Corso di Programmazione in R

Appunti di Corso

18 aprile 2025

Indice

1 Introduzione a R

R è un linguaggio di programmazione e un ambiente di sviluppo specificamente progettato per l'analisi statistica e la visualizzazione dei dati. È ampiamente utilizzato da statistici, data scientist e ricercatori per l'analisi e la manipolazione dei dati, la creazione di modelli statistici e la produzione di grafici di alta qualità.

1.1 Caratteristiche principali di R

- È un linguaggio orientato agli oggetti
- È open-source e gratuito
- Ha una vasta comunità di utenti che contribuiscono a migliaia di pacchetti
- Offre potenti capacità di visualizzazione dei dati
- È altamente estensibile
- Supporta diverse interfacce con altri linguaggi (C, C++, Python)

2 Area di lavoro e comandi di base

2.1 Concetti fondamentali

R è orientato agli oggetti, il che significa che possiamo creare entità denominate oggetti di diverso tipo, come numeri, vettori, matrici, funzioni, o strutture più complesse come data frame o liste.

2.2 Operatori di assegnamento

In R, per assegnare un valore a un oggetto, si possono utilizzare due operatori: `<-` o `=`. Sebbene entrambi funzionino, `<-` è l'operatore di assegnamento tradizionale in R ed è generalmente preferito dalla comunità.

```
a <- 1     # Assegnamento preferito in R
c = 1      # Funziona ma    meno utilizzato
```

Listing 1: Operatori di assegnamento

2.3 Nomi degli oggetti

Le regole per la denominazione degli oggetti in R sono:

- Possono contenere lettere, numeri, punti ('.') e trattini bassi ('_')
- Non possono contenere spazi o altri simboli di punteggiatura
- Devono iniziare con una lettera o un punto (se iniziano con un punto, non può essere seguito da un numero)
- R è case-sensitive (distingue tra maiuscole e minuscole)

```
# Validi
a1 <- 1
a.1 <- 1
a_1 <- 1

# Non validi
```

```r
1a <- 1    # Non pu  iniziare con un numero
a-1 <- 1   # Non pu  contenere trattini
```

Listing 2: Nomi di oggetti validi e non validi

2.4 Gestione dell'environment

L'environment di R contiene tutti gli oggetti creati durante una sessione. Ecco alcuni comandi utili per gestirlo:

```r
ls()           # Elenca tutti gli oggetti nell'environment
rm(a)          # Rimuove l'oggetto 'a'
rm(a, b)       # Rimuove gli oggetti 'a' e 'b'
rm(list=ls()) # Rimuove tutti gli oggetti
```

Listing 3: Gestione dell'environment

2.5 Comandi di base e funzioni

R permette di valutare espressioni e assegnare i risultati a oggetti denominati.

```r
x <- 1+1       # Assegna il risultato dell'espressione a 'x'
x              # Visualizza l'oggetto 'x'

# Funzioni matematiche di base
x <- sqrt(2)   # Radice quadrata
z <- exp(x)    # Funzione esponenziale
w <- log(z)+x # Funzione logaritmica naturale
```

Listing 4: Comandi di base

2.6 Operatori aritmetici

R supporta tutti gli operatori aritmetici standard:

```r
1+2+3          # Addizione
2*3            # Moltiplicazione
3/2            # Divisione
2^3            # Elevamento a potenza
(2 + 3) * 4    # Le parentesi controllano l'ordine delle operazioni
```

Listing 5: Operatori aritmetici

2.7 Operatori relazionali e logici

Gli operatori relazionali confrontano valori e restituiscono un valore logico ('TRUE' o 'FALSE'):

```r
x > 10         # Maggiore di
x <= 10        # Minore o uguale a
x == 10        # Uguale a
x != 10        # Diverso da
```

Listing 6: Operatori relazionali

Gli operatori logici combinano valori logici:

```r
x > 5 & x < 10  # AND logico (entrambe le condizioni devono essere vere)
x < 5 | x > 10  # OR logico (almeno una condizione deve essere vera)
!TRUE           # NOT logico (nega il valore)
```

Listing 7: Operatori logici

2.8 Aiuto in R

R offre un sistema di aiuto completo:

```
?sqrt            # Aiuto su una funzione specifica
help(sqrt)       # Equivalente a ?sqrt
?Arithmetic      # Aiuto sugli operatori aritmetici
??plotly         # Ricerca in tutte le documentazioni installate
```

Listing 8: Ottenere aiuto in R

3 Tipi di dati e strutture in R

3.1 Tipi di dati principali

R supporta diversi tipi di dati primitivi:

- **numeric**: numeri decimali (es. 3.14)
- **integer**: numeri interi (es. 1L, dove L indica che è un intero)
- **character**: stringhe di testo (es. "hello")
- **logical**: valori booleani (TRUE o FALSE)
- **complex**: numeri complessi (es. 1+2i)

```
# Numeric
a <- 10
b <- 3 / 10

# Character
d <- "1"
s <- "hello"

# Integer
var <- 2        # numeric per default
var.int <- 2L   # esplicitamente integer con L

# Logical
x <- TRUE
y <- FALSE
```

Listing 9: Tipi di dati primitivi

3.2 Verificare e convertire tipi di dati

R fornisce funzioni per verificare e convertire i tipi di dati:

```
# Verificare il tipo
is.numeric(a)
is.character(s)
is.integer(var.int)
is.logical(x)

# Convertire il tipo
as.numeric("123")     # Da character a numeric
as.integer(3.14)      # Da numeric a integer (tronca a 3)
as.character(42)      # Da numeric a character
as.numeric(TRUE)      # Da logical a numeric (TRUE diventa 1)
```

Listing 10: Verificare e convertire tipi

La funzione `str()` è molto utile per esaminare la struttura di un oggetto:

```
str(var)       # Visualizza tipo e valore
```

Listing 11: Esaminare la struttura

3.3 Vettori

I vettori sono una struttura dati fondamentale in R che contiene elementi dello stesso tipo. Si creano utilizzando la funzione `c()` (combine):

```
# Vettore numerico
num_v <- c(10, 15, 6.4, 3, 18)

# Vettore di interi
int_v <- c(1L, 6L, 10L)

# Vettore logico
log_v <- c(TRUE, FALSE, T, F)  # T e F sono abbreviazioni

# Vettore di caratteri
ch_v <- c("A", "B", "C")
```

Listing 12: Creazione di vettori

3.3.1 Creazione di sequenze

R offre diversi modi per creare sequenze:

```
v <- 1:10                     # Sequenza da 1 a 10
seq(from=1, to=10)            # Equivalente a 1:10
seq(from=1, to=10, by=3)      # Sequenza con incremento 3
seq(from=1, to=10, length=5)  # 5 numeri equidistanti da 1 a 10

# Ripetizione di elementi
rep(1, 15)                    # Ripete 1 quindici volte
rep(c("a","b"), 5)            # Ripete il vettore 5 volte
rep(c("a","b"), each = 5)     # Ripete ogni elemento 5 volte

# Generare nomi di file
paste("file", 1:5, ".txt", sep="")  # file1.txt, file2.txt, ...
```

Listing 13: Creazione di sequenze

3.3.2 Selezione di elementi e operazioni sui vettori

Gli elementi di un vettore possono essere selezionati utilizzando gli indici (che in R iniziano da 1):

```
x <- c(1, 2, 4, 8, 16, 32)
x[3]                 # Seleziona il terzo elemento
x[2:4]               # Seleziona dal secondo al quarto
x[c(1, 3, 5)]        # Seleziona elementi specifici
x[-4]                # Seleziona tutti tranne il quarto

# Operazioni aritmetiche sui vettori
x <- 1:10
x * 2                # Moltiplica ogni elemento per 2
x > 5                # Confronto elemento per elemento

# Operazioni tra vettori
x <- c(10, 20, 30, 40)
y <- c(1, 2, 3, 4)
```

```
x + y                # Somma elemento per elemento
x * y                # Prodotto elemento per elemento

# Selezione con condizioni logiche
x <- 1:8
x[x > 5]             # Seleziona elementi maggiori di 5
x[x >= 7 | x < 5]    # Operatore OR
```

Listing 14: Selezione e operazioni sui vettori

3.3.3 Funzioni utili per i vettori

R dispone di molte funzioni per operare sui vettori:

```
x <- c(3, 8, 2, 5, 7)

length(x)            # Numero di elementi
sum(x)               # Somma degli elementi
mean(x)              # Media degli elementi
max(x)               # Valore massimo
min(x)               # Valore minimo
sort(x)              # Ordina gli elementi
order(x)             # Restituisce gli indici dell'ordinamento
range(x)             # Valori minimo e massimo

# Operazioni logiche
all(x > 0)           # TRUE se tutti gli elementi sono > 0
any(x < 0)           # TRUE se almeno un elemento    < 0
```

Listing 15: Funzioni per vettori

3.3.4 Nomi degli elementi di un vettore

È possibile assegnare nomi agli elementi di un vettore:

```
x <- c(a = 1, b = 2, c = 3)    # Assegnazione diretta
x <- 1:3
names(x) <- c("a1", "b1", "c1") # Assegnazione successiva
```

Listing 16: Nomi degli elementi

3.4 Matrici ed Array

Le matrici sono array bidimensionali che contengono elementi dello stesso tipo.

```
# Creazione di una matrice
mx <- matrix(c(2, 3, 5, 7, 11, 13), nrow=3)
mx <- matrix(c(2, 3, 5, 7, 11, 13), ncol=2)
mx <- matrix(c(2, 3, 5, 7, 11, 13), ncol=3, byrow=TRUE)

# Dimensioni
nrow(mx)         # Numero di righe
ncol(mx)         # Numero di colonne
dim(mx)          # Dimensioni (righe, colonne)

# Nomi di righe e colonne
rownames(mx) <- c("A", "B")
colnames(mx) <- c("a", "b", "c")

# Selezione di elementi
mx[2, 1]         # Elemento in riga 2, colonna 1
mx[2, ]          # Intera riga 2
mx[ , 3]         # Intera colonna 3
```

```r
mx[ , c("a", "b")]  # Colonne per nome
```

Listing 17: Matrici

3.4.1 Array

Gli array sono generalizzazioni delle matrici a più dimensioni:

```r
# Creazione di un array 3D
x <- 1:20
ax <- array(x, dim=c(5, 2, 2))

# Selezione di elementi
ax[3, 2, 1]      # Elemento specifico
ax[ , 2, ]       # Sottomatrice, seconda colonna per ogni piano
```

Listing 18: Array

3.4.2 Operazioni di algebra lineare

R supporta molte operazioni di algebra lineare:

```r
# Prodotto scalare
crossprod(c(1, 2, 3), c(0, 12, 13))

# Prodotto matriciale
a <- matrix(c(1, 2, 3, 4), ncol = 2, byrow = T)
b <- matrix(c(1, -1, 0, 1), ncol = 2, byrow = T)
a %*% b          # Prodotto matriciale
a * b            # Prodotto elemento per elemento

# Altre operazioni
solve(a)         # Matrice inversa
solve(a, b)      # Risolve il sistema di equazioni Ax = b
diag(a)          # Estrae la diagonale
diag(3)          # Matrice identit  3x3
```

Listing 19: Operazioni di algebra lineare

3.5 Liste

Le liste sono collezioni di oggetti che possono essere di tipi diversi:

```r
# Creazione di una lista
x1 <- 1:3
x2 <- c("A", "B", "C", "D", "E")
x3 <- matrix(1:12, nrow=3)

mylist <- list(x1, x2, x3)  # Lista senza nomi
mylist2 <- list(comp1 = x1, comp2 = x2, comp3 = x3)  # Lista con nomi

# Accesso agli elementi
mylist[[1]]            # Primo elemento
mylist2$comp1          # Elemento per nome
mylist[[3]][1, 1]      # Elemento di una matrice in una lista

# Combinazione di liste
newlist <- c(mylist, mylist2)
```

Listing 20: Liste

3.6 Fattori

I fattori sono utilizzati per variabili categoriali, dove ogni categoria corrisponde a un livello:

```
# Creazione di un fattore
country <- c("Italy", "Germany", "France", "Germany", "Italy")
countryf <- factor(country)

# Esaminare i livelli
levels(countryf)

# Modificare i livelli
countryf2 <- countryf
levels(countryf2) <- c("IT", "DE", "FR")

# Specificare l'ordine dei livelli
factor(country, levels = c("Italy", "Germany", "France"))

# Riordinare i livelli
relevel(countryf, "Italy")  # "Italy" diventa il primo livello

# Tabella delle frequenze
table(countryf)
```

Listing 21: Fattori

I fattori sono particolarmente utili nelle analisi statistiche:

```
# Calcolo della media per gruppo
age <- c(47, 44, 40, 38, 36)
tapply(age, countryf, mean)  # Media dell'et  per paese
```

Listing 22: Uso dei fattori nelle analisi

3.7 Data Frame

I data frame sono tabelle bidimensionali in cui le colonne possono essere di tipi diversi. Sono la struttura dati principale per l'analisi statistica in R:

```
# Creazione di un data frame
countryf <- factor(c("Italy", "Germany", "France", "Germany", "Italy"))
age <- c(47, 44, 40, 38, 36)
genderf <- factor(c(1, 1, 2, 1, 2), labels = c("F", "M"))
under40 <- age < 40

dat <- data.frame(Country=countryf, Age=age, Sex=genderf,
                  Under40=under40)

# Esaminare la struttura
str(dat)
head(dat)       # Prime 6 righe

# Accesso agli elementi
dat[3, 2]       # Riga 3, colonna 2
dat[1:3, 2:4]   # Sottomatrice
dat$Country     # Colonna per nome
dat[ , "Age"]   # Colonna per nome
dat[dat$Sex=="F", ]  # Filtro riga

# Aggiungere nuove colonne
dat$Italy <- dat$Country == "Italy"  # Variabile logica
dat$AgePlus1 <- dat$Age + 1          # Nuova variabile
```

Listing 23: Data frame

3.7.1 Manipolazione dei data frame

```
# Rimuovere colonne
dat$Under40 <- NULL

# Combinare data frame
newdat <- cbind.data.frame(dat, under40)

# Sottoinsiemi avanzati
subset(dat, Country=="Italy", select=c(Age, Sex))
```

Listing 24: Manipolazione dei data frame

3.7.2 Gestione dei valori mancanti

R rappresenta i valori mancanti con 'NA':

```
# Identificare valori mancanti
is.na(dat$Age)
which(is.na(dat$Age))

# Calcolare statistiche escludendo NA
mean(dat$Age, na.rm = TRUE)

# Rimuovere righe con NA
na.omit(dat)            # Rimuove tutte le righe con almeno un NA
dat[!is.na(dat$Age), ] # Rimuove solo le righe con NA in Age
```

Listing 25: Gestione dei valori mancanti

4 Importazione ed esportazione dei dati

4.1 Importazione di dati

R permette di importare dati da vari formati:

```
# Impostare la directory di lavoro
getwd()             # Visualizza la directory attuale
setwd("/path/to/directory")  # Cambia directory

# Importare dati CSV
mydata <- read.csv("file.csv")
mydata <- read.csv("file.csv", header=TRUE, sep=";")

# Importare dati da file di testo
mydata <- read.table("file.txt", header=TRUE)
mydata <- read.table("file.txt", header=TRUE, dec=",")

# Importare dati Excel (richiede un pacchetto)
# install.packages("openxlsx")
# library(openxlsx)
# mydata <- read.xlsx("file.xlsx")
```

Listing 26: Importazione dati

4.2 Esportazione di dati

Allo stesso modo, R permette di esportare dati in vari formati:

```
# Esportare in formato di testo
write.table(mydata, file="mydata.txt")
write.table(mydata, file="mydata.csv", sep=",", row.names=FALSE)
```

```r
write.csv(mydata, file="mydata.csv", row.names=FALSE)

# Salvare l'ambiente R
save(mydata, file="mydata.RData")  # Salva un oggetto specifico
save.image(file="workspace.RData")  # Salva tutto l'ambiente
```

Listing 27: Esportazione dati

5 Funzioni in R

Le funzioni sono blocchi di codice riutilizzabili che eseguono specifiche operazioni:

```r
# Definizione base di una funzione
cube <- function(x) {
  y <- x^3
  return(y)
}

# Funzione con valore di default per un parametro
power <- function(x, exp = 2) {
  return(x^exp)
}

# Funzione che restituisce pi  valori
power_info <- function(x, exp = 2) {
  y <- x^exp
  return(list(result = y, input = x, exponent = exp))
}

# Chiamata delle funzioni
cube(3)
power(2, 3)
power(2)         # Usa il valore di default
power_info(2)    # Restituisce una lista
```

Listing 28: Definizione di funzioni

6 Strutture di controllo

6.1 Condizionali: if/else

```r
# Semplice if
x <- 2
if (x < 3) print("x    minore di 3")

# if-else
if (x < 3) {
  print("x    minore di 3")
} else {
  print("x non    minore di 3")
}

# if-else annidati
if (x < 3) {
  print("x    minore di 3")
} else if (x == 3) {
  print("x    uguale a 3")
} else {
  print("x    maggiore di 3")
}
```

```r
# ifelse (vettorizzato)
y <- 1:10
result <- ifelse(y < 5, "piccolo", "grande")
```

Listing 29: Strutture if-else

6.2 Cicli: for, while, repeat

```r
# Ciclo for
for (i in 1:5) {
  print(i)
}

# Ciclo while
i <- 0
while (i < 5) {
  i <- i + 1
  print(i)
}

# Ciclo repeat (richiede break)
i <- 0
repeat {
  i <- i + 1
  print(i)
  if (i >= 5) break
}
```

Listing 30: Cicli

6.3 Modificatori: break e next

```r
# break (esce dal ciclo)
for (i in 1:10) {
  print(i)
  if (i > 5) break
}

# next (salta all'iterazione successiva)
for (i in 1:10) {
  if (i == 3) next
  print(i)
}
```

Listing 31: Modificatori di cicli

6.4 Funzioni apply

La famiglia di funzioni apply offre un'alternativa più efficiente ai cicli, operando su array, matrici e liste:

```r
# apply - applica una funzione alle righe o colonne di una matrice
z <- matrix(1:20, nrow=4)
apply(z, 1, sum)      # Somma per righe (1)
apply(z, 2, mean)     # Media per colonne (2)

# lapply - applica una funzione a ogni elemento di una lista
x <- list(a=1:5, b=6:10)
lapply(x, sum)        # Restituisce una lista

# sapply - come lapply ma semplifica il risultato
```

```r
sapply(x, sum)        # Restituisce un vettore

# tapply - applica una funzione a sottoinsiemi di un vettore
g <- c(1, 1, 2, 2, 2)
tapply(1:5, g, sum)   # Somma per gruppo
```

Listing 32: Funzioni apply

7 Esempi pratici ed esercizi

7.1 Analisi esplorativa di dati

```r
# Caricare un dataset di esempio
data(mtcars)
head(mtcars)

# Riepilogo statistico
summary(mtcars)

# Media, mediana e deviazione standard
mean(mtcars$mpg)
median(mtcars$mpg)
sd(mtcars$mpg)

# Tabelle di frequenza
table(mtcars$cyl)
table(mtcars$cyl, mtcars$gear)

# Correlazione
cor(mtcars$mpg, mtcars$wt)
cor(mtcars[, c("mpg", "wt", "hp", "disp")])
```

Listing 33: Analisi esplorativa

7.2 Esercizi risolti

7.2.1 Esercizio 1: Manipolazione di vettori

```r
# Definire un vettore y con gli elementi 8, 3, 5, 7, 6, 6, 8, 9, 2
y <- c(8, 3, 5, 7, 6, 6, 8, 9, 2)

# Verificare se gli elementi sono minori di 5
y < 5

# Creare un vettore con gli elementi minori di 5
z <- y[y < 5]
z  # Contiene 3 e 2
```

Listing 34: Esercizio 1

7.2.2 Esercizio 2: Conversione di valori logici

```r
# Conversione di valori logici in numerici
x <- c(TRUE, FALSE, TRUE, TRUE)
x + 0  # Converte TRUE in 1 e FALSE in 0
sum(x) # Somma i valori convertiti (conta i TRUE)
```

Listing 35: Esercizio 2

7.2.3 Esercizio 3: Funzione personalizzata per la varianza

```r
# Funzione per calcolare la varianza
my_var <- function(x) {
  n <- length(x)
  m <- mean(x)
  sum((x - m)^2) / (n - 1)
}

# Test della funzione
x <- 1:10
my_var(x)
var(x)  # Confronto con la funzione incorporata
```

Listing 36: Esercizio 3

7.2.4 Esercizio 4: Riepilogo statistico personalizzato

```r
# Funzione per riepilogo statistico di variabili numeriche
my_summary <- function(df) {
  # Identifica colonne numeriche
  numeric_cols <- sapply(df, is.numeric)

  # Inizializza data frame risultato
  result <- data.frame()

  # Applica funzioni statistiche a ogni colonna numerica
  for(col in names(df)[numeric_cols]) {
    stats <- c(
      min(df[[col]], na.rm = TRUE),
      quantile(df[[col]], 0.25, na.rm = TRUE),
      median(df[[col]], na.rm = TRUE),
      mean(df[[col]], na.rm = TRUE),
      quantile(df[[col]], 0.75, na.rm = TRUE),
      max(df[[col]], na.rm = TRUE)
    )

    # Aggiungi al data frame risultato
    temp <- data.frame(
      Variable = col,
      Min = stats[1],
      Q1 = stats[2],
      Median = stats[3],
      Mean = stats[4],
      Q3 = stats[5],
      Max = stats[6]
    )

    result <- rbind(result, temp)
  }

  return(result)
}

# Test della funzione
data(mtcars)
my_summary(mtcars)
```

Listing 37: Esercizio 4

8 Valori speciali in R

R ha diversi valori speciali per rappresentare situazioni particolari:

- **NA** (Not Available): rappresenta un valore mancante
- **NULL**: rappresenta l'assenza di un oggetto
- **NaN** (Not a Number): risultato di calcoli matematici non definiti, come 0/0
- **Inf** e **-Inf**: infinito positivo e negativo

```
# Esempi di valori speciali
0/0           # NaN
1/0           # Inf
-1/0          # -Inf
NA + 1        # NA (i calcoli con NA producono NA)
is.na(NA)     # TRUE
is.null(NULL) # TRUE
```

Listing 38: Valori speciali

9 Pacchetti in R

Una delle maggiori forze di R è la sua vasta collezione di pacchetti, che estendono le funzionalità di base.

9.1 Installazione e caricamento di pacchetti

```
# Installare un pacchetto dal CRAN
install.packages("ggplot2")

# Caricare un pacchetto
library(ggplot2)

# Verificare i pacchetti installati
installed.packages()

# Verificare i pacchetti caricati
search()

# Aggiornare i pacchetti
update.packages()
```

Listing 39: Gestione pacchetti

9.2 Pacchetti essenziali

Alcuni pacchetti essenziali per l'analisi dei dati:

- **dplyr**: manipolazione efficiente dei dati
- **ggplot2**: visualizzazione avanzata dei dati
- **tidyr**: riorganizzazione dei dati in formato tidy
- **readr**: lettura veloce dei file
- **forcats**: gestione dei fattori

- **stringr**: manipolazione delle stringhe
- **lubridate**: gestione delle date

10 Visualizzazione dei dati

R offre potenti strumenti per la visualizzazione dei dati, sia con funzioni base che con pacchetti specializzati.

10.1 Grafici base

```
# Scatterplot
plot(mtcars$wt, mtcars$mpg, main="Peso vs Miglia per gallone",
     xlab="Peso", ylab="MPG", pch=19)

# Istogramma
hist(mtcars$mpg, breaks=10, main="Distribuzione MPG")

# Boxplot
boxplot(mpg ~ cyl, data=mtcars, main="MPG per cilindri")

# Barplot
barplot(table(mtcars$cyl), main="Frequenza cilindri")

# Pie chart
pie(table(mtcars$cyl), main="Distribuzione cilindri")

# Line chart
plot(1:10, cumsum(1:10), type="l", main="Somma cumulativa")
```

Listing 40: Grafici base

10.2 Grafici con ggplot2

Il pacchetto ggplot2 offre un approccio più potente e flessibile alla visualizzazione:

```
# install.packages("ggplot2")
library(ggplot2)

# Scatterplot base
ggplot(mtcars, aes(x=wt, y=mpg)) +
  geom_point() +
  labs(title="Peso vs MPG", x="Peso", y="Miglia per gallone")

# Scatterplot con colori per gruppi
ggplot(mtcars, aes(x=wt, y=mpg, color=factor(cyl))) +
  geom_point() +
  labs(title="Peso vs MPG per cilindri", color="Cilindri")

# Aggiungere una linea di tendenza
ggplot(mtcars, aes(x=wt, y=mpg)) +
  geom_point() +
  geom_smooth(method="lm") +
  labs(title="Peso vs MPG con trend")

# Boxplot
ggplot(mtcars, aes(x=factor(cyl), y=mpg)) +
  geom_boxplot() +
  labs(title="MPG per cilindri", x="Cilindri", y="MPG")

# Istogramma
```

```r
ggplot(mtcars, aes(x=mpg)) +
  geom_histogram(bins=10) +
  labs(title="Distribuzione MPG")

# Faceting (grafici multipli)
ggplot(mtcars, aes(x=wt, y=mpg)) +
  geom_point() +
  facet_wrap(~cyl) +
  labs(title="Peso vs MPG per cilindri")
```

Listing 41: Grafici con ggplot2

11 Analisi statistica in R

R è nato come linguaggio per l'analisi statistica e offre un'ampia gamma di funzioni per questo scopo.

11.1 Statistiche descrittive

```r
# Carica dati di esempio
data(mtcars)

# Riepilogo
summary(mtcars)

# Media, mediana, deviazione standard
mean(mtcars$mpg)
median(mtcars$mpg)
sd(mtcars$mpg)

# Quantili
quantile(mtcars$mpg)
quantile(mtcars$mpg, probs = c(0.1, 0.9))

# Tavole di contingenza
table(mtcars$cyl, mtcars$gear)
prop.table(table(mtcars$cyl)) # Proporzioni

# Statistiche per gruppo
aggregate(mpg ~ cyl, data = mtcars, FUN = mean)
```

Listing 42: Statistiche descrittive

11.2 Test statistici

```r
# t-test
t.test(mtcars$mpg[mtcars$am == 0], mtcars$mpg[mtcars$am == 1])

# ANOVA
anova_result <- aov(mpg ~ factor(cyl), data = mtcars)
summary(anova_result)

# Test chi-quadro
chisq.test(table(mtcars$cyl, mtcars$am))

# Correlazione
cor.test(mtcars$mpg, mtcars$wt)
```

Listing 43: Test statistici

11.3 Modelli statistici

```r
# Regressione lineare
model <- lm(mpg ~ wt + hp, data = mtcars)
summary(model)

# Predizioni
predict(model, newdata = data.frame(wt = 3, hp = 120))

# Diagnostica
plot(model)

# Regressione logistica
glm_model <- glm(am ~ mpg + wt, data = mtcars, family = "binomial")
summary(glm_model)
```

Listing 44: Modelli statistici

12 Consigli avanzati per la programmazione in R

12.1 Utilizzo efficiente della memoria

```r
# Verificare la memoria utilizzata
object.size(mtcars)

# Rimuovere oggetti inutilizzati
rm(list = ls()) # Attenzione: rimuove tutti gli oggetti!

# Garbage collection
gc()
```

Listing 45: Gestione della memoria

12.2 Ottimizzazione dei cicli con vectorization

R è ottimizzato per operazioni vettoriali. Utilizzare funzioni vettorizzate invece di cicli quando possibile:

```r
# Lento (ciclo for)
result <- numeric(1000)
for(i in 1:1000) {
  result[i] <- sqrt(i)
}

# Veloce (vettorizzato)
result <- sqrt(1:1000)

# Confronto dei tempi di esecuzione
system.time({
  result <- numeric(10000)
  for(i in 1:10000) {
    result[i] <- sqrt(i)
  }
})

system.time({
  result <- sqrt(1:10000)
})
```

Listing 46: Vectorization

12.3 Debugging in R

```
# Funzione con bug
buggy_function <- function(x) {
  if(x < 0) {
    return("Negativo")
  } else if(x = 0) {  # Bug: = invece di ==
    return("Zero")
  } else {
    return("Positivo")
  }
}

# Utilizzo del debugger
debug(buggy_function)
buggy_function(1)  # Entra in modalit  debug
undebug(buggy_function)

# Cattura degli errori
tryCatch({
  buggy_function(1)
}, error = function(e) {
  cat("Si    verificato un errore:", conditionMessage(e), "\n")
})
```

Listing 47: Debugging

13 Risorse per approfondire R

13.1 Libri consigliati

- "R for Data Science" di Hadley Wickham e Garrett Grolemund
- "The Art of R Programming" di Norman Matloff
- "Advanced R" di Hadley Wickham
- "ggplot2: Elegant Graphics for Data Analysis" di Hadley Wickham
- "R Cookbook" di Paul Teetor

13.2 Risorse online

- R Project: https://www.r-project.org/
- RStudio: https://rstudio.com/
- CRAN: https://cran.r-project.org/
- R-bloggers: https://www.r-bloggers.com/
- Stack Overflow - tag R: https://stackoverflow.com/questions/tagged/r
- Tidyverse: https://www.tidyverse.org/

14 Conclusioni

R è un linguaggio potente e flessibile per l'analisi statistica e la visualizzazione dei dati. In questo corso abbiamo esplorato le basi della programmazione in R, dalla manipolazione di strutture dati fondamentali come vettori, matrici e data frame, all'utilizzo di funzioni, strutture di controllo e operazioni statistiche.

Grazie alla sua vasta comunità di utenti e alla grande quantità di pacchetti disponibili, R continua a evolversi e a offrire strumenti sempre più potenti per l'analisi dei dati. Padroneggiare R richiede pratica e approfondimento continuo, ma le basi acquisite in questo corso forniscono un solido punto di partenza per sviluppare competenze avanzate nel campo dell'analisi dei dati e della statistica computazionale.