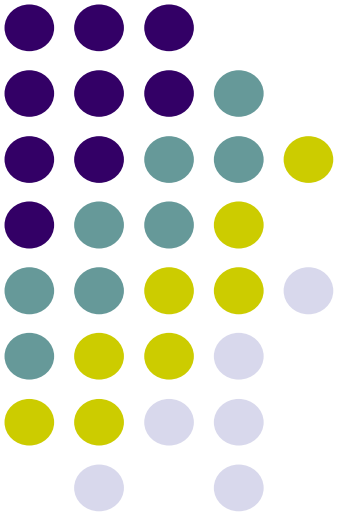
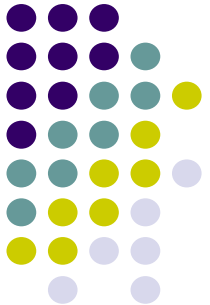
Linguaggio R

Raffaele Miele
Dipartimento di Matematica e Statistica
Università degli Studi di Napoli Federico II
rafmiele@unina.it
http://wpage.unina.it/rafmiele



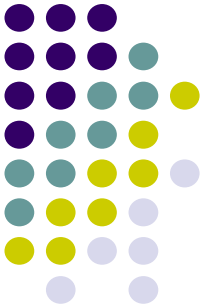
Perché R?



E' un linguaggio di scripting

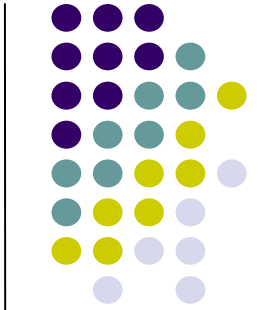
- Non necessita di compilazione
- Adatto ad uso interattivo
- Funzioni ad alto livello
- Multipiattaforma
- Gratuito (derivato da S-Plus)
- Open source
- Specializzato per uso statistico
- Pensato per descrivere modelli anche molto complessi
- E’ un linguaggio object-oriented (può facilmente essere esteso dall’utente)

Pro e contro di R



- Alto livello implica:
  - Funzioni potenti, ma solo per quello che R “sa fare”.
  - Poco potente e flessibile per il resto (meglio linguaggi di programmazione come C++, Java, etc.).
  - Ottimo per applicazioni statistiche di piccole e medie dimensioni.

Open Source

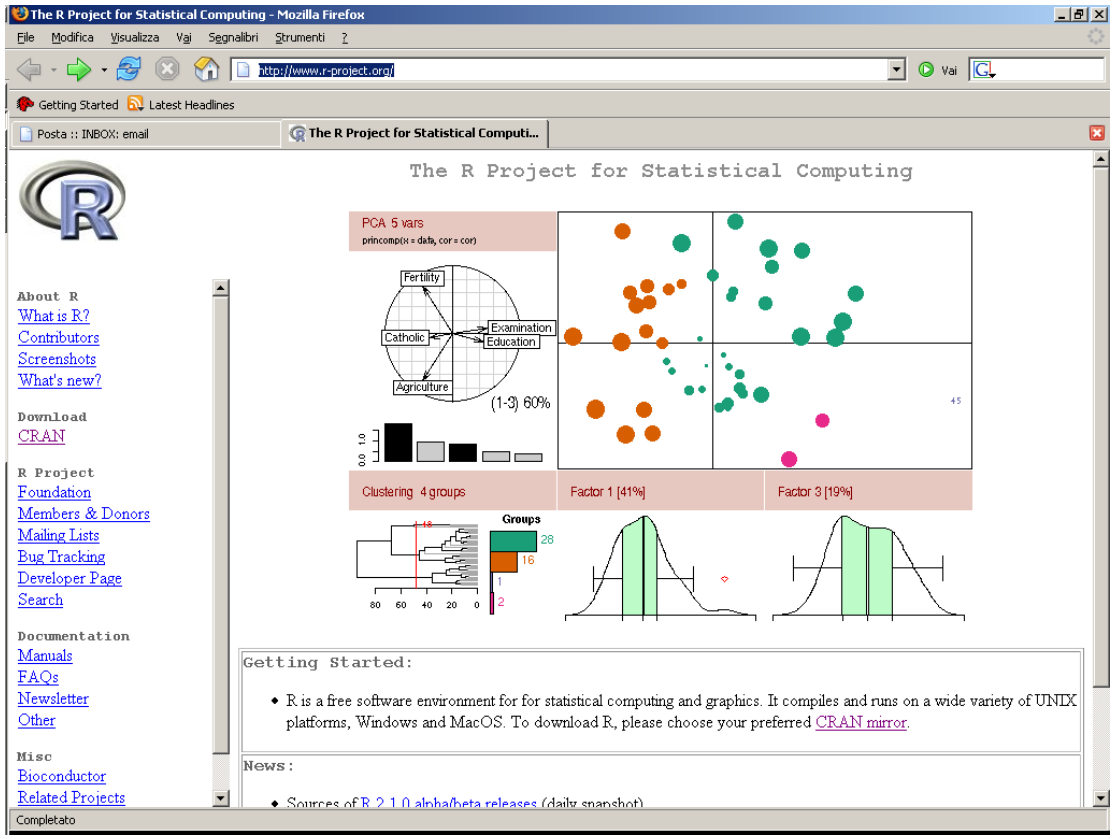


- La licenza open source prevede che il software venga fornito sia nel formato compilato (eseguibile) che in quello sorgente.

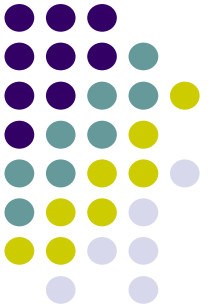
- Sul sorgente può lavorare chiunque sia disponibile a farlo, a titolo gratuito. Su di esso chiunque può eseguire delle modifiche e, nel caso lo faccia, è tenuto a distribuire il nuovo sorgente insieme con il programma nella forma eseguibile.

Ottenere R

- Sito di riferimento : http://www.r-project.org/
- Questo sito è una vera e propria “miniera” di informazioni e materiale su R.
- Da questo sito è possibile
  - Scegliere un mirror da cui scaricare R (disponibile per diverse piattaforme)
  - Scaricare documentazione.
  - Scaricare componenti da integrare con R

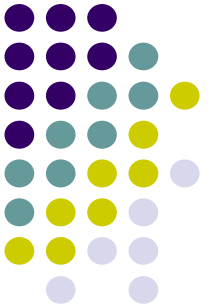


Storia di R



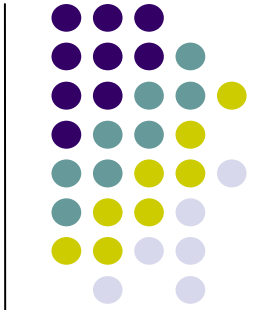
- Dagli anni settanta sono state sviluppate tecniche statistiche che richiedono un notevole supporto computazionale al fine di essere fruibili.
- Negli anni novanta i Bell Laboratories hanno deciso di sviluppare un nuovo ambiente per l'analisi statistica in grado di permettere, oltre l'applicazione delle metodologie conosciute, anche la sperimentazione di nuovi modelli ed idee statistiche. Nasce quindi il linguaggio S

Storia di R



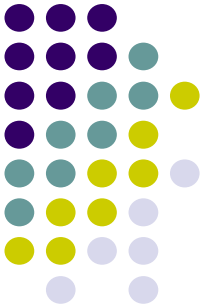
- Il linguaggio S venne poi concesso alla Insightful Corporation che ne ha fatto un software di successo noto come S-Plus.
- Di recente, grazie all'avvento di Internet, è nato il progetto open source R.
- R ed S-Plus sono altamente compatibili.
  - S-Plus è usato maggiormente nelle aziende
  - R è usato nelle università e nei centri di ricerca

L’interfaccia di R



- R è stato pensato essere utilizzato tramite riga di comando
- Esistono anche interfacce grafiche, ma sono limitate alle semplici operazioni ricorrenti (leggere files, etc.)
- Sotto windows usiamo la R-console
- Sotto unix usiamo la shell.

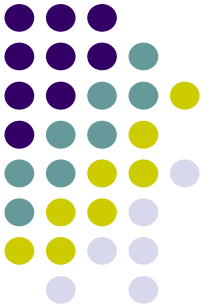
Avviamo R…



…e siamo a contatto con la console.

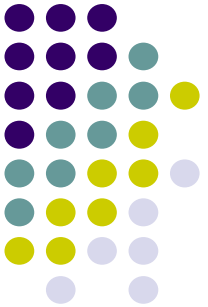
- Tutti i comandi vanno inseriti dopo il prompt
- R ci ricorda che possiamo uscire tramite il comando q()

Workspace



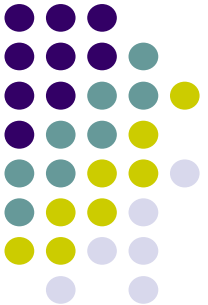
- R salva tutte le variabili frutto di elaborazione in un'area di memoria nominata, per l'utente, workspace.
- E' possibile salvare il workspace all'uscita di R in maniera tale da poter continuare il lavoro da dove l'avevamo lasciato. I dati vengono salvati nei file .RData e .RHistory
- Con il comando ls() visualizziamo il contenuto del workspace.
- Con ls.str() abbiamo più dettagli.

Working directory



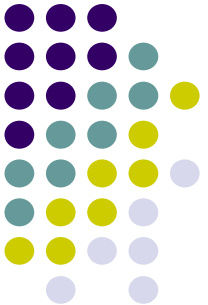
- R, per default, salva i dati nella directory corrente (la directory da cui è stato avviato).
- Tramite il comando setwd(nuovaDirectory) possiamo cambiare tale directory.
- Tramite il comando getwd() visualizziamo tale directory.
- Con il comando dir() visualizziamo il contenuto della working directory.
- E' una buona cosa creare directory diverse per diversi progetti.

Basi del linguaggio



- R è case sensitive (Pippo ≠ pippo)
- Non è necessario dichiarare una variabile
- Operatori aritmetici usuali: +,-,*,/
- Operatori logici: ==, >=, <=, !=
- Parentesi graffe per raggruppare istruzioni
- ; per separare le istruzioni

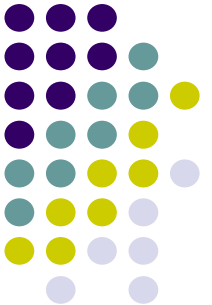
Help di R



L'help di R è molto potente

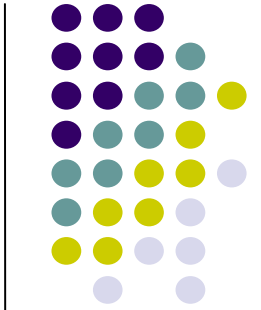
- help() - indicazioni su come usare l'help
- help(comando) – oppure ?comando
- help ≠ help() – usare le parentesi!!!
- example(comando) – provare per credere!
- demo(comando) – idem come sopra.
- Per gli operatori e le parole chiave dei costrutti di programmazione è necessario mettere l'argomento tra apici. es. help("+") oppure help("for")

Help di R



- Con il comando help.search(“exp”) è possibile cercare nell’help gli argomenti che soddisfano l’espressione regolare *exp* fornita al comando.
- Il comando apropos(“exp”) visualizza i comandi il cui nome soddisfa *exp*
- Guardate cosa fa R se digitiamo *help.start()…*

Output in un file



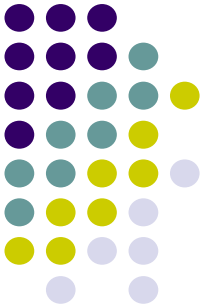
- Con il comando

sink("out.txt")

- si dirotta l'output dei comandi R che digitiamo nel file out.txt
- Per tornare a vedere l'output in console digitiamo

sink()

Basi del linguaggio

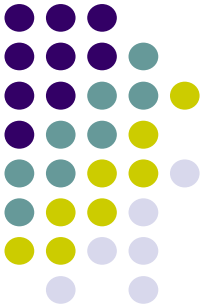


Sintassi di un comando R:

Variabile/oggetto <- comando(par1, par2, …)

- Il simbolo <- è usato al posto dell'uguale (=)
- R supporta il segno = ma ne sconsiglia l'uso
- E' disponibile anche il comando assign (vedi help(assign)). E' utile negli script per le assegnazioni multiple.
- Se non specifichiamo la variabile destinazione il risultato viene tenuto nella variabile *.Last.value*
- Premendo i tasti *freccia su* e *freccia giu* possiamo navigare nella *command history* (la lista dei comandi eseguiti precedentemente)

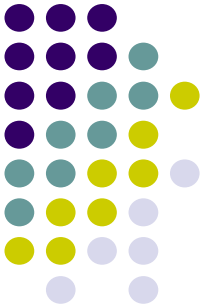
Parametri



- E’ possibile passare i parametri alle funzioni R nell’ordine prestabilito oppure specificando vari nomi. Esempio:
- a <- matrix(1:3)
- a <- matrix(data=1:3)

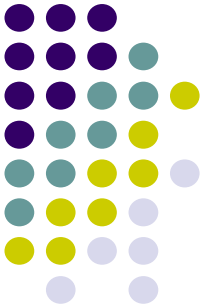
I due comandi sono equivalenti

Pulizia



- Ogni assegnazione in R sovrascrive il contenuto della variabile di destinazione.
- Gli oggetti possono essere rimossi con il comando rm(oggettoDaCancellare)
- rimozioni multiple rm(pippo,pluto, x1)
- con rm(list=ls()) svuotiamo il workspace

Interagire con il sistema operativo



- system(‘applicazione’) avvia un’applicazione
  - es: system(“notepad”)
- Per eseguire script R si usa il comando source
  - source(“nomeScript”)
  - es: source(“fwdTree.R”)

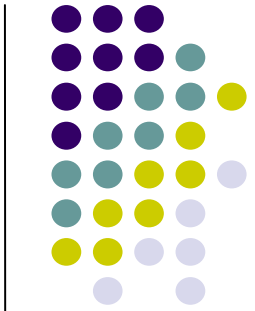
Variabili

```
> a <- 49
> sqrt(a)
[1] 7
```

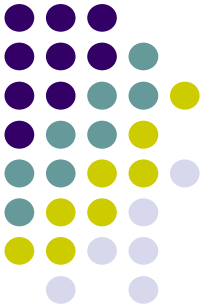
Numeriche

```
> b <- "The dog ate
my homework"
> sub("dog","cat",b)
[1] "The cat ate my
homework"
```

Stringhe

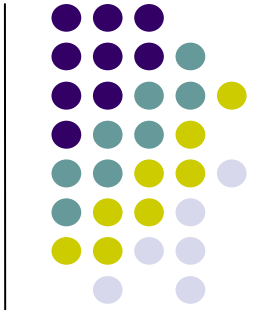


Tipi booleani



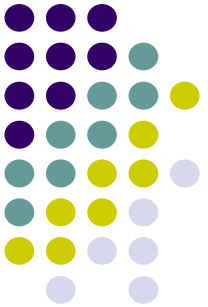
- Variabili che possono avere solo due valori.
- In R i valori si indicano con :
  - TRUE oppure T
  - FALSE oppure F
- Il comando matrix accetta un parametro booleano……

Missing values



- R gestisce i valori mancanti.
- Simbolo: NA (not assigned)
  - NA non equivale a 0
  - NA non equivale a “”
  - Na non equivale a TRUE o FALSE

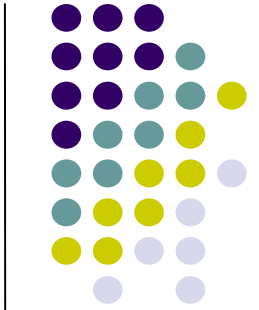
Vettori e variabili



R permette di lavorare con dati strutturati. I tipi più semplici sono gli scalari e i vettori.

- x <- 4; # per assegnare uno scalare a x
- x <- c(1,2,3,4,5,6); # per assegnare un vettore
- tutto quello che c'è dopo il segno # è visto come commento
- Il comando c() concatena gli elementi forniti come parametri

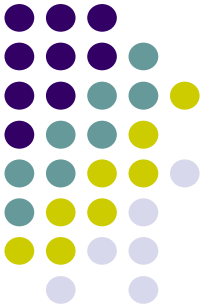
Matrici



Situazione:

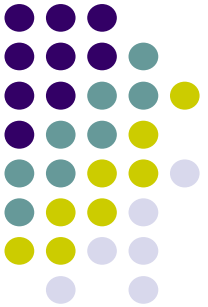
- x <- c(2,4);
- y <- c(2,4);
- Se digitiamo x * y
- otteniamo 4 16

Matrici



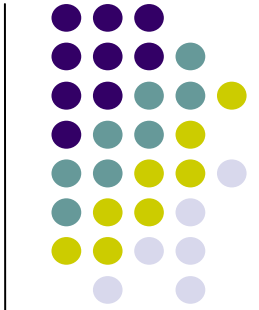
- Se vogliamo usare l'aritmetica dell'algebra lineare dobbiamo usare l'operatore %*%
- R interpreta i vettori come vettori colonna
- La trasposizione di un vettore (o di una matrice) si ottiene tramite il comando t()
- Quindi t(y) %*% y mi restituisce 20
- Per convenzione l'operazione y %*% y equivale a t(y) %*% y

Comando Matrix



- Per creare una matrice ex-novo usiamo il comando matrix(data, nrow, ncol, byrow)
- Consultare l'help…….

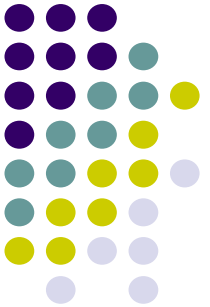
Matrici



Situazione:

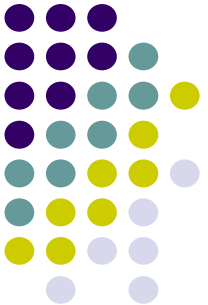
- x <- matrix(c(1,2,3,4),1,4);
- y <- matrix(c(1,2,3,4),4,1);
- Se faccio x * y R mi restituisce un errore.
- Ragione: x e y sono due vettori, quindi non è possibile usare gli operatori che eseguono operazioni termine a termine.
- Usare x %*% y
- Provare help(“+”);

Matrici



- Con il comando *diag(matrice)* otteniamo un vettore con la diagonale della matrice
- Se diamo un numero *k* in pasto al comando diag questo ci restituisce la matrice identità di ordine *k*
- i comandi *lower.tri* e *upper.tri* permettono di estrarre la triangolare inferiore e superiore. Consultare l'help per capire come funzionano…

Matrici



- Per calcolare l'inversa di una matrice

B <- solve(A)

provare

A %% B*

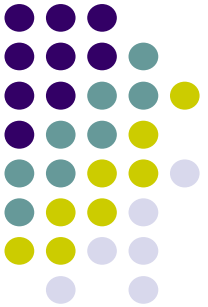
- Per calcolare il determinante di una matrice

det(A)

Operatori più uasti

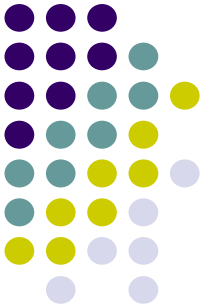
<-	Assign
+	Sum
-	Difference
*	Multiplication
/	Division
^	Exponent
%%	Mod
%*%	Dot product (prodotto matriciale)
%/%	Integer division (restituisce valore intero)

Vettori di indici



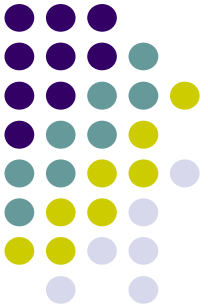
- Serviranno ad accedere agli elementi di altri vettori.
- x <- 1:5;
- oppure l'equivalente seq(1,5);
- generano sequenze di numeri
- è possibile specificare anche il *passo*
- *help(seq)* per i dettagli……
- cercare anche in *help.start()*
- anche il comando *rep()* è interessante

Vettori e matrici



- Creare un vettore sequenza
- sequenza <- -3:8;
- e carichiamone i valori in una matrice
- A <- matrix(sequenza, 2,6);
- Per accedere agli elementi di un vettore (o una matrice) sequenza[3]
- oppure A[2,3]

Vettori e matrici



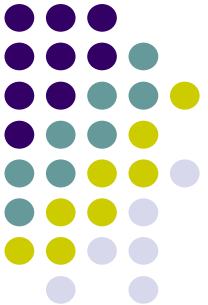
- Per estrarre una riga da una matrice

```
A[1,] #estrae la prima riga
```

- Per estrarre una colonna

```
A[,2] #estrae la seconda colonna
```

Which



Il comando which estrae gli indici di tutti gli elementi di un vettore (o di una matrice) che soddisfano una condizione.

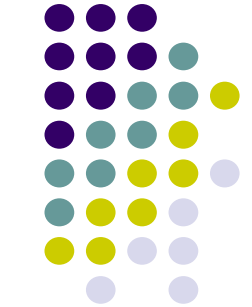
which(condizione)

Es. a <- which(x < 2)

restituisce le posizioni di tutti gli elementi del vettore x di valore inferiore a 2

Che succede se digito x[which(x < 2)] ?

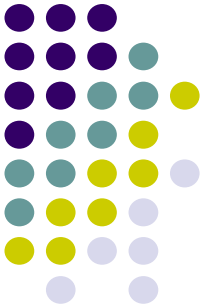
Operatori per le condizioni



|	Or
&	And
<	Less
>	Greater
<=	Less or =
>=	Greater or =
!	Not
!=	Not equal
==	Is equal

Provare a scrivere condizioni più complesse……

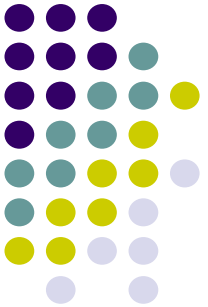
Salvare il contenuto del workspace



Usiamo il comando save

- *save.image()* salva tutto il contenuto del workspace in .RData nellc directory corrente
- *save(x, A, file="prova.R")* memorizza nel file prova.R il contenuto delle variabili x e A
- *save(list=ls(), file="prova.R")* cosa fa?
- *load("prova.R")* carica le variabili in memoria

Oggetti e tipologie di dati



- Ogni cosa in R èun oggetto ed ogni oggetto appartiene ad una classe
- Nomi degli oggetti: sequenze di lettere e numeri
- Un nome non può iniziare con un numero
- ……..

Liste

- E’ un vettore (*contenitore)* di oggetti

esempio
nomi <- c(“pietro”, “francesco”, “walter”); # nomi
x <- c(7,3,29); # numero esami
bool <- c(T,F,F); # frequenta
con il comando

lista <- list(nomi, x, bool)

carichiamo il tutto nella lista

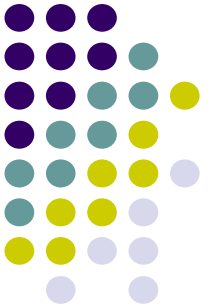
- per accedere agli elementi della lista (es. vettore x)

lista$x

- E’ anche possibile usare la seguente sintassi

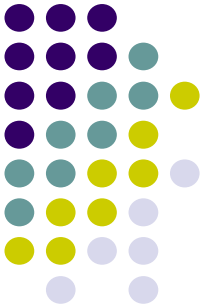
lista[[1]]# primo elemento

Di che tipo sono i miei dati?



- str(oggetto) – informazioni sulla struttura
- mode(oggetto) – tipo di oggetto
- names(lista) – nomi degli oggetti contenuti in lista
- R offre anche svariate funzioni che indicano se una variabile contiene un oggetto di un determinato tipo. (es: is.matrix(), is.numeric(), is.list()). Consultiamo l'help.

Factor



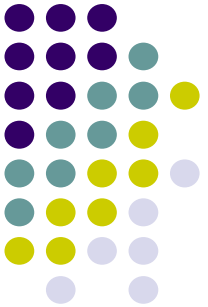
- I dati qualitativi in R sono gestiti da oggetti chiamati *factor*
- l'attributo *order* indica se I dati sono su scala ordinale

es.

eta <- c("giovane", "anziano", "adulto", "adulto", "adulto");

- facEta <- factor (eta);

Ordered factor

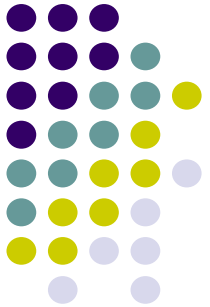


- *ordered*(facEta, levels= levels=c(“giovane”, “adulto”, “anziano”);

oppure

- facEta <- factor(eta, levels=c(“giovane”, “adulto”, “anziano”));

Ricodificare le modalità di un factor



genere <- c(1,2,2,2,2,1,1,2,2,2)

facGenere <- factor(genere)

se digitiamo:

facGenere

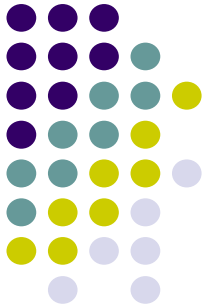
ed R visualizza i livelli

Levels: 1 2

possiamo ricodificarli con

levels(facGenere) <- c(“maschio”, “femmina”);

Dataframe



- E’ l’oggetto più utilizzato, sotto R, per indicare la matrice dei dati.

Immaginiamo di avere 3 variabili:

- nome di tipo factor
- altezza di tipo numeric
- colore.occhi di tipo factor

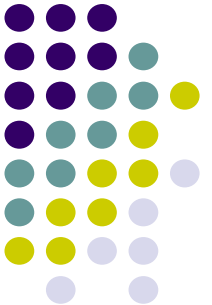
Con il comando

dati <- data.frame(Nome = n, Altezza = a, Colore.occhi= c);

Carichiamo i dati in un dataframe

N.B. le variabili caricate nel dataframe devono avere lo stesso numero di elementi!!!

Dataframe

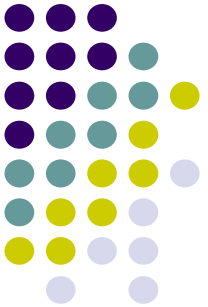


- Per accedere agli oggetti di un dataframe digitiamo:

dati$Nome

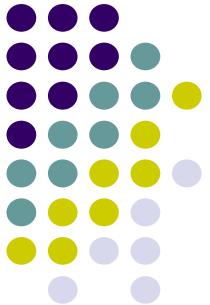
- con il comando attach(dati) possiamo evitare di dover scrivere dati&… per accedere alle variabili del dataframe
- Con il comando detach(dati) ripristiniamo la situazione iniziale

Importare i dati da file di testo



- R usa il comando scan, che però ha una sintassi piuttosto complessa.
- Il comando read.table(data="dati.txt", header=TRUE, sep=",", row.names=7)
  - legge i dati dal file dati.txt
  - estrae le etichette delle variabili dalla prima riga
  - usa il carattere "," per separare I campi
  - usa la colonna 7 come etichette delle righe
- un separatore molto diffuso è la tabulazione, che si indica con "\t"
- Esiste anche il comando *write.table()*

Formato fisso

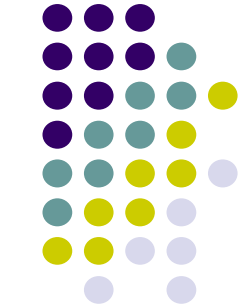


Istruzione

read.fwf(file=“dati.txt”, width=c(3,3,1,2), sep=“\n”);

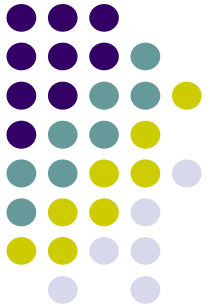
- legge dal file dati.txt
- dimensione dei campi indicata dal parametro *width*
- separatore di riga “\n” (a capo)

Installare ed utilizzare nuove librerie



- Dal sito del CRAN (Comprehensive R archive network) è possibile scaricare una grande quantità di “package” R.
- Un package è una raccolta di funzioni per R.
- Dal sito del CRAN si possono scaricare I packages “ufficiali”, è possibile trovare sul web moltissimi altri packages scritti da ricercatori.
- Il sito www.bioconductor.org , ad esempio, contiene software R per l’analisi del genoma umano.

Installare packages



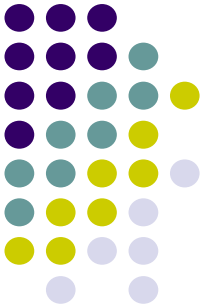
- Dall'interfaccia grafica di R: menu Packages – install packages……
  - è possibile installare direttamente dal sito del CRAN oppure da files zip in locale.
- Una volta installato il package lo si deve richiamare tramite il comando library.

es.

library(forward)

Richiama il package per l'analisi robusta tramite la Forward Search

Comando data



- I package R sono spesso corredati di dataset.
- Per ottenere una lista dei dataset inclusi in un package digitiamo:

  data(package="nomePackage");

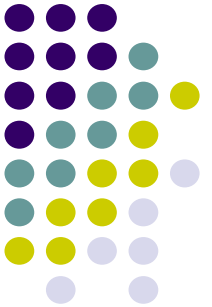
- Per elencare i inclusi in tutti I packages in memoria digitiamo

  data()

- Per caricare in memoria un dataset (dopo aver caricato il package corrispondente) digitiamo:

  data(nomeDataset)

Distribuzioni di frequenza

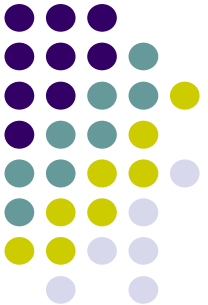


- Carichiamo il dataset “iris” dal package forward
- Con il comando table otteniamo la distribuzione di frequenza

  table(iris$Species)
- per avere le frequenze relative digitiamo:

  table(iris$Species)/length(iris$Species)
- Che fa il comando cumsum?

Variabili quantitative



- Con il comando cut raggruppiamo le unità di una variabile quantitativa. Con

range(iris[[1]])

otteniamo il campo di variazione della prima variabile del dataset iris. Poi con

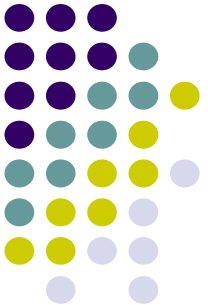
cut(iris[[1]], breaks=c(4.3,6,7.9))

- Dividiamo le unità. Osserviamo il risultato….
- Proviamo anche

table(cut(iris[[1]], breaks=c(4.3,6,7.9)))

- Vedere anche il comando hist()

Produrre grafici

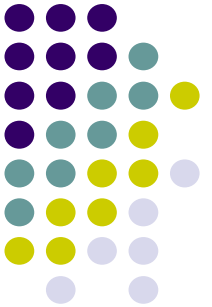


R fornisce un vasto corredo di funzioni per produrre grafici.

esempi:

- *pie(table(iris[[5]]))* # grafico a torta
- *hist(iris[[1]])* # per variabili quantitative
- plot(x<-seq(-100,100), x^2)
- Molti package R implementano dei propri metodi plot per visualizzare l'output delle analisi.

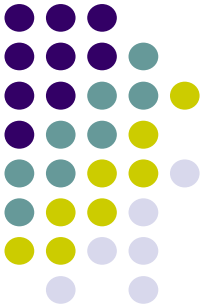
Funzioni utilizzate di frequente



`c`	Concatenate
`cbind, rbind`	Concatenate vectors
`min`	Minimum
`max`	Maximum
`length`	# values
`dim`	# rows, cols
`floor`	Max integer in
`which`	TRUE indices
`table`	Counts

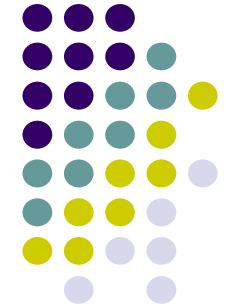
`summary`	Generic stats
`Sort, order, rank`	Sort, order, rank a vector
`print`	Show value
`cat`	Print as char
`paste`	`c()` as char
`round`	Round
`apply`	Repeat over rows, cols

Funzioni statistiche



`rnorm, dnorm, pnorm, qnorm`	Normal distribution random sample, density, cdf and quantiles
`lm, glm, anova`	Model fitting
`loess, lowess`	Smooth curve fitting
`sample`	Resampling (bootstrap, permutation)
`.Random.seed`	Random number generation
`mean, median`	Location statistics
`var, cor, cov, mad, range`	Scale statistics
`svd, qr, chol, eigen`	Linear algebra

Funzioni grafiche spesso utilizzate

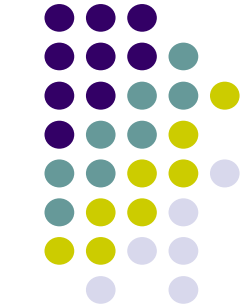


`plot`	Generic plot eg: scatter
`points`	Add points
`lines, abline`	Add lines
`text, mtext`	Add text
`legend`	Add a legend
`axis`	Add axes
`box`	Add box around all axes
`par`	Plotting parameters (lots!)
`colors, palette`	Use colors

Costrutto If

```
if (condizione) {
   statements
}
else {
   alternative statements
}
```

else è opzionale



Cicli

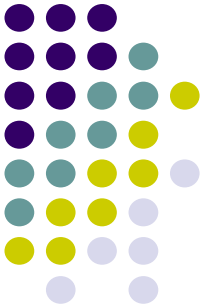
Utili per ripetere azioni simili (o la stessa azione) più volte. Es. tutti gli elementi di una lista o tutte le colonne di un array

```
for(i in 1:10) {
   print(i*i)
}

i<-1
while(i<=10) {
   print(i*i)
   i<-i+sqrt(i)
}
```

Vedi anche: `repeat, break, next`

Scrivere funzioni R



Scrivere le istruzioni in un file all'interno della seguente dichiarazione:

nomefunzione <- function(par1, par2, par3…) {

…istruzioni…

}

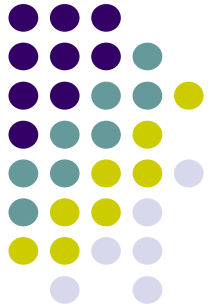
(par1, par2, etc. sono i parametri che la funzione accetta in input)

tale funzione sarà richiamabile utilizzando il comando

source("nomeFunzione");

n.b. il file deve essere raggiungibile (deve stare nella working directory oppure va specificato il percorso completo…attenti al separatore di directory….vedi manuale di R)

Scrivere funzioni R



esempio: copiare il seguente codice in un file di testo dal nome add.R:

```
add <- function(a,b) {
    result <- a+b
    return(result)
}
```

per caricare la funzione in memoria digitiamo

source("add.R")

Se digitiamo *ls()* vediamo, in memoria anche la funzione add. Con *str(add)* vediamo che è una funzione

per usare la funzione digitiamo

add(3,4)

ed ammiriamo l'output…….