- **INTRODUZIONE**

Questa è una dispensa "molto" sintetica dei comandi principali da utilizzare il R. Per una descrizione più approfondita delle funzioni caratterizzanti il software si consiglia la lettura di "***An Introduction to R***" scaricabile dal sito *http://cran.r-project.org/doc/manuals/R-intro.pdf*.

- **INIZIO DI UNA SESSIONE DI LAVORO E ALCUNI PRIMI PASSI ….**

Cliccando sull'icona di R si entra nell'ambiente di lavoro. Per prima cosa appare una scritta relativamente alla versione di R più altre informazioni.....

Poi il sistema si pone in attesa di ricevere ordini con il prompt::

>

Dopo il prompt è possibile in modo interattivo sottoporre comandi da eseguire. Ad esempio, si possono effettuare alcune operazioni elementari come la somma di due numeri:

```
> 3+7
[1] 10
```

La parentesi quadra col numero 1 sta ad indicare il risultato dell'operazione effettuata.

Se vogliamo "**assegnare**" il valore 10 ad un oggetto il cui nome è da noi indicato, si fa:

> test = 10 (oppure test <- 10) le due "assegnazioni" sono **equivalenti;**

L'oggetto denominato da noi 'test' contiene il valore 10. Infatti se scriviamo:

> test

il risultato sarà:

[1] 10

Il comando **objects()** restituisce una lista degli oggetti presenti nella sessione di lavoro. Nel nostro caso al momento abbiamo solo l'oggetto test.

Il comando **ls()** ha la stesso significato di objects().

Se invece si vogliono eliminare tutti gli oggetto che abbiamo creato bisogna digitare:

> **rm(list=ls(all=TRUE))**

- **ALTRI ESEMPI DI COMANDI IN R SONO:**

> y=**c**(1,2,3,4,5,6)	per creare un vettore coi primi 6 numeri naturali
> y=**rnorm**(6)	per creare un vettore con 6 valori estratti da una N(0,1)
> y=x+0.5*y	per trasformare y
> y(y>0)	se y è un vettore di numeri, prende quelli maggiori di 0
> **mean**(y)	calcola la media dei valori contenuti in y
> **var**(y)	calcola la varianza dei valori contenuti in y
> **plot**(x,y)	crea uno scatterplot con x in ascissa e y in ordinata.

I simboli per le operazioni sono:

+	*addizione*
-	*Segno negativo, sottrazione*
*	*Moltiplicazione*
^	*Elevazione a potenza*
/	*Divisione*

- **ALCUNI TASTI SPECIALI**

L'uso delle freccette in alto e basso ha le seguenti funzioni:

Freccia su: per scorrere indietro la lista dei comandi dati durante la sessione **Freccia giù**: per scorrere in avanti la lista dei comandi dati durante la sessione. Sono a disposizione di R numerose librerie. Alcune di esse sono già disponibili nel pacchetto di installazione e si possono interrogare dalla barra di menu con: PACCHETTI – CARICA PACCHETTO. Per richiamare gli oggetti contenuti nelle librerie si usa il comando:

> **library(nome libreria)**

All'interno delle librerie oltre che funzioni sono presenti anche dati sotto forma di data set di R. Per ottenere la lista completa dei data set presenti in R si digita:

> **data()**

Per caricare uno di questi data set basta digitare:

> **data(nome data set)**

- **ALCUNE FUNZIONI DALLA BARRA DI MENU DI R**

Il comando **File** consente di aprire e creare file con la storia di una sessione di lavoro.
Il comando **Edit (Modifica)** consente le consuete funzioni (copy, paste, select all, clear console).
Il menu **Misc (Varie)** ci fa la lista degli oggetti presenti nella sessione di lavoro, dei packages richiamati, oppure ci elimina tutti gli oggetti.
Il comando **Packages (Pacchetti)** merita qualche parola in più. Esso consente di installare packages che non sono presenti nel corredo standard di R (ad esempio, si possono scaricare dalla rete). In tal caso basta selezionare "*Install packages from local zip files*".
Il menu **Windows (Finestre)** consente di selezionare e di disporre come si preferisce (a cascata, accoppiate, ecc.) le finestre di R: quella grafica e quella della sessione di lavoro.
Infine, il menu **Help** contiene una serie di documenti per l'help on line.
Nella pagina dell'**Html Help** è possibile consultare una guida introduttiva di R, vedere i packages (librerie) che abbiamo a disposizione con tutti i loro contenuti e le sintassi dei comandi relativi. Cliccando su 'packages' troviamo, fra gli altri, il package 'base' che contiene le principali funzioni statistiche di base.
Un altro modo di interrogazione è il **Search help** che consente di cercare attraverso delle parole chiave.
Ad ogni modo anche durante la sessione di lavoro si può invocare l'help mediante la funzione:

> **help(*parola chiave)***

il risultato sarà la lista delle funzioni che hanno quella parola chiave e il relativo package di appartenenza.

oppure:

> **apropos("*nome argomento da cercare*")**

1. DATA OBJECTS

1.1 Tipi di dati

R lavora con numeri interi ('**integer'**) e non (tipo '**double**'), e anche con numeri complessi.
Qui di seguito si crea l'oggetto y a cui si assegna 8, e ci si chiede se è intero:

```
> y=8
> is.integer(8)
[1] FALSE
```

ma come si vede la risposta è FALSO perché per default R genera tutti numeri di tipo '**double**'. Allora si specifica che deve essere intero :

```
> y=as.integer(y)
> is.integer(y)
[1] TRUE
```

Abbiamo detto che R opera anche con numeri complessi (a noi non servirà questo, durante il corso, per fortuna !). Ad ogni modo, per curiosità, per generare un numero complesso si fa:

```
> z=as.complex(10+3i)
```

Per vederlo si digita z e si invia ottenendo la risposta 10+3i:

```
> z
[1] 10+3i
```

1.2 Espressioni logiche

I simboli da usare per formare espressioni logiche sono elencati nella tabella seguente

<	*Smaller than*
<=	*Smaller than or equal to*
>	*Larger than*
>=	*Larger than or equal to*
==	*Equal to*
!=	*Unequal to*
&	*AND*
|	*OR*
!	*NOT*

Le operazioni logiche danno una risposta del tipo TRUE/FALSE. Se però le operazioni logiche sono inserite in operazioni numeriche allora si generano valori 0/1 a seconda se l'evento è FALSE/TRUE. Vediamo i seguenti due esempi:

```
> x=9
> y=(3<x) & (x<=10)
> y
[1] TRUE

> y=1*( (3<x) & (x<=10))
> y
[1] 1
```

Un altro esempio è il seguente che usa un vettore di numeri generato dall'operatore **seq(k)** che genera una sequenza di numeri naturali da 1 a k compreso.

```
> x=seq(10)                    x è un vettore di termini 1,2,3,4,5,6,7,8,9,10

> sum(x<6)
[1] 5
```

Il comando sum(x<6) conta il numero di termini del vettore x che sono minori di 6.

1.3 Fattori (scale nominali)

Per fattore intendiamo una variabile con livelli che rappresentano modalità non ordinabili (es. sesso). Supponiamo di avere 5 individui: i primi tre maschi e le ultime due femmine. Vediamo un esempio di creazione di un oggetto '**factor**':

```
> sex=c("m", "m", "m", "f","f")
```

A questo punto chiediamo ad R che tipo di oggetto è sex con il seguente comando:

```
> is(sex)
[1] "character" "vector"
```

Ancora non abbiamo specificato che sex è un **factor**. Lo facciamo così:

```
> sex=factor(sex)
> is(sex)
[1] "factor" "oldClass"

> levels(sex)
[1] "f" "m"
```

Come si può capire, "f" e "m" sono le modalità (livelli) della variabili (fattore) sex.

1.4 Strutture di dati

E' molto importante entrare in questa logica di R perché alcune funzioni matematiche e statistiche già disponibili in R richiedono opportune strutture di dati per poter funzionare.

Le principali strutture dei dati sono le seguenti:

- vettore
- matrice
- array
- data frame
- list

Vettore

E' la più semplice struttura in R. Esso consiste in un certo numero di elementi dello stesso tipo (es. logici, numerici interi, numerici 'double', ecc.). Esempio di vettore con 9 elementi di tipo 'double':

```
> x=c(1,2,3,4,5,6,7,8,9)
```

I vettori si possono concatenare. Esempio:

```
> y=c(x,x)
>y
[1] 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9
```

Un altro modo di generare un vettore (oltre alla funzione c()) è il comando **seq** già introdotto precedentemente.

NOTA : **Tutte le volte che vogliamo conoscere il significato di una funzione e la sua sintassi si può semplicemente digitare nella consol di R il ? seguito dalla funzione interessata. Ad esempio:**

```
> ?seq
```

In questo modo si apre una finestra corrispondente all'help della funzione seq nella quale viene spiegato la sintassi da specificare all'interno delle parentesi tonde.
Ad esempio in questo caso bisogna specificare l'inizio della sequenza, la fine ed il passo:

```
> x=seq(1,12,2)
[1] 1 3 5 7 9 11
```

Un altro modo alternativo di usare **seq()** è: x= **seq** (from=k, to=k1, by=k2). Ad esempio:

```
> x= seq (1,10,0.5)
> x
[1]   1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0
[16] 8.5 9.0 9.5 10.0
```

Matrici

Per generare una matrice si può usare la funzione **dim()**.
Ad esempio creiamo un vettore di elementi da 1 ad 8 con la funzione seq:

```
> x=seq(8)
> x
 [1] 1 2 3 4 5 6 7 8
```

Poi assegnamo a questa sequenza di elementi da 1 ad 8 una matrice di due righe e quattro colonne:

```
> dim(x)=c(2,4)
> x
     [,1] [,2] [,3] [,4]
[1,]   1    3    5    7
[2,]   2    4    6    8
```

In generale una matrice si crea utilizziamo il comando **matrix():**

```
> x=seq(8)
```

```
> x=matrix(x,2,4)
> x
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
```

Se si vuole l'ordinamento dei valori del vettore x per riga:

```
> x=matrix(x,2,4, byrow=TRUE)
> x
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
```

Una matrice può essere anche composta con vettori riga o vettori colonna della stessa dimensione. Se si compone per righe si usa la funzione **rbind(),** se si compone per colonne si utilizza **cbind()**.

Generiamo due vettori x ed y:

```
> x=seq(8)
> x
[1] 1 2 3 4 5 6 7 8
```

Il vettore y lo creiamo in quest'altro modo (**CHE VA COMUNQUE BENE!!!!**):

```
> y=(5:12)
> y
[1] 5 6 7 8 9 10 11 12
```

e poi costruiamo la matrice che si compone di questi due vettori. La chiamiamo MM:

```
> MM=cbind(x,y)
> MM                                   Si digiti MM e si veda il risultato in R
```

Per sapere quante righe e colonne ha:

```
> dim(MM)
[1] 8 2
```

Vediamo ora alcune operazioni sulle matrici. Sia aa una matrice qualsiasi:

```
> aa*aa                                prodotto elemento per elemento

> aa^2                                 potenza elemento per elemento

> max(aa)                              valore massimo dei termini in aa

> min(aaa)                             valore minimo dei termini in aa
```

Per effettuare l'operazione moltiplicazione matrice per vettore si utilizza il simbolo * che fa il prodotto degli elementi omologhi.

Il **prodotto matriciale** si fa con l'operatore %*%. Le dimensioni delle due matrici che si moltiplicano devono essere conformi.

La trasposta di una matrice si fa con **t()**. Esempio riprendiamo la matrice MM precedentemente creata:

```
> tras.MM=t(MM)
> tras.MM                                   Si veda il risultato ottenuto in R
```

Per selezionare un elemento della matrice, ad esempio di riga 2 e colonna 1:

```
> MM[2,1]
> [1] 2
```

Per selezionare tutta una riga della matrice, ad esempio la 3:

```
> MM[3,]
x y
3 7
```

mentre per selezionare tutta una colonna, ad esempio la 1:

```
> MM[,1]
[1] 1 2 3 4 5 6 7 8
```

Alcune operazioni su matrice sono nella tabella seguente. Sia aa una matrice qualsiasi:

ncol(aa)	*Numero di colonne della matrice aa*
nrow(aa)	*Numero di colonne della matrice aa*
diag(x) diag(5)	*Crea una matrice diagonale dal vettore x* *Crea una matrice diagonale con tutti elementi sulla diagonale principale pari a 5*
solve(aa)	*Calcola l'inversa della matrice aa*

Array

Si tratta di una generalizzazione di matrici e vettori. Un vettore è un array ad 1 dimensione (solo righe o solo colonne); una matrice è un array a 2 dimensioni (righe e colonne).
Esempio di generazione di un array a tre dimensioni, (3,4,2) composto da tutti zero:

```
> a=array(0,c(3,4,2))                       Si visualizzi il contenuto di a.
```

Analogamente per creare una matrice di 4 righe e 5 colonne, composta di tutti zero, si fa:

```
> a=array(0,c(4,5))
> a

     [,1] [,2] [,3] [,4] [,5]
[1,]    0    0    0    0    0
[2,]    0    0    0    0    0
[3,]    0    0    0    0    0
[4,]    0    0    0    0    0
```

Data frame

I data frame si possono considerare come una ulteriore estensione delle matrici. In particolare, un data frame ha attributi che indicano il nome delle colonne e delle righe.
Vediamo un esempio di creazione di data frame.

```
> test=matrix(rnorm(21),7,3)          matrice 7*3 con numeri casuali da N(0,1)
> test                                Si veda il risultato ottenuto in R
```

Come possiamo vedere l'oggetto test è una semplice matrice. Trasformiamola in data frame e vediamo che cosa succede.

```
> test=as.data.frame(test)
> test                                Si veda il risultato ottenuto in R
```

Come si vede, sono stati assegnati dei numeri alle righe e dei nomi V1, V2, V3 alle colonne.
Possiamo assegnare in altro modo i nomi di colonna e di riga.

```
> row.names(test)=c("A","Pippo","C","D","E","F","G")
>names(test)=c("X","Y","Z")

> test                                Si veda il risultato ottenuto in R
```

Per riferirsi ai singoli vettori (riga o colonna) bisogna sfruttare la struttura gerarchica del data frame. Per richiamare il vettore X (attenzione, è maiuscolo!) si deve fare:

```
> test$X
[1] 1.33634367 1.01866430 2.38718395 -1.26679172 -0.71931917 -0.07446951
[7] -0.23146199
```

ATTENZIONE: R è sensibile a maiuscolo/minuscolo e quindi bisogna rispettare il carattere. Ad esempio, queste colonne (vettori) sono V1, V2, V3 (V maiuscola) e <u>non</u> v1, v2, v3. Lo stesso discorso vale per i comandi: le lettere maiuscole vanno in maiuscolo, le minuscole in minuscolo).

List

Una lista è tipo un vettore. Tuttavia gli elementi di una lista possono essere oggetti di qualunque tipo. Conseguentemente, possiamo avere una lista che contiene liste ecc.
Si vedrà più avanti un esempio di oggetto lista. E' il risultato del metodo dei minimi quadrati per la stima di un modello di regressione.

2. FUNZIONI STATISTICHE

2.1 Funzioni base

Alcune funzioni principali sono qui rappresentate in tabella.

acf(x)	*Autocorrelazione totale o parziale*
chisq.gof(x)	*Chi-square goodness of fit test*
cor(x,y)	*Correlazione*
mad(x)	*Median absolute deviation*
mean(x)	*Media*
median(x)	*Mediana*
quatile(x,prob)	*Quantici*
range(x)	*Campo di variazione o range*
t.test(x)	*t-test*
var(x)	*Varianza (se x vettore), matrice di covarianza se x matrice*
var(x,y)	*Covarianza*

2.2 Minimi quadrati

Creiamo i seguenti due vettori:

```
> x1=rnorm(100)
> x2=rnorm(100)
> y=rnorm(100)
```

Il comando **lm()** stima un modello y=a+bx1+cx2+errore, ovvero:

```
> out= lm(y ~ x1 +x2)
```

Sui nostri dati generati il risultato è:

```
> out

Call:
lm(formula = y ~ x1 + x2)
Coefficients:
```

```
(Intercept)       x1            x2
-0.064190   0.008453  -0.033130
```

Per vedere i risultati della regressione in modo più completo (perché ci sono) si può fare:

```
> summary(out)

Call:
lm(formula = y ~ x1 + x2)
Residuals:
    Min       1Q       Median    3Q        Max
-2.18089  -0.63506  -0.05141  0.63662  2.56727

Coefficients:
Estimate     Std. Error    t value      Pr(>|t|)
(Intercept)  -0.064190   0.094197  -0.681 0.497
x1            0.008453   0.087116   0.097 0.923
x2           -0.033130   0.089286  -0.371 0.711

Residual standard error: 0.9245 on 97 degrees of freedom
Multiple R-Squared: 0.001473,  Adjusted R-squared: -0.01912
F-statistic: 0.07153 on 2 and 97 DF, p-value: 0.931
```

Le principali funzioni che possono servire in questo caso sono elencate in tabella.:

summary*(object)*	*Riassunto del modello stimato*
coef*(object)*	*Parametri stimati (vettore)*
fitted*(object)*	*Fitted values*
deviance*(object)*	*Somma dei quadrati dei residui*
anova*(object)*	*Tavola dell'analisi della varianza*
plot*(object)*	*Plot diagnostici per l'analisi dei residui*

Fra i plot per l'analisi dei residui, ci sono: scatterplot dei residui verso fitted values, scatterplot dei residui standardizzati verso fitted values, q-q normality plot, Cook's distance plot.

3. LETTURA FILE

Consideriamo qui la lettura di files tipo ASCII, costituiti da un vettore. Il comando è:

> **y=scan("*percorso/nome file*")**

Se il file che vogliamo leggere si chiamo pippo.dat e si trova nella cartella pluto su disco d, il comando sarà:

> **y=scan("d:/pluto/pippo.dat")**

Ricordarsi di usare la barra sopra il tasto col numero 7 (/) e non il backslash (\). Il risultato del comando è un vettore. Se i dati numeri in pippo.dat sono *n*, la risposta è:

Read *n* items

4. GRAFICI

Per i nostri scopi interessa la funzione **plot(*object*)**. Essa però da differenti risultati a seconda dell'oggetto in input. Lo scatterplot viene generato dal comando:

> plot(y,x)

dove y e x sono due vettori delle stesse dimensioni. Se invece facciamo:

> plot(y)

viene prodotto uno scatterplot con y in ordinata ed in ascissa numeri naturali 1,2,…, fini al numero di elementi del vettore y. Se

> **y=ts(y)**

y diventa un oggetto *time series* e quindi, in tal caso:

> plot(y)

produce un time plot.

Se vogliamo sovrapporre un grafico ad un altro, si può usare il comando **lines(*object*)**. Ad esempio, se anche x è una serie storica, per sovrapporre il time plot di x su quello di y si fa:

> plot(y)

> lines(x)

Un altro comando interessante è **abline(h=*valore*)** che consente di sovrapporre al grafico una linea orizzontale nella posizione di ordinata stabilita da *valore*. Ad esempio se sul time plot della serie storica y vogliamo tracciare una linea orizzontale in corrispondenza della media, si fa:

> plot(y)

> abline(h=mean(x))