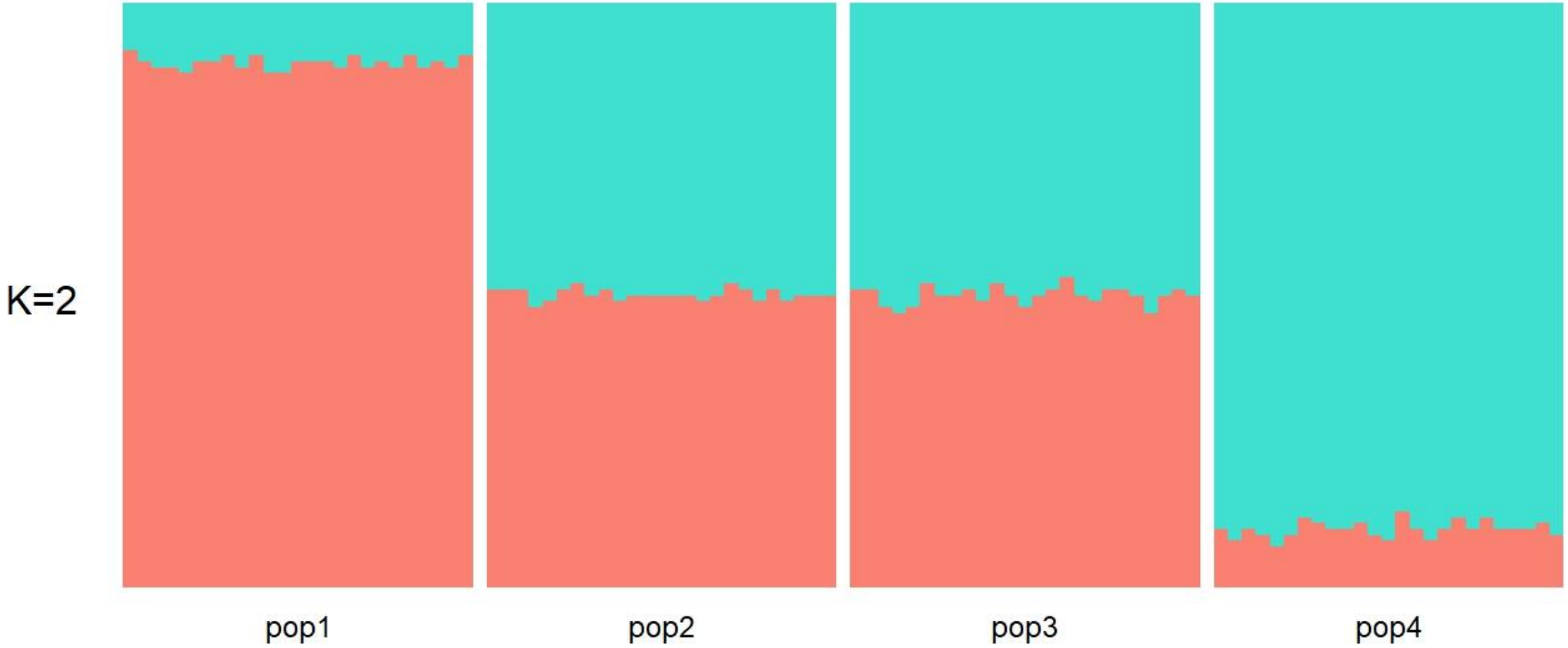


Ogni barra rappresenta un individuo.
Per ciascun individuo:

- **q_1: probabilità di appartenere al cluster 1**
- **q_2: probabilità di appartenere al cluster 2**
- **q_3: probabilità di appartenere al cluster 3**
- **q_4: probabilità di appartenere al cluster 4**

```
df <- read.table("qind.txt", h=T)

head(df)
   ind  pop   q1   q2
1 ind1 pop1 0.91 0.09
2 ind2 pop1 0.91 0.09
3 ind3 pop1 0.90 0.10
4 ind4 pop1 0.89 0.11
5 ind5 pop1 0.90 0.10
6 ind6 pop1 0.90 0.10
```

`df$ind` → 100 individui
`df$pop` → 4 popolazioni diverse
`df$q1` e `df$q2` → probabilità di appartenenza a 2 cluster genetici


K=2
pop1
pop2
pop3
pop4

1. BARPLOT: OGNI BARRA È UN INDIVIDUO

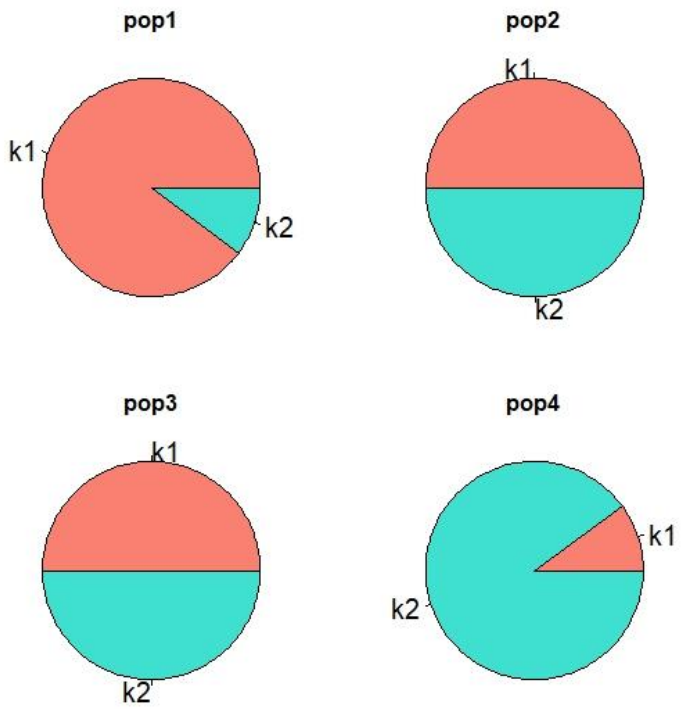
```
# creo un vettore di due colori, uno per ciascun cluster
mycol <- c("salmon", "turquoise")

# separo le quattro popolazioni con una riga vuota di NA
df <- rbind(df[1:25,], NA, df[26:50,], NA, df[51:75,],
NA, df[76:100,])

# isolo la porzione di dataframe di interesse
n <- t(as.matrix(df[,3:4]))
```

1. BARPLOT: OGNI BARRA È UN INDIVIDUO - 2

```
# creo il barplot
pippo <- barplot(n, col=mycol, border=NA, space=0, axes=F)

# indico il numero di cluster k
mtext(2, line=1, text="K=2", cex=2, adj=0.5, las=1)

# posiziono le etichette sotto a ciascuna delle 4 popolazioni
df$pos <- pippo
mtext(1, line=1, at=tapply(df$pos, df$pop, mean),
      text=na.omit(unique(df$pop)), cex=1.5)
```

2. GRAFICO A TORTA: q medio di ciascuna popolazione

```
# calcolo il coefficiente medio di appartenenza di tutte e 4 le popolazioni
q1med <- tapply(df$q1, df$pop, mean)
q2med <- tapply(df$q2, df$pop, mean)
qmed <- rbind(q1med, q2med)

        pop1   pop2 pop3   pop4
q1med 0.8972 0.5016  0.5 0.0996
q2med 0.1028 0.4984  0.5 0.9004

# grafico a torta classico
par(mfrow=c(2,2), mar=c(2,2,2,2))
for(i in 1:4)
     pie(qmed[,i], labels=c("k1", "k2"),
     col=mycol, main=colnames(qmed)[i], cex=1.5)
```



3. GRAFICI A TORTA E MAPPE

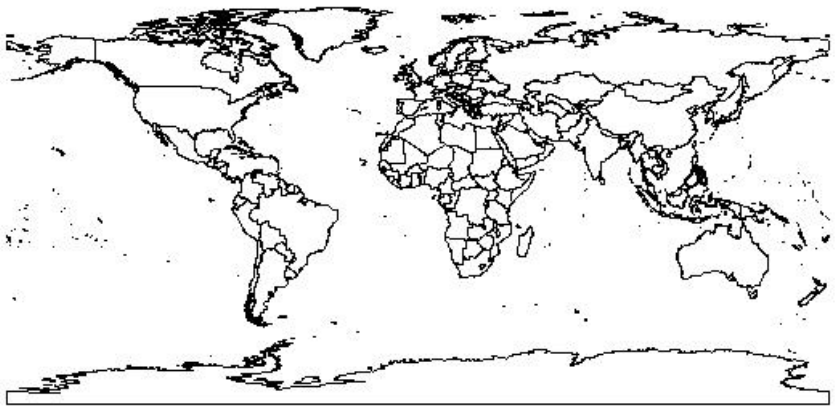
usare `draw.pie` ci permette di essere molto più flessibili e di posizionare i grafici a torta dove ci interessa, per esempio su una cartina

```
#  installare i pacchetti necessari
install.packages("mapplots")
library(mapplots)

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

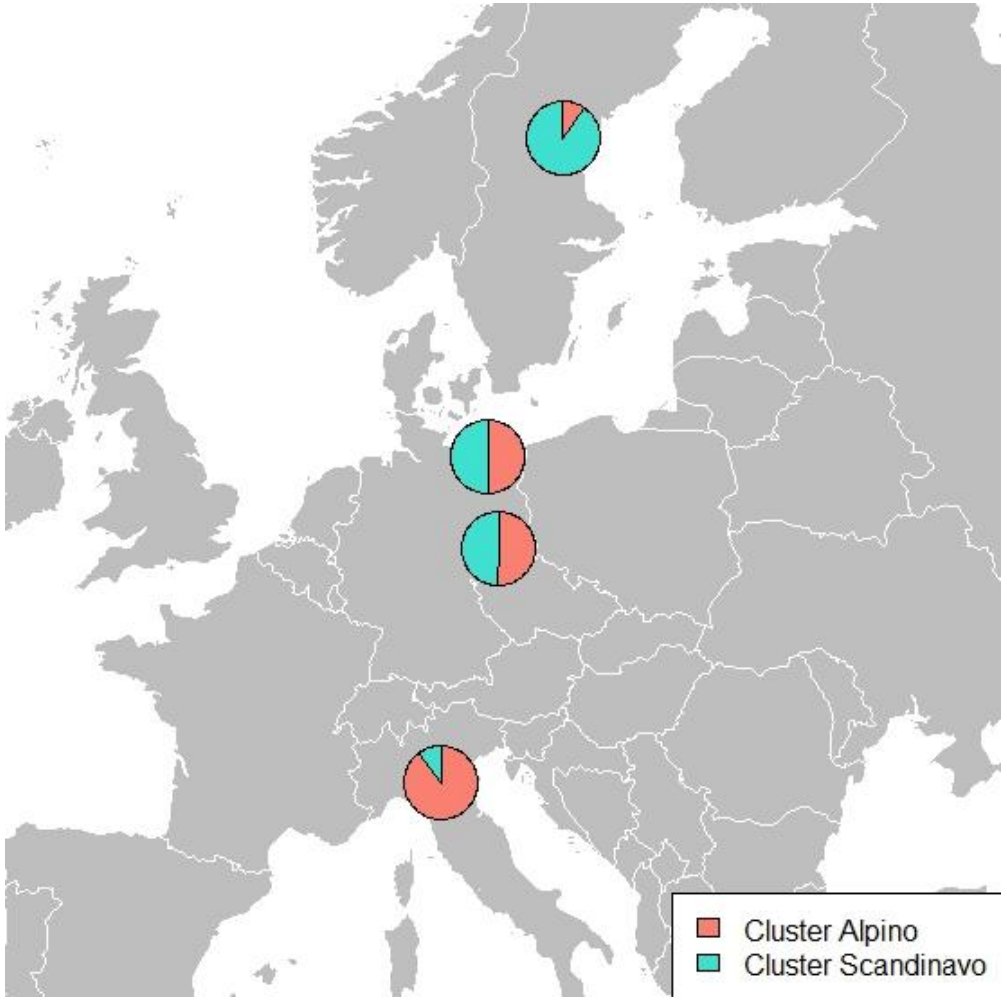
3. GRAFICI A TORTA E MAPPE

```
draw.pie(x, y, z, radius, …)
```

- ✓ x e y # posizione del grafico a torta
- ✓ z # array in cui si trovano le categorie da rappresentare
- ✓ radius # raggio del grafico a torta



```
newmap <- getMap(resolution="low")

# plotto il mondo intero
plot(newmap)

# per plottare solo una porzione di mondo lavoro con xlim e ylim
plot(newmap, xlim=c(7,20), ylim=c(40,65), col="gray",
border="white", lwd=1.5, las=1)
```

3. GRAFICI A TORTA E MAPPE - 2

```
# ottengo le coordinate delle popolazioni
punti <- locator()
points(punti$x, punti$y)

# plotto i grafici a torta
draw.pie(punti$x, punti$y,
t(qmed), radius=1, col=mycol)

# aggiungo una legenda
legend("bottomright", fill=mycol,
legend=c("Cluster Alpino",
"Cluster Scandinavo"), bg="white")
```