# 1. INTRODUZIONE A R --------------------------------------------------------

# Una linea di commento su R inizia con `#'.

# Tutto cio che viene dopo # è ignorato da R, cioè non viene mandato in esecuzione

# R è un software per l'analisi statistica dei dati

# In maniera molto elementare nel funzionamento di R possiamo

# distinguere:

# 1. i COMANDI/FUNZIONI, che ci permettono di effettuare un'analisi o piu

# semplicemente una operazione

# 2. gli OGGETTI, che sono i destinatari delle nostre analisi e elaborazioni

# I comandi/funzioni in R sono seguiti da (),

# all'interno delle parentesi vengono introdotti gli ARGOMENTI della

# funzione

# Le funzioni sono contenute all'interno di pacchetti che devono essere

# installati e richiamati

# Gli argomenti di uno specifico comando\funzione possono essere richiamati con args

args(paste)

args(mean)

# L'interazione con la console di R consiste nel digitare

# una qualche istruzione seguita dal tasto RUN e ricevere un

# risultato.

# Esempio se digitiamo:

3+2

# Otteniamo come output:

# [1] 5

# L'abbastanza criptico [1] di fronte all'output deve essere letto

# in questo modo: "questa linea di output mostra i valori a partire

# dal primo elemento". Questo è particolarmente utile nel caso di risultati

# che contengono molti valori. Per esempio

rnorm(400, mean = 10, sd = 2)

# # 2. Help in R: Sezioni tipiche ---------------------------------------------

# La funzione help.start() apre un help in formato html

help.start()

# La funzione help(), che puo essere abbreviata da ?,

# fornisce informazioni e aiuto relativo a specifiche funzioni o librerie

help()

?mean

# Per una lista completa delle funzioni e delle carattreristiche di

# una libreria/pacchetto si puo usare la seguente sintassi

library(help = "base")

library(help = "utils")

# La funzione example() esegue alcuni esempi di funzioni specifiche

example()

example(plot) # esempi della funzione plot

example(mean)

# # 3. Chiudere la sessione di lavoro e cambiare directory ------------------------------------------

# La funzione q() permette di chiudere la sessione di lavoro

# ovvero ci permette di uscire da R

q()

# Eseguendo q() si aspre una finestra di dialogo che ci chiede se salvare

# o no l'area di lavoro/workspace.

# Se rispondiamo Si salviamo l'area di lavoro nella directory corrente

# Per sapere in che directory ci troviamo usiamo la funzione getwd()

getwd()

# Per cambiare directory si puo usare la funzione setwd(),

# che vuole come argomento il percorso dell'eventuale nuova

# directory di lavoro

setwd("/Users/leonardoalaimo/Desktop")

getwd()

setwd( "/Users/leonardoalaimo/Desktop/Script per Genova")

getwd()

# N.B. utilizzando un progetto abbiamo il vantaggio che sapremo sempre

# in quale directory verra salvato il nostro lavoro relativo a quel

# progetto specifico.

#

# Per prima cosa, quindi, è opportuno creare un progetto in una cartella denominata 'Lezione\_1'.

# salvare tutti i file relativi a R in questa cartella

# # 4. Pacchetti aggiuntivi ---------------------------------------------------

# R possiede una serie di funzioni che sono incluse

# in una serie di pacchetti installati al momento dell'installazione di R

# e che vengono richiamati automaticamente al momento dell'apertura del software

# Ecco una lista dei pacchetti "base"

# base

# stats

# graphics

# datasets

# methods

# utils

# Esistono molti pacchetti utilizzati su R per le più svariate analisi e

# necessita. Al momento in cui scrivo questo script ci sono 12323

# pacchetti disponibili sul cran https://cran.r-project.org

# Per installare un nuovo pacchetto si utilizza la funzione install.packages()

# Per esempio installiamo il pacchetto Compind

install.packages("Compind")

# Dopo aver installato un pacchetto, le relative funzioni sono salvate in R,

# tuttavia per utilizzarle è necessario richiamare il pacchetto attraverso

# la funzione library

library("Compind")

# Per avere una lista delle funzioni del pacchetto

ls(name="package:Compind")

# Pacchetti utili da installare

# cluster foreign sna ggplot2 spatial MASS memisc stargazer tydiverse Compind parsec

# N. B. se non si richiama il pacchetto al momento dell'apertura di R

# le sue funzioni non saranno disponibili

# 5. R come una calcolatrice ------------------------------------------------

# le operazioni principali sono '+, -, \*, /, ^

3 + 2

4 - 1

9 \* 3

9 / 3

9 ^ 2

9^(1/2)

a <- 5 # crea un oggetto di nome 'a' cui si associa il valore 5

# <- Possibile sostituirla con uguale

a=5

a # per vedere il contenuto dell'oggetto digiatiamo il suo nome

a <- 2+2 # un oggetto puo essere il risultato di una operazione

a

# N. B. Come abbiamo posuto osservare, R ha sovrascritto l'oggetto a!

# Inoltre occorre ricordare che R è "case sensitive

a <- 5

a <- 4

A <- 4

# Quante altre operazioni numeriche e con quale sintassi?

?Arithmetic # Operazione Aritmetiche

?Special

help(sqrt)# radice quadrata e valore assoluto

help(Trig)

c <- 3+2.5

c

abs(c)

sqrt(c) # radice quadrata

d <- -4

d

abs(d)

e <- log(2, 10)

e

f <- exp(3)

f

# SEQUENZE DI NUMERI

seq(from = 1, to = 10) # crea una sequenza di numeri da 1 a 10

seq(from = 1, to = 10, length.out = 4) # crea una sequenza di 4 numeri da 1 a 10 distanziati da un intervallo uguale

seq(from = 1, to = 10, by = 2) # crea una sequenza di numeri da 1 a 10 con un incremento di 2 alla volta

# che differenza c'e' tra questi comandi?

seq(from = 4, to = 12, by = 2)

seq(to = 12, by = 2, from = 4)

seq(4, 12, 2)

seq(2, 12, 4) # stessi valori degli argomenti, in ordine diverso

# sequenze di numeri interi consecutivi

1:10

12:4

4:12

# notare la differenza tra questi due comandi

- 1:3

- (1:3)

# # 6. Gli oggetti in R --------------------------------------------

# Tutto in R puo essere immagazzinato in un OGGETTO.

# Un oggetto viene associato il piu delle volte a un NOME che ci

# permette di riferirci al suo contenuto.

# REGOLE SUI NOMI DEGLI OGGETTI

# 1. usare solo caratteri alfanumerici e il simbolo '\_'

# 2. non devono contenere spazi

# 3. devono sempre cominciare con una lettera

# Gli oggetti in R possono immagazinare diversi tipi di informazioni.

# Le entita' piu' semplici che possono costituire un oggetto di R

# (a cui assegnare un nome) sono:

# [1] ELEMENTI SCALARI (numeric)

a <- 10 # NB "<-" è il simbolo per attribuire nomi a oggetti

# oggi si puo utilizzare anche "="

b <- 15.53

c <- 3.51+1.2i

c

# [2] ELEMENTI STRINGHE (character) di caratteri: incluse in ""

d <- "Darth Vader"

e <- "Luke Skywalker"

k <- "odio la statistica in tutte le sue forme"

d

e

k

# [3] ELEMENTI LOGICI (logical)

f <- TRUE

g <- FALSE

h <- F

i <- T

i <- "true"

i

i

# L'insieme degli oggetti creati in una sessione di R viene chiamato "WORKSPACE"

ls() # Restituisce la lista degli oggetti nell'ambiente di lavoro

f # Restituisce il contenuto dell'oggetto f

rm(f)# Cancella l'oggetto f

f

rm(list=ls()) # cancella tutti gli elementi nello spazio di lavoro

# oppure uso la scopetta!

save.image("Lezione\_Genova.Rdata") # salva lo spazio di lavoro in un file Rdata

# N. B. è buona norma tenere in ordine il workspace durante le fasi di lavoro, evitando

# l'utilizzo di nomi insensati per gli oggetti e eliminando oggetti inutili. E ricordarsi

# alla fine di ogni sessione di salvarlo

rm(f,g) # Quando voglio cancellare più elementi

# basta inserirne il nome dentro rm, separandoli con una ,

save.image("Lezione\_Genova.Rdata")

## # 7. Oggetti in R: i vettori --------------------------------------------

# L'oggetto base di R è il "VETTORE".

# Anche quando assegniamo un singolo valore a un oggetto (es. a <- 3)

# stiamo di fatto creando un vettore che contiene un singolo elemento.

# Possiamo definire un vettore come un oggetto che è in grado di immagazzinare

# "un set di valori dello stesso tipo". N.B. non possiamo creare vettori mettendo insieme valori di diverso tipo!!

# Si possono quindi creare vettori di stringhe, di numeri o di valori

# logici. La lunghezza di un vettore equivale al suo numero di elementi e

# puo essere ottenuta con la funzione length().

# Possiamo creare un vettore in r usando la funzione c()

?c() # comando per creare liste di elementi

c(4, 7, 23.5, 76.2, 80)

# Lanciando la riga abbiamo creato un vettore. Tuttavia per salvarlo nello

# spazio si lavoro occorre attribuirgli un nome

v <- c(4, 7, 23.5, 76.2, 80)

v

length(v) # numero di elementi di v

# la funzione mode() restituisce la tipologia di dati dei valori

# contenuti nel vettore

mode(v)

### N.B. COERCIZIONE ###

# Come detto tutti gli elementi di un vettore devono appartenere allo

# stesso tipo. Se ciò non avviene R forza il vettore per coercizione,

# rendendo omogeneo il mode degli elementi.

# Vediamo un esempio:

v1 <- c(4, 7, 23.5, 76.2, 80, "rrt") # tutti numeri tranne l'ultimo

v1

mode(v1)

# Per creare un oggetto che include

# elementi di diverso mode occorre creare una LISTA (vedi più avanti)

# Tutti i vettori possono contenere uno speciale valore NA, che

# indica i valori mancanti.

# NA indica un missing di qualsiasi tipo (numerico, carattere o logico)

u <- c(4, 6, NA, 2)

u

mode(u)

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

k

mode(k)

# INDICIZZAZIONE:

# Per accedere a un elemento di un vettore si puo indicarne

# la posizione fra parentesi quadre

v[3] # restituisce il 3 elemento nel vettore v

# In questo modo è anche possibile cambiare il valore di un

# particolare elemento del vettore

k

k[4] <- T

k

k[5] <- F

k

# R permette anche la formazione di vettori vuoti

vuoto <- vector(length=10,mode="numeric") # numerico

vuoto1 <- vector(length=10,mode="character") # caratteri

vuoto2 <- vector(length=10,mode="logical") # logico

vuoto

vuoto1

vuoto2

vuoto[1] <- 1

vuoto

vuoto <- c(1,0,0,0,0,0,0,0,0,0)

vuoto

vuoto[3] <- 3

vuoto

# E possibile Modificare/trasformare/forzare il modo di un vettore

?as.numeric

?as.character

?as.logical

mode(vuoto1)

vuoto1<- as.numeric(vuoto1)

vuoto1

mode(vuoto1)

mode(vuoto2)

vuoto2 <- as.numeric(vuoto2)

vuoto2

mode(vuoto2)

vuoto2 <- as.logical(vuoto2)

mode(vuoto2)

vuoto2 <- as.character(vuoto2)

## ### Attribuire nomi agli elementi di un vettore

x <- c(1,2)

x

x <- c(a=1,b=2)

names(x)

x

y <- c("leonardo" =40, "giorgio" =23, "sara" = 29)

y

mode(y)

y <- c("leonardo", 40, "giorgio",23, "sara" , 29)

y

mode(y)

names(x)

names(x) <- c("c","d")

names(x)

x

### Attraverso i nomi è possibile anche indicizzare gli elementi di un vettore

# Creaimo un vettore di 6 elementi coi loro nomi

a <- c(a=1,b=2,c=3,d=4,e=5,f=6)

a

# Possiamo selezionare il primo e il terzo elemento del vettore in vari modi:

# 1. Usando un indice numerico

a[c(1,3)]

a[2]

# Utilizzando il nome(perche in qquesto caso c'è) degli elementi

a[c("a","c")]

# Richiamiamo tre oggetti di a

a[c(1,3,5)]

# Utilizzando un indice logico

a[c(TRUE,FALSE,TRUE,FALSE,FALSE,FALSE)]

# # 8. Operazioni coi vettori --------------------------------------------------

### OPERAZIONI SU VETTORI NUMERIC

# Funzioni che manipolano un solo vettore (operazioni SU vettori)

# Nelle "reference cards" ci sono le più comuni

vettore <- 1:15 # crea un vettore numerico coi numeri da 1 a 15

vettore

vettore <- c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15)

# Voglio sommare tutti gli elementi del vettore "vettore"

?sum

somma <- sum(vettore)

somma

class(somma)

mode(somma)

?prod

prodotto <- prod(vettore)

prodotto

?min

?max

minimo <- min(vettore)

massimo <- max(vettore)

minimo

massimo

?mean

?sd

mean(vettore)

sd(vettore)

summary(vettore)

str(vettore)

# Comandi utili per studiare un vettore numeric

length() # numero di elementi di un vettore

sum() # somma degli elementi

mean() # media

sd() # deviazione standard

summary() # statistiche descrittive

str() # informazioni sul vettore

# Funzioni che si applicano a due vettori (operazioni TRA vettori)

# Aspetto fondamentale nell'eleaborazione: operazione eseguita

# tra elementi che occupano la stessa posizione

# il primo elemento del primo vettore con il primo elemento del secondo vettore

# il secondo elemento del primo vettore con il secondo elemento del secondo vettore

# il terzo elemento del primo vettore con il terzo elemento del secondo vettore ..

vettore

vettore1 <- 15:1

vettore

vettore1

vettoresomma <- vettore + vettore1

vettoresomma

vettore +9

vettore - 9

vettore^9

# Affinche le operazioni si eseguano elemento per elemento

# è necessario che i vettori abbiano la stessa lunghezza

# cosa succede se NON hanno la stessa lunghezza?

?length

length(vettore)

vettore\*5

vettore+5

vettore/5

length(vettore)

vettore \* c(1,2) #deve essere un multiplo!!!!! Altrimenti non effettua l'operazione!

vettore\*c(1,2,3)

vettore\*c(5,5,5,5,5)

vettore\*c(1,2,1,2,1)

vettore \* c(11,-2,0.44,200,22700)

vettore\*vettore1

##\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_##

## ### OPERAZIONI SU E TRA VETTORI CHARACTER

?paste # Permette la concatenazione di vettori di tipo character

# é probabilmente una delle funzioni più usate in R

# Gli argomenti della funzione sono:

args(paste)

# a. … = Le stringhe da concatenare

# b. sep = L'elemento che separa ogni termine

# c. collapse = L'elemento che separa ogni risultato

b <- c("Ciao", "Leonardo")

b

b <- paste("Ciao","Leonardo")

length(b)

b

paste("Ro","ma",sep="")

paste("Roma","Milano","Torino",sep="-") # L'argomento "sep" indica un carattere per separare i termini

paste("Roma","Milano","Torino",sep=",")

paste(c("Roma","Milano","Torino"),collapse=" -")

paste("Roma","Milano","Torino",sep=" ")

paste("Roma","Milano","Torino",sep="") # notate la differenza rispetto a prima: qui non aggiunge lo spazio

paste(c("Roma","Milano","Torino"),"Palermo",sep="-")

paste("Roma","Milano","Torino","Palermo",sep="-") # la sintassi è diversa

paste(rep("file", 5), rep("number", 5), seq(1,5,1), sep = "\_")

paste(rep("file", 5), rep("number", 5), seq(1,5,1), sep = "\_", collapse = ",")

# Come possiamo osservare dai risultati dei precedenti esempi,

# SE si specifica un valore nell'argomento collapse, si ottiene un'unica strinfa invece di 5!

# Su R c'è anche un altro comando simile a paste(), paste0(),

# La differenza è che l'argomento sep di default is " " in paste() e "" in paste0()

?paste0

name\_village <- paste("Ma", "con", "do")

name\_village

name\_village <- paste("Ma", "con", "do", sep = "")

name\_village

name\_village <- paste0("Ma", "con", "do")

name\_village

# paste0() è più veloce di paste() se il nostro obiettivo è concatenare stringhe senza spazi

# Sia paste() che paste0() trasformano per coercizione tutto in character

a <- 1:3

b <- 4:6

c <- c("a", "b", "c")

d <- paste(a,b,c, sep = "-")

d

mode(d)

e <- paste0(a,b,c)

e

##\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_##

## ### OPERAZIONI SU E TRA VETTORI LOGICAL

# NOT ovvero Negazione

# [carattere ! o punto esclamativo]

!TRUE

# AND ovvero Congiunzione o Intersezione

# [carattere & o lettera e-commerciale]

TRUE & FALSE

TRUE & TRUE

F & F

c(T,T,F,F)&c(T,F,T,F)

# OR ovvero Alternativa o Unione

# [carattere | o barra verticale]

TRUE | FALSE

c(T,T,F,F)|c(T,F,T,F)

vettore.logico <- c(T,T,F,F,T)

vettore.logico

?any # Dato un vettore logico, ci dice se almeno uno è TRUE

?all # Dato un vettore logico, ci dice se tutti sono TRUE

?which # Dato un vettore logico, ci dice quale elemento è TRUE

any(vettore.logico)

all(vettore.logico)

which(vettore.logico)

# I logici sono importanti perché

# spesso lavoriamo con i confronti tra numeri

# e/o tra stringhe

1<2

c(1,2,3,4,5)>=3

"luca"=="Luca"

# ATTENZIONE!: R è 'case-sensitive'

"luca"<"maria" # Cosa fa???

"paolo" < "maria"

"123456789" =="123456789"

# # 9. Liste ---------------------------------------------------------------

# Come detto tutti gli elementi di un vettore devono appartenere allo

# stesso tipo. Non possiamo mettere insieme "mele e pere"

# Per creare un oggetto che include

# elementi di diverso mode dobbiamo creare una LISTA, usando il comando list()

# Definiamo alcuni vettori di mode differente

vettore1 <- 3

vettore2 <- c(4,5,6)

vettore3 <- c(T,F,T,F)

vettore4 <- c("Luca","Alessandra")

vettore5 <- vector(mode="character",length=0)

# per organizzare elementi di modo diverso dobbiamo usare le liste

lista <- list(vettore3,vettore4,vettore1) # N. B. i vettori possono anche avere lunghezze differenti

lista

# ma anche le liste hanno i nomi ... ma una sintassi leggermente diversa

names(lista)

names(lista) <- c("logico","carattere","numero")

names(lista) <- c("variabile1", "variabile2", "variabile3")

str(lista)

lista

# differenti modi di estrarre elementi da una lista

lista["logico"] # richiama l'elemento "logico" nella lista

lista$ # richiama l'elemento "logico" nella lista

lista[1] # usiamo l'indice

# alcune funzioni da applicare ad una lista

length(lst) # numero di elementi di una lista

str(lst) # informazioni sulla lista

# # 10. Fattori ----------------------------------------------------------------

# I fattori sono un particolare "tipo di vettore" ottenuto con la funzione factor()

# che viene utilizzato per il trattamento di variabili categoriche.

# Ricordiamo un po di statistica....

# Le variabili categoriche sono quelle le cui modalita rappresentano CATEGORIE:

# 1. se le categorie sono tra loro allo stesso livello (es. genere) avremo var. NOMINALI;

# 2. se le categorie possono essere ordinate (es. titolo di studio) avremo var. ORDINALI.

# I fattori sono variabili in R che possono prendere un numero limitato di categorie differenti (le categorie).

# L'importanza dell'uso dei fattori in R è molteplice; possiamo per comodita individuare alcuni ambiti

# in cui il ricorso ai fattori è fondamentale:

# a. I fattori sono importanti per determinare come i dati vengono analizzati e soprattutto visualizzati

# Alcuni pacchetti grafici (es. ggplot2) ricorrono spesso ai fattori per settare alcune impostazioni.

# b. Uno dei piu importanti usi dei fattori è nella modellistica statistica: il loro utilizzo permette

# di trattare le variabili categoriali in modo corretto nei modelli (dummies)

# Un fattore in R viene salvato come:

# un vettore di valori interi

# un set di etichette, corrispondenti alle categorie possibili: i LEVELS

?factor

args(factor)

data <- factor(c(1,2,2,3,1,2,3,3,1,2,3,3,1))

c(1,2,2,3,1,2,3,3,1,2,3,3,1)

data

str(data)

mode(data)

class(data)

# L'unico argomento richiesto dalla funzione factor è un vettore di valori

# che verra trasformato in un vettore di "VALORI FATTORE"

# Ovviamente sia vettori numerici che vettori di caratteri possono essere trasformati in fattori,

# tuttavia i LEVELS saranno sempre valori carattere.

genere <- c(rep("femmina",691), rep("maschio",692))

genere

class(genere)

mode(genere)

levels(genere) # Ovviamente ritorna NULL perche non abbiamo creato un FATTORE ma un VETTORE!

genere <- factor(genere) # trasformo il vettore in fattore

genere

levels(genere)

str(genere)

mode(genere)

class(genere)

as.numeric(genere) # ci mostra le categorie del fattore come numeri

as.character(genere)

# I possibili valori dei LEVELS di un fattore possono essere richiamati attraverso

# la funzione levels(); la funzione nlevels() da il numero di categorie

levels(data)

nlevels(data)

levels(genere)

nlevels(genere)

# dump("data",file="prova.csv") # per esportare gli oggetti di R 'in chiaro' (ascii)

# La funzione levels consente inoltre di cambiare le categorie del vettore. Per esempio cambiamo i livelli

# del fattore data in lettere

levels(data) <- c("a", "b", "c")

data # possiamo notare che ha sostituito ciascun numero nel fattore con la lettera corrispondente

levels(data) <- c("Primo", "Secondo", "Terzo")

data

# 11. Fattori ordinati ----------------------------------------------------

# Finora abbiamo esaminato Fattori non ordinati, in cui cioè le modalita che rappresentano

# le categorie non hanno ALCUNA RELAZIONE DI ORDINE.

# Per ordinare un fattore (per stabilire cioe l'esistenza di una relazione d'ordine fra le sue modalita)

# si aggiunge l'argomento ordered = TRUE

# Creiamo un fattore con i mesi dell'anno

mesi = factor( c("March","April","January","November","January",

"September","October","September","November","August",

"January","November","November","February","May","August",

"July","December","August","August","September","November",

"February","April"))

mesi

table(mesi) # table è una funzione che non abbiamo ancora visto....andiamo a vedere cosa fa

# Risulta evidente che i mesi dell'anno abbiamo una relazione ordinata che non è presente nel

# fattore cosi come lo abbiamo salvato.

# Per cui dobbiamo stabilire una relazione d'ordine fra i LEVELS

mesi <- factor(mesi,levels=c("January","February","March",

"April","May","June","July","August","September",

"October","November","December"),ordered=TRUE)

mesi # in questo modo ho messo in ordine i mesi e stabilito una relazione di ordine

table(mesi)

# 12. Esercizio 1 -------------------------------------------------------------

# 1. Create un nuovo progetto e chiamatelo "Esercitazione"

# 2. All'interno del nuovo progetto create i seguenti vettori:

# Vettore "uno" composto dai primi 100 numeri naturali,

# Vettore "due" dato dal vettore "uno" diviso per 2

# Vettore "tre" dato dai primi 11 numeri naturali

# Vettore "quattro" risultato del vettore "uno" moltiplicato per il vettore "tre"

# Vettore "id" con il vostro nome, cognome e data di nascita

# Vettore "id\_2" con nome.cognome.datadinascita

# Creare un fattore ordinato relativo al "Titolo di studio"

# Create una lista di nome "lista\_uno" che include tutti gli oggetti costruiti costruiti

# # 13. Dati mancanti ---------------------------------------------------------

# ecco uno dei personaggi scomodi spesso

# presenti anche nei dati reali:

# il dato mancante

# in R rappresentato dalle lettere NA (Not Available)

0/4

4/0

0/0

# NaN vuol dire Not a Number

# e una forma indeterminata

# per scovare dove sono i dati mancanti

is.na(c(1,2,3,NA,4))

!is.na(c(1,2,3,NA,4))

# # 14. Matrici e array -----------------------------------------------------------------

# I dati possono essere immagazzinati in oggetti che hanno piu di una dimensione

# Gli ARRAY permettono di immagazzinare i dati in molteplici dimensioni:

# le MATRICI sono un particolare tipo di array che presenta 2 dimensioni.

# In sintesi, non sono nient'altro che VETTORI CON UN PARTICOLORE ATTRIBUTO:

# la DIMENSIONE. Tutti gli elementi contenuti in una matrice o un array devono avere LO STESSO MODE!!

# Vediamo alcuni esempi

# Unione di vettori (colonna), tutti dello stesso tipo

col1 <- c(1, 5, 8, 7)

col2 <- c(10, 0, - 5, 4)

a <- cbind(col1, col2) # il comando cbind unisce due elementi per colonna; rbind come vedremo li unisce per riga

a

class(a)

# Cosa succede se creiamo una matrice con una colonna numerica e una di stringhe?

col3 <- c("a", "b", "c", "d")

col1

col3

b <- cbind(col1, col3) # coercizione!

b

# Creo una matrice da un vettore specificando le dimensioni con la funzione dim(), che speifica il numero di righe (primo argomento)

# e di colonne (secondo argomento) della matrice

x <- 1:100

x

dim(x) <- c(2,50)

x

dim(x) <- c(10,10)

x

# Creo una matrice usando il comando matrix()

?matrix

args(matrix)

# Gli argomenti della funzione matrix sono:

# data = i dati da utilizzare nella matrice: se non specificati viene generata una matrice di NA

# nrow = numero di righe della matrice

# ncol = numero di colonne della matrice

# byrow = si tratta di un argomento logico. Di default il valore è FALSE e la matrice viene riempita per colonna,

# se TRUE la matrice viene riempita per riga

# dimnames = è una lista in cui è possibile inserire due vettori conteneti i nomi di righe e colonne

matrice <- matrix(nrow=5,ncol=4) # crea una matrice di 5 righe e 4 colonne di NA

matrice

mode(matrice)

class(matrice)

matrice <- matrix(0,nrow=5,ncol=4) # diamo il valore 0 a tutti gli elementi della matrice

matrice

mode(matrice)

class(matrice)

matrice <- matrix("ciao",nrow=5,ncol=4) # diamo il valore 0 a tutti gli elementi della matrice

matrice

mode(matrice)

class(matrice)

matrice <- matrix(1:20,nrow=5,ncol=4)

matrice # utilizziamo un vettore per gli elementi della matrice

matrice <- matrix(c(1,2,3,4,5,6,7,8,9,10,

11,12,13,14,15,16,17,18,19,20),nrow=5,ncol=4)

matrice <- matrix(1:20,nrow=5,ncol=4) # utilizziamo un vettore per gli elementi della matrice

matrice <- matrix(seq(1,20),nrow=5,ncol=4)

matrice

mode(matrice)

class(matrice)

matrice <- matrix(1:20,nrow=5,ncol=4, byrow = T) # utilizzando byrow=T riempiamo la matrice per riga e non per colonna

matrice

matrice <- matrix(1:20,nrow=5,ncol=4, byrow = T,

dimnames = list(c("Riga 1","Riga 2", "Riga 3", "Riga 4", "Riga 5"),

c("Colonna 1","Colonna 2", "Colonna 3", "Colonna 4")) ) # cambiamo nome a righe e colonne

matrice

matrice\_trasposta <- t(matrice) # Il comando t() permette di avere la trasposta della matrice

matrice\_trasposta

# alcune funzioni da applicare ad una matrice

dim(matrice) # numero di righe e colonne

nrow(matrice) # numero di righe

ncol(matrice) # numero di colonne

str(matrice) # informazioni sulla matrice

summary(matrice)

# Con la funzione apply() possiamo applicare una specifica funzione alle colonne o alle righe della matrice

# Argomenti di base di apply sono:

?apply

args(apply)

# x = il nome della matrice

# MARGIN = 1 per applicare la funzione per le righe e 2 per applicarla sulle colonne

# FUN = la funzione da applicare

# Calcolare la media per le colonne della matrice

apply(matrice, MARGIN = 2, mean)

# Calcolare la media per le righe della matrice

apply(matrice, MARGIN = 1, mean)

# Gli ARRAY sono oggetti a 3 o piu' dimensioni

# Per crearli usiamo il comando array()

?array

# Gli argomenti della funzione sono:

args(array)

# data = i dati da utilizzare nella matrice: se non specificati viene generata una matrice di NA

# dim = le dimensioni dell'array, da specificare in un vettore dove vengono anche identificate le ampiezze di ciascuna dimensione

# dimnames = è una lista in cui è possibile inserire i vettori con i nomi delle dimensioni

array1 <- array(1:24,dim=c(4,3,2)) #il prodotto di dim è il length dell'array!

array1

array2 <- array(1:81, dim =c(3,3,3,3))

array2

# # 15. Dataframes - Creazione e caricamento -------------------------------------------------------------

# Sono la tipica struttura di R per rappresentare data sets casi\*variabili, vale a dire le classici matrici dei dati della statistica multivariata

# Sono infatti "liste di vettori delle stesse dimensioni/lunghezza"

# Quando richiamiamo un data frame appare come una matrice, con la differenza che puo

# CONTENERE COLONNE DI DATI DI DIFFERENTI TIPOLOGIE

# Utilizziamo come esempio un dataset contenuto all'interno di R

?USJudgeRatings

# Numerosi pacchetti di R contengono datasets di esempio, che consentono di esercitarsi con le funzioni in essi incluse.

# L’installazione standard di R contiene un pacchetto “datasets” che contiene solo dati.

# Ogni volta che si carica un pacchetto, gli eventuali datasets in esso contenuti diventano disponibili,

# e utilizzabili con il comando data()

library(datasets)

library(help = "datasets")

?data()

# Utilizziamo uno di questi dataset per vedere la struttura tipica dei dataframe

?USJudgeRatings # Informazioni sul dataset

data("USJudgeRatings") # Carichiamo il dataset USJudgeRatings

USJudgeRatings[1:15,]

USJudgeRatings[1:15,1]

# Questa è la classica indiciaazione di un dataframe: nelle parentesi quadre si specificano

# le righe da richiamare (prima della virgola)

# le colonne da richiamare (dopo la virgola)

# [1:15,] significa le prime 15 righe e tutte le colonne perche dopo la virgolo non è specificato alcun indice

# Il comando str() ci da una vista differente del data frame

# U.S. State Judge ratings

str(USJudgeRatings)

# Restituisce numero di oss. e variabili, nome variabili e valori delle variabili

# Avendo a che fare con l'analisi dei dati, siamo interessati a utilizzare i nostri dataframe e non

# quelli contenuti in R e utilizzati a fini didattici.

# Possiamo visualizzare un dataframe con il comando View()

View(USJudgeRatings)

### Esistono due modi per usare un dataframe su R:

# Crearlo

# Caricarlo da un file esterno

### a. Creazione di un dataframe ###

# Creo tre variabili: un vettore numerico x e due fattori a e b

x <- rnorm(n=50,mean=1,sd=2)

a <- factor(rep(1:5,10),labels=c("A","B","C","D","E"))

b <- factor(rep(1:2,c(25,25)),labels=c("+","-"))

length(x)

length(a)

length(b)

# Con il comando data.frame() creo il data frame, che mette insieme le tre variabili create.

# Posso anche attribuire un nome alle variabili create

TDF <- data.frame(x,a,b)

class(TDF)

str(TDF)

View(TDF)

# N.B. Il comando data.frame() trasforma i vettori di stringhe in fattori!

# se vogliamo evitarlo usiamo l'argomento stringsAsFactors = FALSE

a <- 1:15

b <- rep(c("a","b", "c"),5)

b

mode(b)

df <- data.frame(a,b)

str(df)

df <- data.frame(a,b, stringsAsFactors = FALSE)

str(df)

### ATTENZIONE alla lunghezza dei vettori !!!!

x1 <- rnorm(n=40,mean=1,sd=2)

a1 <- factor(rep(1:5,10),labels=c("A","B","C","D","E"))

b1 <- factor(rep(1:2,c(25,25)),labels=c("+","-"))

TDF1 <- data.frame(x1,a1,b1)

rm(a1,b1,x1,a,b,x)

# Mostra solo le prime otto righe del dataframe e tutte le colonne.

TDF[1:8,]

# Mostra tutte le righe tranne le prime otto.

TDF[-(1:8),]

# Mostra solo la prima colonna

TDF[,1]

# Mostra tutte le colonne tranne la prima

TDF[,-(1)]

# Per mostrare un elemento specifico se ne inseriscono le coordinate riga e colonna.

TDF[1,1]

# Il comando head(nome.dataframe) restutuisce le prime righe del data frame

head(TDF)

### b. Caricamento di un dataframe ###

# Su R è possibile caricare file di molte tipologie e estensioni (xls,csv, dta,sav, etc.)

# Il caricamento di file esterni con R Studio è semplice e abbastanza intuitivo.

### Esempio di caricamento da file excel ###

# Carichiamo il file excel "Goal\_2" presente nella cartella Lezione5 (cosi iniziamo a capire l'importanza dei progetti)

# per caricare un file excel si usa la funzione read\_excel della libreria "readxl"

library(readxl)

?read\_excel

args(read\_excel)

Goal\_2 <- read\_excel("Goal\_2.xlsx", col\_names = T)

View(Goal\_2)

#N.B. non affrontiamo qui il caricamento di file di altro formato

# nella considerazione che valgono regole simili a quelle dei file excel (che spesso sono quelli che usiamo di piu)

# e che R Studio permette di caricare senza conoscere le righe di comando.

# Vediamo un esempio con il file Sintesi.csv

class(Sintesi)

# Dopo aver caricato un dataframe è sempre buona prassi controllarne la classe e le variabili

class(Goal\_2) # la classe è un po strana...

# "tbl\_df" "tbl" "data.frame" .... convertiamo in un dataframe

Goal\_2 <- as.data.frame(Goal\_2)

class(Goal\_2)

summary(Goal\_2) # da le statistiche di base delle variabili e del dataframe

View(Goal\_2)

# Come vediamo il dataframe non contiene il nome delle righe originali,

# che R ha salvato come una nuova variabili chiamata in questo caso "Unita"

# Attribuiamo a ciascuna riga il suo nome originale con il comando rownames

rownames(Goal\_2) <- Goal\_2[,1]

View(Goal\_2) # Possiamo decidere di tenere o cancellare la variabile Unita.....

# Cancellarla è semplice ma non facciamolo per ora: cmq ecco la sintassi

# Goal\_2 <- Goal\_2[,-1]

# Cambiamo i nomi delle righe nel database Sintesi

# class(Sintesi)

# Sintesi <- as.data.frame(Sintesi)

# rownames(Sintesi) <- Sintesi[,1]

# Sintesi <- Sintesi[,-1]

# View(Sintesi)

# Allo stesso modo possiamo cambiare il nome delle colonne con il comando colnames

colnames(Goal\_2) <- c("Unita", "Ripartizione geografica", "Luogo", "x1", "x2", "x3", "x4", "x5")

View(Goal\_2)

# Un'altra operazione che spesso è opportuno fare, in base ai nostri obiettivi,

# è ridurre le cifre decimali presenti nelle variabili del dataframe.

# Per farlo usiamo il comando round()

View(Sintesi) # le variabili hanno troppe cifre decimali...

Sintesi <- round(Sintesi,2) # il comando round() permette di ridurre le cifre decimali

# N.B. il comando va applicato solo a elementi numeric senno restituisce errore

View(Sintesi)

## # 16. Dataframes - Accesso ---------------------------------------------------

# Le variabili in un dataframe possono essere richiamate usando $ e []

# Per esempio accediamo a due variabili usando '$'

mean(USJudgeRatings$CONT) + mean(USJudgeRatings$INTG)

# Usiamo []

mean(USJudgeRatings[,1]) + mean(USJudgeRatings[,2])

Goal\_2$Luogo

# Le variabili in un data frame possono essere accessibili come "gli oggetti nel workplace"

# attraverso il comando attach(nome..dataframe).

# Cio significa che il database viene ricercato da R durante la valutazione

# di una variabile, per cui gli oggetti nel database possono essere raggiunti

# semplicemente indicando i loro nomi.

?attach()

attach(Sintesi)

Sintesi

G5\_2010

x1 # Ovviamente non trova l'oggetto perche il dataframe in cui si trova non è attach() !!!!

CONT # Ovviamente non trova l'oggetto perche il dataframe in cui si trova non è attach() !!!!

# Dal momento che ho usato attach() sul dataframe Sintesi, posso richiamare

# la variabile G5\_2010 direttamente come se fosse nel Global Environment.

# attach() modifica il percorso di ricerca, diventa quindi necessario, una volta finite

# le operazioni sul dataframe usare detach()

detach(Sintesi)

# Vediamo la stessa operazione fatta prima attraverso l'uso di attach() detach()

attach(USJudgeRatings)

mean(CONT) + mean(INTG)

detach(USJudgeRatings)

# Infine un buon modo per accedere alle variabili di un dataframe

# e usare la funzione with(), che permette un accesso temporaneo al dataframe.

# with(nome.dataframe, {})

?with

args(with) # l'operazione che voglio compiere sul dataframe va dentro la parentesi graffa.

with(USJudgeRatings, {

mean(CONT) + mean(INTG)

})

a <- with(Goal\_2, {

Unita

})

a

## # 17. Dataframes - Subsetting --------------------------------------------------

# Ci sono vari modi per creare un suttoinsieme da un dataframe. Ne vediamo due:

#-1 indicizzando righe e colonne coi comandi $ o []

str(Sintesi)

View(Sintesi)

subset.Sintesi1 <- Sintesi[2:5,3:5]

subset.Sintesi1

subset.Sintesi2 <- data.frame(Sintesi$G5\_2011, Sintesi$G5\_2013)

subset.Sintesi2

#-2 utilizzando la funzione subset()

?subset()

args(subset)

subset.Sintesi3 <- subset(Sintesi, select = c(G5\_2010, G5\_2012)) # l'argomento select seleziona le variabili da inbcludere nel dataset

subset.Sintesi3

subset.Sintesi4 <- Sintesi[1:5,] # selezionare i casi e inserirli in un subset!

#Puliamo lo spazio di lavoro

rm(subset.Sintesi1,subset.Sintesi2, subset.Sintesi3, subset.Sintesi4)

## # 18. Dataframes - la funzione within() ------------------

# Come detto in precedenza, le funzioni attach() e with() facilitano l'accesso a un dataframe o

# in generale a una lista, permettendo di accedere alle variabile del dataframe come se

# fossero nello spazio di lavoro.

# Tuttavia queste due funzioni NON PERMETTONO LA CREAZIONE DI NUOVE VARIABILI

# nel dataframe.

# Per modificare le variabili dentro il dataframe si puo usare la funzione within()

# Il primo elemento della funzione within e il dataframe.

# Il secondo un'espressione di assegnazione o varie espressioni di assegnazione racchiuse

# fra parentesi graffe {}.

x <- 1:50

y <- rnorm(50, 3,5)

z <- factor(rep(1:5, 10), labels = c("a", "b", "c", "d", "e"))

TDF <-data.frame(x,y,x)

rm(x,y,z)

?within

args(within)

TDF\_new <- within(TDF,

{ g <- x - mean(x)

}) # in questo modo creo un nuovo dataframe..... Contiene le variabili originali e la nuova

TDF

TDF\_new

with(TDF\_new, mean((g)))

rm(TDF\_new)

TDF <- within(TDF,

{ g <- x - mean(x)

}) # modifico nello stesso dataframe creando la nuova variabili e sovrascrivendo

with(TDF, mean((g)))

# Ovviamente possiamo fare piu operazioni contemporanemante.

# Quello che conta è che:

# 1. le operazioni siano tutte incluse nella graffa

# 2. le varie operazioni NON DEVONO ESSERE SEPARATE DA VIRGOLE

# Creiamo un nuovo database Goal2 che contiene le variabili centrate

Goal2\_centred <- within(Goal\_2, {

x1\_centred <- x1 -mean(x1)

x2\_centred <- x2 - mean(x2)

x3\_centred <- x3 - mean(x3)

x4\_centred <- x4 - mean(x4)

x5\_centred <- x5- mean(x5)

}) # se chiamo le nuove variabili in nome diverso, R copiera anche le originali

View(Goal2\_centred)

Goal2\_centred <- within(Goal\_2, {

x1 <- x1 -mean(x1)

x2 <- x2 - mean(x2)

x3 <- x3 - mean(x3)

x4 <- x4 - mean(x4)

x5 <- x5- mean(x5)

}) # se chiamo le variabili con lo stesso nome delle originali,

# R sovrascrive le variabili con lo stesso nome....ma questo lo sappiamo gia!

View(Goal2\_centred)

rm(Goal2\_centred, TDF\_new)

## # 19. Dataframes - Recoding ----------------------

# R NON SUPPORTA una delle principali operazioni della preparazione

# dei dati nelle scienze sociali: la RICODIFICA DELLE VARIABILI

# Ci sono sistemi per emulare questo processo ma spesso questo puo diventare abbastanza noioso.

# Alcuni pacchetti permettono invece di usufruire di specifiche funzioni di codifica.

# Fra i principali ci sono car() e memisc(). Per chi fosse appassionato puo vedersi i diversi metodi nel dettaglio!

# Qui ci concentreremo sul sistema più semplice e intuitivo: l'uso della funzione recode della libreria memisc()

# Creiamo il dataframe df

df <- data.frame(a = 1:7, f = factor(letters[1:7]))

df

#### Codifica usando le funzioni di memisc()

library(memisc)

library(help = "memisc")

library(psych)

?recode

args(recode)

df <- within(df,{

a3 <- recode(a,

c(1,2,3)->1,

c(4,5) ->2,

c(6,7) ->3)

f3 <- recode(f,

c("a","b","c")->"a",

c("d","e") ->"b",

c("f","g") ->"c")

}) # in pratica dico di reicodificare dentro df (uso infatti within) le variabili a e f chiamandole a3 e f3

df

# Se con la funzione recode() non ricodifichiamo tutti i valori/etichette della variabile

# originale, a questa verranno assegnati i valori NA

df <- data.frame(a = 1:7, f = factor(letters[1:7]))

df

df <- within(df,

r <- recode(a, 1 -> 5,

2:3-> 2,

7 -> 6))

# Posso riassegnarle cosi. Oppure c'?? il modo piu semplice descritto di seguito.

df

df$r

# Si puo usare otherwise() per assegnare un valore standard ai restanti.

within(df,

r <- recode(a, 1 -> 5,

2:3-> 2,

7 -> 6,

otherwise="copy")) # otherwise = "copy" capia nella nuova variabile i restanti valori originali

within(df,

r <- recode(a, 1 -> 5,

2:3-> 2,

7 -> 6,

otherwise=100))

within(df,

r <- recode(a, 1 -> "a",

2:3 -> "b",

7 -> "c"))

within(df,

r <- recode(f,letters[1:3] -> "a",

letters[4:5] -> "b",

"f" -> "c",

otherwise="z"))

## APPROFONDIMENTO ##

# Il pacchetto memisc() oltre a recode() offre anche un'altra funzione molto utile cases().

# Cases() permette di distinguere fra differenti casi e consente la creazione di codici o

# livelli di fattori appropriati.

# La funzione prende diversi fattori logici come argomento e restituisce un fattore o un

# vettore numerico

# `cases` crea un fattore con labels generati automaticamente

df <- data.frame(

x = rnorm(100),

y = rnorm(100)

)

library(memisc)

df <- within(df,{

x4 <- cases(x < -1,

x >= -1 & x < 0,

x >= 0 & x < 1,

x >= 1)

y4 <- cases(y < -1,

y >= -1 & y < 0,

y >= 0 & y < 1,

y >= 1)

})

with(df,table(x4,y4))

# `cases` crea un fattore con labels generati dall'utente

library(memisc)

df <- within(df,{

x5 <- cases(a = x < -1,

b = x >= -1 & x < 0,

c = x >= 0 & x < 1,

d = x >= 1)

y5 <- cases(a = y < -1,

b = y >= -1 & y < 0,

c = y >= 0 & y < 1,

d = y >= 1)

})

with(df,table(x5,y5))

str(df)

# `cases` crea un vettore numerico

df <- within(df,{

x6 <- cases(1 <- x < -1,

2 <- x >= -1 & x < 0,

3 <- x >= 0 & x < 1,

4 <- x >= 1)

y6 <- cases(1 <- y < -1,

2 <- y >= -1 & y < 0,

3 <- y >= 0 & y < 1,

4 <- y >= 1)

})

with(df,table(x6,y6))

str(df)

## # 20. Dataframe: Merging -----------------------------------------------------------------

# E possibile unire due o piu dataframe:

# 1. per colonna (devono avere lo stesso numero di righe) -> funzione cbind()

# 2. per riga (devono avere lo stesso numero di colonne) -> funzione rbind()

# Uniamo i due dataframe Sintesi e Goal\_2 per righe

# Sintesi ha un caso in piu che eliminiamo per permettere il merge

Sintesi <-Sintesi[-23,]

databasecompleto <- cbind(Goal\_2, Sintesi)

class(databasecompleto)

View(databasecompleto)

## # 21. Dataframe: Salvataggio -------------------------------------------------------

# Salvare in csv

write.table(databasecompleto, "databasefinale.csv", sep="," , dec=".", row.names=T, col.names=T )

# Salvare in txt

write.table(df, file="df.txt", quote=T, sep=" ", dec=".", na="NA", row.names=T, col.names=T)

# Salvare in xls

library(openxlsx)

write.xlsx(df, 'df.xlsx')

# Salvare in SPSS SAS STATA -> usiamo la libreria foreign()

library(foreign)

# Salvare per SPSS

write.foreign(df, "mydata.txt", "mydata.sps", package="SPSS")

# Salvare per SAS

write.foreign(df, "mydata.txt", "mydata.sas", package="SAS")

# Salvare per STATA

write.dta(df, "df.dta")

# 22. ESERCIZIO ---------------------------------------------------------------

# 1. creare un data frame con due colonne:

# esame = nomi degli esami sostenuti durante il corso di laurea magistrale

# voto = vot1 preso all'esame

# 2. calcolare il voto medio

# 3. selezionare solo gli esami con voto superiore al voto medio

# 4. creare una nuova variabile con il voto normalizzato

# 5. Creare un fattore cosi codificato

# "Superiori alla media" - "Inferiori alla media" in funzione del vodo di esame

# 4. salvare il data frame in formato .xlsx

esame <- factor(rep(1:10,1),labels=c("antropologia","economia politica","soc proc cult","soc conosc","logica","psicologia soc","metodologia","storia","sociologia","diritto pubbl"))

voto <- c(28,25,31,31,30,29,30,24,31,30)

carriera <- data.frame(esame,voto)

mean(carriera$voto)

subset\_carriera <-subset(carriera, voto > mean(carriera$voto), select=c(esame,voto))

carriera2 <- within(carriera,

{ votostand <- scale(carriera$voto)

})

library("memisc")

carriera3 <- within(carriera2,{

voto\_f <- recode(votostand,

"sotto la media" <- range(min,0),

"sopra la media" <- range(0,max)

)

})

library("openxlsx")

write.xlsx(carriera3, 'carriera.xlsx')

# # 1. Dataframes caricamento e creazione -------------------------------------------------------------

# Sono la tipica struttura di R per rappresentare data sets casi\*variabili, vale a dire le classici matrici dei dati della statistica multivariata

# Sono infatti "liste di vettori delle stesse dimensioni/lunghezza"

# Quando richiamiamo un data frame appare come una matrice, con la differenza che puo

# CONTENERE COLONNE DI DATI DI DIFFERENTI TIPOLOGIE

# Numerosi pacchetti di R contengono datasets di esempio, che consentono di esercitarsi con le funzioni in essi incluse.

# L’installazione standard di R contiene un pacchetto “datasets” che contiene solo dati.

# Ogni volta che si carica un pacchetto, gli eventuali datasets in esso contenuti diventano disponibili,

# e utilizzabili con il comando data()

library(datasets)

library(help = "datasets")

?data()

## 1. CARICAMENTO

# Su R è possibile caricare file di molte tipologie e estensioni (xls,csv, dta,sav, etc.)

# Il caricamento di file esterni con R Studio è semplice e abbastanza intuitivo.

### Esempio di caricamento da file excel ###

# Carichiamo il file excel "Goal\_2" presente nella cartella di lavoro

# per caricare un file excel si usa la funzione read\_excel della libreria "readxl"

library(readxl)

?read\_excel

args(read\_excel)

Goal\_2 <- read\_excel("Goal\_2.xlsx", col\_names = T)

View(Goal\_2)

#N.B. non affrontiamo qui il caricamento di file di altro formato

# nella considerazione che valgono regole simili a quelle dei file excel (che spesso sono quelli che usiamo di piu)

# e che R Studio permette di caricare senza conoscere le righe di comando.

### Esempio di caricamento da file excel ###

# Cariachiamo il file Sintesi.csv usando le socrciatoie offerte da RStudio

class(Sintesi)

# Dopo aver caricato un dataframe è sempre buona prassi controllarne la classe e le variabili

class(Goal\_2) # la classe è un po strana...

# "tbl\_df" "tbl" "data.frame" .... convertiamo in un dataframe

Goal\_2 <- as.data.frame(Goal\_2)

class(Goal\_2)

summary(Goal\_2) # da le statistiche di base delle variabili e del dataframe

View(Goal\_2)

# Come vediamo il dataframe non contiene il nome delle righe originali,

# che R ha salvato come una nuova variabili chiamata in questo caso "Unita"

# Attribuiamo a ciascuna riga il suo nome originale con il comando rownames

rownames(Goal\_2) <- Goal\_2[,2] # abbiamo messo i codici

View(Goal\_2) # Possiamo decidere di tenere o cancellare la variabile Unita.....

# Cancellarla è semplice ma non facciamolo per ora: cmq ecco la sintassi

# Goal\_2 <- Goal\_2[,-1]

# Cambiamo i nomi delle righe nel database Sintesi

# class(Sintesi)

# Sintesi <- as.data.frame(Sintesi)

# rownames(Sintesi) <- Sintesi[,1]

# Sintesi <- Sintesi[,-1]

# View(Sintesi)

# Allo stesso modo possiamo cambiare il nome delle colonne con il comando colnames

colnames(Goal\_2) <- c("Unita", "Codice", "Ripartizione geografica", "Luogo", "x1", "x2", "x3", "x4", "x5")

View(Goal\_2)

# Un'altra operazione che spesso è opportuno fare, in base ai nostri obiettivi,

# è ridurre le cifre decimali presenti nelle variabili del dataframe.

# Per farlo usiamo il comando round()

View(Sintesi) # le variabili hanno troppe cifre decimali...

Sintesi <- round(Sintesi,2) # il comando round() permette di ridurre le cifre decimali

# N.B. il comando va applicato solo a elementi numeric senno restituisce errore

View(Sintesi)

# Ovviamente possiamo caricare file RData in modo simile a quanto visto finora.

# Carichiamo il file "carico.RData" che contiene due dataframe:

# a. Il dataframe "lavoro" contiene variabili sul lavoro per le regioni italiane.

# b. Il dataframe "dati\_AVQ" contiuene un sottoinsieme di unità e variabili

# estratti dall'indagine aspetti della vita quotidiana di Istat

# Possiamo notare che in questo momento sono presenti ben 4 dataframe

# sull'ambiente di lavoro!

# Salviamo

save.image("dati.Rdata")

# # 2. Analisi della struttura dei dati in un dataframe ---------------------------------------

# 2.1. Per scoprire le caratteristiche del data-set, ed in generale di un oggetto di R,

# si utilizza la funzione str()

str(lavoro)

str(dati\_AVQ)

# 2.2. In alternativa, si può impiegare la funzione head()

# che mostra le prime righe della matrice dei dati (di default le prime sei).

head(lavoro)

head(dati\_AVQ)

# 2.3. La funzione View() apre una finestra simile ad un foglio di lavoro Excel,

# che permette di visualizzare l'intero data-set.

View(lavoro)

View(dati\_AVQ)

# 2.4. Un'utile funzione che restituisce il numero di righe e colonne di un dataframe

# è dim()

dim(lavoro)

dim(dati\_AVQ)

# 2.5. La funzione colnames() restituisce i nomi delle colonne (le variabili);

# la funzione rownames() i nomi delle righe (le unità).

colnames(lavoro)

rownames(lavoro)

colnames(dati\_AVQ)

rownames(dati\_AVQ)

# 2.5.1 è possibile cambiare tutti o una parte dei nomi sia delle righe che delle colonne

# 2.5.1.1 Prima salviamo in un vettore i nomi originali cosi ce li ricordiamo

nomivariabili <- colnames(lavoro)

# 2.5.1.2 A questo punto cambiamo i nomi delle variabili

colnames(lavoro) <- c("Reg", "TCP", "TD", "TO", "TMPL", "NEET", "IL", "PI", "COD", "RIP")

str(lavoro)

rownames(lavoro)

lavoro$COD

# Cambiare il nome delle righe utilizzando i miei codici regione che

# sono contenuti nella variabile lavoro$COD

rownames(lavoro) <- lavoro$COD

View(lavoro)

# # 3. Dataframes - Accesso ---------------------------------------------------

# 3.1 Le variabili in un dataframe possono essere richiamate usando $ e []

lavoro$TCP

lavoro[,2]

lavoro$NEET

dati\_AVQ$CITT

dati\_AVQ[,5]

# 3.2 Le variabili in un data frame possono essere accessibili come "gli oggetti nel workplace"

# attraverso il comando attach().

# Cio significa che il database viene ricercato da R durante la valutazione

# di una variabile, per cui gli oggetti nel database possono essere raggiunti

# semplicemente indicando i loro nomi.

# 3.2.1 Proviamo a richiamare la variabile TCP del dataset lavoro

TCP # ovviamente non trova la variabile perchè è dentro il dataframe lavoro!

lavoro$TCP

# Per poter richiamare la variabile in modo diretto usiamo attach() su lavoro

attach(lavoro)

TCP

# 3.2.2 Dal momento che ho usato attach() sul dataframe, posso richiamare

# la variabile TCP direttamente come se fosse nel Global Environment.

# attach() modifica il percorso di ricerca, diventa quindi necessario, una volta finite

# le operazioni sul dataframe usare detach()

detach(lavoro)

# 3.3 Infine un buon modo per accedere alle variabili di un dataframe

# e usare la funzione with(), che permette un accesso temporaneo al dataframe.

# with(nome.dataframe, {}).

?with

args(with) # l'operazione che voglio compiere sul dataframe va dentro la parentesi graffa.

with(lavoro, {

mean(TCP) + mean(TO)

})

# 3.3.1 Per esempio salviamo in un vettore a i nomi delle Regioni contenute nella

# variabile Reg di lavoro.

a <- with(lavoro, {

Reg

})

a

## # 4. Dataframes - within() ------------------

# 4.1 Come detto in precedenza, le funzioni attach() e with() facilitano l'accesso a un dataframe o

# in generale a una lista, permettendo di accedere alle variabile del dataframe come se

# fossero nello spazio di lavoro.

# Tuttavia queste due funzioni NON PERMETTONO LA CREAZIONE DI NUOVE VARIABILI

# nel dataframe.

# 4.2 Per modificare le variabili dentro il dataframe si puo usare la funzione within()

# Il primo elemento della funzione within e il dataframe.

# Il secondo un'espressione di assegnazione o varie espressioni di assegnazione racchiuse

# fra parentesi graffe {}.

# 4.2.1 Creiamo all'interno del dataset lavoro una nuova variabili

# uguale alla differenza fra i valori della variabili IL e PI

# che chiamiamo prova

lavoro\_1 <- with(lavoro,

{ prova <- IL - PI

}) # ha creato un vettore! ma noi volevamo creare una nuova

# variabile nostro dataset

lavoro <- within(lavoro,

{ prova <- IL - PI

}) # modifico nello stesso dataframe creando la nuova variabile e sovrascrivendo

lavoro$prova

# 4.3 Ovviamente possiamo fare piu operazioni contemporanemante.

# Quello che conta è che:

# 1. le operazioni siano tutte incluse nella graffa

# 2. le varie operazioni NON DEVONO ESSERE SEPARATE DA VIRGOLE

# 4.3.1 Creiamo un nuovo database Goal2 che contiene le variabili centrate

lavoro <- within(lavoro, {

TCP\_centred <- TCP - mean(TCP)

TD\_centred <- TD - mean(TD)

TO\_centred <- TO - mean(TO)

TMPL\_centred <- TMPL - mean(TMPL)

NEET\_centred <- NEET - mean(NEET)

IL\_centred <- IL - mean(IL)

PI\_centred <- PI - mean(PI)

})

## # 5. Dataframes - Recoding ----------------------

# R NON SUPPORTA una delle principali operazioni della preparazione

# dei dati nelle scienze sociali: la RICODIFICA DELLE VARIABILI

# Ci sono sistemi per emulare questo processo ma spesso questo puo diventare abbastanza noioso.

# Alcuni pacchetti permettono invece di usufruire di specifiche funzioni di codifica.

# Fra i principali ci sono car() e memisc(). Per chi fosse appassionato puo vedersi i diversi metodi nel dettaglio!

# Qui ci concentreremo sul sistema più semplice e intuitivo: l'uso della funzione recode della libreria memisc()

library(memisc)

?recode

# Il dataset dati\_AVQ contiene una serie di variabili:

# ETA = anni del soggetto

# SESSO = sesso del soggetto:

# 1 - Maschio

# 2 - Femmina

# STCIV = stato civile:

# 1 - celibe/nubile

# 2 - coniugato/a coabitante con il coniuge; unito/a civilmente

# 3 - separato/a di fatto; separato/a legalmente; divorziato/a; gia' in unione civile (per scioglimento unione)

# 6 - vedovo/a; gia' in unione civile (per decesso del partner)

# ISTR: livello di istruzione:

# 1 - laurea e post-laurea

# 2 - diploma

# 3 - licenza di scuola media

# 4 - licenza di scuola elementare, nessun titolo di studio

# CITT: cittadinanza:

# 1 - ITALIANA

# 3 - STRANIERA

# COND: condizione lavorativa:

# 1 - occupato

# 2 - in cerca di occupazione

# 3 - inattivo

# Andiamo a controllare la natura di queste variabili

str(dati\_AVQ) # sono tutte numeric!

# Possiamo trasformarle usando la funzione recode() in combinazione con within()

#### Codifica usando le funzioni di memisc()

library(memisc)

# SESSO

table(dati\_AVQ$SESSO)

dati\_AVQ <- within(dati\_AVQ,{

Sesso <- recode(SESSO,

1 -> "Maschio",

2 -> "Femmina")

})

dati\_AVQ$Sesso

# Livello Istruzione

table(dati\_AVQ$ISTR)

dati\_AVQ <- within(dati\_AVQ,{

Istr <- recode(ISTR,

1 -> "Laurea",

2 -> "Diploma",

3 -> "Licenza media",

4 -> "Licenza elementare" )

})

dati\_AVQ$ISTR

# STATO CIVILE

table(dati\_AVQ$STCIV)

dati\_AVQ <- within(dati\_AVQ,{

Statciv <- recode(STCIV,

1 -> "Celibe/Nubile",

2 -> "Coniugato/a",

3 -> "Separato/a",

6 -> "Vevovo/a" )

})

dati\_AVQ$Statciv

# Cittadinanza

table(dati\_AVQ$CITT)

dati\_AVQ <- within(dati\_AVQ,{

Citt <- recode(CITT,

1 -> "Italiana",

3 -> "Straniera")

})

dati\_AVQ$Citt

# Condizione professionale

table(dati\_AVQ$COND)

dati\_AVQ <- within(dati\_AVQ,{

Cond <- recode(COND,

1 -> "Occupato",

2 -> "Disoccupato",

3 -> "Inattivo")

})

dati\_AVQ$Cond

# Guardiamo il risultato

View(dati\_AVQ)

str(dati\_AVQ)

# La ricodifica ha creato in automatico dei fattori, cosi da permetterci di trattare le

# varibili come categoriali

dati\_AVQ$Istr # si tratta di una varaibile ordinale!

# ordiniamo dunque il fattore

dati\_AVQ$Istr <-factor (dati\_AVQ$Istr, levels = c("Licenza elementare", "Licenza media",

"Diploma", "Laurea"), ordered = T)

dati\_AVQ$Istr # ORDINATO

# Salviamo

save.image("dati.Rdata")

## # 6. Dataframe: Salvataggio -------------------------------------------------------

# Salvare in csv

write.table(lavoro, "lavoro.csv", sep="," , dec=".", row.names=T, col.names=T )

# Salvare in txt

write.table(lavoro, file="lavoro.txt", quote=T, sep=" ", dec=".", na="NA", row.names=T, col.names=T)

# Salvare in xls

install.packages("openxlsx")

library(openxlsx)

write.xlsx(lavoro, "lavoro.xlsx")

# Salvare in SPSS SAS STATA -> usiamo la libreria foreign()

install.packages("foreign")

library(foreign)

# Salvare per SPSS

write.foreign(lavoro, "lavoro.txt", "lavoro.sps", package="SPSS")

# Salvare per SAS

write.foreign(lavoro, "lavoro.txt", "lavoro.sas", package="SAS")

# Salvare per STATA

write.dta(lavoro, "lavoro.dta")

# # 7. Alcune funzioni statistiche ---------------------------------------------------------------------

# 7.1 Vediamo alcune funzioni per lavorare sui nostri dati

# mean() calcola la media aritmetica di un vettore di dati.

# median() calcola la mediana di un vettore di dati.

# var() calcola la varianza di un vettore di dati, la covarianza tra due vettori,

# o la matrice di varianze e covarianze di una matrice di dati.

# sd() calcola lo scarto quadratico medio (standard error) di un vettore di dati.

# summary() riporta le principali statistiche descrittive di un vettore

# o di una matrice di dati.

# Vediamo alcune applicazioni sul dataset lavoro

summary(lavoro)

mean(lavoro$TCP) # indichiamo la vatriabile usando $

mean(lavoro[,2]) # indichiamo la variabile usando []

median(lavoro$PI)

var(lavoro$TCP) # varianza della variabile TCP

var(lavoro) # matrice di covarianza. Per chi corra eseguire statistiche di alto profilo

sd(lavoro$PI)

# 7.2 Nei nostri lavori è importante spesso riportare una tabella con le

## # statistiche di sintesi dei nostri dati.

# Su R sono disponibili alcuni pacchetti che ci permettono non solo

# di calcolare direttamente le statistiche sintetiche, ma anche di

# realizzare tabelle già formattate in diversi formati (Latex, Text, Html).

# 7.2.1 Una di queste librerie è stargazer. La funzione stargazer() permette

# di calcolare le statistoche sintetiche delle variabili in un dataframe e inoltre

# consente di specificare il tipo di formato che vogliamo per la tabella ottenuta,

# attraverso l'argomento type=c ("latex","html", "text").

# è possibile formattare la tabella come più ci aggrada specificando

# i vari argomenti diponibili.

install.packages("stargazer")

library(stargazer)

?stargazer

stargazer(lavoro, type="text") # Argomento type stabilisce il tipo di tabella che creiamo

stargazer(lavoro, type= "latex")

stargazer(dati\_AVQ, type="html")

# N.B. stargazer elimina in automatico le variabili categoriali e caratteri!

# 7.2.2 Un'altra libreria che permette di creare facilmente tabelle

# di statistiche descrittive è psych.

# di calcolare le statistoche sintetiche delle variabili in un dataframe e inoltre

# consente di specificare il tipo di formato che vogliamo per la tabella ottenuta,

# attraverso l'argomento type=c ("latex","html", "text").

# è possibile formattare la tabella come più ci aggrada specificando

# i vari argomenti diponibili.

library(psych)

sint\_lavoro = describe(lavoro, quant=c(.25,.75))

View(sint\_lavoro)

# crea un dataframe che puo essere esportatom per esempio in xlsx

# N.B. in questo caso, se vogliamo avere anche i nomi delle variabili

# nel file excel, dobbiamo aggiungere rownames fra le variabili

sint\_lavoro <- cbind(rownames(sint\_lavoro), sint\_lavoro)

library(openxlsx)

write.xlsx(sint\_lavoro, "stat\_desc\_lavoro.xlsx")

# Possiamo usare la funzione kable() del pacchetto knitr per esportare in latex ad esempio!

?kable

knitr::kable(sint\_lavoro[,c("min", "max","Q0.25", "median", "Q0.75", "mean", "skew", "kurtosis")],

format = "latex", digits = 3)

# Stiamo sempre attenti alle variabili che trattiamo!

# la funzione describe ha il vaantaggio di segnalaarci con un asterisco le variabili

# categoriali

?describe

describe(dati\_AVQ, quant=c(.25,.75)) # ATTENTI!

# 7.3 Il pacchetto DescTools ci offre alcune funzioni importanti per l'analisi descrittiva

library(DescTools)

library(help = "DescTools")

?Gini # indice di eterogeneità di gini

Gini(table(dati\_AVQ$Sesso))

Gini(table(dati\_AVQ$Statciv))

?Gmean # media geometrica

Gmean(lavoro$IL)

# # 8. Tabulazione -------------------------------------------------------------

# 

# Cerchiamo di vedere come è possibile sintetizzare distribuzioni univariate

# in distribuzioni di frequenza...

# Cos'è una distribuzione di frequenza?...andate a ripassarlo!!!!

## # 8.1 Tabulazione di una variabile categoriale

# La funzione di base per la tabulazione è table()....

# ce se stanno anche altre!

# Prediamo come riferimento la distrbuzione unitaria del Sesso per Tipologia nel dataframe "dato\_AVQ"

# ovvero della variabile (categoriale) SESSO

str(dati\_AVQ$Sesso)

table(dati\_AVQ$Sesso) # ci da quante osservazioni sono state osservate per ciascuna categoria

# l'output non è il massimo ma dopo vedremo come migliorarlo

tabella\_sesso <- table(dati\_AVQ$Sesso) # salviamo la tabella

# cosa contiene? che tipo di oggetto e'? quali caratteristiche ha?

# quante dimensioni?

tabella\_sesso

View(tabella\_sesso) # forse meglio....

str(tabella\_sesso)

mode(tabella\_sesso)

class(tabella\_sesso)

attributes(tabella\_sesso)

names(tabella\_sesso)

# La funzione table(...) in pratica ha:

# 1) Ha estratto le modalita' distinte del fattore

# dati\_AVQ$Sesso

# 2) Ha contato quante unita' hanno quella modalita

# Sistemiamo meglio la nostra tabella

tabella\_sesso <- as.data.frame(tabella\_sesso)

View(tabella\_sesso)

colnames(tabella\_sesso) <- c("Sesso", "Frequenza")

rownames(tabella\_sesso) <- tabella\_sesso[,1]

View(tabella\_sesso)

## # 8.2 Tabulazione di una variabile quantitativa (continua)

# Prendiamo in consderazione la variabile ETA nel dataframe dati\_AVQ

eta <- dati\_AVQ$ETA # per semplicita di scrittura di codice la salviamo in un vettore

table(eta) # FA SCHIFO!!!!! non è proprio di semplice lettura

tabella\_1 <- table(eta)

View(tabella\_1)

# Nel caso di variabili continue per avere una tabella decente dobbiamo

# raggruppare in classi le singole modalita: creare cioe classi di intervalli

?cut # la funzione cut permette di creare intervalli

# specifichiamo il vettore da trasformare in classi

# i breaks sono i limiti delle nostre classi

cut(eta,breaks=c(0,20,30,40,50,60,70,100))

out.cut <- cut(eta,breaks=c(0,20,30,40,50,60,70,100)) # salviamo la nuova variabile in un oggetto

str(out.cut)

class(out.cut) # è un fattore!

table(out.cut)

eta\_bella <- table(out.cut)

View(eta\_bella)

eta\_bella <- as.data.frame(eta\_bella)

colnames(eta\_bella) <- c("Classi di età", "Frequenza")

# Di fatto si fa una vera e propria ricodifica!!

# Prendiamo l'esempio di prima....

# N.B. Non esiste una regola generale per la creazione di classi....

# molto dipende dai dati che abbiamo di fronte

# Per scegliere in questo caso le classi di intervallo ci conviene dare un'occhiata

# sommaria alla distribuzione

tabella <- table(dati\_AVQ$Sesso)

tabella1 <- table(cut(eta,breaks=c(0,20,30,40,50,60,70,100)))

## # 8.3 Come costruire invece una tabella con frequenze relative/percentuali

# N.B. Ricordiamoci sempre che l'oggetto deve essere una tabella non un dataframe!

tabella <- table(dati\_AVQ$Sesso)

tabella1 <- table(cut(eta,breaks=c(0,20,30,40,50,60,70,100)))

#1. a mano

tabella/sum(tabella)

tabella1/sum(tabella\_1)

# oppure

prop.table(tabella)

prop.table(tabella1)

round(prop.table(tabella)\*100,2) # con la funzione round ottengo un numero prefissato da me di cifre decimali

round(prop.table(tabella1),2)

## # 8.4 Tabelle per coppie di variabili: tabelle a doppia entrata

table(dati\_AVQ$ETA, dati\_AVQ$SESSO)

table(dati\_AVQ$STCIV, dati\_AVQ$SESSO)

table(dati\_AVQ$STCIV\_Fact, dati\_AVQ$SESSO\_Fact)

# Un pacchetto interessante per le tabelle

# è il pacchetto epiDisplay

install.packages("epiDisplay", dependencies = T)

library(epiDisplay)

# La funzione tab1(). Ha una serie di argomenti molto utili

# cum.percent che se messo = T vi restituisce in automatico

# le frequenze percentuali

# graph che se messo = T vi restituisce in automatico

# una rappresentazione grafica.

# occorre ricordarsi che l'argomento principale della funzione è la variabile non la tabella!

tab1(dati\_AVQ$Statciv,cum.percent=F,graph=F)

tab1(dati\_AVQ$Statciv,cum.percent=F) # se non specifichiamo graph=F otteniamo anche un grafico adatto!

tab1(dati\_AVQ$Statciv,cum.percent=F,

main = "Distribuzione dello stato civile",

col = c("blue", "red", "orange", "green", "white"),

xlab = "Categorie", ylab = "Frequenze")

?tab1

tab1(dati\_AVQ$Cond,cum.percent=T,

main = "Distribuzione della condizione professionale",

xlab = "Categorie", ylab = "Frequenze")

# Altri interessanti argomenti!

?tab1

tab1(out.cut,

main = "Distribuzione per classi di età",

ylab= "Frequenza",

col = "grey50",

sort.group = "decreasing",

bar.values = "percent",

xlab = "Calssi di età")

# Per le tabelle a doppia entrata usiamo la funzione tabpct()

tabpct(dati\_AVQ$Statciv, dati\_AVQ$Sesso, las=2)

tabpct(dati\_AVQ$Statciv, dati\_AVQ$Sesso,

las=2, cex.axis=1, main="",

xlab="", ylab="",

col=c("blue","purple"))

title(main="Stato civile per età", cex.main=1.8,

xlab="Stato civile",ylab="Sesso", cex.lab=1.5)

# Magari provate a migliorarlo

# # Grafici Introduzione ----------------------------------------------------------------

# R è in grado di produrre diversi tipi di grafico.

# Grazie all'estrema flessibilità del programma è possibile generare

# grafici essenziali o estremamente ricchi, in base alle esigenze dell'utilizzatore.

# E' inoltre possibile inserire più grafici in un'unica finestra grafica.

# I comandi per i grafici possono essere divisi in tre gruppi:

# a. funzioni di alto livello, che creano un nuovo grafico sulla finestra grafica;

# b. funzioni di basso livello, che aggiungono parti ad un grafico già esistente;

# c. funzioni per grafici interattivi, che consentono di aggiungere

# interattivamente informazioni, o di estrarne, da un grafico esistente.

# In R è disponibile una lunga serie di parametri grafici,

# che possono essere modificati dall'utilizzatore.

# La funzione par() permette di settare qualsiasi parametro grafico.

?par # la vedremo piu avanti nel dettaglio

# 9.4 Per vedere una panoramica dei grafici che è possibile realizzare R,

# basta eseguire il comando demo(graphics)

demo(graphics)

# 9.4 Iniziamo banalmente con le coordinate di 200 punti

x <- 1:200

y <- c(1:100,100:1)

# grafico di 20 punti su un piano cartesiano

?plot # funzione principale in R per i grafici

plot(x,y)

plot(y,x)

# nota1: è stata aperta automaticamente una nuova finestra

# (device grafico o dev)

# vedremo successivamente come aprire e gestire più finestre per

# visualizzare in finestre separate rappresentazioni grafiche diverse

# nota2: assenza di titolo e descrizione automatica dell'asse x e y

# attraverso il nome dei vettori usati

# per personalizzare il titolo principale

plot(x,y,main="Grafico cartesiano")

# per personalizzre gli assi si usa xlax per le ascisse e ylab per le ordinate

plot(x,y,main="Grafico cartesiano",xlab="Asse x",ylab="Asse y")

# per aggiungere anche un titolo secondario in fondoal centro

plot(x,y,main="Grafico cartesiano",sub="[dati inventati]",

xlab="Asse x",ylab="Asse y" )

# oppure i titoli possono aggiungersi in un secondo momento come segue

plot(x,y)

title(main="Grafico cartesiano")

title(sub="dati inventati")

# per personalizzare le etichette per la descrizione degli assi

plot(x,y,xlab="x",ylab="y")

plot(x,y,xlab="",ylab="")

# oppure

plot(x,y,main="Grafico cartesiano",sub="[dati fittizi]",

ylab="ordinata",xlab="ascissa")

## # Punti, simboli e dimensioni ------------------------------------------------------------------

# argomento pch specificato con un numero permette di stabilire

# il simbolo da rappresentare

?pch

plot(1:25,pch=17,main="Grafico con un simbolo diverso",sub="[pch=17]")

plot(1:25,pch=16,main="Grafico con un simbolo diverso",sub="[pch=16]")

plot(1:25,pch=25,main="Grafico con un simbolo diverso",sub="[pch=25]")

plot(1:25,main="Grafico con un simbolo diverso",sub="[default]")

a <- 1:25

# si possono usare anche simboli diversi per punti diversi

plot(a,pch=1:25,main="Grafico con alcuni dei simboli possibili")

# argomento pch specificato con un carattere

plot(1:25,pch="&", sub='[pch="&"]',

main="Grafico con un carattere alfanumerico")

plot(1:25,pch="?", sub='[pch="?"]',

main="Grafico con un carattere alfanumerico")

letters

class(letters)

plot(1:25,pch=letters[1:25],main="Grafico con un carattere alfanumerico")

plot(26:50) # per default se manca l'argomento x e vi è un solo

# argomento nella funzione plot questo sarà interpretato come valore/i

# dell'asse y e per default l'asse x vedrà assegnati

# i primi numeri naturali da 1 fino alla lunghezza

# dell'unico argomento di plot(unicoargomento)

plot(100:76, 26:50)

# i caratteri/simboli corrispondenti a ciascun punto rappresentato

# possono essere ingranditi con l'argomento cex

plot(1:25,pch=1,cex=0.5)

plot(1:25,pch=1,cex=5)

plot(1:25,pch=1:25,cex=seq(0.5,3.5,length=25),

main="Grafico con alcuni dei simboli possibili ingranditi da 0.5 a 3.5 volte")

## # Colori -----------------------------------------------------------------

# Continuiamo il nostro esempio per illustrare i colori disponibili

# ed il controllo dei colori sia per i punti rappresentati che per gli altri

# elementi della rappresentazione grafica come il titolo

plot(1:25,pch=16,cex=2)

plot(1:25,pch=16,cex=2,col="orange")

# per usare i colori più comuni è sufficiente specificare come testo

# il corrispondente nome in inglese ad esempio

# L'argomento col.main permette di cambiare il colore del titolo

# col.sub permette di cambiare il colore del sottotitolo

plot(1:25,pch=16,cex=2,main="Simboli di colore arancione e titolo pricipale in blu",

col="orange",col.main="blue",sub="[uso esagerato di colori diversi]",

col.sub="red",font.sub=2,

xlab="", ylab="Vettore teorico")

# per una lista completa usa

colors()

# sono in tutto

length(colors())

plot(1:25,pch=16,cex=2,main="Simboli di colore arancione e titolo pricipale in blu",

col="orange",col.main="violetred",sub="[uso esagerato di colori diversi]",

col.sub="turquoise3",font.sub=2,

xlab="", ylab="Vettore teorico")

# oppure uso i primi numeri interi

plot(1:20,col=1:20,cex=3,pch=16)

# Posso rappresentare punti su un piano cartesiano

# fornendo come unico argomento di plot(...)

# una matrice di elenti di modo "nummeric" (che corrispondono alle coordinate)

mat=cbind(1:5,1:5)

mat

plot(mat)

# per vederli tutti insieme possiamo metterli in una griglia

# 26x26

# ordinandoli per riga partendo dalla riga in basso

expand.grid(1:3,1:2)

plot(expand.grid(1:26,1:26),col=colors(),cex=2,pch=16)

# per vederli tutti insieme possiamo metterli in una griglia

# ordinandoli per riga partendo dalla riga in alto

plot(expand.grid(1:26,26:1),col=colors(),cex=2,pch=16)

plot(1:25,pch=16,cex=2,main="Grafico con colori nei simboli e nel titolo",col="red",

col.main="blue")

# per usare sfumature di grigi si può usare la funzione gray(...)

# che a ben vedere restituisce una rappresentazione del colore con

# uncodice stringa esadecimale

gray(0.5)

# [1] "#808080"

plot(1:20,col=gray(seq(1,0,length=20)),cex=10,pch=16)

# per usare sfumature di grigi si possono usare

# palette(...)

# rainbow(...)

# terrain.colors(...) # per le mappe topografiche

# e loro varianti

plot(1:20,col=rainbow(20),cex=10,pch=16)

# esistono altri modi di definire colori attraverso i canali rgb

# red-green-blue

rgb(1,0,0)

# [1] "#FF0000"

# colore rosso

rgb(0,0,0)

plot(1:3,1:3,col=c(rgb(1,0,0),rgb(0,1,0),rgb(0,0,1)),pch=16,cex=10)

plot(1:3,1:3,col=c(rgb(1,0,0),rgb(0,0.2,0),rgb(0,0,0)),pch=16,cex=10)

colors()

rgb(0.5,0.5,0.5)

# [1] "#808080"

plot(1:3,1:3,col=c(rgb(1,0,0),rgb(0.5,0.5,0.5),rgb(0,0,1)),pch=16,cex=10)

# colore grigio intermedio

# non approfondiremo più di tanto se non per sottolineare che con

# rgb si possono gestire le \*trasparenze\* con l'argomento alpha

plot(1:20,col=rgb(0.5,0.5,0.5,alpha=0.5),cex=10,pch=16)

plot(1:20,col=rgb(0.5,0.5,0.5,alpha=0.1),cex=10,pch=16)

plot(1:20,col=rgb(0.5,0.5,0.5,alpha=1),cex=10,pch=16)

# nota ORA la differente sovrapposizione del colore!!

plot(1:20,col=rainbow(20,alpha=0.5),cex=10,pch=16)

# colori simili

library("RColorBrewer")

display.brewer.all()

?display.brewer.all

?brewer.pal

brewer.pal(9,"Blues")

?rect

# argomento xlim = ... i limiti dell'asse delle ascisse

# un vettore numerico di due elementi:

# il primo è il limite inferiore e il secondo il limite superiore

# argomento ylim = ... i limiti dell'asse delle ordinate

# un vettore numerico di due elementi:

# il primo è il limite inferiore e il secondo il limite superiore

plot(0,0)

plot(0,0,xlim=c(0,9),ylim=c(0,9))

plot(0,0,xlim=c(0,9),ylim=c(0,9),type="n",axes=F,xlab="",ylab="") # eliminare tutto

0:8

i=0:8

rect(0+i,0,1+i,9,border=NA,col=colorRampPalette(brewer.pal(9,"Blues"))(9))

plot(0,0,xlim=c(0,100),ylim=c(0,100),type="n",axes=F,xlab="",ylab="")

i=0:99

rect(0+i,0,1+i,100,border=NA,col=colorRampPalette(brewer.pal(100,"Blues"))(100))

## # Legenda -------------------------------------------------------------

# per vedere come si introduce una legenda e come si posiziona

# usiamo finalmente dei dati veri (il dataset iris)

data(iris)

str(iris)

lunghezza <- iris$Petal.Length

ampiezza <- iris$Petal.Width

head(iris$Species)

head(as.numeric(iris$Species))

specie <- as.numeric(iris$Species)

plot(lunghezza,ampiezza,col=specie+1,pch=16)

plot(lunghezza,ampiezza,col=specie+1,

main="I petali del dataset Iris",pch=16,cex=1,type="p")

specie

colori.scelti <- c("violet","orange","black")

colore.specie <- colori.scelti[specie]

colore.specie

plot(lunghezza,ampiezza,col=colore.specie,main="I petali del dataset Iris",

pch=16,cex=2)

# aggiungliamo una legenda per segnalare quale colore corrisponde

# alle diverse specie di iris

levels(iris$Species)

1:nlevels(iris$Species)

corrispondenza.livellonumerico.specie <- cbind(1:nlevels(iris$Species),levels(iris$Species))

corrispondenza.livellonumerico.specie

?legend

legend(x=1,y=2.5,legend=levels(iris$Species),pch=16,col=colori.scelti)

plot(lunghezza,ampiezza,col=colore.specie,main="I petali del dataset Iris",pch=16,cex=2)

legend(x=1,y=2.5, legend=levels(iris$Species),pch=16,col=colori.scelti,cex=0.5)

plot(lunghezza,ampiezza,col=colore.specie,main="I petali del dataset Iris",pch=16,cex=2)

legend(x=1,y=2.5,legend=levels(iris$Species),pch=16,col=colori.scelti,pt.cex=.5)

plot(lunghezza,ampiezza,col=colore.specie,main="I petali del dataset Iris",pch=16,cex=2)

legend(x=4.481903,y=0.6343091,legend=levels(iris$Species),pch=16,col=colori.scelti,pt.cex=2)

# Puo essere utile capire cove posizionare esattamente la nostra legenda sul grafico.

# per farlo possiamo usare il seguente comando

locator(1)

# restituisce una lista di coordinate x e y individuate dalla posizione

# del mouse al momento del nostro click con il tasto sinistro

plot(lunghezza,ampiezza,col=colore.specie,main="I petali del dataset Iris",pch=16,cex=2)

legend(x=4.879094,y=0.9571784,legend=levels(iris$Species),pch=16,col=colori.scelti,pt.cex=2)

# possiamo sfruttare la funzione locator interattivamente

plot(lunghezza,ampiezza,col=colore.specie,main="I petali del dataset Iris",pch=16,cex=2)

legend(locator(1),legend=levels(iris$Species),pch=16,col=colori.scelti,pt.cex=2)

# ovviamente la funzione locator può servire analogo scopo in altri contesti

# di rappresentazioni grafiche

## # Impostazioni grafiche ---------------------------------------------------

# per memorizzare le impostazoni correnti

oldpar <- par()

str(oldpar)

# per modificarle d'ora in poi con nuove scelte

par(col="blue",bg="lightyellow")

plot(1:10,pch=16,cex=2)

plot(lunghezza,ampiezza,pch=16,cex=2)

par(col="blue",bg="lightyellow", font.main=4)

plot(1:10,pch=16,cex=2)

plot(lunghezza,ampiezza,pch=16,cex=2, main ="PROVA")

# per ritornare alle vecchie impostazioni

par(oldpar)

dev.off()

## # Più grafici in una finestra ---------------------------------------------

# se vogliamo possiamo rappresentare più di un grafico contemporaneamente

# sulla stessa finestra grafica.

par(mfrow=c(1,2)) # una sola riga con 2 colonne per due grafici affiancati

plot(1:20,col=rainbow(20, alpha=0.5),cex=10,pch=16,main="Grafico 1")

# notare le proporzioni dei cerchi colorati

plot(lunghezza,ampiezza,col=colore.specie,main="Grafico 2 \nI petali del dataset Iris",pch=16,cex=2)

dev.off() # per tornare alla modalità classica

plot(1:20,col=rainbow(20, alpha=0.5),cex=10,pch=16,main="Grafico 1")

par(mfrow=c(2,1)) # due righe una colonna

plot(1:20,col=rainbow(20),cex=10,pch=16,main="Grafico 1")

# notare le proporzioni dei cerchi colorati

plot(lunghezza,ampiezza,col=colore.specie,main="Grafico 2 \nI petali del dataset Iris",pch=16,cex=2)

par(mfrow=c(3,2)) # per righe

# attenzione all'ordine di riempimento

for(i in 1:6){

plot(1:10,pch=16,main=i)

title(sub="ciao")

}

par(mfcol=c(3,2)) # per colonne

# attenzione all'ordine di riempimento

for(i in 1:6) plot(1:10,pch=16,main=i)

title(sub="ciao")

dev.off()

## # Gestire spazi interni e esterni al grafico -----------------------------

?par

par(oma=c(5,5,5,5)) # i margini esterni del nostro grafico:

# c(bottom, left, top, right)

plot(lunghezza,ampiezza,col=colore.specie,main="I petali del dataset Iris",pch=16,cex=2,oma=c(14,14,14,14))

legend(x=1,y=2.5,legend=levels(iris$Species),pch=16,col=colori.scelti,pt.cex=2)

par(oma=c(0,0,0,0))

plot(lunghezza,ampiezza,col=colore.specie,main="I petali del dataset Iris",pch=16,cex=2,oma=c(14,14,14,14))

legend(x=1,y=2.5,legend=levels(iris$Species),pch=16,col=colori.scelti,pt.cex=2)

par(oma=c(5,5,5,5))

plot(lunghezza,ampiezza,col=colore.specie,main="I petali del dataset Iris",pch=16,cex=2,oma=c(14,14,14,14))

# per scrivere nell'abbondante spazio esterno al grafico che abbiamo creato

mtext(side=1,line=0,"line 0",outer=T)

mtext(side=1,line=1,"line 1",outer=T)

mtext(side=1,line=2,"line 2",outer=T)

mtext(side=1,line=3,"line 3",outer=T)

mtext(side=1,line=4,"line 4",outer=T)

mtext(side=1,line=-1,"line -1",outer=T)

mtext(side=1,line=-1,"line -1",outer=T,adj=1)

mtext(side=1,line=-1,"line -1",outer=T,adj=0)

# per capirne l'utilità potremmo aggiungere una nota a margine del grafico

par(oma=c(5,0,0,0))

plot(lunghezza,ampiezza,col=colore.specie,main="I petali del dataset Iris",pch=16,cex=2,oma=c(14,14,14,14))

legend(x=1,y=2.5,legend=levels(iris$Species),pch=16,col=colori.scelti,pt.cex=2)

mtext(side=1,line=3,'Nota: si veda la figura la sezione 12.5.3 a pagina 78

\ndel manuale ufficiale "An Introduction to R"',

outer=T,adj=0,col="blue",font=3)

# esiste un altro margine del grafico che è

# quello controllato dall'argomento mar() della funzione par()

par(mfcol=c(3,2),mar=c(4,2,4,2))

# è utile ad esempio quando ci sono tanti grafici insieme

for(i in 1:6){

plot(1:10,pch=16,main=i)

}

# più vicine le due colonne meno vicine le righe

par(mfcol=c(3,2),mar=c(1,1,1,1))

# è utile ad esempio quando ci sono tanti grafici insieme

for(i in 1:6){

plot(1:10,pch=16,main=i)

}

# righe e colonne sono molto vicine

par(mfcol=c(3,2),mar=c(2,2,2,2))

# è utile ad esempio quando ci sono tanti grafici insieme

for(i in 1:6){

plot(1:10,pch=16,main=i)

}

## # Alcune funzioni secondarie comuni ---------------------------------------

dev.off()

plot(lunghezza,ampiezza,col=colore.specie,

main="Grafico 2 \nI petali del dataset Iris",pch=16,cex=2)

grid(col="grey", lty=1.5) # aggiunge una griglia specificando colore e larghezza delle righe

abline(v=3.2, col="black", lwd=3) # aggiunge una linea verticale alla coordinata v =

abline(h=2.1, col="red", lwd=2) # aggiunge una linea orizzontale alla coordinata h =

points(lunghezza, pch=10, col="maroon3") # aggiunge dei punti contenuti nel vettore

box(col="grey", lwd=3) # aggiunge un box

# # Barplot -----------------------------------------------------------------

# Il barplot permette di visualizzare una variabile quantitativa discreta.

barplot(lavoro$TO) # il grafico è un po grezzo. Proviamo a migliorarlo con i suoi argomenti

# Aggiungiamo le etichette, un box e il titolo del grafico

barplot(lavoro$TO,names.arg = lavoro$COD ) # l'argomento names.arg vuole un vettore con le etichette per l'asse delle ascisse

box() # aggiunge un box

title("Tasso di occupazione nelle regioni italiane")

# Possiamo cambiare il colore delle barre usando l'argomento col

barplot(lavoro$TO,names.arg = lavoro$COD , col="cyan") # col è l'argomento per il colore

box() # aggiunge un box

title("Tasso di occupazione nelle regioni italiane")

# Possiamo invertire l'ordine degli assi con l'argomento horiz

barplot(lavoro$TO,names.arg = lavoro$COD , col="cyan", horiz = T) # inverte assi

box() # aggiunge un box

title("Tasso di occupazione nelle regioni italiane")

# è possibile agire su font, colore e orientamento degli assi

# col gli argomenti font.axis, col.axis e las

barplot(lavoro$TO,names.arg = lavoro$COD , col="cyan", horiz = F)

barplot(lavoro$TO,names.arg = lavoro$COD , col="cyan", horiz = F,

font.axis = 2, col.axis = "Red", las = 2 ) # inverte assi

# Valori di las ammissibili

# 0 - Y = Vertical, X = Horizontal

# 1 - Y = Horizontal, X = Horizontal

# 2 - Y = Horizontal, X = Vertical

# 3 - Y = Vertical, X = Vertical

# Possiamo per esempio rappresentare più variabili in un

# barplot. Lo Stacked BarPlot permette di individuare all'interno di ciascun rettangolo

# aree relative a un altra variabile

counts <- table(dati\_AVQ$Sesso, dati\_AVQ$Citt) # creiamo una tabella di frequenza

#delle variabili che ci interessano

counts

# Versione stacked

barplot(counts, main="Distribuzione della popolazione per genere e cittadinanza",

xlab="cittadinanza", col=c("darkblue","cyan"),

ylim = c(0,3000),

legend = rownames(counts))

# Aggiungendo l'argomento beside=T è possibile avere le

# colonne affiancate

## 

## # Versione Grouped

barplot(counts, main="Distribuzione della popolazione per genere e cittadinanza",

xlab="cittadinanza", col=c("darkblue","red"), ylim = c(0,3000),

legend = rownames(counts), beside=TRUE)

# Possiamo aggiungere una serie di elementi aggiuntivi visti in precedenza

grid(col="grey", lty=1.5)

box(col="grey", lwd=3) # aggiunge un box

?barplot

# 

# # Pie chart ---------------------------------------------------------------

# Per le torte si usa la funzione pie

# Prendiamo in considerazione il Fattore relativo alla condizione lavorativa

tabella <- table(dati\_AVQ$Cond)

pie(tabella)

# Possiamo migliorarlo con gli argomenti

pie(tabella,col=rainbow(9),main="Grafico a torta")

par(oma=c(0,0,0,0), bg="grey")

# Possiamo migliorarlo con gli argomenti

pie(tabella,col=rainbow(9),main="Grafico a torta")

barplot(tabella, col=rainbow(9),main="Grafico a barre",

ylim=c(0,1200))

dev.off()

# # Boxplot --------------------------------------------------

# Il boxplot è un grafico statistico che si utilizza per variabili quantitative.

# E’ molto utile per capire se la distribuzione è simmetrica oppure asimmetrica

# e per confrontare la forma di più distribuzioni.

# Ma soprattutto permette di identificare in modo rapido e preciso

# valori anomali e outliers.

#

par(oma=c(2,2,2,2))

boxplot(dati\_AVQ$ETA)

# Aggiungiamo alcuni titoli

boxplot(dati\_AVQ$ETA,col="orange",range=0,main="Distribuzione delle età",

ylab="Età in anni")

# boxplot affiancati

boxplot(lavoro$TO,lavoro$TD, horizontal=T,col=c("red","blue"),

main="Boxplot appaiato",xlab="Percentuali") # Guardate i pallini!

par(oma=c(0,0,0,0))

boxplot(lavoro$TO,lavoro$TD, horizontal=F,col=c("red","blue"),

main="Boxplot appaiato",xlab="Percentuali")

# Aggiungiamo una legenda interattiva

legend(locator(1),legend=c("Tasso occupazione","Tasso disoccupazione"),

col=c("red","blue"),pch=16)

# # Istogramma --------------------------------------------------------------

# Altro grafico importante per la rappresentazione di variabili quantitative è

# l'istrogramma. La funzione per creare istogrammi su R è hist()

?hist

dati\_AVQ$ETA

hist(dati\_AVQ$ETA)

# Aggiungiamo i titoli corretti

hist(dati\_AVQ$ETA,col="orange",

xlab="Età",ylab="Frequenza",

main="Istogramma della distribuzione per età")

hist(dati\_AVQ$ETA,col=gray(seq(1,0,length=10)),

xlab="Età",ylab="Frequenza",

main="Istogramma della distribuzione per età")

# Sistemiamo la lungezza degli assi con le funzioni

# xlim e ylim

par(oma=c(5,1,1,1))

hist(dati\_AVQ$ETA,col=gray(seq(1,0,length=10)),

xlab="Età",ylab="Frequenza",main="Istogramma della distribuzione per età",

ylim =c(0,500), xlim = c(0,100))

# La corretta visualizzazione di un istogramma richiede che

# sull'asse delle ordinate ci sia la densità di frequenza e non la frequenza.

# Per farlo aggiungiamo l'argomento freq=F.

# In questo caso ylim va ridefinito

hist(dati\_AVQ$ETA,col=gray(seq(1,0,length=10)),

xlab="Età",ylab="Densità di frequenza relativa",

main="Istogramma della distribuzione per età",

xlim = c(0,100), ylim=c(0,0.02),

freq=F)

# aggiungiamo un riferimento alla media e alla mediana della distribuzione

# possiamo aggiungere un pallino rosso nelle giuste coordinate

points(mean(dati\_AVQ$ETA),0,pch=16,col="red",cex=1)

# possiamo aggiungere un triangolino rosso nelle giuste coordinate

points(median(dati\_AVQ$ETA),0,pch=17,col=rgb(0,0,1,alpha=0.5),cex=1)

# Aggiungiamo delle linee che esprimono valori significativi

abline(v=median(dati\_AVQ$ETA),lwd=2,col="blue", lty="dotted") # linea retta verticale su tutto l'interno del grafico

abline(v=mean(dati\_AVQ$ETA),lwd=2,col="red",lty= "dotted") # linea tratteggiata

# per fare una griglia usiamo vettori

abline(h=seq(0,100,10),lty="dotted")

abline(v=seq(0,0.02,0.001),lty="dotted",col="blue")

# agiungiamo una scatola

box()

sd(dati\_AVQ$ETA)

text(locator(1), "S.q.m. = 23.825", col="black")

text(locator(1), "ciao", col="red")

# # Scatterplot -------------------------------------------------------------

# Lo scatterplot, o grafico a dispersione, permette la rappresentazione

# di una matrice di dati bi-dimensionale.

# La funzione utilizzata è plot, impiegata anche per molti altre tipologie di grafici.

# La sintassi di base è: plot(x, y, ...) dove

# x e y sono due vettori che rappresentano le coordinate dei punti nel grafico.

# Tra gli altri argomenti della funzione, quelli di uso più comune sono:

# type che specifica il tipo di grafico. Di default il grafico è a punti,

# il che equivale a scrivere "p".

# Altre possibili opzioni possono essere

# type="l", per disegnare delle line

# type="b" per disegnare sia linee che punti.

# L'opzione type="n" è utile per lasciare il grafico vuoto

# e riempirlo successivamente con altre funzioni di basso livello

# Vediamo i possibili type con un esempio

plot(lavoro$TO, lavoro$TD, type="p")

plot(lavoro$TO, lavoro$TD, type="l")

plot(lavoro$TO, lavoro$TD, type="b")

plot(lavoro$TO, lavoro$TD, type="o")

plot(lavoro$TO, lavoro$TD,type="h")

plot(lavoro$TO, lavoro$TD, type="s")

plot(lavoro$TO, lavoro$TD, type="n")

# Tutte le opzioni

x=1:10

y=x

par(pch=22, col="blue") #

par(mfrow=c(2,4))

varie.opzioni = c("p","l","o","b","c","s","S","h")

for(i in 1:length(varie.opzioni)){

titolo = paste("type =",varie.opzioni[i])

plot(x, y, type="n", main=titolo)

lines(x, y, type=varie.opzioni[i])

}

dev.off() # serve a resettare l'ambiente grafico alle impostazioni di partenza

# Adesso facciamo un esempio di scatterplot usando il dataset lavoro

plot(lavoro$TO, lavoro$TD)

# sistemiamo assi e titolo

plot(lavoro$TO, lavoro$TD, main="Scatterplot",

sub = "Occupazione nelle regioni italiane",

xlab = "Tasso di occupazione", ylab = "Tasso di disoccupazione")

# Cambiamo il colore e il tipo di punti

plot(lavoro$TO, lavoro$TD, main="Scatterplot",

sub = "Occupazione nelle reigoni italiane",

xlab = "Tasso di occupazione", ylab = "Tasso di disoccupazione",

pch=8, col="gold")

# è anche possibile inserire una variabile di strato

# per esempio coloriamo i diversi punti in funzione della ripartizione territoriale

lavoro <- within(lavoro,{

Ripartizione <- as.factor(RIP)

})

lavoro$Ripartizione

plot(lavoro$TO, lavoro$TD, main="Scatterplot",

sub = "Occupazione nelle reigoni italiane",

xlab = "Tasso di occupazione", ylab = "Tasso di disoccupazione",

pch=8, col=lavoro$Ripartizione) # la variabile stratificazione deve essere un fattore!

legend(locator(1),legend=levels(lavoro$Ripartizione),

pch=8, col=c(1,2,3,4))

# Colori e punti diversi

plot(lavoro$TO, lavoro$TD, main="Scatterplot",

sub = "Occupazione nelle reigoni italiane",

xlab = "Tasso di occupazione", ylab = "Tasso di disoccupazione",

pch=1:4,

col=lavoro$Ripartizione) # la variabile stratificazione deve essere un fattore!

legend(locator(1),legend=levels(lavoro$Ripartizione),

pch=1:4, col=c(1,2,3,4))

# # Scatterplot 3d ------------------------------------------------------

# Nel caso di 3 variabili e possibile rappresentare graficamente i punti

# in tre dimensioni corrispondenti alle singole unita.

# Si effettua quindi una estensione del diagramma di dispersione (scatterplot),

# relativo al caso di 2 variabili.

# Tuttavia, l'estensione in 3 dimensioni presenta alcuni inconvenienti:

# 1 Lettura non chiara della configurazione dei punti: risulta difficile percepire

# chiaramente le 3 dimensioni corrispondenti alle 3 variabili.

# 2 Se i punti sono numerosi, si possono manifestare sovrapposizioni che creano

# confusione.

# 3 Il metodo non è generalizzabile (a 4 e più dimensioni).

# Per crearlo possiamo usare la funzione scatterplot3d() contenuta nell'omonimo pacchetto.

# Gli argomenti principali della funzione sono:

# x, y e z, le coordinate dei punti;

# box è un parametro logico, il cui valore di default è TRUE,

# che permette di disegnare una scatola attorno al grafico.

# highlight.3d è un parametro logico, il cui valore di default è FALSE.

# Se posto uguale a {\tt TRUE} evidenzia con colori diversi i punti,

# a seconda della loro posizione, rispetto all'asse delle y.

# Maggiore è il valore dell'unità rispetto a questa variabile,

# più scuro sarà il punto.

# type determina il tipo di grafico. Di default è posto uguale a "p".

# Se "h", viene disegnata anche l'altezza del punto,

# rispetto al piano (x, y).

# Inoltre possono essere impiegati argomenti già visti per la funzione plot.

library(scatterplot3d)

scatterplot3d(x = lavoro$TD, y = lavoro$TO, z = lavoro$NEET,

xlab = "Disoccupazione", ylab = "Occupazione", zlab = "NEET",

box = TRUE, highlight.3d = TRUE, type = "p", pch = 16,

main = "Dati sull'occupazione", sub = "Scatterplot 3d")

scatterplot3d(x = lavoro$TD, y = lavoro$TO, z = lavoro$NEET,

xlab = "Disoccupazione", ylab = "Occupazione", zlab = "NEET",

box = F, highlight.3d = TRUE, type = "h", pch = 16,

main = "Dati sull'occupazione", sub = "Scatterplot 3d")

# # Bubbleplot --------------------------------------------------------------

# Il grafico a bolle, o bubbleplot è un grafico bi-dimensionale

# che permette di rappresentare contemporaneamente l'associazione

# tra tre variabili, x, y e z.

# E' il caso di notare che anche il grafico a dispersione

# con la stratificazione delle variabili consente di considerare

# tre variabili, ma la terza deve essere per forza una variabile di strato.

# Il grafico a bolle può essere impiegato per qualsiasi

# ``terna'' di variabili quantitative.

# La terza variabile può essere rappresentata sostituendo

# i punti corrispondenti alle unità con opportuni simboli,

# la cui dimensione è proporzionale ai valori della terza variabile.

# Di solito si considerano circonferenze con superficie proporzionale

# ai valori assunti dalla terza variabile nelle diverse unità.

# Per la generazione del bubbleplot in R si impiega la funzione symbols(), i cui

# argomenti principali sono:

# 1 x e y, le coordinate delle prime due variabili;

# 2 circles che stabilisce la circonferenza delle bolle, proporzionali ai valori

# assunti dalla terza variabile;

# 3 inches che fissa la massima ampiezza delle bolle;

# 4 fg che specifica il colore della circonferenza (di default, nero = 1);

# 5 bg per determinare il colore con cui vanno riempite le bolle

# (di default non viene riempita).

symbols(x = lavoro$TD, y = lavoro$TO, circles = lavoro$NEET,

inches = 0.5, fg = 2,

xlab = "Disoccupazione", ylab = "Occupazione",

main = "Lavoro: disoccupazione e occupazione",

sub = "Simboli proporzionali ai NEET")

# e possibile associare ciascuna unità allo strato cui appartiene.

# In questo caso si utilizzano, in maniera del tutto analoga a quanto visto

# precedentemente, l'argomento fg o l'argomento bg.

symbols(x = lavoro$TD, y = lavoro$TO, circles = lavoro$NEET,

inches = 0.5,

xlab = "Disoccupazione", ylab = "Occupazione",

main = "Lavoro: disoccupazione e occupazione",

sub = "Simboli proporzionali ai NEET",

bg=lavoro$Ripartizione)

legend(locator(1), legend = levels(lavoro$Ripartizione),

pch = rep(16, 4), col = 1:4)

# # Scatterplot matrix ------------------------------------------------------

# 

# Un metodo per la rappresentazione di 3 variabili è la matrice dei diagrammi

# di dispersione (scatterplot matrix) che permette di evidenziare:

# 1 i valori anomali, sia unidimensionali sia bidimensionali;

# 2 La relazione eventualmente esistente tra ogni coppia di variabili, evidenziando

# anche se essa può ritenersi almeno approssimativamente lineare.

# La funzione per la costruzione di una matrice di diagrammi di dispersione in

# R è pairs().

# La funzione si applica ad una matrice di dati quantitativi, per cui vanno

# eventualmente selezionate dalla matrice di dati iniziale solo le colonne

# appropriate.

# Gli argomenti principali della funzione, di uso più comune, sono:

# 1 x, la matrice dei dati;

# 2 labels, che è il vettore dei nomi delle variabili.

# Di default la funzione inserisce i nomi delle variabili nella matrice dei dati;

# 3 altri argomenti, di uso comune, sono identici a quelli visti per le precedenti

# funzioni;

# 4 in particolare, in maniera analoga a quanto visto precedentemente, pch e/o bg

# possono essere impiegati per distinguere le unità

# a seconda del gruppo cui appartengono.

pairs(x = lavoro[,c(3,4,6)], bg = lavoro$Ripartizione,

labels = c("Disoccupazione", "Occupazione", "NEET"))