Marco Plebani

Analisi dei dati con R ed RStudio

Primi passi
Modelli lineari
Modelli non lineari
Modelli generalizzati
Modelli ad effetti misti

versione gratuita

Analisi dei dati con R ed RStudio - versione gratuita

Primi passi - Modelli lineari e non lineari - Modelli lineari generalizzati - Modelli ad effetti misti

Marco Plebani

Analisi dei dati con R ed RStudio – versione gratuita

Prima edizione: agosto 2026

Questo libro è stato compilato in RStudio e generato in R con il pacchetto “bookdown” di Yihui Xie et al. (2025).

Copertina: ©Marco Plebani

Nota alla versione gratuita: questa edizione, disponibile gratuitamente in formato pdf all’indirizzo https://data.marcoplebani.com/libro/, si basa sul mio libro “Analisi dei dati con R ed RStudio”. Rispetto all’edizione completa a pagamento, alla versione gratuita mancano alcuni esempi ed alcuni approfondimenti. Le sezioni rimosse dalla versione gratuita sono indicate nel testo.

Potete acquistare il libro completo qui:

- https://www.lulu.com/it/shop/marco-plebani/analisi-dei-dati-con-r-ed-rstudio/ebook/product-yvkp2ng.html (e-book)
- https://www.lulu.com/it/shop/marco-plebani/analisi-dei-dati-con-r-ed-rstudio/paperback/product-je6k8y6.html (in formato cartaceo)

Buona lettura!

- l’autore

INDICE

1 Nota introduttiva

Cara lettrice, caro lettore,

Questo libro è pensato per essere un corso introduttivo all'analisi dei dati. Lo ho sviluppato sulla base della mia esperienza come studente, insegnante, e ricercatore universitario. I miei anni di studente mi hanno fornito una solida formazione teorica nell'ambito della biologia e dell'ecologia quantitativa, offrendomi però poche occasioni per svolgere della ricerca di tipo sperimentale. Come risultato, all'inizio della mia carriera di scienziato mi mancavano gli strumenti pratici per "fare" scienza. Avevo familiarità con la teoria del metodo scientifico e con i princìpi della statistica, ma non sapevo come metterli in pratica.

Questo libro si propone di costruire un ponte tra la teoria della statistica e la pratica dell'analisi dei dati. Il libro si concentra su esempi di tipo biologico, ma descrive procedure analitiche che si possono applicare a una vasta gamma di contesti.

Il processo di prova ed errore è parte integrante dell'apprendimento e della ricerca in generale, e il vostro percorso di apprendimento dell'analisi dei dati non farà eccezione. Proprio per questo il libro è ricco di esempi pratici: la pratica e l'esercizio saranno cruciali affinchè assorbiate efficacemente i contenuti di questo manuale. Con questo libro mi propongo di fornirvi degli strumenti che vi aiuteranno ad avventurarvi nel mondo dell'analisi dei dati in maniera il più possibile indolore e - perchè no? - divertente.

Buona lettura!

Marco

Bergamo, 19 gennaio 2026

2 Requisiti per questo corso

Questo manuale presuppone che abbiate seguito un corso introduttivo di statistica e che abbiate dunque dimestichezza con concetti quali: popolazione statistica, campione statistico, distribuzione di frequenza, media, varianza, modello lineare, test statistico, soglia di significatività, eccetera. Alcuni di questi concetti saranno rivisti in termini pratici/applicati, ma in generale sarà dato per scontato che ne abbiate una conoscenza teorica. Se avete bisogno di un ripasso vi suggerisco i manuali di Underwood (1996) o di Gotelli & Ellison (2012), ma qualunque manuale introduttivo di analisi statistica dovrebbe servire allo scopo.

Questo corso utilizza l'ambiente informatico R come strumento di analisi e visualizzazione dei dati. Per sfruttare al meglio questo manuale non è richiesta nessuna esperienza pregressa con R nè con la programmazione. Dal momento che il linguaggio di programmazione di R è basato sulla lingua inglese, un minimo di familiarità con quest'ultima è vantaggioso ma non indispensabile.

La semplice lettura di questo libro non sarà sufficiente a farvi imparare ad analizzare i dati. Sarà anche necessario che riproduciate gli esempi qui riportati, soffermandovi sul linguaggio di programmazione e divertendovi a "smanettare". I linguaggi di programmazione, per quanto relativamente semplici ed intuitivi quale è R, richiedono pratica per essere appresi, esattamente come lo richiede l'imparare una nuova lingua.

Al di là di questo vi servirà un computer con connessione internet.

3 Contenuti del corso

Questo corso si concentra sull'analisi di dati univariati, ovvero dati in cui ogni osservazione della variabile di risposta è rappresentata da un singolo valore numerico (e.g. lunghezza, peso, ecc.).

Le analisi descritte in questo corso sono di tipo frequentista. L'approccio frequentista all'analisi dei dati stima la probabilità del verificarsi di un evento in base alla frequenza con cui quell'evento è osservato. Questo approccio permette di stimare in che misura il verificarsi di un evento sia correlato ad una o più variabili esplicative (o predittori), di definire una serie di modelli che potrebbero descrivere l'andamento osservato, e di identificare tra questi candidati il modello più probabile.

Il corso si propone di insegnare sia un metodo affidabile di analisi dei dati, sia la sua esecuzione con il programma/linguaggio R. A questo scopo prenderemo in esame interrogativi di ricerca di complessità crescente, in modo da introdurre concetti ed elementi del linguaggio di R in modo graduale e "digeribile".

4 Primi passi

4.1 Cos'è R e come installarlo

L'ambiente informatico R è sia un programma, sia un linguaggio di programmazione orientato verso l'analisi statistica e la rappresentazione grafica dei dati. R è gratuito e disponibile per i sistemi operativi Linux, MacOS e Windows. Creato nel 1993 dagli statistici Ross Ihaka e Robert Gentleman, dalla sua nascita ad oggi R ha conquistato un vasto seguito di utilizzatori nel mondo della ricerca scientifica e dell'analisi dei dati.

Per ottenere R, rechiamoci al sito internet https://cran.r-project.org/ e scarichiamo la versione di R per il nostro sistema operativo. Per installarlo basta fare doppio clic sul file di installazione appena scaricato e seguire le istruzioni. Al momento della scrittura di queste righe R è giunto alla versione 4.5.2 (2025), che è quella su cui si basa questo libro. I contenuti di questo manuale si basano su caratteristiche per lo più stabili e assodate di R e dovrebbero potersi applicare senza grossi intoppi anche a versioni future del programma.

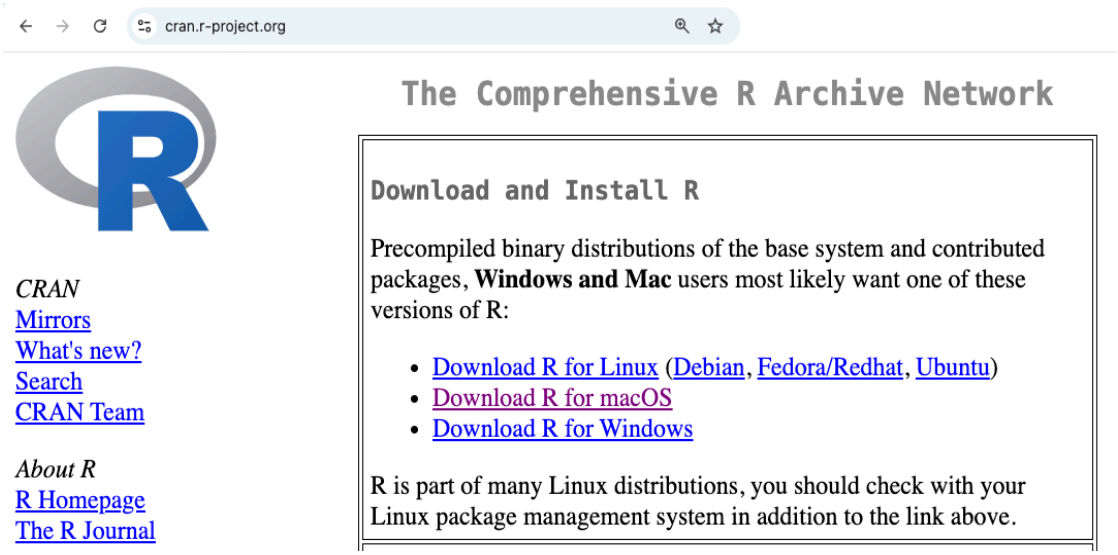

Figura 4.1: Il sito del Comprehensive R Archive Network, dal quale scaricare R.

4.2 Cos'è RStudio e come installarlo

RStudio è una interfaccia che presenta le finestre di R in maniera ordinata ed accessibile ed offre una serie di estensioni che permettono la facile gestione di progetti e la pubblicazione di articoli e relazioni. Questo programma non è essenziale per l'utilizzo di R, ma il suo uso è sempre più diffuso. Potete scaricare gratuitamente RStudio Desktop al sito internet https://posit.co/download/rstudio-desktop/. Per la stesura di questo libro ho utilizzato RStudio nella versione 2023.06.2+561.

4.3 Primi passi con R

Cominciamo con l'aprire R facendo doppio clic sull'icona del programma. Si aprirà una semplice interfaccia di testo chiamata "R console", anche nota semplicemente come *consolle* o "terminale di R". Il terminale è il nostro principale canale di comunicazione con R: vi scriveremo le istruzioni che vogliamo siano eseguite da R stesso.

Proviamo a digitare la seguente operazione aritmetica nella *consolle* e a premere il tasto di invio:

```
1 + 1

## [1] 2
```

Nel terminale appaiono le istruzioni che abbiamo appena fornito a R e, al di sotto, il risultato di queste istruzioni. Ecco alcune altre operazioni di base, che vi invito a sperimentare:

```
5 * 3
2/3
3 * 1 - 2
(3 * 1) - 2
3 * (1 - 2)
```

Per calcolare la potenza di un numero di usa ^:

```
3^2

## [1] 9
```

Per calcolare radici quadrate possiamo usare una apposita **funzione**: `sqrt()`. Il concetto di funzione è cruciale in R: nel linguaggio di R, le funzioni sono i "verbi" che ci permettono di comunicare l'azione che vogliamo eseguire. In R esistono numerosissime funzioni, ognuna con compiti più o meno specifici e con nomi che richiamano il loro compito espresso in inglese: ad esempio, `sqrt()` è l'abbreviazione di *square root.* Le funzioni sono organizzate in "pacchetti" *(packages)*, solitamente organizzati per area tematica. Tra le parentesi della funzione va inserito il suo *argomento*, ovvero l'oggetto su cui vogliamo che la funzione operi. Ad esempio:

```
sqrt(2)

## [1] 1.414214
```

Sapendo il nome di una funzione possiamo richiedere informazioni al riguardo a R digitando `?sqrt` nel terminale. Si apre così una pagina d'aiuto che spiega che `sqrt()` è una di varie "Miscellaneous Mathematical Functions" contenute nel pacchetto `base` (indicato in alto a sinistra nella pagina di aiuto, dopo il nome della funzione e tra parentesi graffe). La pagina di aiuto riporta, tra l'altro: gli argomenti accettati dalla funzione in questione; una lista di dettagli informativi; alcuni esempi di codice eseguibile.

Possiamo salvare il prodotto di una funzione assegnandogli un nome:

```
il.mio.primo.oggetto.R <- sqrt(9)
```

Il simbolo `<-` assegna il nome alla sua sinistra al prodotto delle operazioni alla sua destra. Possiamo leggere il codice nell'esempio come: "`il.mio.primo.oggetto.R` *chiama* `sqrt(9)`". Se eseguiamo quella linea di codice nel terminale, R non scrive più il risultato della funzione, ma la salva come un "oggetto" nella propria memoria. Per visualizzare il risultato salvato come `il.mio.primo.oggetto.R` basta scriverne il nome nel terminale e premere 'invio':

```
il.mio.primo.oggetto.R

## [1] 3
```

Ricordate che i nomi degli oggetti non possono contenere spazi nè iniziare con un numero.

Il vantaggio di salvare il risultato di una funzione come un oggetto nella memoria di R è che rende agevole svolgere ulteriori operazioni sull'oggetto stesso. Ad esempio:

```
il.mio.primo.oggetto.R * 1/2

## [1] 1.5
```

Introduciamo ora una nuova funzione, `c()`, che combina tutto ciò che le forniamo come argomento in un unico gruppo. Gli oggetti prodotti da `c()` sono chiamati **vettori**. Ad esempio:

```
vettore1 <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

L'oggetto `vettore1` è un vettore dei numeri interi che vanno da 1 a 10. Il vantaggio dei vettori è che, se forniti come argomento ad una funzione, quella funzione opera su tutti gli elementi del vettore in questione. Ad esempio, vediamo come calcolare la radice quadrata di tutti i valori contenuti in `vettore1`:

```
sqrt(vettore1)

##  [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490
##  [7] 2.645751 2.828427 3.000000 3.162278
```

Oppure possiamo calcolarne l'esponenziale, con l'apposita funzione `exp()`:

```
exp.vettore1 <- exp(vettore1)
```

Rappresentiamo ora il nostro primo grafico in R per visualizzare la correlazione tra i valori di `vettore1` ed il loro esponenziale:

```
plot(x = vettore1, y = exp.vettore1)
```

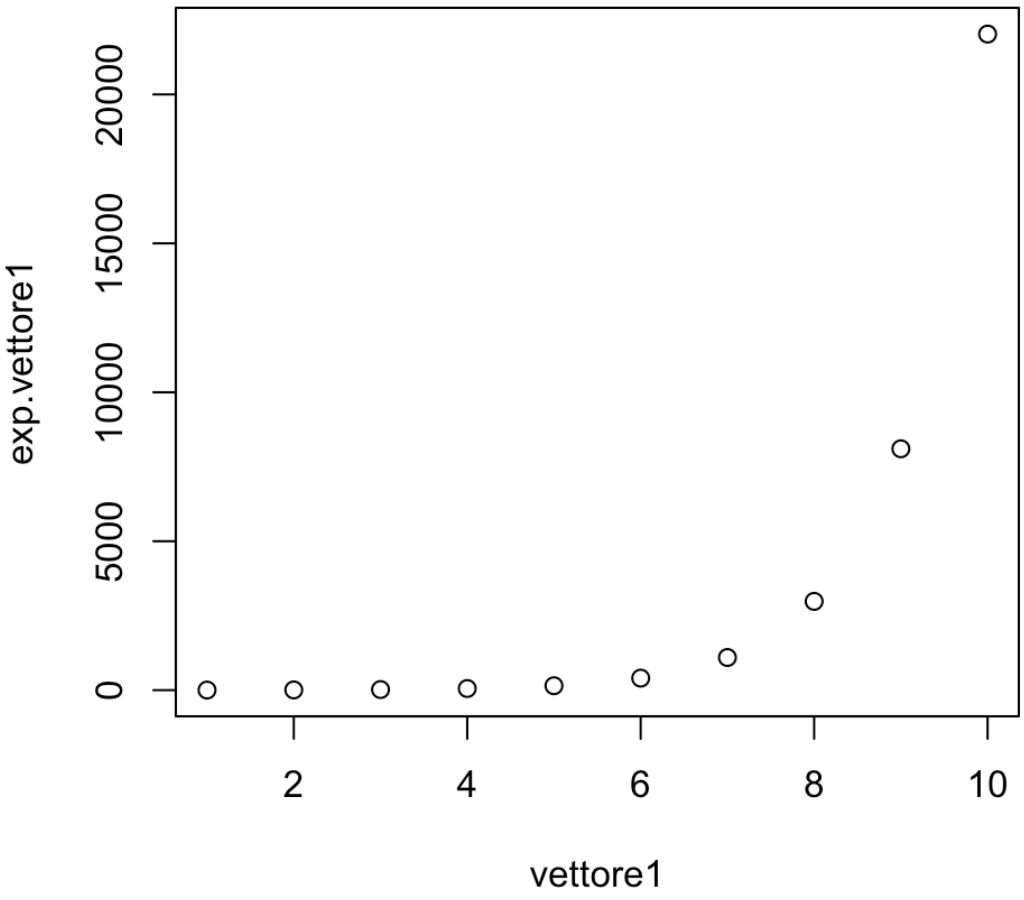

Figura 4.2: Il nostro primo grafico.

Il risultato è in fig. 4.2.

La funzione `plot()` produce grafici di tipo "scatterplot" (in italiano: grafici a dispersione). A questo scopo gli abbiamo fornito due argomenti `x=` e `y=`, che definiscono rispettivamente l'ascissa e l'ordinata dei valori da rappresentare. Chiediamo dettagli sull'uso di `plot()` a R:

```
?plot
```

In questo caso R ci dice che esistono due funzioni chiamate `plot()` contenute in differenti *packages* o pacchetti. È un evento poco comune, ma può succedere. In questi casi, per essere sicuri che R usi la funzione dal pacchetto desiderato, lo si può indicare nel codice come segue:

```
graphics::plot(x = vettore1, y = vettore.esponenziale)
```

Nel caso di `plot()`, la funzione contenuta nel pacchetto `graphics` e quella contenuta nel pacchetto `base` sono identiche.

La pagina di aiuto per la funzione `plot()` nel pacchetto `graphics` indica che, oltre ai valori dell'ascissa `x` e dell'ordinata `y`, c'è poi una quantità di altri argomenti che permettono di definire vari parametri per abbellire il nostro grafico. Vediamone alcuni nel prossimo esempio. Produciamo una nuova sequenza di valori, stavolta usando la funzione `seq()`:

```
vettore2 <- seq(from = 0, to = 5, by = 0.1)
```

La funzione `seq()` permette di produrre sequenze di numeri. Il suo argomento `by=` permette di definire la distanza tra i valori consecutivi creati da `seq()`. Otteniamo ora l'esponenziale di `vettore2`:

```
exp.vettore2 <- exp(vettore2)
```

Produciamo ora un grafico che rappresenti ciascun valore di `exp.vettore2` in relazione a ciascun valore di `vettore2`:

```
plot(x = vettore2,
    y = exp.vettore2,
    type="l", # connette i dati con segmenti
    lwd=2, # Line WiDth, ovvero spessore della linea
    col="red",
    # etichette degli assi:
    xlab="x",
    ylab="exp(x)"
    )
```

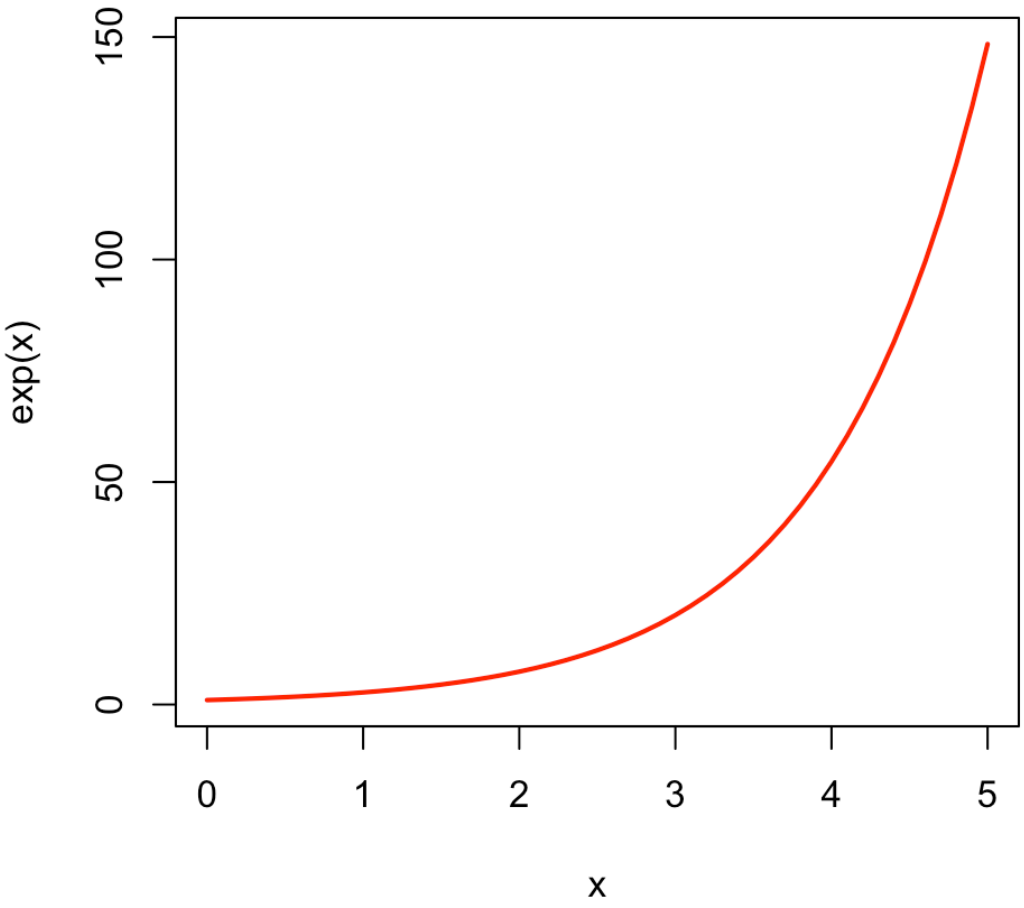

Figura 4.3: Il nostro secondo grafico.

Il risultato è in fig. 4.3. Oltre agli argomenti `x` ed `y`, che indicano a `plot()` su che dati operare, gli abbiamo indicato anche una serie di altri argomenti opzionali che ci hanno permesso di modificare i parametri grafici della figura prodotta. Notate come ho usato il simbolo del cancelletto, `#`, per annotare il mio codice. R ignora le scritte a destra del simbolo `#`, che ci per mette quindi di aggiungere annotazioni al nostro codice. Notate inoltre la formattazione del codice soprastante: ho scelto di andare a capo tra un argomento e l'altro della stessa funzione. Così facendo, il codice si legge più chiaramente. Ricordate che andare a capo non è opzionale ma necessario alla fine di ogni funzione (ovvero dopo la parentesi chiusa) e alla fine di ogni commento indicato da `#`.

Complimenti! siete riusciti ad installare R e ad utilizzarlo con successo, avete cominciato a prendere dimestichezza con le funzioni di R, ed avete prodotto i vostri primi grafici con R. Potete chiudere R.

4.4 Gli “script”

Se riaprite R noterete che tutti gli esercizi che abbiamo svolto fin’ora sono andati perduti. Se provate ad aprire `il.mio.primo.oggetto.R` oppure `vettore1`, R restituirà un errore: gli oggetti in questione non possono essere trovati. Questo succede perchè il terminale di R serve ad eseguire comandi, non è un luogo per salvare codice ed oggetti di R a lungo termine. Per salvare i comandi che fornite a R dovete scriverli all’interno di un documento chiamato **script**. Per creare un nuovo script aprite R, cliccate su “File” e, nel menù a tendina, cliccate su “New Document”. Se vi trovate in RStudio potete fare lo stesso cliccando su *File > New File > R Script*. Si aprirà uno script vuoto dove potete scrivere tutto il codice che vi serve. Per eseguire codice da uno script nel terminale di R selezionate il codice di interesse e premete:

- *Ctrl+R* in Windows;
- *cmd+Invio* in MacOS;
- Ctrl+Invio o cmd+Invio in RStudio.

Salvate lo script con un nome a piacere. Il risultato sarà un file di tipo `.R`. Per aprire uno script basta fare doppio click sul file ed esso si aprirà automaticamente in R. Potete anche scegliere RStudio come programma predefinito per aprire file di tipo `.R`. Potete anche selezionare gli script da aprire aprendo R e cliccando su *File > Open Document…*, oppure aprendo RStudio e cliccando su *File > Open File…*.

Usare gli script ha numerosi vantaggi:

- potete salvare il lavoro svolto e continuare a lavorarci in seguito;
- potete tornare sui vostri passi per svolgere controlli, modifiche e correzioni con facilità;
- uno script documenta il vostro processo di lavoro in modo trasparente;
- gli script possono essere condivisi;
- col tempo accumulerete una vostra personale “biblioteca” di script, e ogni qual volta dovrete svolgere delle operazioni che avete già svolto in passato potrete semplicemente copiare e incollare frammenti di codice da uno script all’altro e apportare le modifiche necessarie senza dover rifare tutto il lavoro da zero.

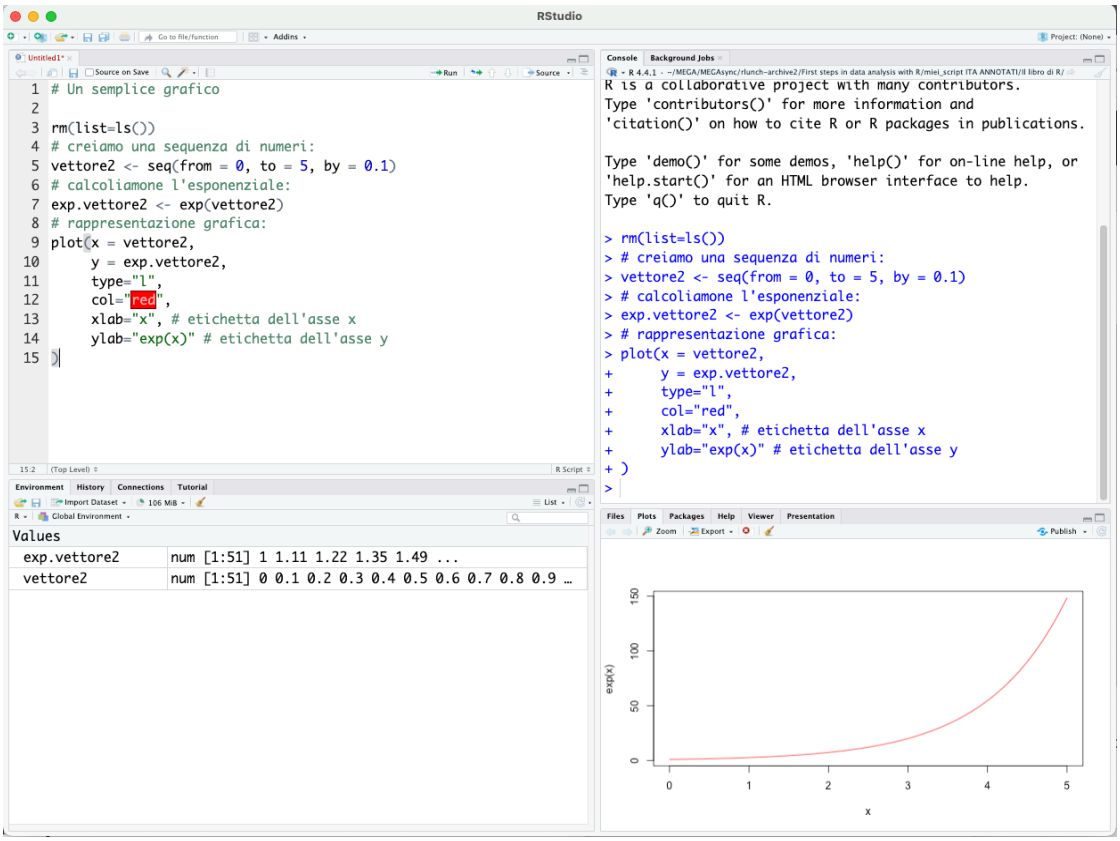

Figura 4.4: RStudio con la sua tipica suddivisione in quattro quadranti. Partendo dal quadrante in alto a sinistra, in senso orario si vedono: uno script aperto; il terminale di R con delle righe di codice già eseguite; il quadrante Environment, che mostra gli oggetti attivi; una finestra grafica con il grafico prodotto dal codice appena eseguito.

4.5 Impostare uno script

Suggerisco di iniziare i vostri script come segue:

```
### TITOLO
## Breve descrizione del contenuto
## data di inizio dello script
## nome e indirizzo email
```

Come già accennato, R ignora qualunque cosa a destra del cancelletto #, per cui lo si può usare per aggiungere commenti, annotazioni e promemoria ai nostri script. R li ignorerà, ma serviranno da riferimento a noi o a chiunque si trovi ad utilizzare il nostro codice. Commentate ampiamente i vostri script! All'inizio vi servirà per imparare, poi vi servirà per ricordare a voi stessi cosa avete fatto nel caso doveste rimettere mano a uno script dopo qualche tempo.

Oltre a titolo, descrizione e nome dell'autore, i miei script solitamente iniziano con:

```
rm(list = ls())
```

Questo pezzo di codice serve per assicurarsi che non ci sia nulla di salvato nella memoria temporanea di R, detta *workspace*. La funzione `rm()` serve per rimuovere oggetti e richiede una lista di oggetti come argomento. La funzione `ls()` restituisce una lista degli oggetti presenti nella memoria temporanea di R. Ho detto in precedenza che, ogni volta che chiudiamo R, tutto ciò che è contenuto nella sua memoria viene dimenticato. Questa è la sua impostazione di default, che può essere modificata cliccando su *R* > *Preferences* > *Startup* in R o su *Tools* > *Global Options…* > *General* in RStudio. Io preferisco assicurarmi che la memoria di R sia "pulita" all'inizio di ogni sessione di lavoro, per evitare confusione.

4.6 Archiviare gli script: le cartelle di lavoro

Col tempo vi troverete a destreggiarvi tra molteplici script per ciascun progetto di analisi. Vi consiglio di creare una cartella per ciascun progetto, dentro la quale archivierete tutti gli script relativi ad esso. R chiama queste cartelle *working directories*, ovvero "cartelle di lavoro". Ad esempio, io ho chiamato lo script che abbiamo appena creato `ilmioprimoscript.R` e lo ho salvato in una cartella che ho chiamato "Corso_R". La cartella "Corso_R" sarà la cartella di lavoro dove salverò tutti gli script creati nel corso di questo libro.

5 Confrontare gruppi di dati usando il t test

5.1 Confronto tra due popolazioni

Nel capitolo precedente abbiamo mosso i primi passi nell'ambiente di R. Ora vediamo come utilizzare R per un semplice esempio di analisi dei dati. Supponiamo di avere due popolazioni di animali, *Popolazione 1* e *Popolazione 2*, e di voler sapere se le masse corporee individuali differiscono significativamente nelle due popolazioni. Per operare questo confronto, l'ideale sarebbe misurare ogni singolo individuo delle popolazioni di interesse, calcolare le loro distribuzioni di frequenza, e calcolare in che misura queste distribuzioni di frequenza si sovrappongono. In realtà misurare tutti gli individui delle popolazioni di interesse non è quasi mai fattibile, per cui ci si limita a misurare dei campioni *rappresentativi* delle popolazioni di interesse. In questo caso immaginiamo di avere misurato venti individui per ciascuna popolazione, ottenendo i seguenti valori (in chilogrammi):

```
rm(list = ls())
gruppo1 <- c(10.06, 9.47, 9.25, 9.72, 10.16, 9.91, 9.47, 10.57,
    8.87, 9.62, 8.97, 8.92, 10.94, 10.14, 9.29, 9, 9.6, 9.71,
    8.88, 9.26)

gruppo2 <- c(19.42, 19.29, 21.71, 20.26, 18.42, 20.61, 18.86,
    20.91, 20.22, 20.44, 20.23, 20, 21.71, 20.7, 20.63, 19.64,
    21.84, 21.25, 19.63, 19.56)
```

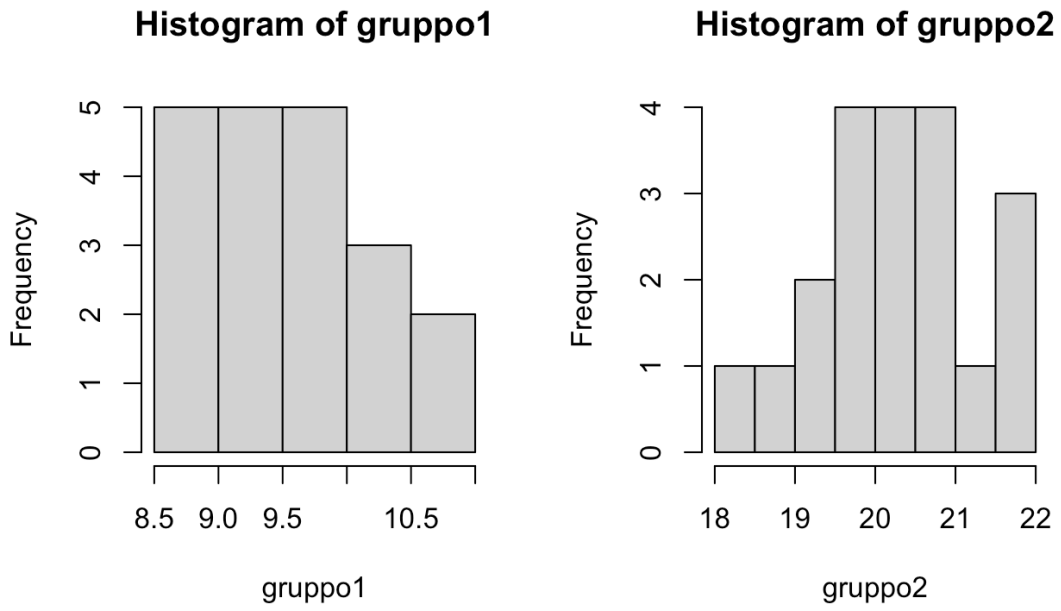

Figura 5.1: Distribuzioni di frequenza delle masse corporee osservate in `Popolazione1` *e* `Popolazione2` *sulla base dei campioni raccolti.*

Cominciamo con il visualizzare i dati raccolti. Usiamo `hist()` per creare degli istogrammi:

```
# definiamo una finestra grafica con una riga e due colonne:
par(mfrow=c(1,2))
hist(gruppo1)
hist(gruppo2)
```

La funzione `par()` permette indicare una serie di parametri da applicare ai grafici che la seguono. Uno degli argomenti di `par()`, "`mfrow=`", permette di creare una finestra grafica suddivisa in righe e/o colonne. I grafici creati successivamente vengono inseriti nei pannelli della finestra grafica così creata.

Osservando gli istogrammi in fig. 5.1 possiamo vedere che i dati raccolti per *Popolazione 1* vanno da ~8.5 Kg a ~11 Kg, mentre quelli raccolti per *Popolazione 2* vanno da ~18 Kg a ~22 Kg.

Per un confronto più agevole ed un risultato grafico esteticamente più appagante possiamo anche rappresentare i due istogrammi sullo stesso sistema di assi. In primo luogo ricreiamo l'istogramma per `gruppo1`, assicurandoci che l'ascissa (l'asse delle *x*) includa sia i valori di `gruppo1` sia i valori di `gruppo2`:

```
par(mfrow=c(1,1))
hist(gruppo1,
     col="grey90", # colore dell'istogramma
     xlim=c(5, 25), # limiti dell'asse x
     xlab="massa corporea [Kg]", # etichetta dell'asse x
     ylab="Numero di individui", # etichetta dell'asse y
     main="Popolazione 1 vs. Popolazione 2" # titolo del grafico
    )
```

In questo caso, oltre ad indicare alla funzione `hist()` i dati da usare, ho usato una serie di altri argomenti che permettono di raffinare i dettagli del grafico.

Aggiungiamo ora il secondo istogramma al medesimo sistema di assi usando, all'interno della funzione `hist()`, l'argomento `add=TRUE`:

```
hist(gruppo2, add = TRUE, col = "grey60")
```

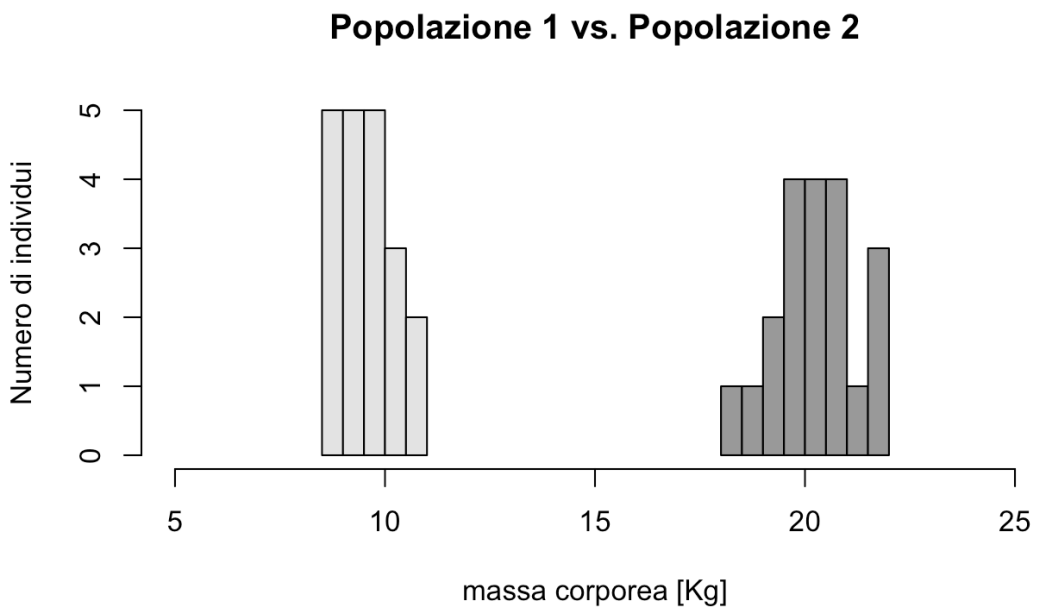

Figura 5.2: Distribuzioni di frequenza delle masse corporee osservate in `Popolazione1` *(in grigio chiaro) e* `Popolazione2` *(in grigio scuro) rappresentate sullo stesso sistema di assi cartesiani.*

La figura 5.2 mostra che le distribuzioni di massa corporea osservate per i due campioni sono ben distinte. Possiamo misurare la significatività statistica di questa differenza usando il test t, che valuta l'ipotesi secondo la quale le due popolazioni da cui provengono i nostri campioni hanno la stessa media. In R questo test può essere eseguito usando la funzione `t.test()`:

```
t.test(gruppo1, gruppo2)  # test t di Welch

##
##  Welch Two Sample t-test
##
## data:  gruppo1 and gruppo2
## t = -43.175, df = 31.445, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -11.18003 -10.17197
## sample estimates:
## mean of x mean of y
##    9.5905   20.2665
```

La funzione `t.test()` restituisce:

- il valore di t, che rappresenta la differenza tra le medie campionarie standardizzata in base alle loro varianze, ed i corrispondenti gradi di libertà *(degrees of freedom,* `df`);
- il *p-value*, ovvero la probabilità di osservare una differenza tra le medie campionarie uguale o maggiore a quella osservata nel caso in cui l'ipotesi nulla fosse vera. L'ipotesi nulla afferma che non c'è differenza tra le medie delle popolazioni a confronto. In questo

caso, se l'ipotesi nulla fosse vera, la probabilità di osservare una differenza tra le medie campionarie uguale o maggiore di quella osservata sarebbe meno di una su 10^{16};

- i limiti dell'intervallo di confidenza al 95% per la differenza tra le medie campionarie, pari a -11.18 e -10.17. Questo significa che, se ripetessimo lo studio un numero infinito di volte, la differenza tra le medie campionarie cadrebbe tra questi due valori nel 95% dei casi;
- le medie campionarie per i due gruppi.

Potreste aver notato che i risultati della funzione `t.test()` vanno sotto il titolo di *"Welch Two Sample t-test"*, ovvero test t di Welch. A differenza del più familiare test t di Student, che assume che le popolazioni a confronto abbiano la stessa varianza (assunzione di *omoscedasticità* o omogeneità delle varianze), il test t di Welch non si basa su questa assunzione ed è dunque di più ampia applicazione. È comunque possibile eseguire un test t di Student, che si basa sull'assunzione di omoscedasticità, aggiungendo l'argomento `var.equal=TRUE` (scrivere `var.equal=T` è sufficiente):

```
t.test(gruppo1, gruppo2, var.equal = T)  # test t di Student

##
##  Two Sample t-test
##
## data:  gruppo1 and gruppo2
## t = -43.175, df = 38, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -11.17658 -10.17542
## sample estimates:
## mean of x mean of y
##    9.5905   20.2665
```

In questo caso i test t nella versione di Student e in quella di Welch producono lo stesso risultato: le masse corporee medie degli individui delle due popolazioni a confronto, stimate sulla base delle medie campionarie, differiscono significativamente l'una dall'altra.

5.2 Confronto tra una media campionaria e un valore di interesse

La funzione `t.test()` può anche essere usata per valutare se il valor medio della variabile di interesse per un determinato campione sia significativamente diverso da un determinato valore μ. Di default, R valuta se la media della massa corporea nella popolazione in esame differisce significartivamente da 0:

```
t.test(gruppo1)

##
##  One Sample t-test
##
## data:  gruppo1
## t = 74.407, df = 19, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  9.320724 9.860276
## sample estimates:
## mean of x
##    9.5905
```

La media delle masse corporee misurate in `gruppo1` differisce significativamente da zero con $p < 0.001$. E se volessimo confrontare la media delle masse corporee misurate in `gruppo1` rispetto a 9.5 Kg?

```
t.test(gruppo1, mu = 9.5)

##
##  One Sample t-test
##
## data:  gruppo1
## t = 0.70213, df = 19, p-value = 0.4911
## alternative hypothesis: true mean is not equal to 9.5
## 95 percent confidence interval:
##  9.320724 9.860276
## sample estimates:
## mean of x
##    9.5905
```

La media della massa corporea per la Popolazione 1, stimata in base ai valori di `gruppo1`, *non* differisce significativamente da 9.5 con $p = 0.491$.

5.3 Confronto per dati appaiati

A volte si vogliono confrontare gruppi di dati che sono per loro natura *appaiati*. Supponiamo, ad esempio, che i dati misurati in `gruppo1` e `gruppo2` nella sezione precedente non si riferissero ad individui differenti di popolazioni differenti, ma alle masse corporee di un gruppo di individui misurati all'inizio e alla fine di un trattamento sperimentale, come ad esempio un cambio di dieta:

```
prima <- gruppo1
dopo <- gruppo2
```

Potremmo semplicemente confrontare la massa corporea media del campione di individui prima e dopo il trattamento sperimentale, ma così facendo si ignorerebbe il fatto che le misurazioni non sono indipendenti. Esaminare la media delle variazioni di massa corporea in ciascuno degli individui studiati permette di prendere in considerazione la non-indipendenza delle osservazioni pre- e post-trattamento sperimentale. Possiamo integrare questa informazione nell'analisi con l'argomento `paired=TRUE`:

```
t.test(x = prima, y = dopo, paired = TRUE)

##
##  Paired t-test
##
## data:  prima and dopo
## t = -48.98, df = 19, p-value < 2.2e-16
## alternative hypothesis: true mean difference is not equal to 0
## 95 percent confidence interval:
##  -11.13221 -10.21979
## sample estimates:
## mean difference
##         -10.676
```

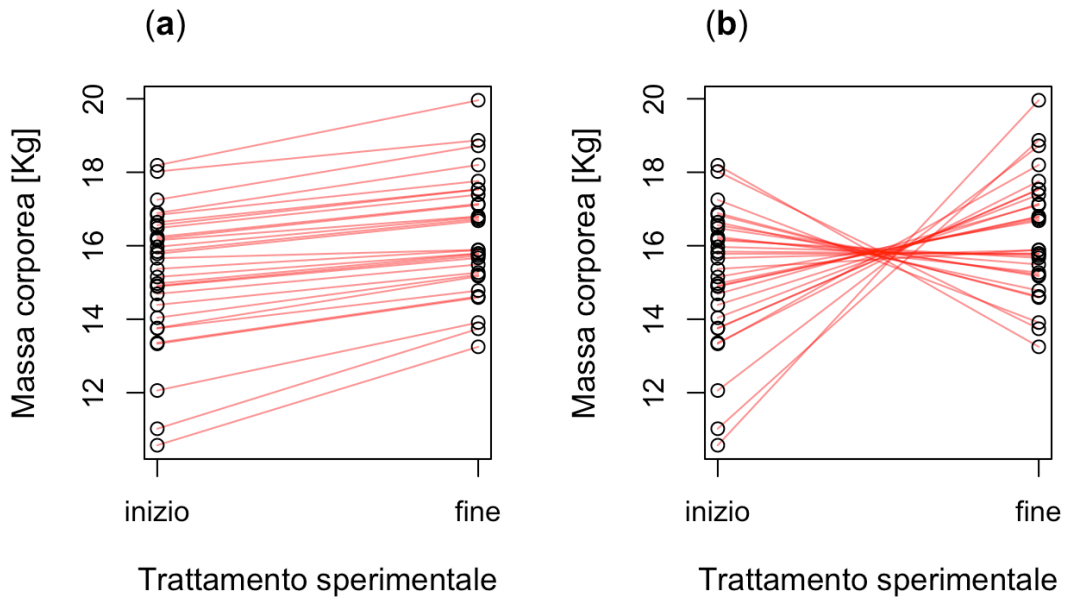

Figura 5.3: Due possibili risposte della massa corporea di un gruppo di individui ad un trattamento sperimentale. Le linee uniscono la massa corporea di ciascun individuo all'inizio e alla fine del trattamento sperimentale. La massa corporea individuale media all'inizio e alla fine del trattamento sperimentale è uguale nei due casi, ma nel caso (a) tutti gli individui rispondono similmente al trattamento sperimentale, mentre nel caso (b) le risposte variano ampiamente.

In questo caso i due gruppi campionari sono talmente differenti tra loro (fig. 5.2) che il test t dà risultati simili, a prescindere da che si prenda in considerazione la non-indipendenza delle osservazioni oppure no. La figura 5.3 mostra due esempi più sfumati. In figura 5.3a-b, le linee uniscono la massa corporea di ciascun individuo all'inizio e alla fine del trattamento sperimentale. I valori misurati all'inizio e alla fine del trattamento sperimentale sono identici nello scenario mostrato in figura 5.3a e in figura 5.3b, ma nel caso mostrato in figura 5.3a tutti gli individui hanno risposto similmente al trattamento sperimentale, aumentando in massa corporea; al contrario, nello scenario mostrato in figura 5.3b le risposte degli individui al trattamento sperimentale differiscono ampiamente per direzione e magnitudine. Se confrontassimo le distribuzioni delle misure di massa corporea all'inizio e alla fine dell'esperimento nel caso (b) considerandole indipendenti potremmo erroneamente dedurre che il trattamento sperimentale ha prodotto un aumento significativo nella massa corporea degli individui in studio. Prendendo in considerazione la non-indipendenza delle misurazioni con un test t per dati appaiati noteremmo invece che la media delle variazioni in massa corporea tra la fine e l'inizio dell'esperimento non è significativamente diversa da zero: questo perchè le risposte dei singoli individui al trattamento sperimentale sono spesso in controtendenza tra loro. Vi invito a verificare queste affermazioni voi stessi. I valori utilizzati per produrre la figura 5.3b sono i seguenti:

```
casoB_prima <- c(10.57, 11.02, 12.06, 13.33, 13.36, 13.75, 13.76, 14.04, 14.3
9, 14.69, 14.89, 14.91, 14.97, 15.15, 15.37, 15.66, 15.78, 15.84, 15.97, 16.1
5, 16.19, 16.24, 16.48, 16.56, 16.64, 16.84, 16.89, 17.25, 18.02, 18.19)

casoB_dopo <- c(19.96, 18.87, 18.72, 18.20, 17.76, 17.54, 17.53, 17.39, 17.14,
 17.11, 16.80, 16.78, 16.73, 16.68, 15.89, 15.88, 15.79, 15.78, 15.73, 15.67,
 15.49, 15.27, 15.21, 15.17, 14.78, 14.62, 14.59, 13.91, 13.74, 13.25)
```

Potete confrontarli utilizzando la funzione `t.test()` con argomento `paired=TRUE` e `paired=FALSE`. Noterete che, in base alla consueta soglia di significatività del 5% ($p \leq 0.05$), il test t per dati appaiati ci porta a non rigettare l'ipotesi nulla per la quale la differenza reale tra le medie all'inizio e alla fine dell'esperimento è uguale a zero. Al contrario, con il test t classico per dati indipendenti concluderemmo che c'è una differenza significativa tra la massa corporea media degli individui all'inizio e alla fine dell'esperimento.

6 Confrontare due gruppi di dati usando la funzione `lm()`

Nel capitolo precedente abbiamo imparato a confrontare due gruppi di dati con un test t. Per svolgere il test in R abbiamo digitato i dati direttamente, creando due vettori con la funzione `c()`, e li abbiamo confrontati con la funzione `t.test()`. Di solito, però, i dati da analizzare sono salvati in un file. In questo capitolo vedremo come inserire in R dei dati salvati in un file esterno. Parleremo inoltre di alcune regole di "buona etichetta" per compilare i nostri dataset in modo che R (e i software di analisi dei dati in generale) li possa leggere ed analizzare facilmente e senza errori. Introdurremo infine la funzione `lm()`, una funzione versatile e potente per analizzare modelli lineari.

6.1 Inserire dati in R

Potete scaricare i dataset che utilizzeremo in questo libro visitando l'indirizzo: https://github.com/marcoplebani85/datasets/ e cliccando su: *Code* > *Download ZIP*. Decomprimete il file `.zip` che avete scaricato e salvate la cartella così ottenuta all'interno della vostra *working directory* (ne abbiamo parlato nella sezione 4.6). Negli esempi che vedremo, i dataset saranno salvati nella cartella *"my_data"* all'interno della cartella *"Corso_R"*. Vi consiglio di usare questo approccio per tutti i vostri progetti di analisi: create una cartella per ciascuno di essi, e all'interno di essa create un'altra cartella dove salvare i dati relativi a quel progetto.

In questo capitolo ci occuperemo di confrontare la massa corporea di due ipotetiche specie animali, *Species A* e *Species B*. Useremo i dati all'interno del file `compare_2_species.csv`, che contiene le misure di massa corporea per 50 individui di ciascuna specie in Kg. Come nel capitolo precedente vogliamo confrontare le distribuzioni delle masse corporee individuali nelle due specie e vedere se sono significativamente differenti. Procediamo a caricare il set di dati in R con la funzione `read.csv()`:

```
rm(list=ls()) # ripulisce la memoria di R
df <- read.csv("~/Desktop/Corso_R/my_data/compare_2_species.csv")
```

Mi raccomando di sostituire la directory del file dei dati con quella che avete scelto sul vostro computer. Inoltre, se usate R in Windows bisogna assicurarsi di sostituire "\" con "/". Ad esempio, se la mia working directory si trovasse sul desktop di un computer Windows, userei:

```
df <- read.csv(
  "C:/Users/Marco/Desktop/Corso_R/my_data/compare_2_species.csv"
  )
```

La sigla "csv" sta per "comma-separated values", ovvero valori separati da virgole: infatti, di norma i file `.csv` usano la virgola per separare i valori tra loro. Le versioni di *Excel* che usano la virgola anzichè il punto come separatore decimale generano file `.csv` usando il punto e virgola per separare i valori tra loro. È una impostazione comune nelle versioni di Excel distribuite in Italia, e potete verificare quale carattere sia stato usato come separatore aprendo i file `.csv` con un visualizzatore di testi come TextEdit o TextPad. Per specificare queste impostazioni a R si usano gli argomenti `sep=` e `dec=` all'interno di `read.csv()`. Ad esempio, se la mia working

directory si trovasse sul desktop di un computer Windows e se i miei dati fossero in un file `.csv` creato con una versione di Excel che usa ";" come separatore e "," per i decimali, userei:

```
df <- read.csv(
  "C:/Users/Marco/Desktop/Corso_R/my_data/compare_2_species.csv",
  sep=";", dec="."
  )
```

L'impostazione `sep=","` e `dec="."` è quella di default. L'oggetto che abbiamo creato, `df`, appartiene alla classe `data.frame` e contiene i dati contenuti nel file `compare_2_species.csv`. Una nota importante: non date ai dati che inserite in R il nome `data`. In R, `data()` è una funzione, e dare ad un oggetto il nome di una funzione può creare problemi. Per brevità, di solito chiamo i miei dati `df` (abbreviazione di `data.frame`) o `dd`.

Visualizziamo ora la "testa" e la "coda" del nostro dataset, per farci un'idea di com'è composto:

```
# Prendiamo confidenza coi dati:
head(df)  # le prime sei righe del data.frame df
```

```
##   Sample_ID   Species body.mass
## 1   SpA_001 Species A  95.39405
## 2   SpA_002 Species A  97.69891
## 3   SpA_003 Species A 101.75060
## 4   SpA_004 Species A 104.98992
## 5   SpA_005 Species A  99.85557
## 6   SpA_006 Species A  96.22698
```

```
tail(df)  # le ultime sei righe del data.frame df
```

```
##     Sample_ID   Species body.mass
## 95    SpB_045 Species B  109.1885
## 96    SpB_046 Species B  113.6186
## 97    SpB_047 Species B  109.7411
## 98    SpB_048 Species B  110.5616
## 99    SpB_049 Species B  108.1049
## 100   SpB_050 Species B  117.4926
```

Vediamo la struttura del dataset:

```
str(df)
```

```
## 'data.frame':    100 obs. of  3 variables:
##  $ Sample_ID: chr  "SpA_001" "SpA_002" "SpA_003" "SpA_004" ...
##  $ Species  : chr  "Species A" "Species A" "Species A" "Species A" ...
##  $ body.mass: num  95.4 97.7 101.8 105 99.9 ...
```

Il dataset consiste di 100 righe e tre colonne. La prima colonna, `Sample_ID`, consiste di caratteri, come indicato dalla sigla `chr`. Ogni valore di `Sample_ID` identifica univocamente ciascuna misurazione. La seconda colonna, `Species`, indica la specie da cui quella misurazione è stata ottenuta. La terza colonna, `body.mass`, riporta le masse corporee in Kg ed è di tipo numerico, come indicato dalla sigla `num`. R interpreta il contenuto della colonna `Species` come una stringa

di caratteri ma noi sappiamo che `Species` è in realtà una variabile categorica, anche detta variabile fattoriale o semplicemente fattore *(factor)*. Comunichiamolo a R usando la funzione `as.factor()`:

```
df$Species <- as.factor(df$Species)
```

L'espressione `df$Species` significa *"la colonna_ `Species` nel data.frame `df`"*. Il segno del dollaro è uno dei modi utilizzati per selezionare specifiche colonne di un set di dati in R. La funzione `as.factor()` ha sovrascritto la colonna `df$Species` con una nuova colonna con lo stesso nome e lo stesso contenuto, ma di formato *factor* anzichè *character*.

Per assicurarci che la formattazione della colonna `Species` sia avvenuta con successo possiamo usare la funzione `class()` per controllare a che classe di oggetti di R appartenga il contenuto della colonna:

```
class(df$Species)

## [1] "factor"
```

Identificare `Species` come una variabile fattoriale ci aiuterà nelle analisi successive. Le categorie di una variabile categorica sono anche note come "livelli". Nel nostro caso, `Species` è una variabile fattoriale con due livelli: *Species A* e *Species B*.

6.2 Analisi grafica

Siamo ora pronti per analizzare i nostri dati per rispondere alla domanda: "C'è una differenza significativa tra le masse corporee degli individui di *Species A* e *Species B*?"

Prima di lanciarsi alla cieca in analisi numeriche è buona regola iniziare con una analisi visiva, ovvero con una rappresentazione grafica dei nostri dati. Nel capitolo precedente abbiamo usato degli istogrammi. Un altro strumento grafico utile è rappresentato dai grafici scatola-e-baffi (fig. 6.1), che possiamo produrre con la funzione `boxplot()`:

```
boxplot(body.mass ~ Species, data = df)
```

Il risultato è mostrato in figura 6.1. Notate che la funzione `boxplot()` richiede che le variabili da rappresentare siano fornite con una speciale formula. L'espressione `body.mass ~ Species` significa: "`body.mass` *varia in funzione di* `Species`". La "~" (tilde) significa: *"varia in dipendenza da (o in funzione di)"*. Useremo "~" spessissimo, per cui verificate come ottenerlo sulla vostra tastiera.

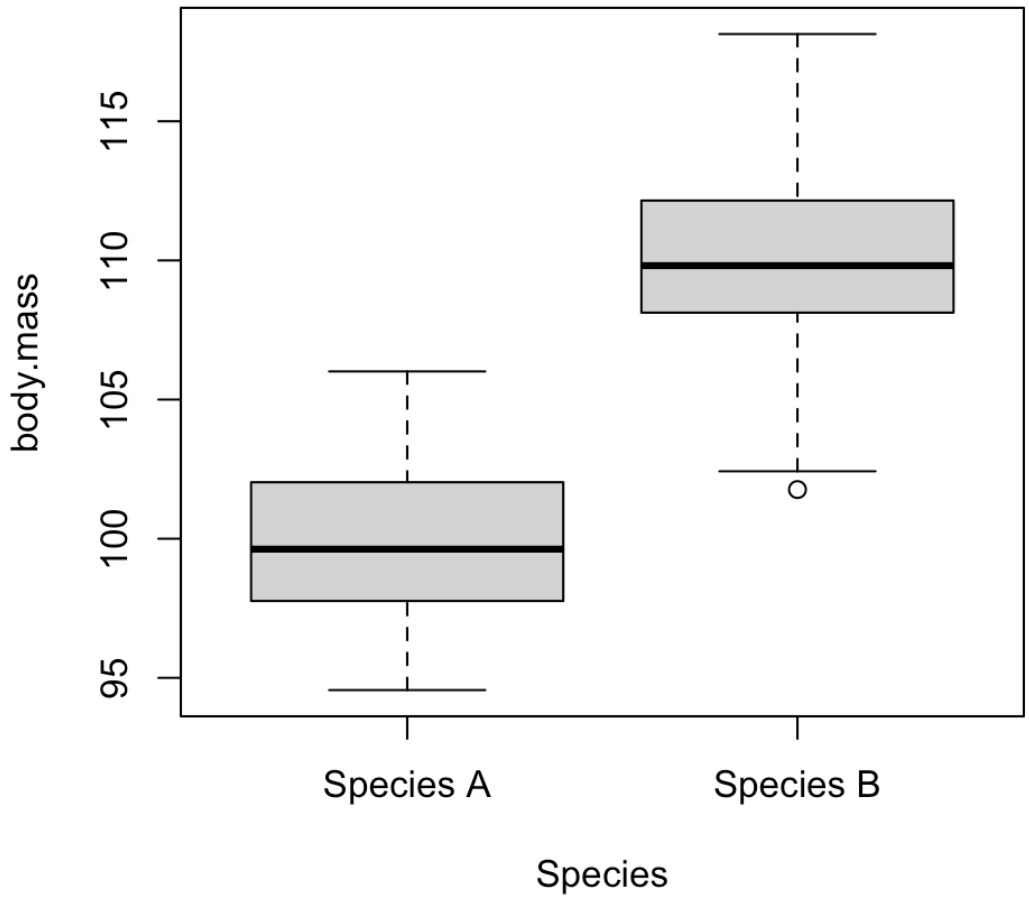

Figura 6.1: Distribuzione delle masse corporee misurate per SpeciesA e SpeciesB.

Nei grafici scatola-e-baffi la "scatola" è delimitata dal primo e terzo quartile ed è "tagliata" dalla mediana (ovvero dal secondo quartile). Le estremità dei "baffi" indicano i valori massimo e minimo di ciascun gruppo. Eventuali valori che sono sospettati di essere *"outliers"* sono indicati dai cerchiolini oltre le estremità dei baffi. Gli *"outliers"* sono valori estremi che si discostano grandemente da tutti gli altri valori osservati in un certo gruppo. Torneremo sul concetto di *"outlier"* e sulle loro implicazioni nel corso di questo capitolo (approfondiremo ulteriormente questo argomento in sezione 16).

La figura 6.1 mostra che la massa corporea differisce in maniera visivamente evidente nei campioni delle due specie a confronto. Come verificare se la differenza osservata sia statisticamente significativa? Potremmo usare la funzione `t.test()` che abbiamo visto in precedenza. Useremo invece una funzione più versatile e potente chiamata `lm()`.

6.3 I modelli lineari e la funzione `lm()`

Nel capitolo precedente abbiamo usato il test t per verificare se ci fosse una differenza significativa nella massa corporea di due specie. In altre parole abbiamo usato il test t per valutare la significatività statistica dell'affermazione: "la massa corporea (`body.mass`) varia in funzione dell'identità di specie (`Species`)". Possiamo esprimere questa affermazione in maniera matematica con la seguente equazione:

$$body.mass_i = \mu + b_k \cdot Species_k + \epsilon_i \qquad (6.1)$$

Dove:

- $body.mass_i$ è la *i*-esima osservazione di massa corporea;
- μ è la media di tutte le osservazioni a prescindere dalla specie di appartenenza;
- b_k è un parametro specie-specifico che misura lo scostamento medio della *k*-esima specie dalla media generale μ;
- $Species_k$ è la *k*-esima specie, alla quale la *i*-esima osservazione di *body.mass* appartiene;
- ϵ_i rappresenta lo scostamento di $body.mass_i$ dalla media della specie di appartenenza dovuto a variabilità individuale.

Questa equazione descrive la massa corporea della i-esima osservazione usando, come prima approssimazione, la massa corporea media osservata tra tutti gli individui misurati, a prescindere dalla loro specie di appartenenza. A questa prima stima si aggiunge una correzione sulla base della specie a cui quell'osservazione appartiene ($b_k \cdot Species_k$). Infine si aggiunge una ulteriore correzione dovuta allo scostamento della massa dell'i-esimo individuo dalla massa media della sua specie di appartenenza (ϵ_i).

L'equazione soprastante rappresenta un *modello lineare.* I modelli lineari sono modelli che descrivono i valori osservati nella variabile di risposta in funzione di una combinazione lineare di parametri e variabili esplicative (o predittori). I modelli lineari possono essere validati con vari strumenti statistici:

- modelli con un singolo predittore categorico con due livelli possono essere validati con un test t;
- modelli con uno o più predittori categorici possono essere validati con l'analisi della varianza, o ANOVA;
- modelli con un singolo predittore continuo possono essere validati con un'analisi di regressione lineare;
- modelli con due predittori di cui uno continuo ed uno categorico possono essere validati con l'Analisi della Covarianza, o ANCOVA;
- modelli con due predittori continui possono essere validati con l'analisi della regressione multipla.

Quelli descritti sono approcci statistici differenti per validare la stessa ipotesi generale: cioè che la variabile dipendente *y* (`body.mass` in questo caso) varii linearmente al variare di uno o più predittori. In questo manuale presenterò separatamente gli scenari tradizionalmente analizzati

con test t, ANOVA, ANCOVA, e regressione lineare, ma mostrerò un unico protocollo di validazione dei modelli che può essere applicato a ciascuna delle situazioni appena descritte.

Torniamo al nostro esempio: vogliamo confrontare la massa corporea di due ipotetiche specie animali, *Species A* e *Species B*, e vogliamo capire se l'identità di specie è un predittore importante per la variabilità osservata nelle misure di massa corporea. Possiamo definire l'affermazione "`body.mass` varia in funzione di `Species`" all'interno della funzione `lm()` come segue:

```
m1 <- lm(body.mass ~ Species, data = df)
```

La formula `body.mass ~ Species` all'interno di `lm()` corrisponde al modello descritto nell'equazione (6.1). Notate che è la stessa formula usata nella funzione `boxplot()`. Ricordiamo che l'espressione `body.mass ~ Species` significa: "`body.mass` *varia in funzione di* `Species`".

L'oggetto `m1` contiene la formula del modello in esame e le stime dei suoi parametri:

```
m1

##
## Call:
## lm(formula = body.mass ~ Species, data = df)
##
## Coefficients:
##      (Intercept)  SpeciesSpecies B
##            99.94             10.16
```

Anzichè stimare la media totale μ ed i parametri b_k come visto nell'Equazione (6.1), R sceglie uno dei livelli del fattore `Species` come livello di riferimento, ne calcola la media, e riporta le stime medie della variabile di risposta per ciascun livello come *differenze* rispetto al livello di riferimento. Nel nostro esempio R ha scelto *Species A* come livello di riferimento per `Species` (semplicemente perchè è il primo livello di `Species` in ordine alfanumerico). La media di `body.mass` per il livello di riferimento, *Species A*, è 99.94 Kg. La differenza media tra la `body.mass` di *Species B* e quella del livello di riferimento è 10.16 Kg: in altre parole, la massa corporea media stimata per *Species B* è 99.94 + 10.16 = 110.1 Kg. Possiamo confrontare queste stime con i dati presentati nel grafico scatola-e-baffi.

6.4 Verificare le assunzioni del modello con i grafici diagnostici

Per ora abbiamo ottenuto solo le medie campionarie della massa corporea per *Species A* e *Species B*, senza i corrispondenti intervalli di confidenza e senza una validazione statistica del modello di interesse. Il nostro obiettivo è quello di testare la significatività statistica del modello `m1` nel descrivere i dati contenuti in `df`. Prima di fare ciò dobbiamo assicurarci che il modello `m1` rispetti le assunzioni dei modelli lineari, ovvero:

- omoscedasticità (omogeneità delle varianze);
- indipendenza delle osservazioni;
- normalità della distribuzione dei residui del modello;
- linearità della correlazione tra predittori e variabile di risposta (questo apparirà più chiaro nei capitoli successivi, quando ci occuperemo della regressione lineare).

Bisogna anche valutare se ci siano valori estremi *(outliers)* nella nostra variabile di risposta che potrebbero sfalsare i risultati del nostro test statistico.

Se questi requisiti non fossero verificati, un modello lineare non sarebbe l'approccio adatto per rispondere al nostro interrogativo di ricerca, e dovremmo quindi optare per altri approcci analitici (ne vedremo alcuni più avanti nel corso).

Ci sono vari metodi per verificare che le assunzioni dei modelli lineari siano rispettate. Un metodo semplice ed efficace consiste nella valutazione visiva dei cosiddetti grafici diagnostici. Se forniamo il modello `m1` come argomento alla funzione `plot()`, essa produrrà automaticamente quattro grafici che ci aiutano a valutare visivamente se le assunzioni del modello siano verificate. Come visto in sez. 5, possiamo usare `par(mfrow=...)` per visualizzare i quattro grafici diagnostici in un'unica finestra grafica:

```
# suddividiamo la finestra grafica in due righe e due colonne:
par(mfrow=c(2, 2))
plot(m1)
```

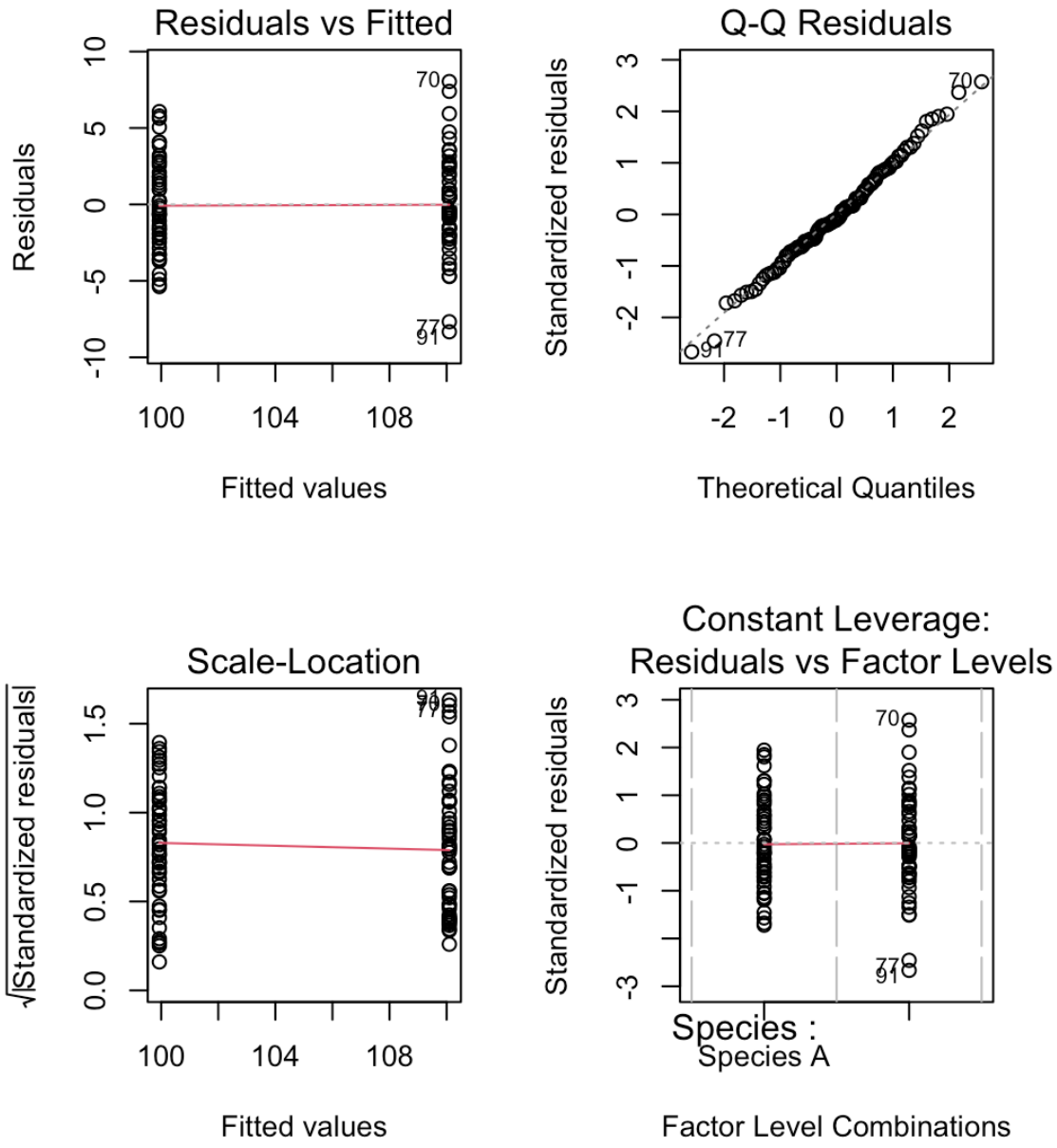

Figura 6.2: Grafici diagnostici per il modello m1.

Il risultato è in figura 6.2. I quattro grafici analizzano in vario modo i *residui* del modello, ovvero gli scostamenti di ciascun valore osservato rispetto ai valori attesi dal modello. Nel loro insieme, i residui riflettono la frazione di variabilità che il modello non è in grado di spiegare. Vediamo di seguito il significato di ciascun grafico:

- **Residuals vs Fitted**: mostra i residui in relazione ai valori stimati dal modello. Permette di valutare linearità della correlazione tra variabile di risposta e predittore, nonchè l'omoscedasticità (omogeneità della varianza) dei residui. Scostamenti seriali dei residui

dai valori stimati indicherebbero l'inadeguatezza del modello a catturare la variabilità osservata. Una disposizione orizzontale "a imbuto" dei residui indicherebbe che la loro varianza non è omogenea. Nel nostro caso i residui sono distribuiti omogeneamente attorno ai valori stimati e la loro variabilità è uniforme attorno a ciascuno dei valori stimati.

- **Normal Q-Q**: il grafico quantile-quantile *("Q-Q plot")* mostra i quantili dei residui standardizzati del modello contro i quantili attesi secondo una distribuzione normale con media = 0 e deviazione standard = 1. Nel nostro caso la corrispondenza tra valori osservati e valori attesi è quasi perfetta: i residui del modello seguono una distribuzione normale.

- **Scale-Location**: mostra la radice quadrata dei residui in funzione dei valori stimati dal modello. È un modo per identificare eteroscedasticità. In questo caso le radici quadrate dei residui sono simili per tutti i livelli del predittore, che suggerisce omoscedasticità dei residui.

- **Leverage**: questo grafico quantifica l'"effetto leva" di ciascuna osservazione, cioè il suo influsso sulla stima dei parametri del modello. Valori con un effetto leva estremo, noti come valori estremi o *outliers*, possono portare a stime fuorvianti dei parametri del modello (sez. 16). Una stima dell'"effetto leva" di una osservazione è la distanza di Cook, *D*. La distanza di Cook *D* per una certa osservazione corrisponde alla somma delle differenze tra i valori stimati dal modello in base all'intero dataset ed i valori stimati escludendo l'osservazione in questione dal dataset. Osservazioni con una distanza di Cook elevata potrebbero essere *outliers*. Nel grafico "Leverage" le eventuali linee tratteggiate con etichette "0.5", "1", ecc. sono le isolinee per i corrispondenti valori di *D*. Esploreremo questo argomento con maggiore dettaglio in sezione 16. Per il momento limitiamoci a notare che nel grafico "Leverage" in figura 6.2 nessuno dei valori si discosta grandemente dagli altri, per cui non si direbbe che ci siano valori estremi di cui preoccuparsi.

Dal momento che i grafici diagnostici in figura 6.2 non mostrano nulla di sospetto possiamo considerare rispettate le assunzioni su cui si basano i modelli lineari.

6.5 Testare la significatività del modello

Possiamo ora procedere a testare la significatività statistica del modello. Vogliamo rispondere alla domanda: il predittore `Species` spiega una porzione significativa della variabilità osservata in `body.mass`? A questo scopo introduciamo la funzione `anova()`. La funzione `anova()` può:

1. svolgere una classica analisi della varianza (ANOVA) sul modello per valutare se la variabilità spiegata dai predittori è significativamente maggiore della variabilità residua;
2. confrontare due modelli verificando se la quantità di variabilità residua (non spiegata dal modello) differisca significativamente nei modelli a confronto.

In questo libro mi concentrerò sul secondo approccio, che considero più versatile del primo e meno soggetto a differenze filosofiche di interpretazione[1].

Il modello `m1` riflette l'ipotesi per la quale "la massa corporea (`body.mass`) varia in funzione dell'identità di specie (`Species`)". Possiamo testare la validità di questa ipotesi confrontando il modello `m1` contro il modello nullo, secondo il quale `body.mass` cambia solo in funzione di scostamenti individuali dalla media generale di tutte le osservazioni:

$$H_0: body.mass_i = \mu + \epsilon_i \qquad (6.2)$$

Possiamo esprimere l'equazione (6.2) in R con:

```
m0 <- lm(body.mass ~ 1, data = df)
```

La dicitura `~ 1` sta ad indicare che non abbiamo indicato nessun predittore nel modello: l'unico "predittore" di `body.mass` in `m0` è la media delle masse corporee osservate.

Possiamo ora confrontare `m1` ed `m0` fornendoli come argomenti alla funzione `anova()`:

```
anova(m1, m0)
```

```
## Analysis of Variance Table
##
## Model 1: body.mass ~ Species
## Model 2: body.mass ~ 1
##   Res.Df    RSS Df Sum of Sq      F    Pr(>F)
## 1     98  975.7
## 2     99 3555.9 -1   -2580.2 259.15 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

La funzione `anova()` usa un test F per confrontare la variabilità residua dei due modelli, stimata come la somma dei quadrati dei residui (la *Residual Sum of Squares*, o *RSS*) divisa per i loro gradi di libertà (i *degrees of freedom* dei residui, $Res.Df$). Tanto minore è la variabilità residua di un modello, tanto maggiore è la frazione di variabilità osservata che esso è in grado di descrivere. Se le misure di variabilità residue differiscono significativamente (ovvero: se il *p-value* è al di sotto della soglia di significatività), il modello con il valore di $RSS/Res.Df$ più basso è quello che meglio descrive la variabilità osservata tra i modelli a confronto. Nel caso in cui i modelli a confronto non differissero significativamente potremmo considerare il modello più semplice come in migliore dei due, secondo il principio del rasoio di Occam.

Nel nostro caso i modelli `m1` e `m0` differiscono significativamente, come evidenziato dal valore di p notevolmente inferiore alla tradizionale soglia di significatività di 0.05. Tra i due modelli, `m1` è quello con la variabilità residua minore. Possiamo rigettare il modello `m0` e la corrispondente ipotesi nulla. Il modello `m1`, che include il predittore `Species`, spiega una quantità di variabilità totale osservata in `body.mass` significativamente maggiore di quanto non faccia `m0`. Ne consegue

[1] Si veda: *Anova – Type I/II/III SS explained.* di F. Scholer, https://archive.fo/ZqRTy

che l'identità di specie è un importante predittore della variabilità della massa corporea, nelle specie a confronto.

Possiamo ora ottenere una tavola riassuntiva per il modello `m1` con la funzione `summary()`:

```
summary(m1)

##
## Call:
## lm(formula = body.mass ~ Species, data = df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -8.3291 -1.9816 -0.2943  2.0665  8.0388
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)        99.9357     0.4462   223.9   <2e-16 ***
## SpeciesSpecies B  10.1591     0.6311    16.1   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.155 on 98 degrees of freedom
## Multiple R-squared:  0.7256, Adjusted R-squared:  0.7228
## F-statistic: 259.1 on 1 and 98 DF,  p-value: < 2.2e-16
```

La funzione `summary()`:

- comincia col fornirci un promemoria del modello lineare in esame e dei dati usati per stimarne i parametri;
- fornisce alcune informazioni sulla distribuzione dei residui del modello, ovvero i valori minimo e massimo dei residui del modello, nonchè il loro primo, secondo, e terzo percentile;
- fornisce i coefficienti del modello ed il relativo errore standard, nonchè i risultati di un test t circa la loro differenza da zero.

L'interpretazione dei coefficienti del modello così come li fornisce `summary()` merita un pizzico di approfondimento. `(Intercept)` è la stima per il livello di riferimento del predittore fattoriale `Species`, scelto da R in base all'ordine alfa-numerico. In questo caso l'intercetta è il peso medio stimato per *Species A*, pari a 99.94 Kg, ed è significativamente diversa da zero con un p-value $p<0.0001$. Come già accennato, i successivi coefficienti non si riferiscono agli altri livelli di `Species`, ma alle *differenze* tra i livelli e quello di riferimento. Nel nostro caso la stima della differenza tra *Species A* e *Species B* è 10.16 Kg, e questa differenza differisce significativamente da zero con $p<0.0001$ (che è un altro modo per dire che la massa corporea differisce significativamente nelle due specie). La `body.mass` media per *Species B* è dunque 110.1 Kg, ovvero la somma dell'intercetta (99.94 Kg) e della differenza tra *Species A* e *Species B* (10.16 Kg).

La funzione `summary()` fornisce anche due stime di R quadro R^2: un *Multiple R-squared* ed un *Adjusted R-squared*. Entrambe indicano la percentuale di variabilità osservata spiegata dal modello, che in questo caso è circa il 72%. Consiglio di fare sempre riferimento all'R quadro "adjusted" per misurare il potere esplicativo di un modello. È infatti dimostrato che il *Multiple R-squared* (cioè l'R quadro *non-adjusted*) aumenta in maniera artificiosa all'aumentare del numero dei predittori, anche nel caso in cui essi non abbiano potere esplicativo. Al contrario, l'R quadro *adjusted* riflette in maniera più fedele la quantità di variabilità spiegata da un modello[2]. Finchè abbiamo a che fare con modelli con un solo predittore la differenza tra *Multiple R-squared* ed *Adjusted R-squared* è minima, ma tanto vale prendere buone abitudini fin dall'inizio.

6.6 Comunicazione

Per comunicare i dettagli dello studio che abbiamo appena svolto è importante evidenziare l'interrogativo di ricerca, la natura dei dati, i metodi di analisi, i risultati. Ecco un esempio:

Abbiamo studiato la massa corporea delle specie animali Species A e Species B per capire se esse differiscano significativamente.

Metodi. Abbiamo ottenuto le misure di massa corporea per 50 individui di ciascuna specie (in Kg). Per capire se l'identità di specie fosse un predittore importante della massa corporea degli individui delle due specie abbiamo confrontato un modello lineare con "identità di specie" come predittore (modello M1) ed un modello lineare corrispondente all'ipotesi nulla (modello M0). Abbiamo svolto il confronto tra i modelli con un test F sulle loro variabilità residue.

Risultati. La variabilità residua differisce significativamente nei due modelli (Tab. 1) ed è minore nel modello M1, che include "identità di specie" come predittore (Tab. 1). Il modello M1 cattura il ~72% della variabilità osservata nei dati. La massa corporea media stimata per Species A è pari a 99.94 Kg, mentre la massa corporea media stimata per Species B è pari a 110.1 Kg.[3]

Confronto tra modelli.

Modello	Predittori	RSS	df	F	p
M1	Identità di specie	975.7	98		
M0	Media generale (modello nullo)	3555.9	99	259.15	<0.0001

[2] Qui trovate una dimostrazione del prof. Owen Petchey sulla base di dati simulati: https://archive.fo/QlDRF (pagina di archivio aggiornata ad Aprile 2023).

[3] È buona prassi calcolare e comunicare non solo le stime del modello, ma anche i rispettivi intervalli di confidenza. Vedremo come fare ciò nei prossimi capitoli.

6.7 Raccomandazioni circa la formattazione dei dati

I dati contenuti nel file `compare_2_species.csv`, che abbiamo appena finito di analizzare, sono organizzati in modo che ciascuna riga del dataset si riferisca ad una diversa misurazione. A ciascuna misurazione di massa è assegnato un codice univoco di identificazione, `Sample_ID`, ed è associata la specie a cui si riferisce, `Species`. Questo tipo di formattazione dei dati è noto come *long format* o "formato lungo" ed è necessario per svolgere analisi univariate in R. Nel formato "lungo" dei dataset ogni riga corrisponde ad una diversa misurazione della variabile di risposta. Un esempio generico di questo tipo di formato è mostrato in Tab. 2.

esempio di dati inseriti in formato 'lungo'.

Species	Replicates	Values
Species A	Rep1	95.39
Species A	Rep2	97.69
Species A	Rep3	101.75
…	…	…
Species B	Rep1	110.53
Species B	Rep2	97.69
Species B	Rep3	101.75
…	…	…

Se i dati in Tab. 2 fossero inseriti in un formato "largo", come quello mostrato in Tab. 3, sarebbe necessario riformattarli prima di poter svolgere le analisi descritte in questo libro. Vedremo come cambiare il formato di un dataset da "lungo" a "largo" e vice versa nel capitolo 21.

esempio di dati inseriti in formato 'largo'.

Replicates:	Rep1	Rep2	Rep3	…
Species A	95.39	97.69	101.75	…
Species B	110.53	113.61	110.56	…

Altri aspetti del formato di un dataset che possono rendere problematica l'analisi dei dati in R sono:

- celle vuote nel vostro dataset. Se manca una informazione per una data cella non lasciatela in bianco ma scrivetevi `NA`, che sta per "not available" (ovvero non disponibile);
- l'uso della virgola invece del punto per indicare valori decimali;
- la presenza di caratteri speciali, doppi spazi, lettere maiuscole usate invece delle minuscole o viceversa (ad esempio: R considera *"species_A"* e *"Species_A"* come valori differenti), eccetera. Assicuratevi che i vostri dati siano inseriti in maniera uniforme.

6.8 Il protocollo di analisi dei dati

In questo capitolo abbiamo imparato ad usare le funzioni `lm()` ed `anova()` per definire e confrontare modelli lineari in R, e la funzione `summary()` per visualizzare i parametri stimati per il modello, nonchè il loro grado di significatività.

Abbiamo inoltre messo in pratica un protocollo di analisi dei dati con i seguenti passaggi fondamentali (modificato a partire dal protocollo proposto da Beckerman & Petchey, 2012):

1. elaborazione di un interrogativo di ricerca chiaro e statisticamente verificabile;
2. ottenimento dei dati necessari per rispondere al questito di ricerca;
3. inserimento dei dati in R;
4. visualizzazione e interpretazione visiva dei dati;
5. definizione di un modello sulla base dell'interrogativo di ricerca di cui al punto 1;
6. controllo che le assunzioni del modello siano verificate;
7. verifica della validità del modello confrontandolo con modelli più semplici;
8. stima dei parametri del modello e del potere esplicativo del modello;
9. comunicazione degli interrogativi di ricerca, metodi di analisi, e risultati.

Questo protocollo ci accompagnerà per il resto del libro.

7 Confrontare più di due gruppi: lo scenario dell'ANOVA a una via

Il protocollo di analisi dei dati delineato nel precedente capitolo per confrontare due gruppi di interesse si può facilmente estendere al confronto tra più di due gruppi e, in generale, a modelli lineari con un solo predittore categorico con più di due livelli. Lo "strumento" statistico tradizionalmente usato per esaminare modelli lineari con predittori fattoriali è l'Analisi della Varianza, o ANOVA. Per modelli con un singolo predittore fattoriale si parla di ANOVA ad una via. Continuo a chiamare questo tipo di scenario "scenario dell'ANOVA a una via" anche se, come accennato nel capitolo precedente, il protocollo di analisi proposto in questo libro si discosta dall'ANOVA tradizionale.

Immaginiamo il seguente scenario: vogliamo confrontare la massa corporea di sei specie animali per vedere se ci siano differenze significative, ovvero se la massa corporea media di almeno una delle specie in esame differisca in maniera statisticamente significativa dalle altre. A questo scopo, immaginiamo di aver pesato 500 individui per ciascuna specie e di aver organizzato le misurazioni in un dataset, `compare_6_species.txt`, che trovate a https://github.com/marcoplebani85/datasets/. Questa volta i dati sono salvati come file delimitato da tabulazioni in formato `.txt` (e non `.csv` come visto nell'esempio precedente). Per caricare dati con questo tipo di formato in R esiste la funzione `read.delim()`.

Prima di caricare i dati in R voglio introdurre la funzione `setwd()`, che permette di indicare a R la nostra *working directory*. Definiamo la cartella *Corso R* come la nostra *working directory*:

```
rm(list = ls())  # ripulisce la memoria di R
setwd("~/Desktop/Corso_R")
```

In RStudio è possibile scegliere la *working directory* cliccando su *Session* > *set working directory* > *choose directory*. Noterete che RStudio esegue direttamente il comando nella *consolle* senza salvarlo nello script; per completezza e riproducibilità vi consiglio di copiare la linea di codice anche nello script.

Definire una *working directory* non è indispensabile, ma rende meno tedioso comunicare a R dove trovare i nostri files. A questo punto, infatti, per caricare i dati in R è sufficiente scrivere:

```
df <- read.delim("my_data/compare_6_species.txt")
```

Infatti R sa già che la cartella *my_data* si trova nella nostra *working directory*, cioè nella cartella *Corso_R*.

Prendiamo confidenza coi dati:

```
str(df)  # struttura dei dati

## 'data.frame':    3000 obs. of  2 variables:
##  $ Species  : chr  "A" "A" "A" "A" ...
##  $ body.mass: num  37.6 27.8 25.8 27.7 30.6 ...
```

Comunichiamo a R che `Species` è un predittore categorico, ovvero un fattore:

```
df$Species <- as.factor(df$Species)
class(df$Species)

## [1] "factor"

levels(df$Species)

## [1] "A" "B" "C" "D" "E" "F"
```

Procediamo ora a visualizzare i dati. Cogliamo l'occasione per personalizzare l'estetica del grafico utilizzando `par()` ed alcuni argomenti aggiuntivi all'interno di `boxplot()`:

```
par(family="Times", # tipo di caratteri da usare
    # tipo di "scatola" da tracciare attorno al grafico:
    bty="L" # indica di tracciare solo gli assi
    # anzichè un rettangolo completo
    )
boxplot(body.mass ~ Species, data=df,
    col="white", # colore di riempimento delle "scatole"
    ylim=c(0,70), # limiti dell'asse y
    # etichette personalizzate per gli assi:
    xlab="Specie",
    ylab="Massa corporea [Kg]"
    )
```

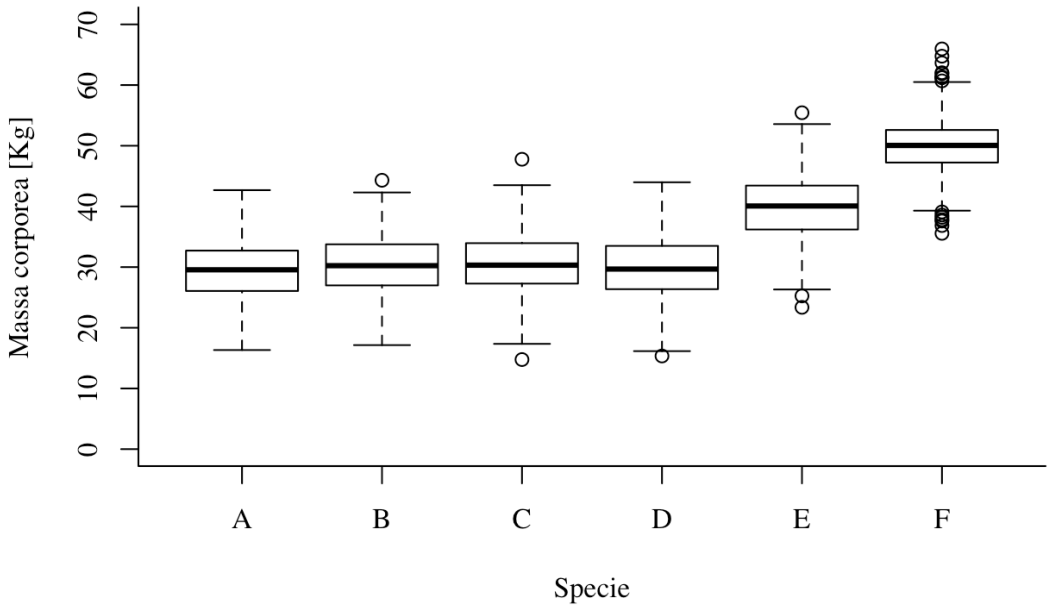

Figura 7.1: Distribuzione delle masse corporee delle specie A, B, C, D, E, ed F.

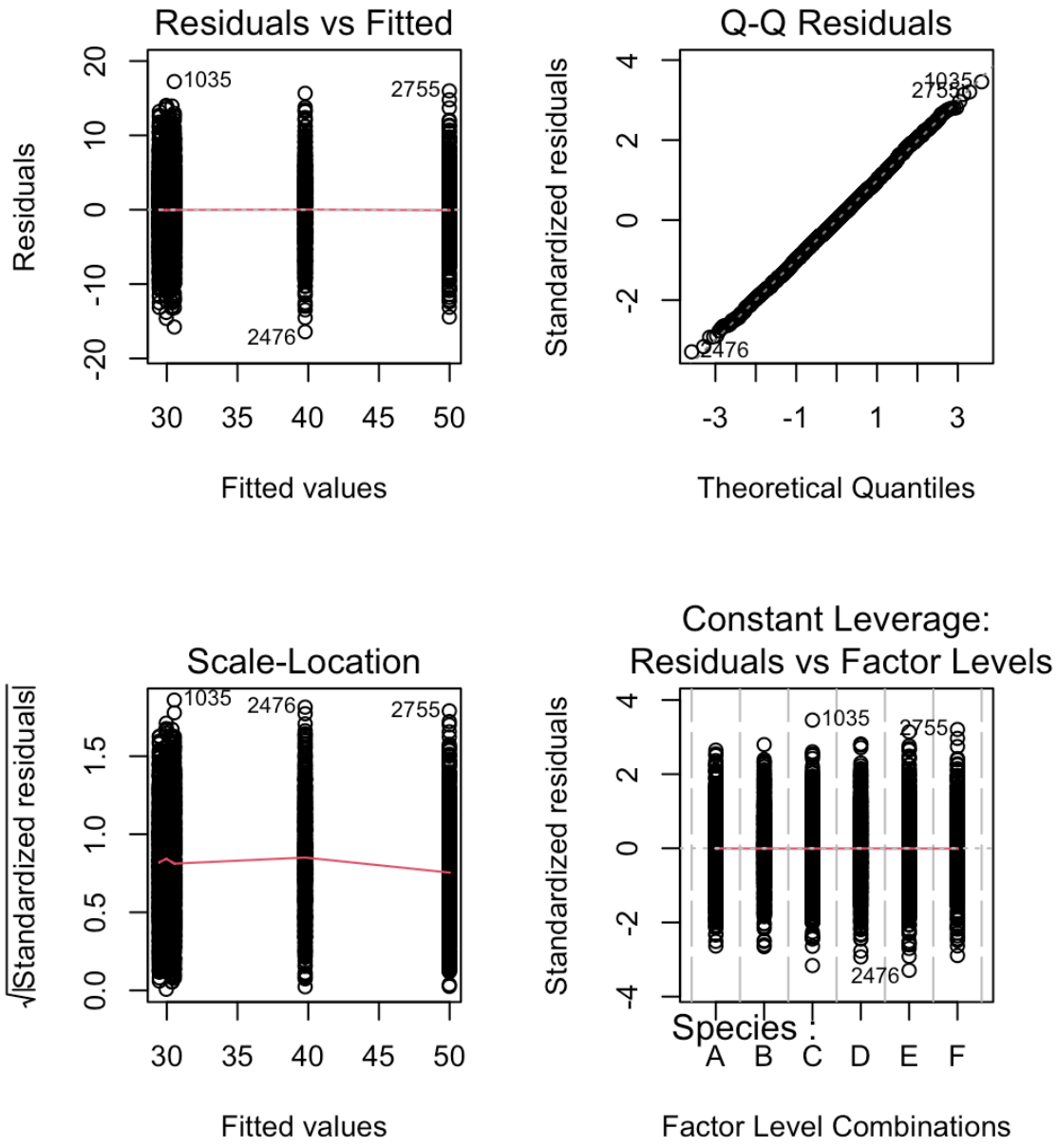

Figura 7.2: Grafici diagnostici per il modello m1.

La figura 7.1 non mostra differenze evidenti tra le masse corporee medie delle specie A-D, mentre quelle delle specie E ed F si discostano sia l'una dall'altra, sia dalle specie A-D.

Procediamo dunque a definire il modello secondo il quale l'identità di specie spiega una quantità significativa della variabilità osservata nelle masse corporee:

```
m1 <- lm(body.mass ~ Species, data = df)
```

Visualizziamo i grafici diagnostici per il modello `m1`:

```r
par(mfrow = c(2, 2))  # 2 righe, 2 colonne
plot(m1)
```

I grafici diagnostici in figura 7.2 non mostrano nulla di sospetto. Possiamo dunque definire il modello nullo `m0` e confrontare i due modelli tramite la funzione `anova()`:

```r
m0 <- lm(body.mass ~ 1, data = df)

anova(m1, m0)
```

```
## Analysis of Variance Table
##
## Model 1: body.mass ~ Species
## Model 2: body.mass ~ 1
##   Res.Df    RSS Df Sum of Sq      F    Pr(>F)
## 1   2994  74613
## 2   2999 246907 -5   -172295 1382.7 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

C'è una differenza significativa tra i due modelli. Il rapporto $RSS/Res.Df$ è maggiore nel modello `m0`, che significa che il modello `m0` ha una quantità maggiore di variabilità residua rispetto al modello `m1`. Di conseguenza, il modello `m1` ha un maggiore potere esplicativo rispetto al modello nullo `m0`. Possiamo rigettare l'ipotesi nulla secondo la quale le specie in studio non differiscono significativamente tra loro per massa corporea. Secondo il modello `m1` c'è *almeno* una specie la cui massa corporea media differisce significativamente da quella delle altre specie.

Come identificare quali sono le paia di specie che differiscono significativamente l'una dall'altra? Usare `summary()` permette di capire solo quali siano le specie che differiscono significativamente dalla specie scelta da R come livello di riferimento:

```r
summary(m1)
```

```
##
## Call:
## lm(formula = body.mass ~ Species, data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -16.4126  -3.3733  -0.0466   3.3199  17.2394
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  29.4721     0.2233 132.013  < 2e-16 ***
## SpeciesB      0.8931     0.3157   2.829 0.004703 **
## SpeciesC      1.0585     0.3157   3.353 0.000811 ***
## SpeciesD      0.4882     0.3157   1.546 0.122174
## SpeciesE     10.3105     0.3157  32.657  < 2e-16 ***
## SpeciesF     20.5047     0.3157  64.945  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Residual standard error: 4.992 on 2994 degrees of freedom
## Multiple R-squared:  0.6978, Adjusted R-squared:  0.6973
## F-statistic:  1383 on 5 and 2994 DF,  p-value: < 2.2e-16
```

…E se fossimo interessati nella differenza tra le specie B e C? Per ottenere tutti i possibili contrasti a coppie in un colpo solo possiamo usare la funzione `pairwise.t.test()`, che permette inoltre di correggere le stime dei *p-values* sulla base del fatto che stiamo svolgendo confronti multipli:

```
pwt <- pairwise.t.test(x = df$body.mass,
            g = df$Species,
            p.adjust.method="bonferroni")
pwt

##
##  Pairwise comparisons using t tests with pooled SD
##
## data:  df$body.mass and df$Species
##
##   A      B      C      D      E
## B 0.071  -      -      -      -
## C 0.012  1.000  -      -      -
## D 1.000  1.000  1.000  -      -
## E <2e-16 <2e-16 <2e-16 <2e-16 -
## F <2e-16 <2e-16 <2e-16 <2e-16 <2e-16
##
## P value adjustment method: bonferroni
```

L'argomento `p.adjust.method=` si riferisce al fatto che confronti multipli aumentano il rischio di errore di tipo I (rigettare una ipotesi nulla vera) e correzioni come quella di Bonferroni adeguano il test statistico di conseguenza.[4]

Un'alternativa a `pairwise.t.test()` che useremo spesso durante il corso è offerta dal pacchetto `emmeans` (Lenth 2023). Si tratta di un pacchetto che non è preinstallato in R. Per installarlo introduciamo la funzione `install.packages()`, che permette di installare pacchetti presenti sul Comprehensive R Archive Network (CRAN, la stessa piattaforma da cui abbiamo scaricato R):

```
install.packages("emmeans")
```

Dopo l'installazione possiamo caricare `emmeans` usando la funzione `library()`:

```
library(emmeans)
```

Possiamo ora usare la funzione `emmeans()` (ha lo stesso nome del pacchetto a cui appartiene) e confrontare il risultato con quello fornito da `pairwise.t.test()`:

[4] Si veda: https://it.wikipedia.org/wiki/Correzione_di_Bonferroni e la bibliografia ivi riportata.

```
emmeans.test <- emmeans(m1,
        pairwise ~ Species,
        adjust = "bonferroni"
        )
emmeans.test
```

```
## $emmeans
##  Species emmean    SE   df lower.CL upper.CL
##  A         29.5 0.223 2994     29.0     29.9
##  B         30.4 0.223 2994     29.9     30.8
##  C         30.5 0.223 2994     30.1     31.0
##  D         30.0 0.223 2994     29.5     30.4
##  E         39.8 0.223 2994     39.3     40.2
##  F         50.0 0.223 2994     49.5     50.4
##
## Confidence level used: 0.95
##
## $contrasts
##  contrast estimate    SE   df t.ratio p.value
##  A - B      -0.893 0.316 2994  -2.829  0.0705
##  A - C      -1.058 0.316 2994  -3.353  0.0122
##  A - D      -0.488 0.316 2994  -1.546  1.0000
##  A - E     -10.311 0.316 2994 -32.657 <0.0001
##  A - F     -20.505 0.316 2994 -64.945 <0.0001
##  B - C      -0.165 0.316 2994  -0.524  1.0000
##  B - D       0.405 0.316 2994   1.283  1.0000
##  B - E      -9.417 0.316 2994 -29.828 <0.0001
##  B - F     -19.612 0.316 2994 -62.116 <0.0001
##  C - D       0.570 0.316 2994   1.806  1.0000
##  C - E      -9.252 0.316 2994 -29.304 <0.0001
##  C - F     -19.446 0.316 2994 -61.592 <0.0001
##  D - E      -9.822 0.316 2994 -31.110 <0.0001
##  D - F     -20.017 0.316 2994 -63.398 <0.0001
##  E - F     -10.194 0.316 2994 -32.288 <0.0001
##
## P value adjustment: bonferroni method for 15 tests
```

La funzione `emmeans()` ci fornisce due tabelle:

- la prima tabella contiene le stime delle masse corporee medie di ciascuna specie con i limiti inferiore e superiore dell'intervallo di confidenza al 95% (`lower.CL` e `upper.CL`);
- la seconda tabella contiene le differenze tra le medie per ciascuna coppia di specie, i parametri dei test t per valutare se siano significativamente diversi da zero, ed i corrispondenti valori di *p*.

Notate che la sintassi di `emmeans` è un po' diversa da quanto abbiamo usato finora. Bisogna un po' abituarcisi ma ne vale la pena, viste la sua potenza e flessibilità. L'argomento `adjust=` è analogo all'argomento `p.adjust.method=` della funzione `pairwise.t.test()`. L'argomento `pairwise` permette di specificare i predittori da utilizzare per i confronti a coppie. In questo caso

i predittori vanno specificati a destra di una tilde, ~: è una piccola variazione della sintassi usata per speficicare un modello all'interno della funzione `lm()`.

Possiamo esportare le tabelle prodotte da `emmeans()` (per inserirle in un articolo o in un report, ad esempio) con la funzione `write.csv()`. È necessario indicare a `write.csv()` il nome dell'oggetto da esportare ed il nome del file `.csv` da produrre:

```
write.csv(emmeans.test$contrasts,
          file="pairwise_tests.csv",
          row.names=F)
```

Così facendo, R produrrà un file di nome `pairwise_tests.csv` e lo salverà all'interno della nostra *working directory*.

Notate che ho selezionato quale elemento esportare tra quelli contenuti in `emmeans.test` usando il simbolo `$`, che abbiamo già usato in precedenza per selezionare una colonna all'interno di un `data.frame`. L'oggetto `emmeans.test` non è un `data.frame`, ma il simbolo `$` funziona ugualmente come "selezionatore".

8 Regressione lineare

Finora abbiamo esaminato esempi in cui la variabilità osservata nella variabile di risposta era spiegata da predittori di natura categorica. Vedremo ora come esaminare il potere esplicativo e predittivo di un predittore di tipo continuo sulla variabile di risposta di interesse. A differenza dei predittori categorici, i predittori di tipo continuo sono predittori che possono assumere qualunque valore numerico entro un certo intervallo. Immaginiamo il seguente scenario: stiamo studiando una specie d'alga e ci sembra di aver notato che l'alga raggiunga lunghezze differenti a profondità differenti. Vogliamo dunque verificare se ci sia una relazione lineare tra la lunghezza dell'alga e la profondità a cui cresce. Questo tipo di scenario è quello classico della regressione lineare. Vedremo che il protocollo di analisi che abbiamo delineato per modelli con predittori categorici si applica anche a questo scenario.

Immaginiamo di avere misurato una specie d'alga a varie profondità e di avere salvato i dati raccolti in un file che abbiamo chiamato *regression_data.csv*, che trovate a https://github.com/marcoplebani85/datasets/. Carichiamo i dati in R:

```
rm(list = ls())  # ripulisce la memoria di R
# definiamo la working directory:
setwd("~/Desktop/Corso_R")
# carichiamo il file di dati:
df <- read.csv("my_data/regression_data.csv")
str(df)

## 'data.frame':    85 obs. of  2 variables:
##  $ depth      : int  2 3 4 5 6 7 8 9 10 11 ...
##  $ alga.length: num  5.79 7.64 7.11 11.07 10.23 ...
```

Il dataset contiene due colonne numeriche: la variabile di risposta `alga.length`, cioè la lunghezza dei nostri campioni di alga, e la variabile esplicativa `depth`, profondità. Entrambe le variabili sono numeriche e continue. Rappresentiamo i dati:

```
par(family="Times", # tipo di caratteri da usare
    bty="L" # tipo di "scatola" da tracciare attorno al grafico
    )
plot(alga.length ~ depth, data=df,
     xlab="Profondità [m]", # personalizzazione dei nomi degli assi
     ylab="Lunghezza algale [cm]",
     xlim=c(0,20), # limiti degli assi
     ylim=c(0,25)
     )
```

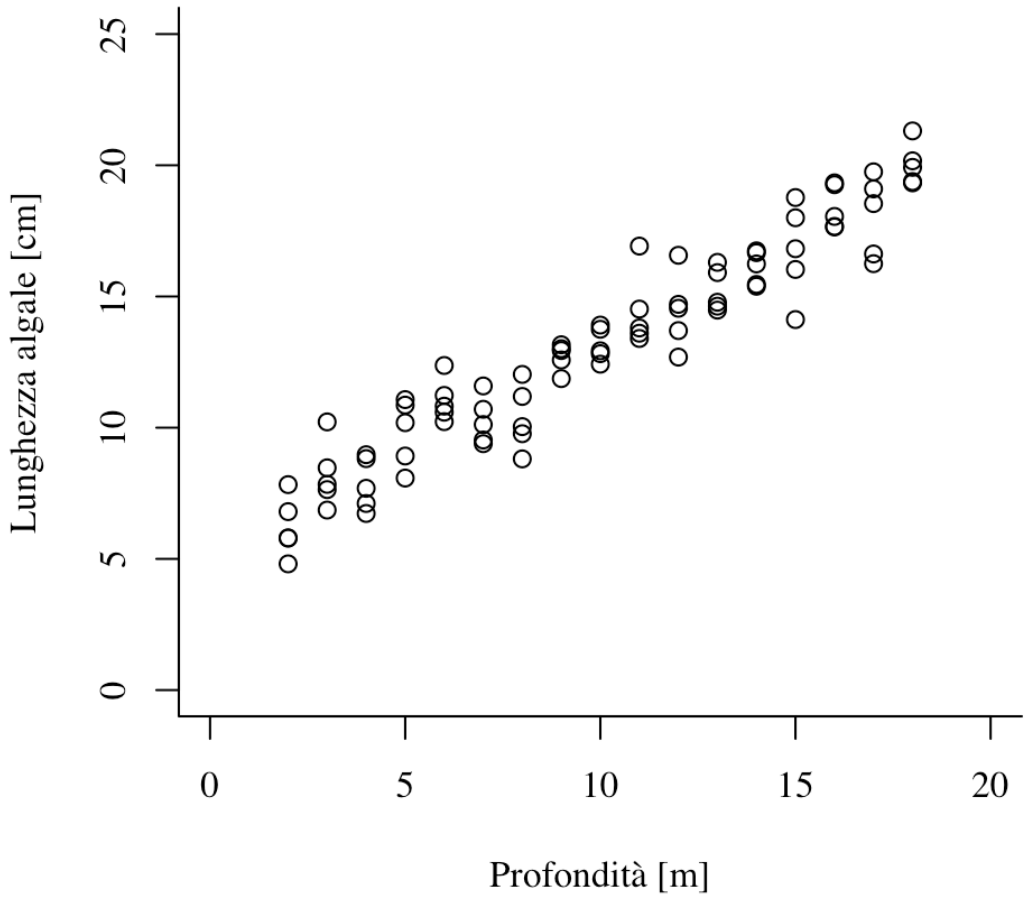

Figura 8.1: Lunghezza algale misurata a diverse profondità.

Osservando la figura 8.1 possiamo notare un'apparente correlazione positiva tra la lunghezza dell'alga e profondità. Definiamo il modello corrispondente, in modo da poterne stimare i parametri e valutarne la significatività statistica:

```
lm1 <- lm(alga.length ~ depth, data = df)
```

L'oggetto `lm1` definisce il modello secondo cui `alga.length` varia in funzione di `depth`, e specifica che i dati con cui stimare i parametri del modello sono contenuti nell'oggetto `df`.` Ispezioniamo ora i grafici diagnostici per assicurarci che il modello non violi le assunzioni dei modelli lineari:

```
par(mfrow = c(2, 2))
plot(lm1)
```

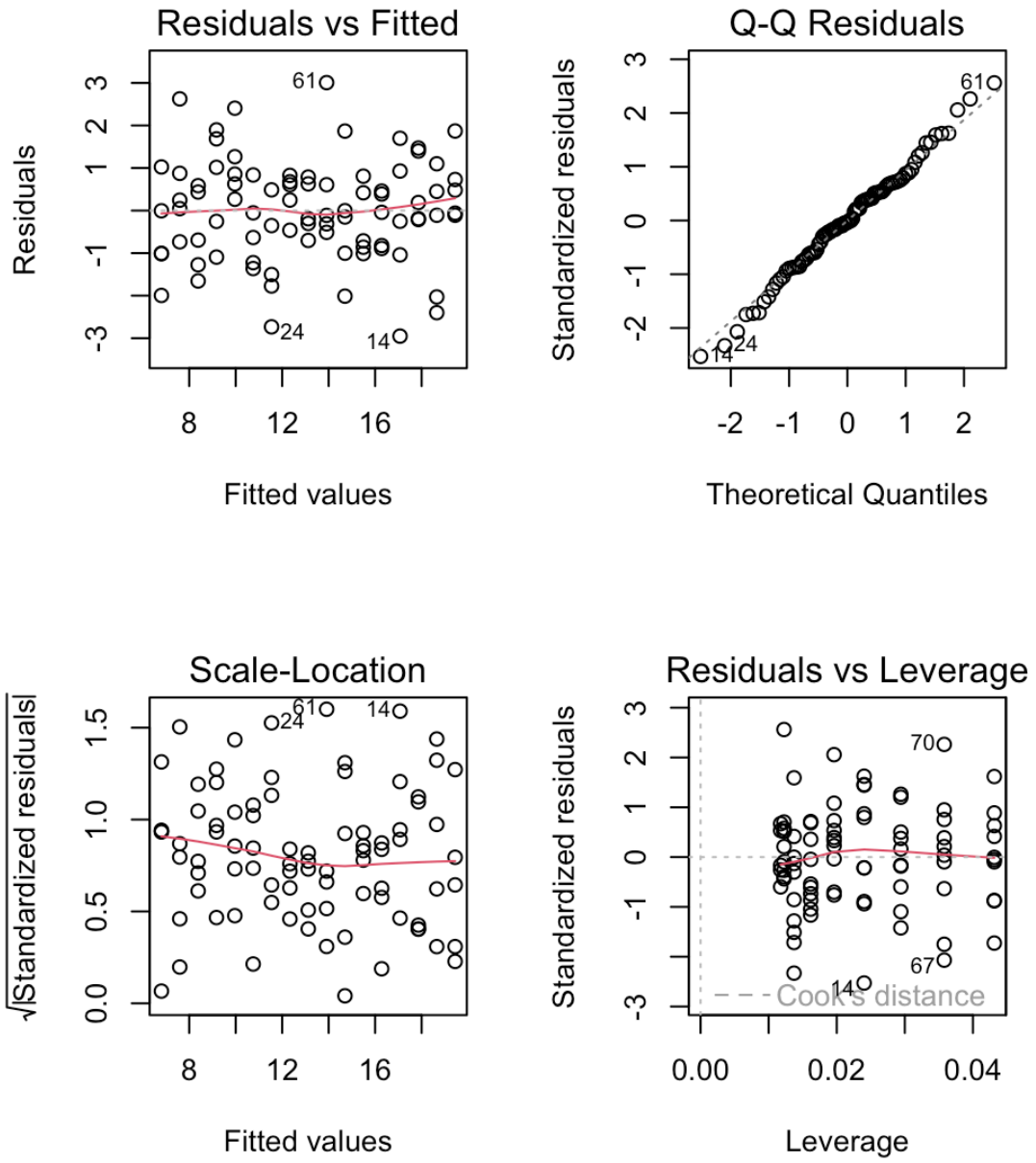

Figura 8.2: Grafici diagnostici per il modello di regressione lineare lm1.

Osservando la figura 8.2 non notiamo nulla di sospetto:

- i residui non mostrano andamenti rispetto ai valori stimati dal modello, suggerendo che un modello di correlazione lineare è adeguato ai dati;
- il grafico quantile-quantile indica che i residui hanno una distribuzione normale;
- il grafico *Scale-Location* non indica eteroscedasticità;
- il grafico *Residuals vs Leverage* non indica valori estremi.

Procediamo dunque a valutare il modello `lm1` rispetto al modello nullo, per il quale la lunghezza dell'alga in studio *non* varia in funzione della profondità a cui si trova. Definiamo il modello nullo `lm0`:

```
lm0 <- lm(alga.length ~ 1, data = df)
```

Confrontiamo ora `lm1` ed `lm0` usando un test F sulla variabilità residua dei due modelli:

```
anova(lm1, lm0)

## Analysis of Variance Table
##
## Model 1: alga.length ~ depth
## Model 2: alga.length ~ 1
##   Res.Df     RSS Df Sum of Sq      F    Pr(>F)
## 1     83  115.85
## 2     84 1388.01 -1   -1272.2 911.41 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

C'è una differenza significativa nella variabilità residua due modelli. I rapporti $RSS/Res.Df$ indicano che la variabilità residua di `lm1` è minore rispetto a quella di `lm0`, per cui possiamo rigettare il modello nullo (e di conseguenza l'ipotesi nulla). Il predittore `depth` spiega una porzione significativa della variabilità osservata in `alga.length`.

Nel caso della regressione lineare, `summary()` è di facile interpretazione:

```
summary(lm1)

##
## Call:
## lm(formula = alga.length ~ depth, data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.95102 -0.73478 -0.04133  0.72992  3.00773
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  5.22573    0.29128   17.94   <2e-16 ***
## depth        0.78969    0.02616   30.19   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.181 on 83 degrees of freedom
## Multiple R-squared:  0.9165, Adjusted R-squared:  0.9155
## F-statistic: 911.4 on 1 and 83 DF,  p-value: < 2.2e-16
```

La sezione centrale, `Coefficients`, ci informa sui parametri del modello. L'intercetta è data dall'`Estimate` per `(Intercept)` ed il coefficiente angolare è dato dall'`Estimate` per `depth`.

L'equazione della retta di regressione che descrive la relazione tra `alga.length` e `depth` è dunque:

$$\widehat{alga.length} = 5.23 + 0.79 \cdot depth \qquad (8.1)$$

Il sommario prodotto da `summary(lm1)` ci informa anche che l' *Adjusted R-squared* del modello è pari a 0.9155, ovvero il modello descrive il 92% circa della variabilità osservata nei dati.

Come rappresentare le predizioni del modello insieme ai nostri dati? Vedremo dei metodi esteticamente più appaganti in seguito, ma per ora possiamo usare la funzione `abline()` fornendole il modello di interesse come argomento (fig. 8.3):

```
par(mfrow=c(1,1),
    family="Times", # tipo di caratteri da usare
    bty="L" # tipo di "scatola" da tracciare attorno al grafico
    )
plot(alga.length ~ depth, data=df,
     xlab="Profondità [m]", # personalizzazione dei nomi degli assi
     ylab="Lunghezza algale [cm]",
     xlim=c(0,20), # limiti degli assi
     ylim=c(0,25)
     )
abline(lm1)
```

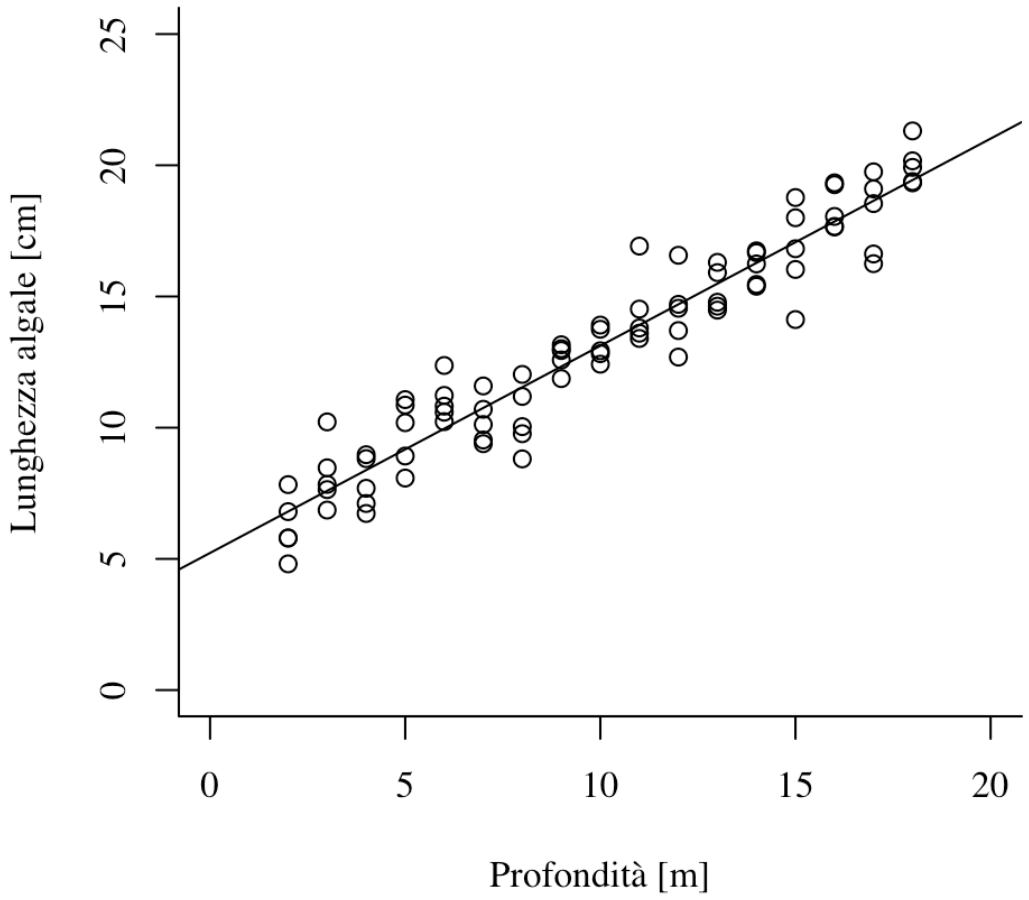

Figura 8.3: Lunghezza algale misurata a diverse profondità. La linea rappresenta i valori attesi secondo il modello `lm1`.

8.1 Comunicazione

Possiamo comunicare gli obiettivi, i metodi ed i risultati del nostro studio come segue:

Ci siamo posti l'obiettivo di quantificare la relazione tra la lunghezza di una specie d'alga e la profondità a cui cresce. Abbiamo campionato l'alga a profondità di 2-18 m sotto il livello del mare, ottenendo cinque repliche indipendenti a ciascuna profondità per un totale di 85 osservazioni.

Analisi. Sulla base dei dati raccolti abbiamo confrontato un modello di regressione lineare, secondo il quale la lunghezza algale varia linearmente al variare della profondità, con il modello nullo usando un test F sulla variabilità residua dei due modelli. Abbiamo svolto tutte le analisi

con R 4.5.2 (R Core Team, 2025). Abbiamo definito i modelli lineari con la funzione `lm()` e li abbiamo confrontati con la funzione `anova()`.

Risultati. Il modello di regressione lineare è risultato significativamente migliore di quello nullo nel descrivere la variabilità osservata (Tab. 4). La lunghezza algale varia al variare della profondità secondo il modello:

$$lunghezza_i = 5.23 + 0.79 \cdot profondita_i + \epsilon_i \qquad (8.2)$$

Ovvero: la lunghezza algale aumenta mediamente di 0.79 cm per ogni aumento di profondità pari ad un metro (Fig. 8.3; Tab. 5). Il modello spiega il 92% della variabilità totale osservata nelle misure di lunghezza algale.

Confronto tra modelli.

Predittori	df	RSS	F	p
Profondità	83	115.9		
Nessuno (modello nullo)	84	1388	911.4069	<0.0001

Parametri del modello di regressione lineare.

Parametri	Stima	Errore standard	t	p
Intercetta	5.226	0.291	17.94	<0.0001
Pendenza	0.79	0.026	30.19	<0.0001

8.2 Correlazione non implica causalità!

Bisogna resistere alla tentazione di parlare di un *effetto* del predittore sulla variabile di risposta. Solo in un contesto sperimentale e controllato potremmo inferire, con cautela, una causalità tra la variazione delle condizioni sperimentali e la variabilità osservata nella variabile di risposta. L'esempio che stiamo esaminando è di tipo osservativo, non sperimentale, e non permette questo tipo di inferenza. Ad esempio, la correlazione osservata tra lunghezza algale e profondità potrebbe essere una conseguenza di differenze di pressione, luminosità, abbondanza di nutrienti, densità di erbivori… Non avendo informazioni sperimentali al riguardo, le nostre affermazioni devono essere adeguatamente circostanziate.

9 Visualizzazione avanzata dei dati: introduzione a ggplot2

Finora siamo stati in grado di rappresentare adeguatamente i dati usando le funzioni `plot()`, `boxplot()` e `hist()`. Nei prossimi capitoli cominceremo ad avere a che fare con dataset più complessi e con modelli con più di un predittore, che non sono facilmente rappresentabili con le funzioni grafiche che abbiamo usato finora. Useremo invece le funzioni contenute nel pacchetto `ggplot2` (Wickham, 2016), che introdurrò in questo capitolo.

Il pacchetto `ggplot2` non è preinstallato in R e dovremo installarlo dal CRAN, ma prima di fare ciò procediamo ad installare il pacchetto `pacman` (Rinker e Kurkiewicz, 2018):

```
rm(list = ls())  # ripulisce la memoria di R

# carica i pacchetti:
if (!require(pacman)) install.packages("pacman")
library(pacman)
```

Il codice qui sopra contiene un `if` *statement* che dice a R: "se il pacchetto `pacman` non è installato, installalo; dopodichè, caricalo". Il pacchetto `pacman` non è indispensabile, ma contiene l'utilissima funzione `p_load()`. La funzione `p_load()` permette di caricare pacchetti presenti sul CRAN senza preoccuparci se siano installati oppure no: `p_load()` li installerà per noi, se necessario. In pratica la funzione `p_load()` svolge il compito delle funzioni `install.packages()` e `library()`, ed io la uso sempre al loro posto.

Procediamo quindi ad installare e caricare il pacchetto `ggplot2`:

```
p_load(ggplot2)
```

9.1 Rappresentazione di dati in relazione ad un predittore continuo

Carichiamo ora i dati relativi alla variazione della lunghezza algale a diverse profondità già visti nei capitoli precedenti:

```
setwd("~/Desktop/Corso_R")
df_regr <- read.csv("my_data/regression_data.csv")
```

Vediamo come produrre un grafico con `ggplot`:

```
pp1 <- ggplot(data = df_regr, aes(x = depth, y = alga.length))
```

L'argomento `aes` permette di definire i parametri estetici *(aesthetics)* del grafico. L'argomento `aes` permette, ad esempio, di specificare le variabili da riportare sugli assi *x* ed *y*.

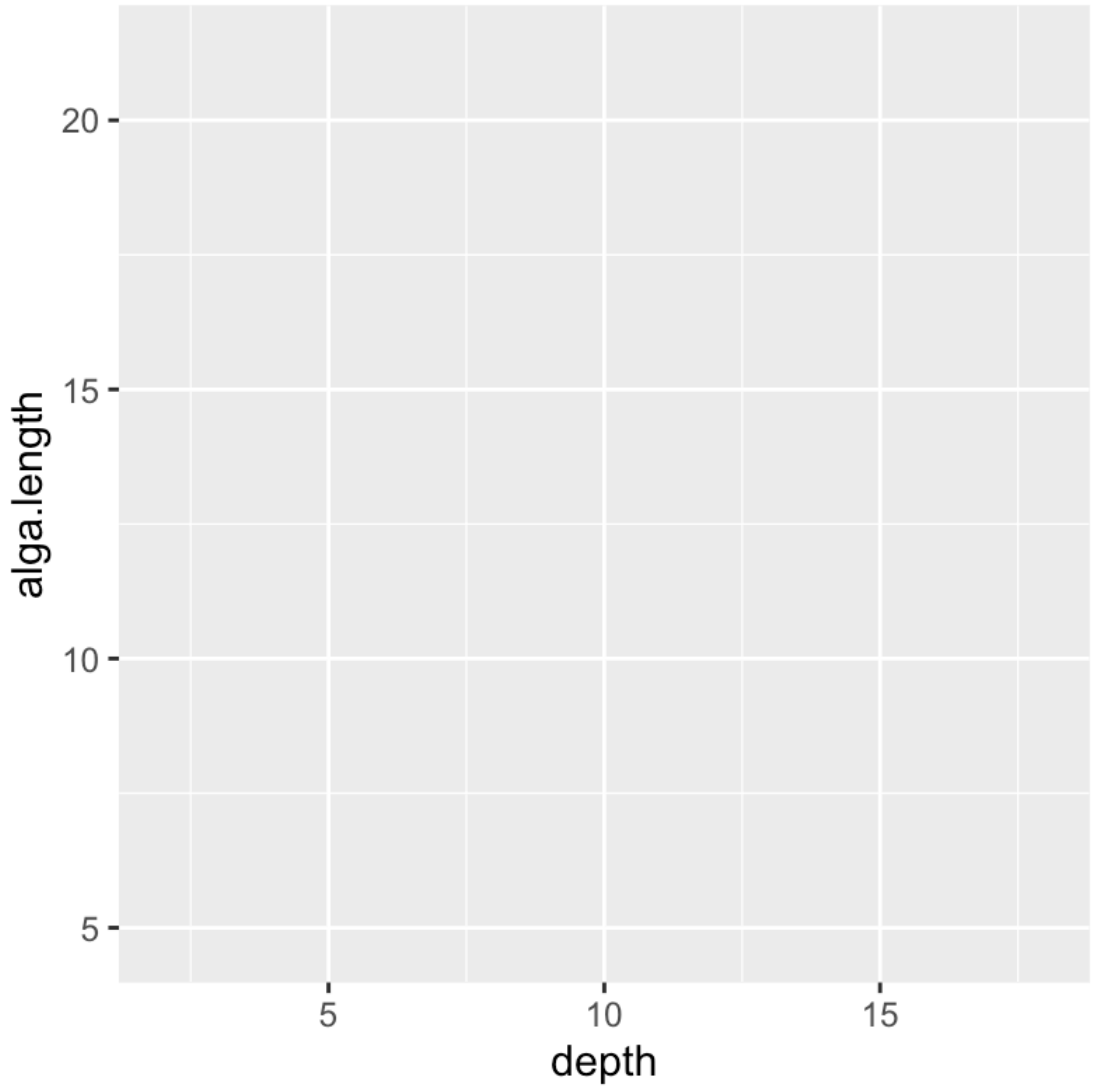

Figura 9.1: L'oggetto `pp1` *è il livello base su cui andremo a rappresentare i nostri dati.*

Se visualizziamo `pp1` notiamo che si tratta di un sistema di assi cartesiani vuoto (fig. 9.1). Il motivo è che non abbiamo ancora comunicato a `ggplot` che tipo di grafico vogliamo produrre. Per ora abbiamo solo creato il "livello base" del nostro grafico, la tela su cui rappresentare i nostri dati e modelli. Una volta creato questo "livello base" vi si possono sovrapporre altri livelli, chiamati "geoms":

```
pp1 +
    geom_point() + # rappresenta i dati con uno scatterplot
    geom_smooth() # aggiunge una smoothing line
```

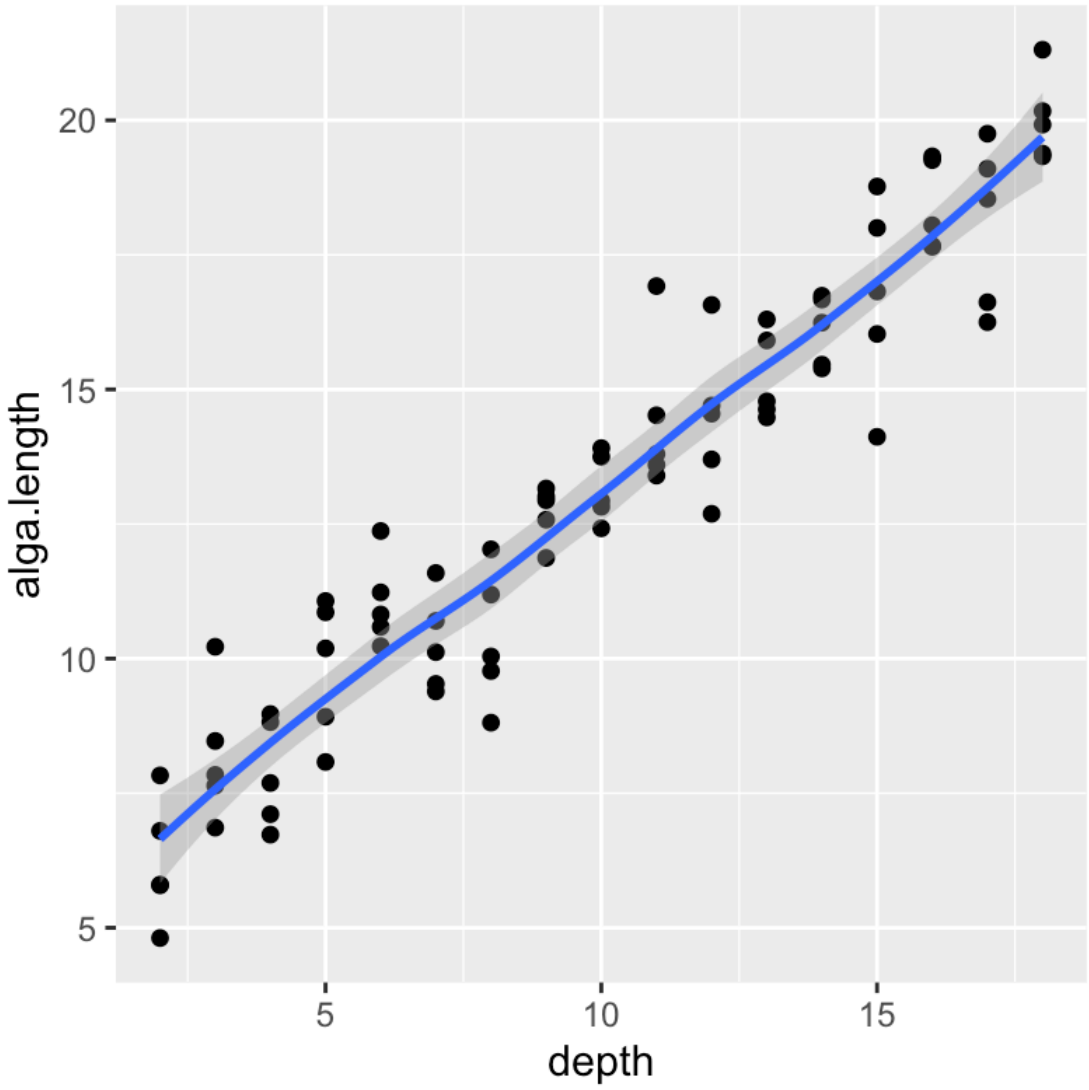

Figura 9.2: Variabilità della lunghezza algale a diverse profondità; la linea blu è una smoothing spline il cui intervallo di confidenza è rappresentato dall'area grigia.

La sintassi delle funzioni del pacchetto `ggplot2` è facile da comprendere tenendo presente che le sue funzioni agiscono per addizione: i grafici da esse creati sono una figura composta da vari livelli, ed ogni funzione crea o modifica un livello grafico. La figura 9.2 è stata ottenuta creando un livello base, `pp1`, con la funzione ggplot(). A questo livello base abbiamo sovrapposto i dati con la funzione `geom_point()`, che produce un grafico a dispersione. Abbiamo poi rappresentato una linea di *smoothing* (anche nota come *spline*) aggiungendo un ulteriore livello con la funzione `geom_smooth()`.

Possiamo usare `geom_smooth()` anche per rappresentare una retta di regressione lineare, specificando `method = "lm"` tra i suoi argomenti (fig. 9.3):

```
gg <- pp1 +
  geom_point() + # crea uno scatterplot
  geom_smooth(method = "lm", color="red") + # aggiunge retta di regressione i
```

```
n rosso
  theme_bw() # crea uno sfondo bianco anzichè grigio
```

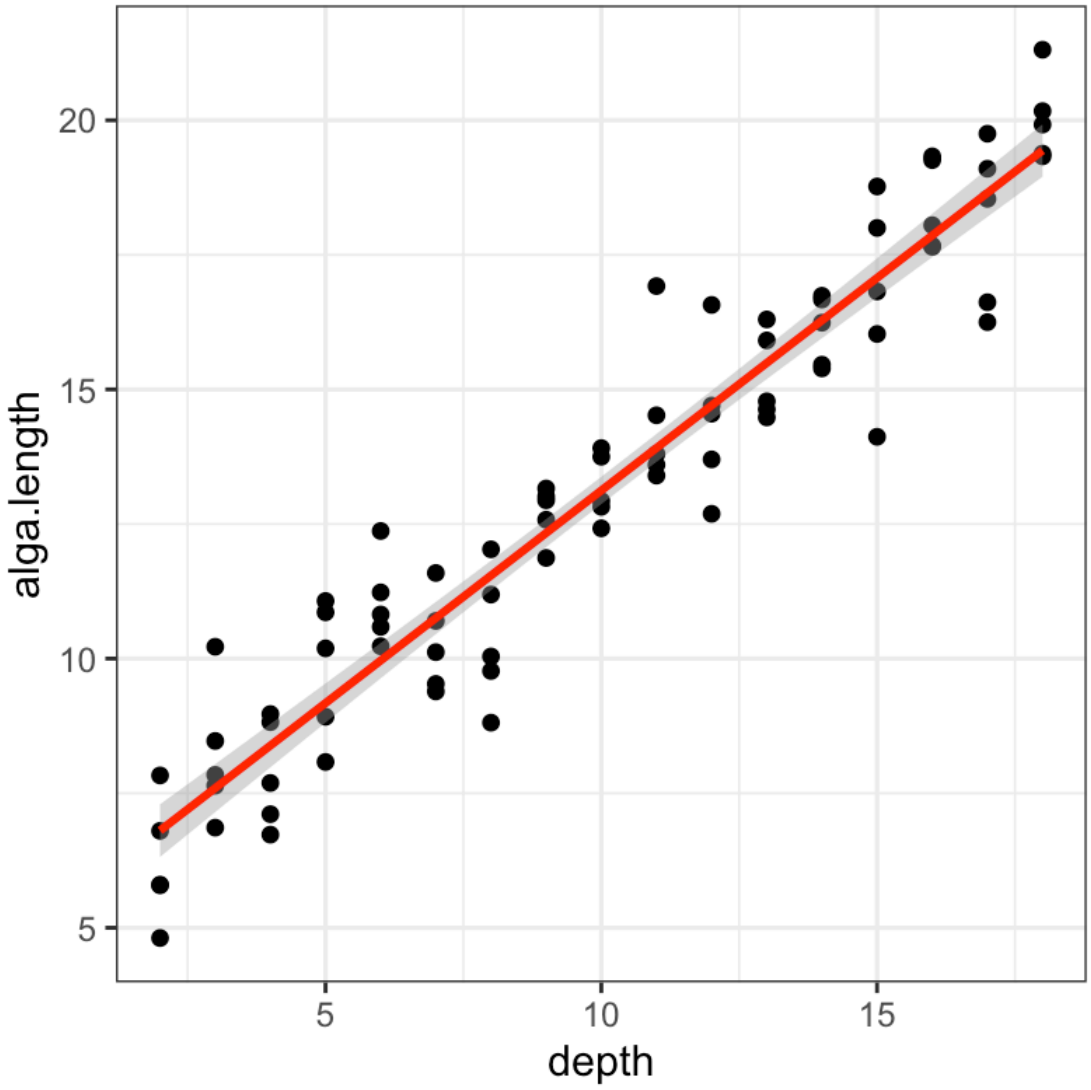

Figura 9.3: Variabilità della lunghezza algale a diverse profondità. La linea rossa è la retta di regressione associata ai dati, con l'intervallo di confidenza rappresentato in grigio.

Per ora, la funzione `ggplot()` ha solo creato l'oggetto `gg` contenente il grafico. Per visualizzare il grafico (fig. 9.3) dobbiamo digitare ed eseguire `gg` nel terminale di R.

Possiamo personalizzare il grafico mostrato in figura 9.3 usando le funzioni `labs()` e `theme()` per modificare gli assi ed il titolo di `gg` (fig. 9.4):

```
gg +
  labs(title = " ", x = "Profondità [m s.l.m.]", y = "Lunghezza algale [cm]")
 +
  # titolo (lasciato in bianco), etichette degli assi
  theme(plot.title = element_text(size = 20, hjust = 0.5), # dimensioni titol
```

```
o
      axis.text.x = element_text(size = 15), # dimensioni valori asse x
      axis.text.y = element_text(size = 15), # dimensioni valori asse y
      axis.title.x = element_text(size = 15), # dimensioni etichetta dell'ass
e x
      axis.title.y = element_text(size = 15) # dimensioni etichetta dell'asse
 y
      ) +
  geom_smooth(method = "lm", color="red") + # aggiunge retta di regressione
  theme_classic() # tema "classico" (simile a theme_bw ma senza reticolo)
```

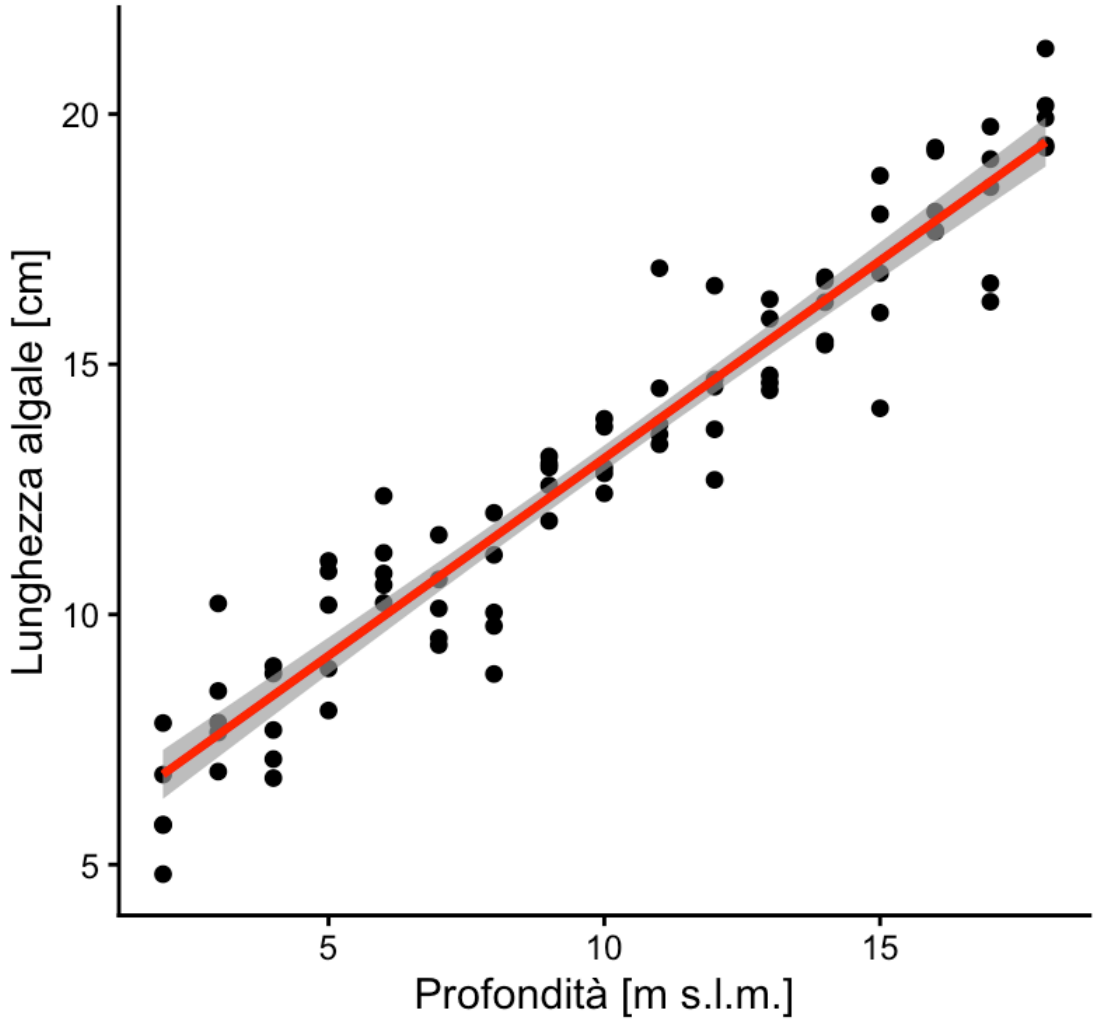

Figura 9.4: Variabilità della lunghezza algale a diverse profondità. La linea rossa è la retta di regressione associata ai dati, con l'intervallo di confidenza rappresentato in grigio.

9.2 Rappresentazione di dati in relazione ad un predittore categorico

Vediamo ora come rappresentare un dataset con un predittore categorico. Carichiamo i dati relativi alla massa corporea delle specie A-F già visti nella sez. 7:

```
df_6sp <- read.delim("my_data/compare_6_species.txt")
```

Cominciamo con il creare il primo livello del grafico, in cui comunichiamo a `ggplot()` quali dati rappresentare e cosa rappresentare sull'ascissa e sull'ordinata:

```
g0 <- ggplot(df_6sp, aes(y = body.mass, x = Species))
```

Possiamo rappresentare i dati con un grafico "scatola-e-baffi" usando `geom_boxplot()` (fig. 9.5a):

```
g1 <- g0 +
          geom_boxplot() + # grafico scatola-e-baffi
          labs(x="Specie", y = "Massa corporea [kg]") + # etichette degli ass
i
          theme_classic()
```

Possiamo anche rappresentare i dati con un "violin plot" usando `geom_violin()` (fig. 9.5b):

```
g2 <- g0 +
            geom_violin(aes(fill=Species)) + # per creare un violin plot
            theme_classic() +
            # personalizza le etichette degli assi:
            labs(x="Specie", y = "Massa corporea [kg]") +
            theme(legend.position = "none") # rimuove legenda
```

La funzione `geom_histogram()` permette di creare istogrammi (fig. 9.5c):

```
g3 <- ggplot(df_6sp, aes(x = body.mass, fill = Species)) + # informazioni bas
e
  geom_histogram(alpha = 0.4, position = "identity", bins=55, color="white")
+
  # personalizza le etichette degli assi:
  labs(x = "Massa corporea [kg]", y = "Numero di individui") +
  # personalizza il titolo della legenda:
  labs(fill="Specie") +
  theme_classic()
```

9.3 Creare figure contenenti più di un grafico

Come per i grafici prodotti da `plot()`, è possibile rappresentare più di un grafico di `ggplot2` all'interno di un unico pannello. Per farlo serve il pacchetto `ggpubr` (Kassambara, 2023):

```
p_load(ggpubr)  # per organizzare grafici prodotti da ggplot
```

Vediamo come usare `ggpubr` per "assemblare" i grafici `g1`, `g2`, e `g3` in un'unica figura (fig. 9.5):

```
ggarrange(g1, g2, g3, # grafici da rappresentare insieme
          labels = c("(a)", "(b)", "(c)"), # numerazione dei grafici
          ncol = 1, nrow = 3 # disposizione: un rigo, tre colonne
          )
```

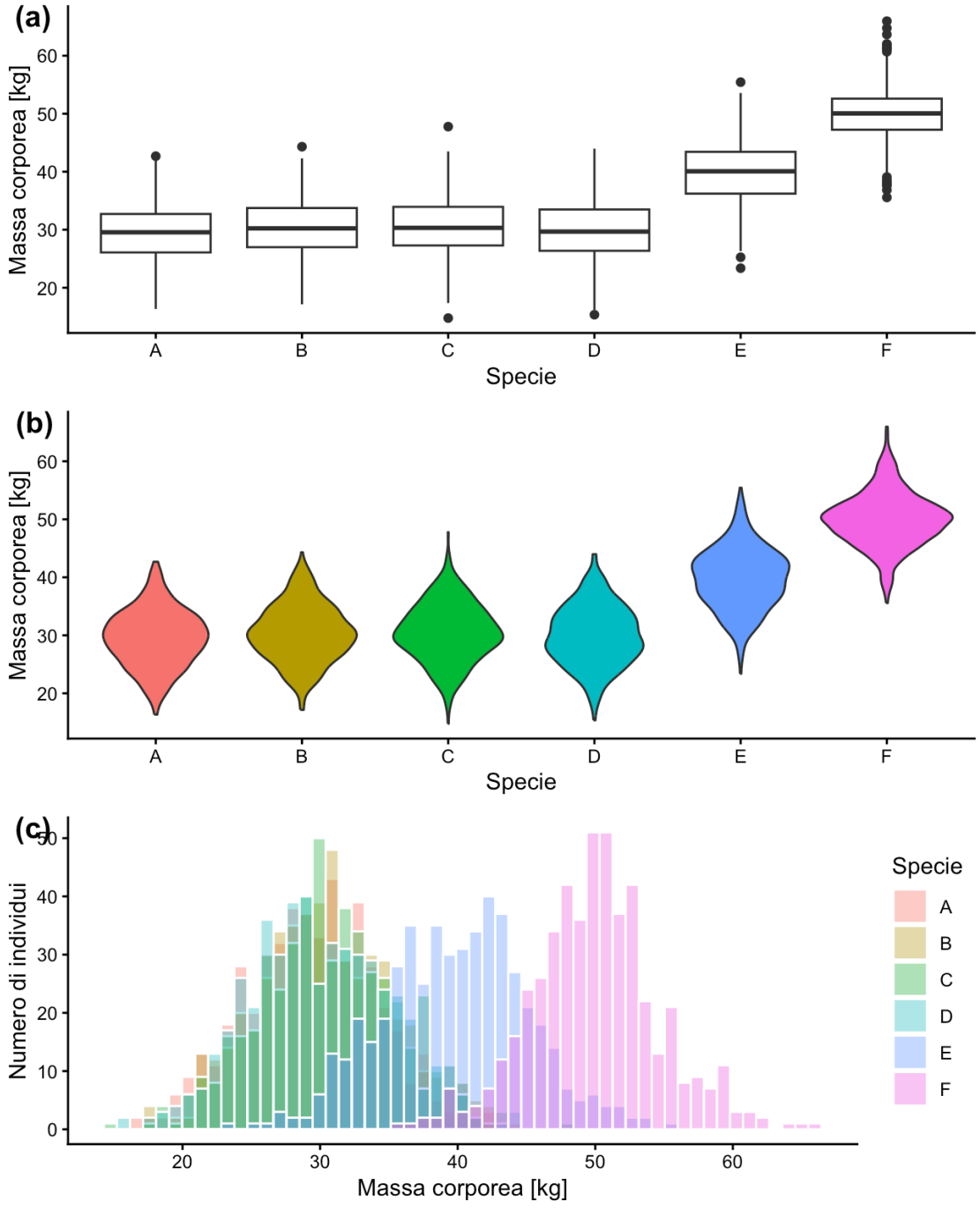

Figura 9.5: Variabilità delle masse corporee osservate per le specie A-F rappresentate in forma di grafico scatola-e-baffi (a), violin plot (b), e distribuzioni di frequenza (c).

La funzione `ggarrange()` si può usare all'interno di se stessa per creare layout più complessi (fig. 9.6):

```
ggarrange(g3, # prima riga.
          # seconda riga:
          ggarrange(g1, g2, ncol = 2, labels = c("(b)", "(c)")),
          nrow = 2,
          labels = "(a)" # etichetta di g3
          )
```

9.4 Esempi di pacchetti per usi specifici basati su `ggplot2`

Il pacchetto `ggpubr` è un esempio di uno dei tanti pacchetti satelliti a `ggplot2` che sono stati sviluppati per usi specifici. Un altro esempio è il pacchetto `ggridges` (Wilke, 2022):

```
p_load(ggridges)
```

Il pacchetto `ggridges` permette di rappresentare distribuzioni di frequenza su piani sfalsati (fig. 9.7). È particolarmente utile per casi in cui le distribuzioni di frequenza che si vogliono rappresentare sono largamente sovrapposte, come nel caso delle specie A-D del nostro dataset `df_6sp`:

```
ggplot(df_6sp, aes(x = body.mass, fill = Species, y = Species)) + # informazi
oni base
          ggridges::geom_density_ridges(alpha = 0.4, scale = 10) +
          # personalizza le etichette degli assi:
          labs(x = "Massa corporea [kg]",
               y = "Frequenza") +
          # personalizza il titolo della legenda:
          labs(fill="Specie") +
          theme_classic()
```

9.5 Considerazioni conclusive

Il pacchetto `ggplot2` usa una sintassi un po' diversa rispetto a quella a cui ci siamo abituati finora. Ci siamo abituati a modulare il funzionamento delle funzioni specificando argomenti all'interno della funzione stessa, ma nel caso delle funzioni del pacchetto `ggplot2`, esse possono anche interagire "per somma". Ad esempio, abbiamo visto che un grafico a dispersione prodotto con `ggplot2` è la somma delle funzioni `ggplot(...)` e `geom_point()`, mentre la somma `ggplot(...) + geom_boxplot()` produce un grafico scatola-e-baffi, e così via. Vedremo presto che la capacità di `ggplot2` di produrre figure complesse e pronte per la pubblicazione vale lo sforzo di imparare la sintassi specifica di questo pacchetto.

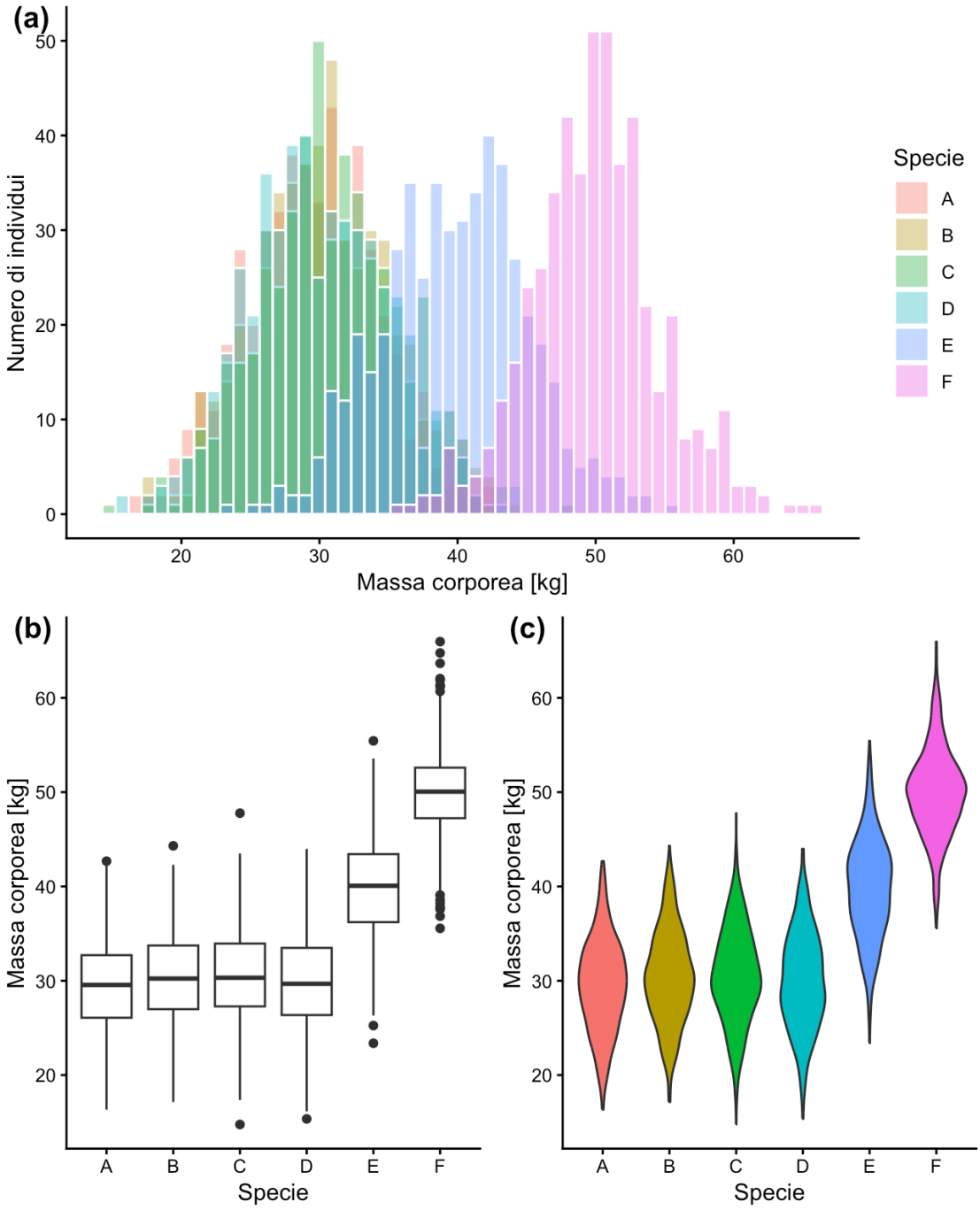

Figura 9.6: Variabilità delle masse corporee osservate per le specie A-F rappresentate in forma di distribuzioni di frequenza (a), grafico scatola-e-baffi (b), e violin plot (c).

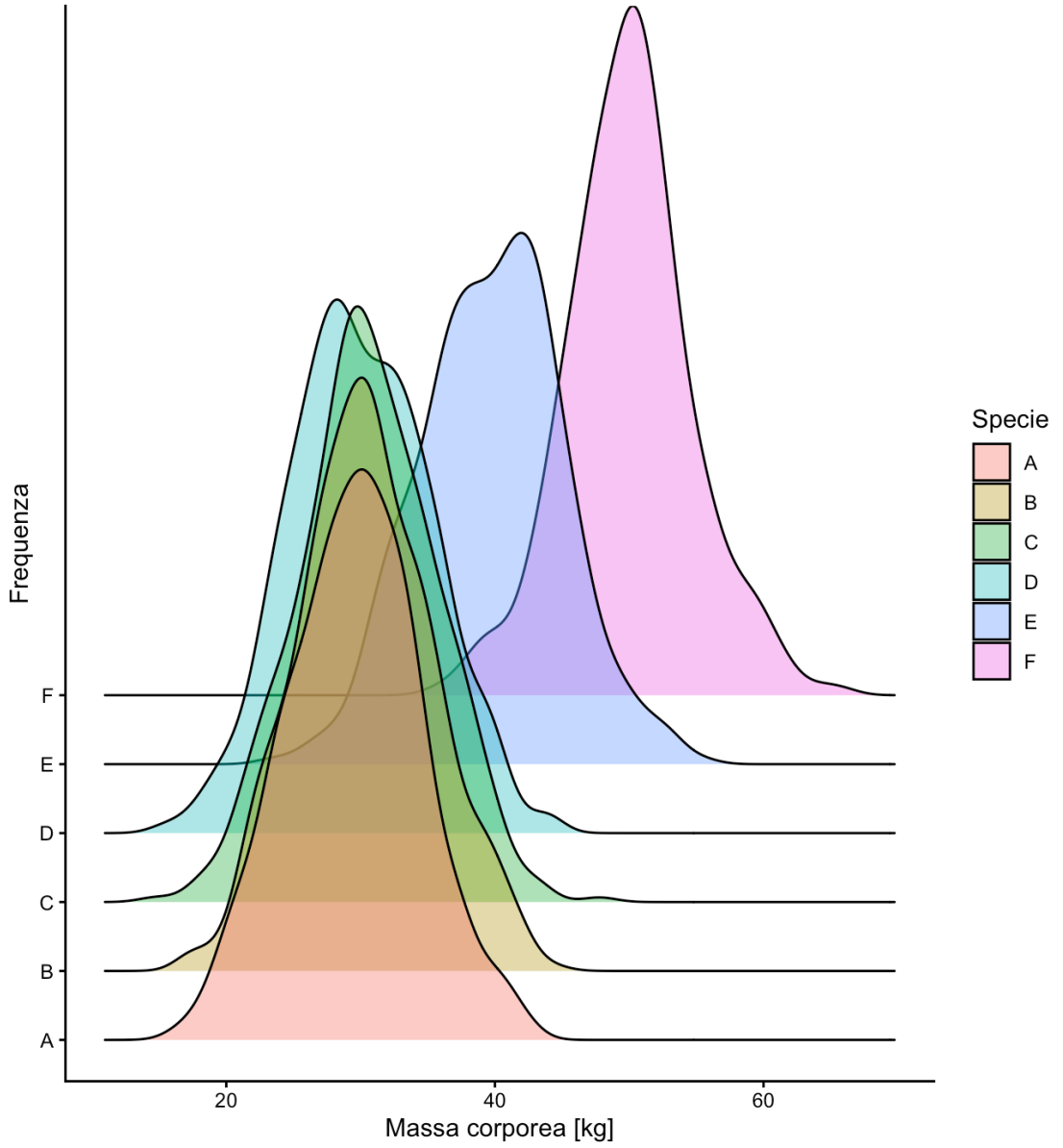

Figura 9.7: Distribuzioni di frequenza delle masse corporee osservate per le specie A-F.

Una lista di pacchetti basati su `ggplot2`, con esempi sul loro uso, è disponibile all'indirizzo: https://exts.ggplot2.tidyverse.org/.

10 Modelli lineari con più di un predittore

Finora abbiamo avuto a che fare con modelli ad un singolo predittore: data una variabile di risposta di interesse, volevamo capire se e come essa variasse in dipendenza del predittore. Nel caso in cui si voglia studiare l'effetto di più di un predittore sulla variabile di risposta le cose si complicano, ma si fanno anche più interessanti.

10.1 Capire le interazioni tra predittori

Consideriamo il caso in cui si voglia studiare l'effetto di due predittori x_1 e x_2 su una variabile di risposta y. Sono possibili i seguenti scenari, in ordine di complessità:

1. nessuno dei due predittori ha un effetto significativo sulla variabile di risposta y (ipotesi nulla).
2. solo uno dei due predittori ha un effetto significativo sulla variabile di risposta y.
3. entrambi i predittori hanno un effetto sulla variabile di risposta, e l'effetto di x_1 e quello di x_2 su y sono indipendenti l'uno dall'altro.
4. i predittori hanno un effetto *interattivo* sulla variabile di risposta: l'effetto di x_1 su y è modulato da x_2 e vice versa.

Per capire meglio ciascuno di questi scenari, consideriamo il seguente esempio ed esaminiamolo graficamente. Vogliamo esaminare la relazione tra la biomassa di *due* specie di piante, *Specie A* e *Specie B*, e l'abbondanza di zinco (Zn) nel terreno su cui crescono. In particolare, vogliamo capire se la relazione tra abbondanza di zinco e biomassa differisca nelle due specie, ovvero se le due specie rispondano diversamente all'abbondanza di zinco. In questo esempio la biomassa vegetale è la variabile di risposta, mentre i predittori sono l'abbondanza di zinco (di tipo continuo) e l'identità di specie (di tipo categorico ovvero fattoriale).

10.1.1 Scenario 1: l'ipotesi nulla

È possible che nessuno dei due predittori abbia un effetto significativo sulla biomassa vegetale: ovvero, che la biomassa vegetale media sia statisticamente indistinguibile nelle sue specie, e che non varii significativamente al variare della disponibilità di zinco. Questo scenario è rappresentato in figura 10.1a e corrisponde al seguente modello lineare:

$$H_0: biomassa_i = \mu + \epsilon_i \qquad (10.1)$$

Dove μ è la biomassa vegetale media a prescindere dall'identità di specie, e la variabilità attorno a μ è spiegata dagli scostamenti ϵ_i delle singole osservazioni.

10.1.2 Scenario 2

Un'altra possibilità è che la biomassa vegetale vari solo in dipendenza dell'identità di specie oppure della disponibilità di zinco nel terreno. Questi scenari sono rappresentati rispettivamente in figura 10.1b e 10.1c e corrispondono ai seguenti modelli lineari:

$$biomassa_i = \mu + b_k \cdot Specie_k + \epsilon_i \qquad (10.2)$$

$$biomassa_i = \mu + c_j \cdot Zinco_j + \epsilon_i \qquad (10.3)$$

10.1.3 Scenario 3: il modello *additivo*

È possibile che la biomassa vegetale vari linearmente in dipendenza sia dell'identità di specie, sia della disponibilità di zinco nel terreno. Questa situazione, rappresentata in fig. 10.1d, è descritta da un modello di tipo *additivo*:

$$biomassa_i = \mu + b_k \cdot Specie_k + c_j \cdot Zinco_j + \epsilon_i \qquad (10.4)$$

In questo caso la correlazione tra biomassa vegetale e i due predittori è detta di tipo *additivo* perchè la variabilità totale spiegata dal modello corrisponde alla somma della variabilità spiegata dall'identità di specie e della variabilità spiegata dalla disponibilità di zinco.

10.1.4 Scenario 4: il modello *interattivo*

È anche possibile che la correlazione tra biomassa vegetale e disponibilità di zinco differisca nelle due specie, ovvero che il grado di pendenza della retta di regressione che descrive la correlazione tra biomassa vegetale e disponibilità di zinco dipenda dall'identità di specie. Entrambe queste situazioni si possono descrivere con un modello di tipo *interattivo*:

$$\begin{aligned} biomassa_i &= \mu \\ &+ b_k \cdot Specie_k \\ &+ c_j \cdot Zinco_j \\ &+ d_{jk} \cdot Specie_k \cdot Zinco_j \\ &+ \epsilon_i \end{aligned} \qquad (10.5)$$

La correlazione tra biomassa vegetale e i due predittori in questo caso è di tipo *interattivo* perchè la variabilità totale spiegata dal modello corrisponde alla somma della variabilità spiegata dall'identità di specie, della variabilità spiegata dalla disponibilità di zinco, e della variabilità spiegata *dall'interazione* tra identità di specie e disponibilità di zinco.

Le figure 10.1e-f mostrano due possibili scenari risultanti da un effetto interattivo dei predittori sulla variabile di risposta. In fig. 10.1e vediamo il caso in cui la biomassa di *Specie B* aumenta all'aumentare della disponibilità di zinco, mentre la biomassa di *Specie A* non varia al variare della disponibilità di zinco. In fig. 10.1f vediamo il caso in cui la correlazione tra biomassa vegetale e disponibilità di zinco è positiva nel caso di *Specie A* e negativa nel caso di *Specie B*. In entrambi i casi, per capire l'effetto della disponibilità di zinco sulla biomassa vegetale è necessario prendere in considerazione l'identità della specie.

Nei capitoli seguenti vedremo come adattare il protocollo di analisi dei dati che abbiamo sviluppato finora per valutare modelli con predittori multipli.

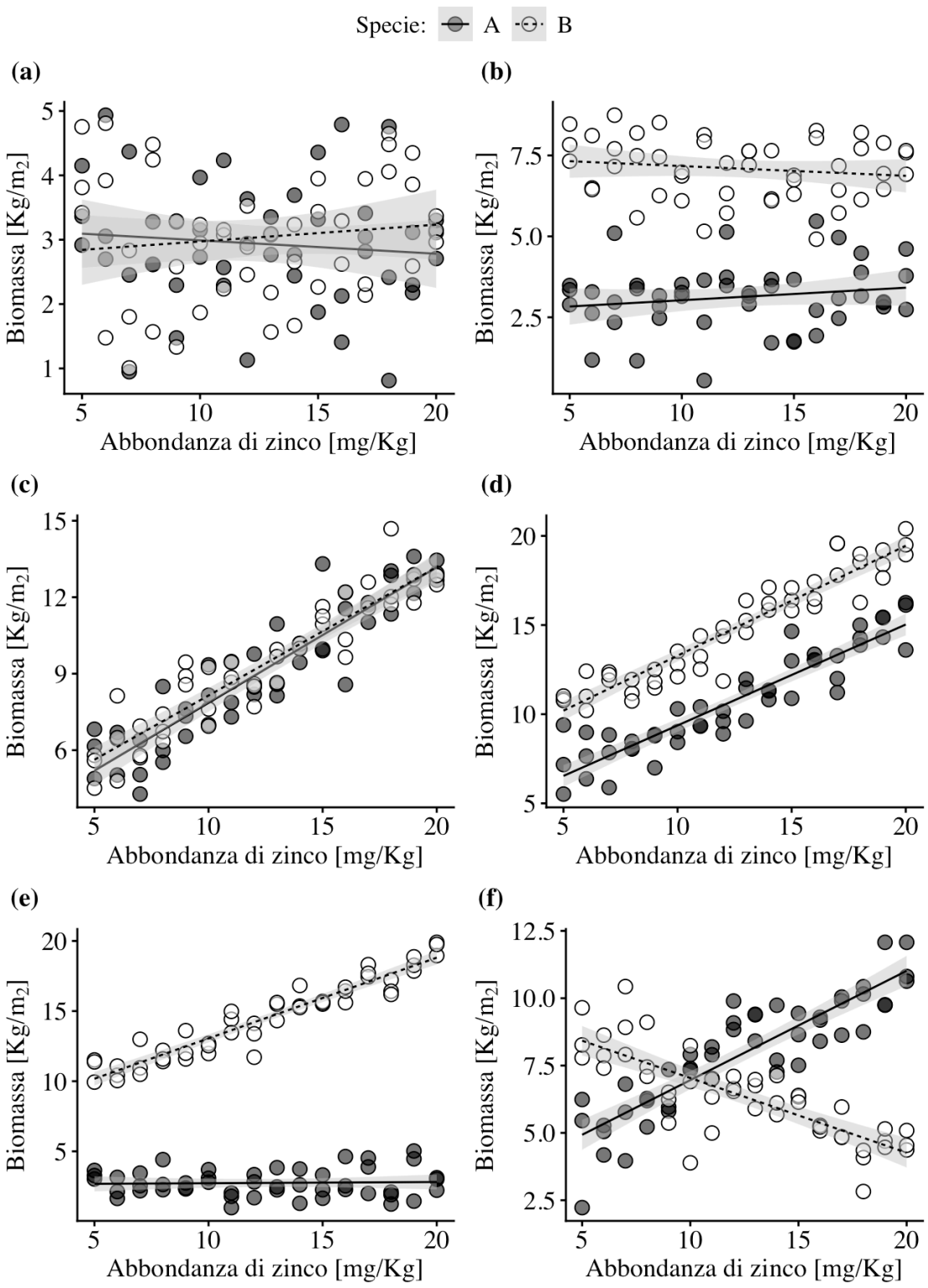

Figura 10.1: Scenari possibili.

10.2 Inclusione degli effetti principali nei modelli con interazione

Abbiamo visto che una variabile di risposta può variare in relazione all'effetto dei predittori nonchè in relazione all'effetto interattivo tra due o più predittori. L'effetto dei singoli predittori sulla variabile di risposta è noto come *effetto principale* o *main effect*, per distinguerlo dai possibili effetti interattivi tra predittori. Sebbene modelli caratterizzati da soli effetti interattivi senza *main effects* siano teoricamente possibili, nell'ambito dell'analisi dei dati è pratica comune includere sempre nel modello i *main effects* dei predittori nel caso in cui il modello includa la loro interazione.[5]

[5] Due pagine web rappresentative delle diverse scuole di pensiero al riguardo sono: https://archive.fo/3DcOc (archiviata il 9 febbraio 2025) e https://archive.fo/6vKiS (archiviata il 6 aprile 2022).

11 Il caso di un predittore continuo ed uno categorico: lo scenario "ANCOVA"

Vediamo un'applicazione pratica della casistica descritta nella sez. 10.1. Supponiamo di voler esaminare la relazione tra la lunghezza di *due* specie di alghe, *Specie A* e *Specie B*, e la profondità a cui crescono. In particolare, vogliamo capire se la relazione tra lunghezza algale e profondità differisce nelle due specie. Supponiamo di aver proceduto con il campionamento di entrambe le specie d'alga a profondità differenti, e di aver salvato i dati raccolti in un file `.csv` chiamato `ANCOVA_data.csv`, che trovate a https://github.com/marcoplebani85/datasets/.

I modelli lineari con un predittore fattoriale ed uno continuo sono tradizionalmente analizzati tramite Analisi della Covarianza, abbreviato come ANCOVA. In questa sezione non useremo l'ANCOVA; adatteremo invece il protocollo di analisi che abbiamo sviluppato finora per modelli con un solo predittore ai modelli lineari con predittori multipli. La differenza fondamentale tra l'analisi dei modelli con un solo predittore e quella dei modelli a predittori multipli è che, per questi ultimi, i modelli possibili da confrontare aumentano. Nel caso di un modello lineare con un singolo predittore l'unico modello lineare alternativo è quello nullo. Nel caso di un modello lineare a due predittori, i modelli a confronto sono tutti quelli descritti in sez. 10.1:

- il modello interattivo;
- il modello additivo;
- i modelli con un solo predittore;
- il modello nullo.

Il processo di selezione del modello rifletterà ciò.

Per prima cosa carichiamo i pacchetti che ci serviranno:

```
rm(list = ls())  # ripulisce la memoria di R
# carica il pacchetto 'pacman':
if (!require(pacman)) install.packages("pacman")
library(pacman)
# In pacman c'è la funzione p_load() che carica pacchetti previo

# installazione, se necessaria. Io lo uso sempre al posto di library().

p_load(ggplot2)  # per produrre grafici complessi
p_load(emmeans)  # per stimare i parametri e le predizioni del modello
```

Indichiamo ora la directory di lavoro a R, come mostrato in sez. 7:

```
setwd("~/Desktop/Corso_R")
```

Fatto ciò possiamo caricare i dati contenuti in `ANCOVA_data.csv`:

```
df <- read.csv("my_data/ANCOVA_data.csv")
```

Comunichiamo a R che `Species` è una variabile categorica:

```
df$Species <- as.factor(df$Species)
```

Visualizziamo i dati:

```
ggplot(data=df, aes(x=depth, y=alga.length, color=Species)) +
  geom_point() +
  labs(x = "Profondità [m]", y = "Lunghezza algale [cm]") +
  theme_classic()
```

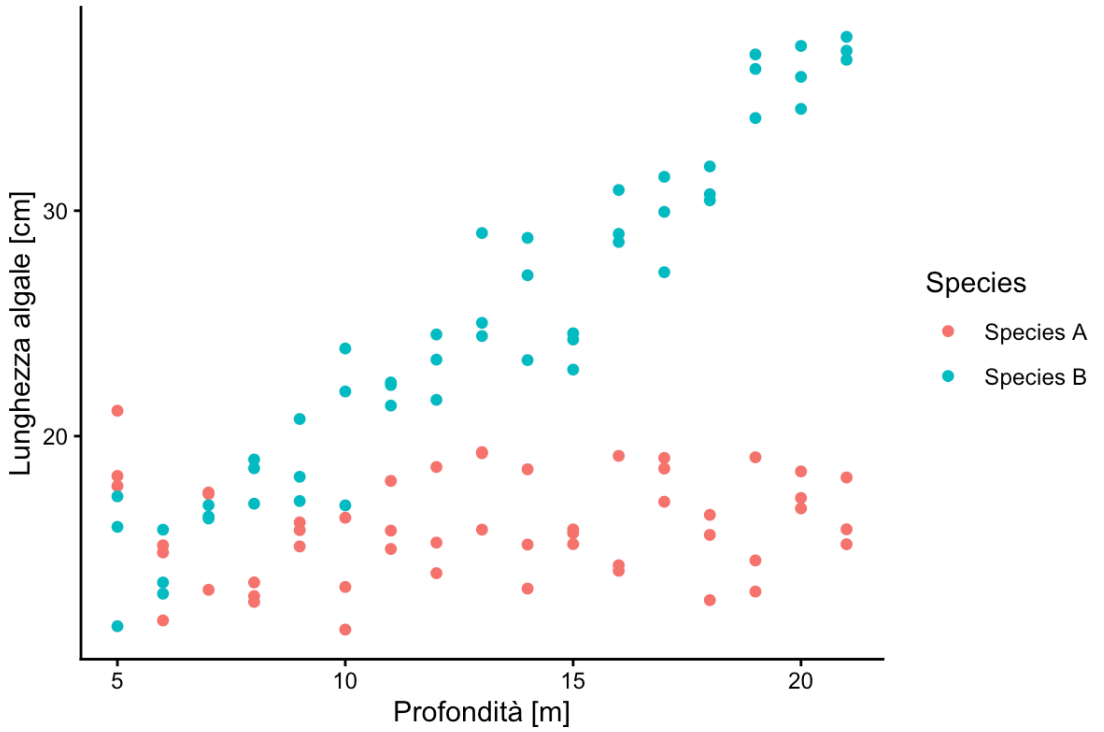

Figura 11.1: Lunghezza delle specie algali A e B a profondità di 5-21 m sotto il livello del mare.

Ad un esame visivo si direbbe che la correlazione tra taglia e profondità differisca nelle due specie d'alga (fig. 11.1). Si tratta di uno scenario che potrebbe essere ben descritto da un modello lineare di tipo interattivo. Possiamo definire questo modello in R come segue:

```
lm.interactive <- lm(alga.length ~ Species * depth, data = df)
```

Il modo in cui ho definito `lm.interactive` è la versione "compatta" del modello:

```
alga.length ~ Species + depth + Species:depth
```

I due punti in `Species:depth` stanno ad indicare l'effetto interattivo tra i due predittori. In R questi due modi di scrivere il modello sono equivalenti. Se scriviamo solo `alga.length ~ Species * depth`, R capisce che stiamo sottintendendo la presenza nel modello degli effetti di `Species` e `depth` - detti "main effects" - e della loro interazione, `Species:depth`.

Definiamo ora i possibili modelli lineari alternativi a quello interattivo (anche detto *full model* o modello completo). Il modello additivo si esprime come:

```
lm.additive <- lm(alga.length ~ Species + depth, data = df)
```

Gli altri possibili modelli lineari sono:

```
lm.only.species <- lm(alga.length ~ Species, data = df)
lm.only.depth <- lm(alga.length ~ depth, data = df)
lm.null <- lm(alga.length ~ 1, data = df)
```

Ciascuno di questi modelli rappresenta una semplificazione di vario grado rispetto al modello interattivo. I grafici diagnostici per ciascun modello sono mostrati nelle figure 11.3-11.7 (in appendice a questa sezione). Come menzionato nei capitoli precedenti, se i grafici diagnostici di un modello hanno un aspetto problematico potrebbe significare che il modello in questione sia inadeguato per descrivere i dati. Le possibili ragioni possono essere:

- non-normalità dei residui;
- la relazione tra le variabili potrebbe non essere lineare;
- il modello potrebbe non includere importanti predittori.

Dei grafici diagnostici "belli" sono un indizio (non una prova) della bontà di un modello. Nel nostro esempio gli unici grafici diagnostici che non hanno un aspetto problematico sono quelli del modello interattivo `lm.interactive`.

Procediamo con una selezione formale del migliore tra i cinque modelli in esame. Cominciamo col confrontare il più complesso e quello di complessità immediatamente minore, ovvero `lm.interactive` e `lm.additive`, con un test F sulle loro variabilità residue:

```
anova(lm.interactive, lm.additive)
```

```
## Analysis of Variance Table
##
## Model 1: alga.length ~ Species * depth
## Model 2: alga.length ~ Species + depth
##   Res.Df     RSS Df Sum of Sq      F    Pr(>F)
## 1     98  450.74
## 2     99 1607.92 -1   -1157.2 251.59 < 2.2e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

I due modelli differiscono significativamente. Il valore di $RSS/Res.Df$, che stima la variabilità residua dei modelli, è inferiore nel modello `lm.interactive`: di conseguenza il modello interattivo, `lm.interactive`, spiega una frazione maggiore della variabilità osservata rispetto al modello additivo, `lm.additive`. Come già menzionato in precedenza, se non avessimo riscontrato una differenza significativa tra i modelli avremmo rigettato il più complesso dei due e considerato quello più semplice come il "vincitore", secondo il principio del rasoio di Occam. Se avessimo idenfiticato il modello `lm.additive` come il migliore tra i due, il passo successivo sarebbe stato confrontarlo con i modelli `lm.only.species` ed `lm.only.depth`.

Un altro approccio alla selezione del modello migliore è tramite l' *Akaike Information Criterion* (AIC) usando la funzione `AIC()`:

```
AIC(lm.interactive, lm.additive, lm.only.species, lm.only.depth,
    lm.null)

##                 df      AIC
## lm.interactive   5 451.0282
## lm.additive      4 578.7511
## lm.only.species  3 639.7869
## lm.only.depth    3 660.7879
## lm.null          2 691.2850
```

L'AIC è una misura della bontà di un modello che penalizza i modelli con molti parametri. L'obiettivo dell'AIC è di identificare il modello che approssima al meglio i dati osservati con il minor numero di parametri, ovvero il modello che coniuga al meglio semplicità e potere esplicativo[6]. Più basso è l'AIC, migliore è il modello. La soglia di differenza per considerare due modelli "significativamente" diversi secondo l'AIC non è univocamente riconosciuta. Harrison et al. (2018) suggeriscono come valore-soglia una differenza di AIC (ΔAIC) di almeno sei unità.

Nota: il test F sulla variabilità dei residui di due modelli può essere eseguito solo per confrontare modelli in cui uno è una versione semplificata dell'altro. Ad esempio, il test F sul seguente confronto non è fattibile:

```
anova(lm.only.species, lm.only.depth)

## Analysis of Variance Table
##
## Model 1: alga.length ~ Species
## Model 2: alga.length ~ depth
##   Res.Df  RSS Df Sum of Sq F Pr(>F)
## 1    100 2983
## 2    100 3665  0   -681.98
```

Al contrario, l'AIC può essere usato per comparare la bontà di modelli anche non gerarchizzati tra loro, a patto che siano applicati allo stesso set di dati (Akaike, 1985). Ad esempio:

```
AIC(lm.only.species, lm.only.depth)

##                 df      AIC
## lm.only.species  3 639.7869
## lm.only.depth    3 660.7879
```

Abbiamo dunque appurato che `lm.interactive` è il modello migliore tra quelli in esame per descrivere i dati osservati. Possiamo usare `summary(lm.interactive)` per estrarre l' *adjusted R-squared*, che è ~0.908: il modello interattivo spiega il 90.8% della variabilità osservata nella lunghezza algale di *Specie A* e *Specie B*.

[6] https://archive.fo/5H3dN

Possiamo usare le funzioni del pacchetto `emmeans` per estrarre i parametri del modello. Nella sezione 7 abbiamo introdotto la funzione `emmeans()` per calcolare le medie marginali di un modello con un predittore fattoriale. Nel caso di modelli con un mix di predittori continui e categorici conviene usare la funzione `emtrends()`. Possiamo distinguere le due funzioni come segue: `emmeans()` calcola stime e statistiche per singoli punti; `emtrends()` calcola stime e statistiche per andamenti e pendenze. Per ottenere le stime di pendenza delle due specie e confrontarle con un test t eseguiamo:

```
pendenze <- emtrends(lm.interactive, # modello di interesse
      pairwise ~ Species, # predittori categorici
      var = "depth") # predittore continuo
pendenze
```

```
## $emtrends
##  Species   depth.trend     SE df lower.CL upper.CL
##  Species A       0.062 0.0613 98  -0.0597    0.184
##  Species B       1.437 0.0613 98   1.3154    1.559
##
## Confidence level used: 0.95
##
## $contrasts
##  contrast              estimate     SE df t.ratio p.value
##  Species A - Species B    -1.38 0.0867 98 -15.862 <0.0001
```

Il risultato della funzione `emtrends()` è suddiviso in due sezioni. La sezione `$emtrends` fornisce la stima della pendenza (`depth.trend`) delle rette che descrivono la relazione tra lunghezza algale e profondità nelle due specie, oltre ai rispettivi errori standard ed ai limiti degli intervalli di confidenza. La sezione `$contrasts` produce i risultati del test t sulla differenza delle pendenze nelle due specie, che in questo caso differiscono significativamente l'una dall'altra ($p < 0.0001$).

Per ottenere la significatività di ciascuna pendenza (ovvero per sapere se differiscono significativamente da zero) eseguiamo:

```
emmeans::test(pendenze)
```

```
## $emtrends
##  Species   depth.trend     SE df t.ratio p.value
##  Species A       0.062 0.0613 98   1.011  0.3147
##  Species B       1.437 0.0613 98  23.442 <0.0001
##
##
## $contrasts
##  contrast              estimate     SE df t.ratio p.value
##  Species A - Species B    -1.38 0.0867 98 -15.862 <0.0001
```

In questo caso la sezione `$emtrends` fornisce nuovamente la stima della pendenza della retta che descrive la relazione tra lunghezza algale e profondità in ciascuna specie, associata però ai risultati dei test t sulla differenza di ciascuna pendenza da zero. In questo caso la pendenza

associata alla specie A non differisce significativamente da zero (p=0.3147), mentre la pendenza associata alla specie B differisce significativamente da zero (p<0.0001).

Per ottenere le stime di intercetta eseguiamo:

```
emmeans(lm.interactive,
            specs = ~ Species * depth,
            at = list(depth = 0)
            )
```

```
##  Species   depth emmean    SE df lower.CL upper.CL
##  Species A     0  15.16 0.852 98    13.47    16.85
##  Species B     0   6.27 0.852 98     4.58     7.96
##
## Confidence level used: 0.95
```

Queste intercette indicano la taglia algale attesa ad una profondità pari a zero (`depth=0`), ovvero in superficie. I nostri dati però si riferiscono a misurazioni ottenute a profondità tra i 5 e i 21 m sotto il livello del mare. Se le due specie d'alga crescono solo a queste profondità, i valori di intercetta calcolati per 0 m di profondità non si prestano ad una interpretazione biologica. Si può chiedere ad `emmeans()` di stimare la taglia algale ad altre profondità usando l'argomento "`at=`":

```
emmeans(lm.interactive,
        specs = ~ Species * depth,
        at = list(depth = c(5, 20))
        )
```

```
##  Species   depth emmean    SE df lower.CL upper.CL
##  Species A     5   15.5 0.575 98     14.3     16.6
##  Species B     5   13.5 0.575 98     12.3     14.6
##  Species A    20   16.4 0.524 98     15.4     17.4
##  Species B    20   35.0 0.524 98     34.0     36.0
##
## Confidence level used: 0.95
```

Rappresentiamo ora le predizioni del modello sui dati (Fig. 11.2), lavorando sui dettagli del grafico in modo che sia pronto per la pubblicazione. Per prima cosa cominciamo col creare una nuova colonna `Specie` che riporta le stesse informazioni della colonna `Species` ma in italiano:

```
df$Specie <- ifelse(df$Species == "Species A", "A", "B")
```

Questa colonna ci permette di creare agevolmente i grafici dei nostri dati senza dover tradurre la legenda.

Cominciamo col creare la "base" del grafico:

```
grafico_alghe <- ggplot(data=df, aes(x=depth, y=alga.length, fill=Specie, col
or=Specie)) +
  geom_point(shape=21, cex=3.5, color="black") +
  # shape indica il tipo di punto da usare
  # fissiamo i limiti dell'asse y:
  scale_y_continuous(limits=c(9, 35)) +
```

```
  # etichettiamo gli assi:
  labs(x = "Profondità [m]", y = "Lunghezza algale [cm]") +
  theme_classic() +
  # impostiamo i caratteri di testo:
  theme(text = element_text(family="Times", size=15))
```

Ora aggiungiamo a `grafico_alghe` le informazioni per personalizzare i colori e le trasparenze e per aggiungere le predizioni del modello:

```
grafico_alghe +
  scale_color_manual(values=c("black", "black")) +
  scale_fill_manual(values=c(alpha("black", 0.5), alpha("white", 0.5))) +
  geom_smooth(method="lm",  fill="grey", linewidth=0.5, aes(linetype=Specie))
```

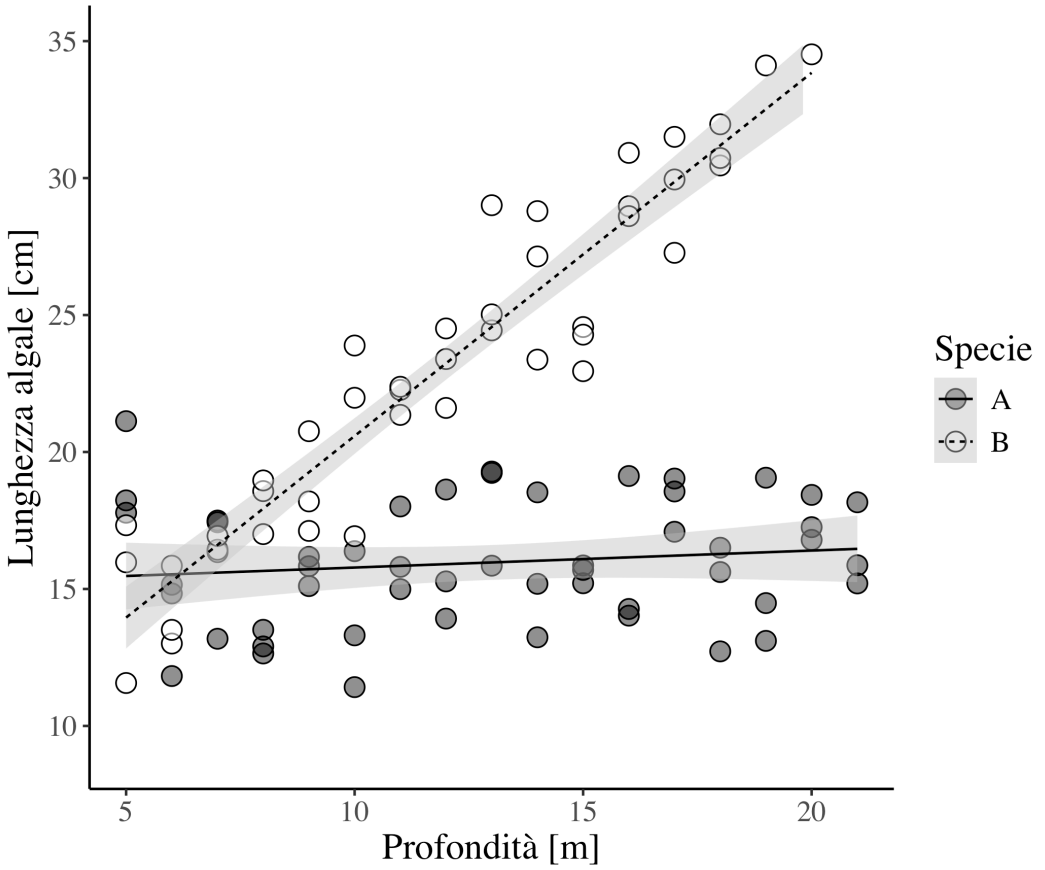

Figura 11.2: Dati e predizioni per il modello con interazione.

11.1 Appendice: grafici diagnostici

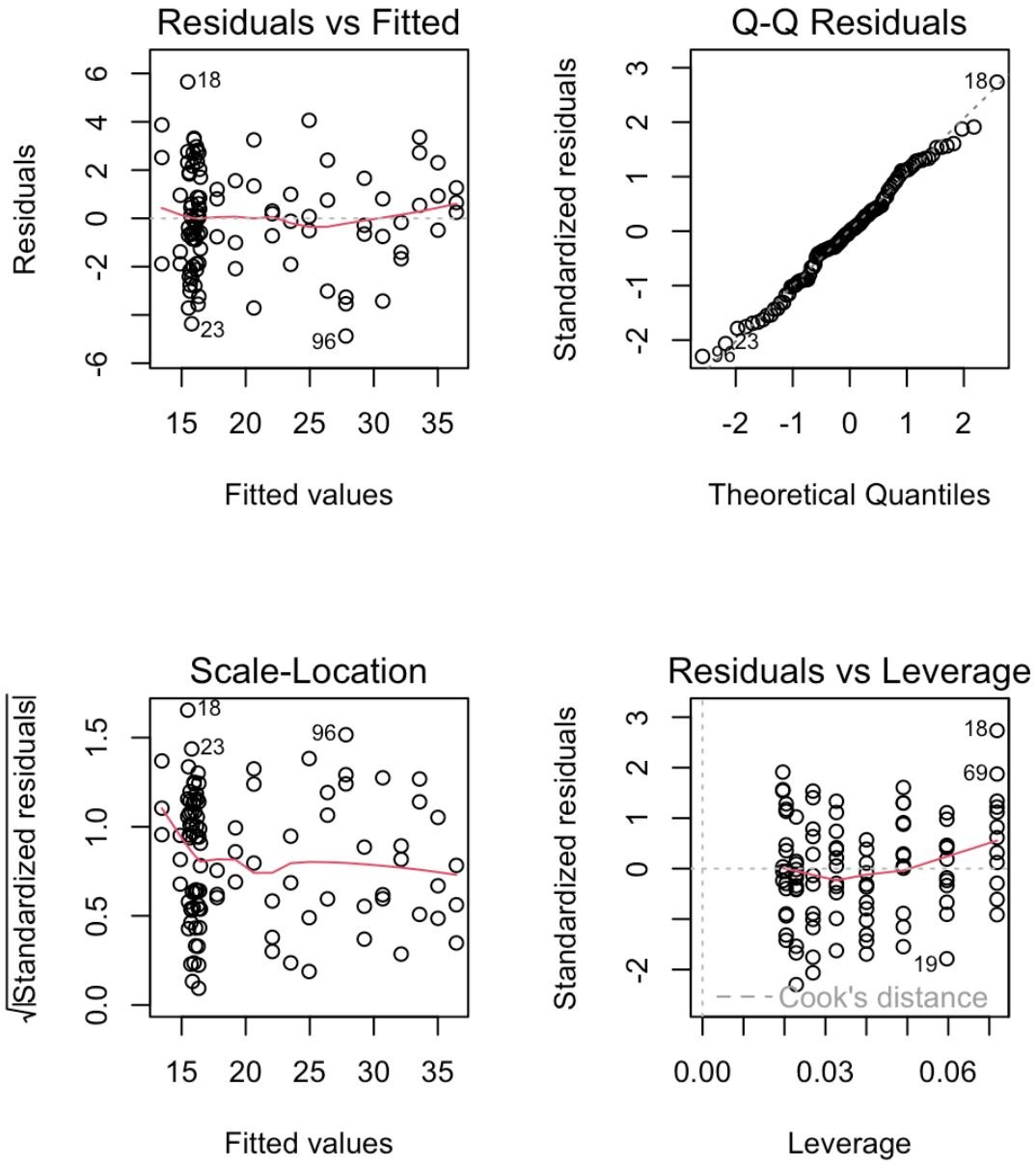

Figura 11.3: I grafici diagnostici per `lm.interactive` *non suggeriscono violazioni delle assunzioni.*

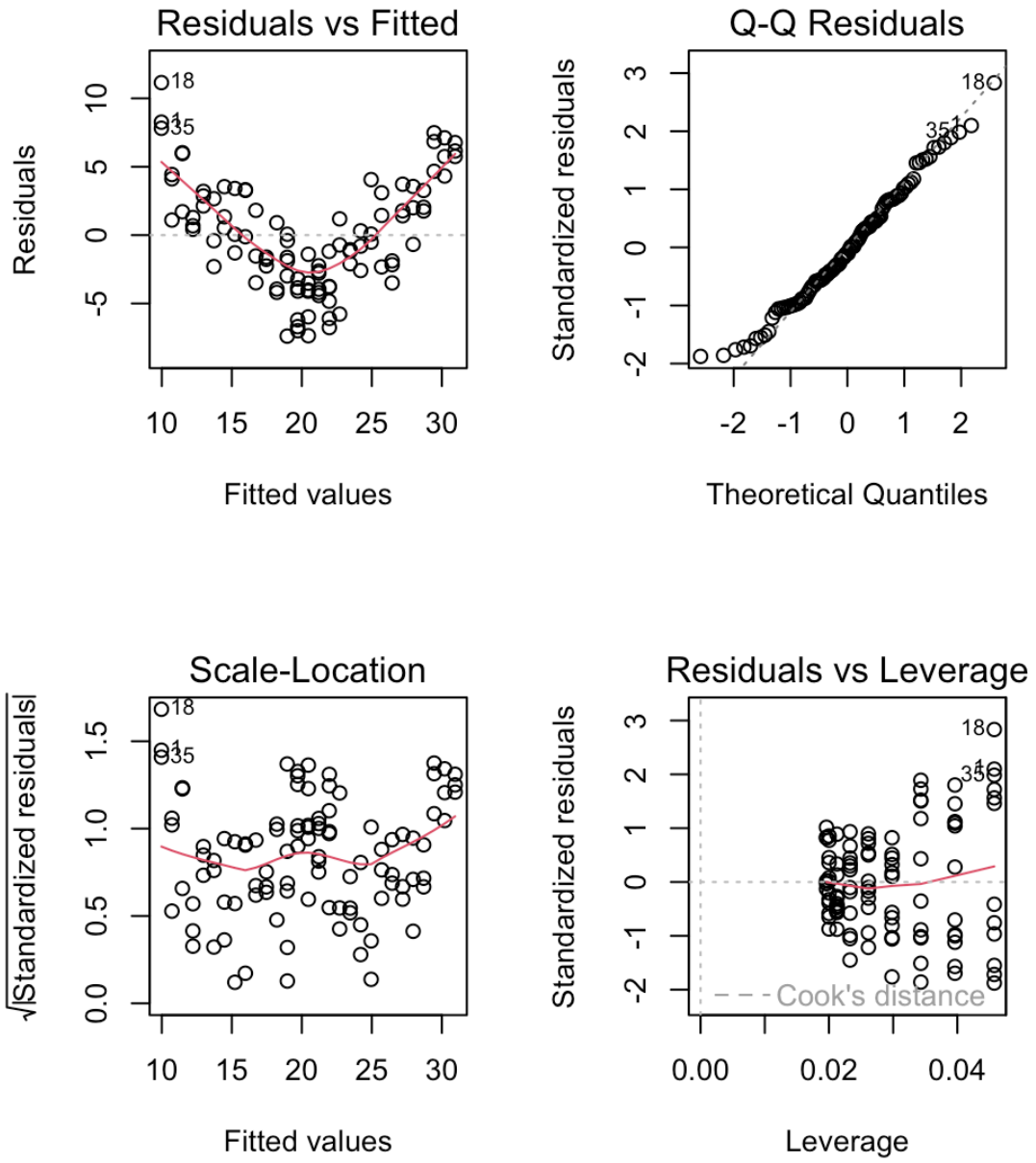

Figura 11.4: Grafici diagnostici per `lm.additive`. *Il grafico* `residuals vs fitted` *suggerisce una relazione non-lineare tra i residui ed i valori stimati.*

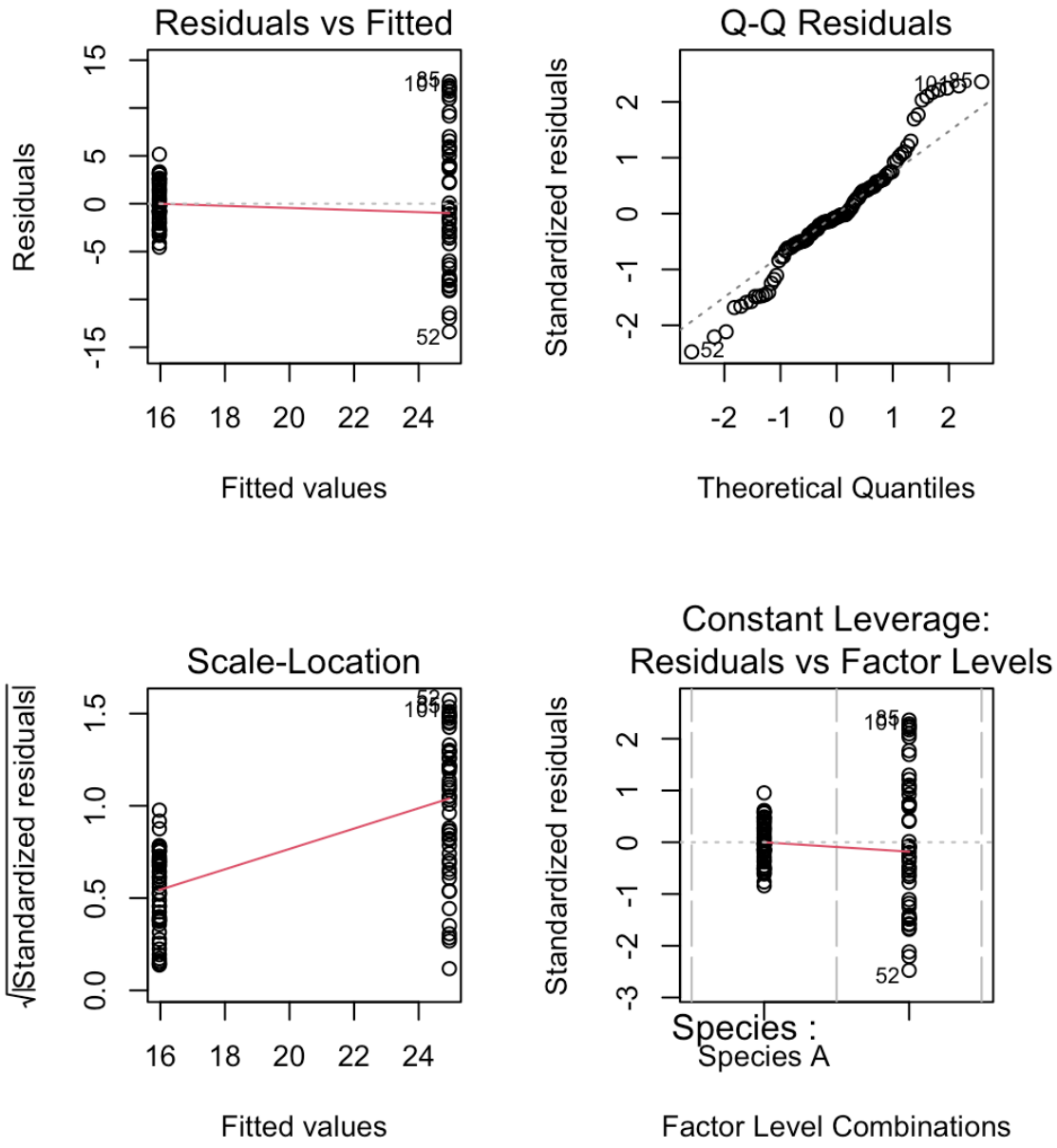

Figura 11.5: I grafici diagnostici per `lm.only.species` *suggeriscono non-normalità ed eteroscedasticità dei residui.*

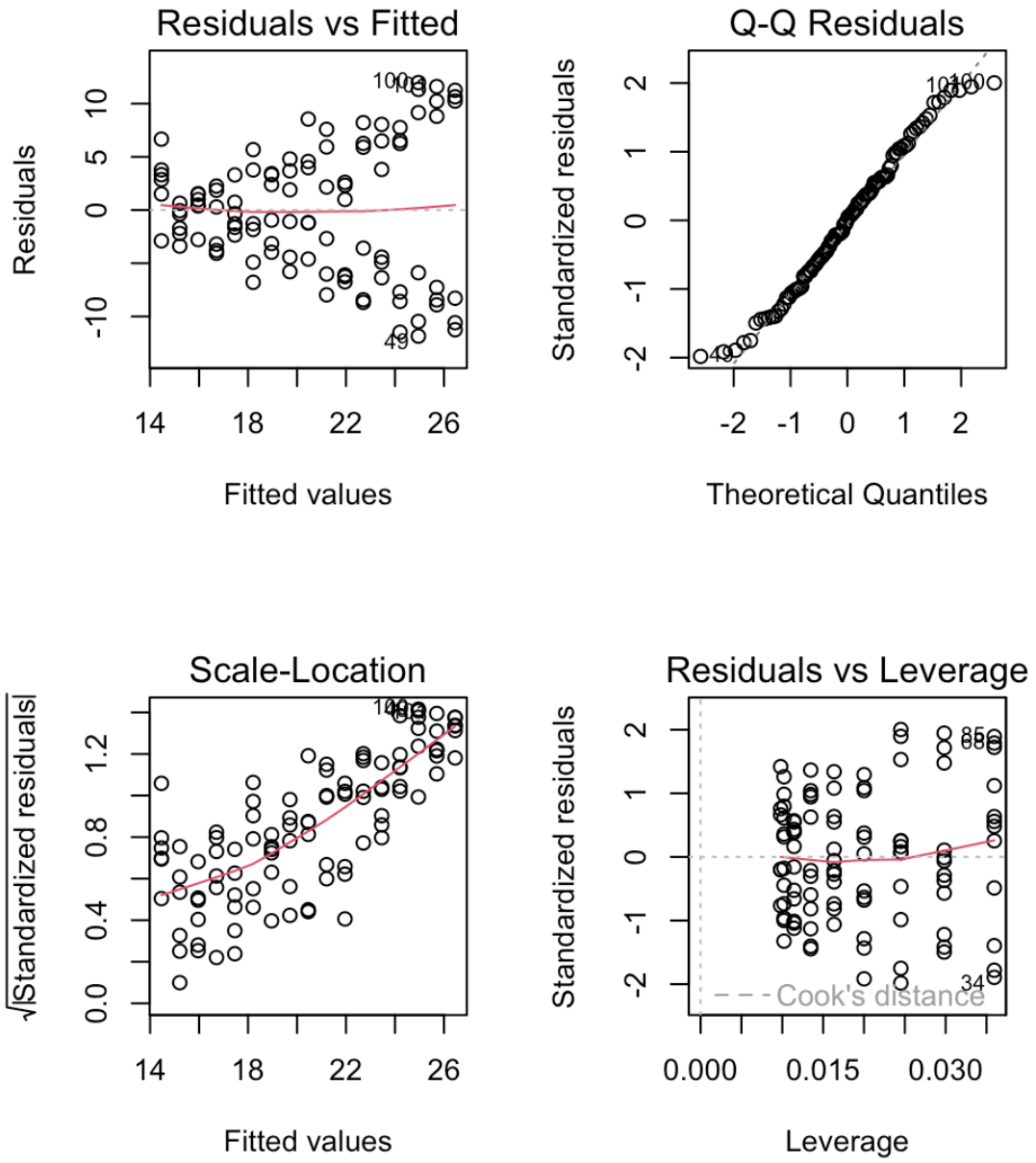

Figura 11.6: Grafici diagnostici per `lm.only.depth`. *Il grafico* `Scale − Location` *indica eteroscedasticità dei residui.*

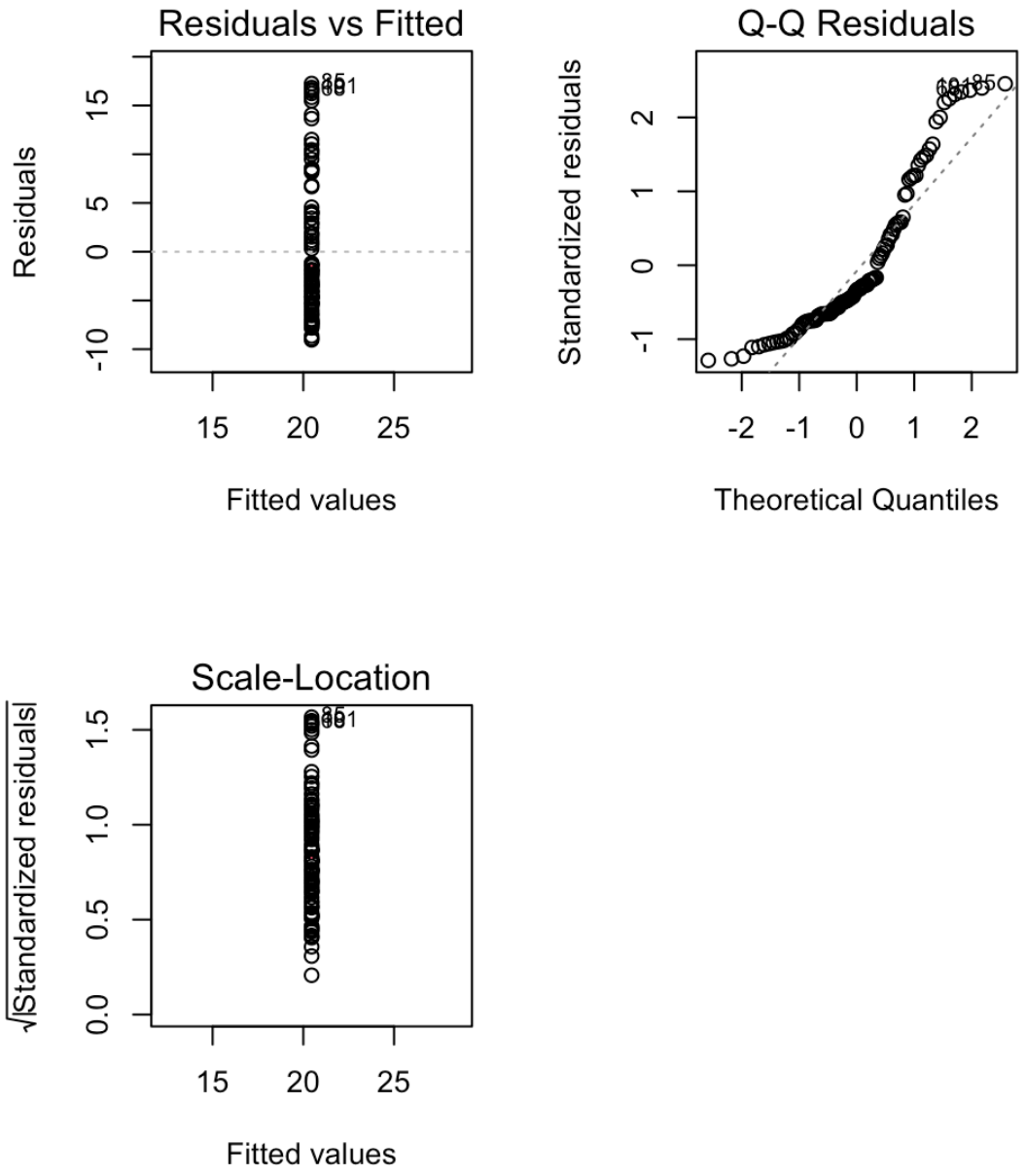

Figura 11.7: Grafici diagnostici per lm.null*. Il Q-Q plot indica che i residui non seguono una distribuzione di frequenza normale.*

12 Il caso di due (o più) predittori categorici: lo scenario "ANOVA a due vie"

Vediamo ora il caso di un modello lineare con due predittori di tipo categorico. Si tratta di una situazione tradizionalmente analizzata con una ANOVA a due vie ma, come vedremo, saremo in grado di utilizzare il protocollo già usato per analizzare modelli lineari con un predittore continuo ed uno categorico (sez. 11) con minime variazioni.

Supponiamo di voler studiare se due specie, *Specie A* e *Specie B*, rispondano diversamente a due trattamenti sperimentali: *dieta 1* e *dieta 2* (dieta naturale e dieta con integratori, ad esempio). Per farlo svolgiamo il seguente esperimento: assegniamo un gruppo di individui di ciascuna specie a ciascuno dei trattamenti sperimentali e a fine esperimento misuriamo la nostra variabile di risposta (massa corporea in chilogrammi nel nostro caso).

12.1 Gli scenari possibili

Vediamo ora, come già fatto nel capitolo precedente, alcuni dei possibili risultati di questo esperimento. In generale, anche al di là di una situazione di apprendimento, è buona prassi sforzarsi di illustrare i possibili risultati di uno studio *ancor prima* di svolgerlo: aiuta a definire i nostri interrogativi e le nostre ipotesi, aiuta a comprendere meglio il sistema oggetto dello studio, ed aiuta a individuare e risolvere i punti deboli di uno studio e/o di un esperimento prima ancora di metterli in atto.

Nel caso del nostro esperimento immaginario, l'ipotesi nulla è che né l'identità di specie né il tipo di dieta abbiano un effetto significativo sulla massa corporea degli individui delle specie in esame (fig. 12.1a). Alternativamente la massa corporea potrebbe in variare solo in funzione dell'identità di specie (fig. 12.1b), solo in funzione della dieta (fig. 12.1c), o in funzione di entrambi i fattori in maniera additiva (fig. 12.1d). Infine ci potrebbe essere un effetto interattivo dell'identità di specie e del tipo di dieta sulla massa corporea che si somma ai possibili effetti dei due fattori presi singolarmente (due possibili scenari: fig. 12.1e-f).

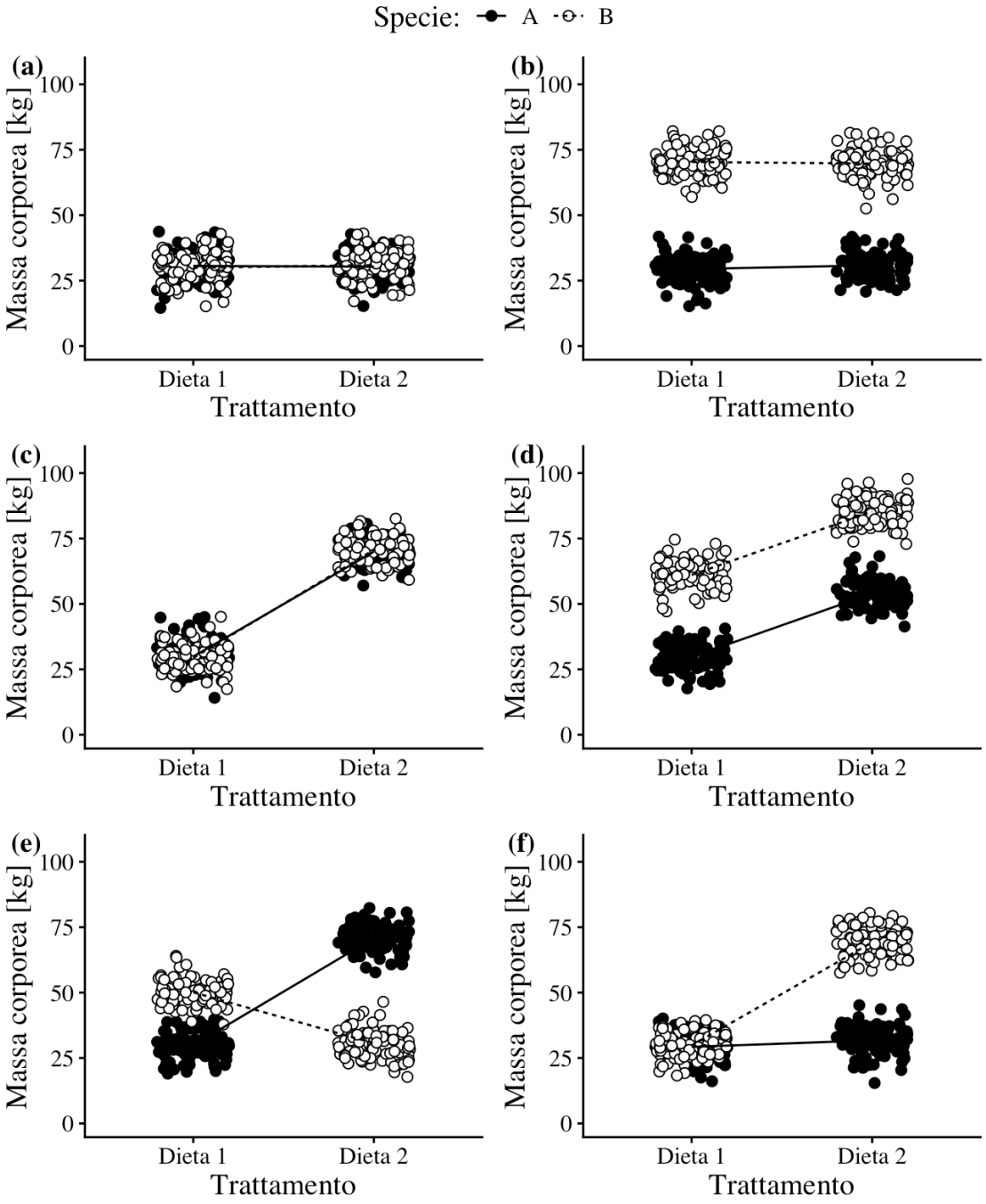

Figura 12.1: Alcune delle possibili risposte della variabile di risposta a diverse combinazioni di livelli dei predittori categorici. Ciascun punto rappresenta una singola osservazione.

12.2 Analisi

Ora che abbiamo chiaro in mente non solo il nostro interrogativo di ricerca, ma anche le possibili risposte, possiamo procedere ad analizzare i dati. Cominciamo come sempre impostando il nostro script e caricando i pacchetti che ci serviranno:

```
rm(list = ls())  # ripuliamo la memoria di R

# carichiamo 'pacman' per poter usare p_load():
if (!require(pacman)) install.packages("pacman")
library(pacman)

# carichiamo i pacchetti che ci serviranno:
p_load(ggplot2)
p_load(ggpubr)
p_load(emmeans)
```

Carichiamo i dati in R (li trovate a https://github.com/marcoplebani85/datasets/):

```
setwd("~/Desktop/Corso_R")
df <- read.csv("my_data/2way_ANOVA_data.csv")
str(df)

## 'data.frame':    400 obs. of  3 variables:
##  $ Species  : chr  "A" "A" "A" "A" ...
##  $ Treatment: chr  "diet1" "diet1" "diet1" "diet1" ...
##  $ body.mass: num  72.1 73.3 66.4 74 70.4 ...
```

R "vede" sia `Species` sia `Treatment` come stringhe di caratteri ("`chr`" sta per *character*). Indichiamo a R che si tratta di predittori categorici, ovvero fattori:

```
df$Species <- as.factor(df$Species)
df$Treatment <- as.factor(df$Treatment)
```

Il passo successivo è visualizzare di dati. Per farlo ho scelto di usare la funzione `ggline()` del pacchetto `ggpubr`, uno dei tanti pacchetti basati su `ggplot2`. La funzione `ggline()` permette di visualizzare i dati come una nuvola di punti e di congiungere con dei segmenti colorati i valori medi di ciascun gruppo. Cominciamo col creare la "base" del grafico:

```
basegrafico <- ggline(df, # provenienza dei dati
    x = "Treatment", y = "body.mass", color = "black",
    add=c("jitter", "mean"),
    # indicazioni su come tracciare
    # le linee che collegano i valori medi:
    plot_type="l", linetype="Species",
    # forma, dimensione e colore dei punti:
    add.params = list(shape = 21, size=2, fill = "Species"))
```

L'argomento `add="jitter"` aggiunge un po' di rumore alle coordinate dell'asse x così che ci sia minore sovrapposizione tra i punti che rappresentano le misurazioni. L'argomento `add="mean"` serve a calcolare il valore medio delle osservazioni in ciascun gruppo sperimentale in modo da

poterli collegare con linee per visualizzare più agevolmente le differenze tra gruppi. Personalizziamo l'oggetto `basegrafico` in modo da produrre la figura finale:

```
basegrafico +
  ylim(0,105) + # limiti dell'asse y
  # etichettiamo gli assi:
    labs(y = "Massa corporea [kg]", x="Trattamento") +
    theme_classic() +
  # per definire manualmente i colori da usare:
    scale_fill_manual("Specie:", values = c("grey45", "white")) +
  scale_linetype_discrete(name = "Specie:") +
  scale_x_discrete(labels=c("Dieta 1", "Dieta 2")) +
  # tipo e dimensione dei caratteri:
    theme(text = element_text(family = "Times")) +
    theme(text = element_text(size = 14))
```

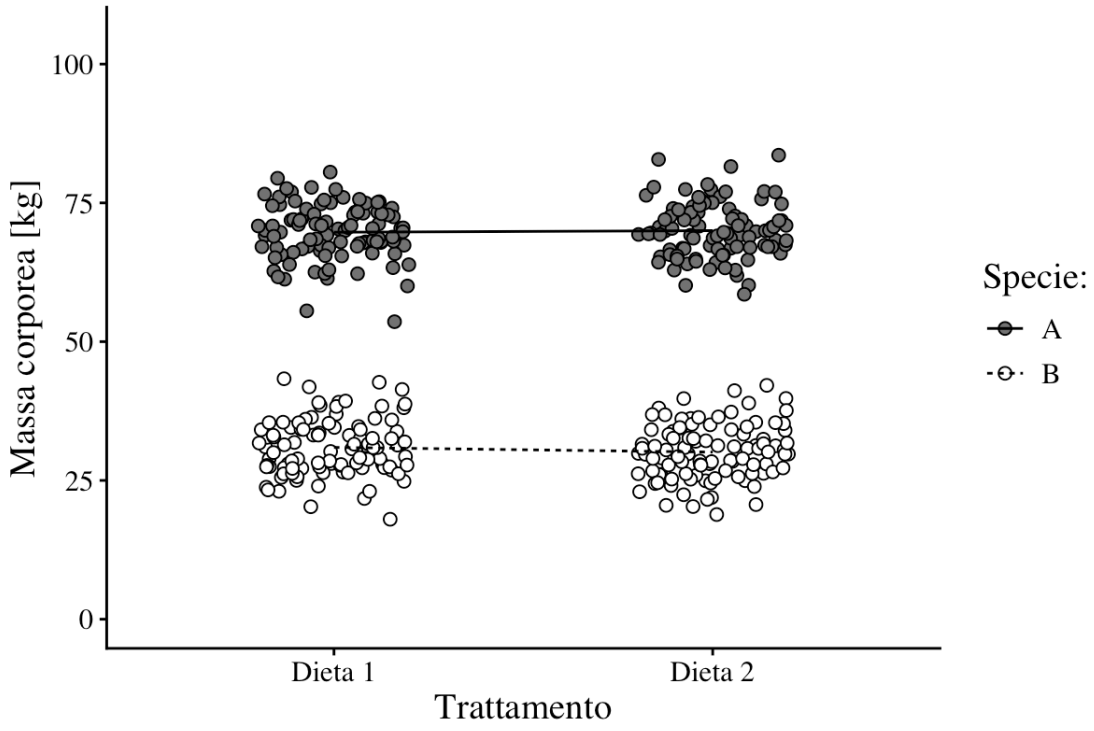

Figura 12.2: Masse corporee di individui appartenenti alle specie A e B esposti a due diete sperimentali. Ciascun punto rappresenta una singola osservazione.

Si direbbe che gli individui delle due specie abbiano masse corporee differenti ma non rispondano diversamente alle due diete sperimentali (fig. 12.2). Ci possiamo aspettare che il modello lineare che meglio descrive i dati sia quello in cui il solo predittore significativo della massa corporea sia l'identità di specie:

```
lm.only.species <- lm(body.mass ~ Species, data = df)
```

Verifichiamo questa ipotesi operando una selezione tra `lm.only.species` e i seguenti modelli lineari:

```
lm.interactive <- lm(body.mass ~ Species * Treatment, data = df)
lm.additive <- lm(body.mass ~ Species + Treatment, data = df)
lm.only.Treatment <- lm(body.mass ~ Treatment, data = df)
lm.null <- lm(body.mass ~ 1, data = df)
```

Per ciascun modello dobbiamo assicurarci che le assunzioni siano rispettate. I grafici diagnostici per ciascun modello sono riportati in appendice al capitolo. I grafici diagnostici per i modelli `lm.interactive`, `lm.additive`, e `lm.only.species` non indicano violazioni delle assunzioni (fig. 12.3-12.5), mentre i residui dei modelli `lm.only.Treatment` e `lm.null` hanno distribuzioni non-normali (fig. 12.6, 12.7).

Cominciamo col confrontare il modello "completo" (con interazione) e quello additivo:

```
anova(lm.interactive, lm.additive)
```

```
## Analysis of Variance Table
##
## Model 1: body.mass ~ Species * Treatment
## Model 2: body.mass ~ Species + Treatment
##   Res.Df    RSS Df Sum of Sq      F Pr(>F)
## 1    396 9771.1
## 2    397 9803.9 -1   -32.788 1.3288 0.2497
```

I modelli non differiscono significativamente. Dal momento che l'inclusione dell'effetto dell'interazione non migliora significativamente il potere esplicativo del modello consideriamo `lm.additive` il migliore tra i due.

Il modello additivo è migliore del modello con il solo trattamento (dieta) come predittore nel descrivere i dati, infatti la sua variabilità residua ($RSS/Res.Df$) è significativamente minore:

```
anova(lm.additive, lm.only.Treatment)
```

```
## Analysis of Variance Table
##
## Model 1: body.mass ~ Species + Treatment
## Model 2: body.mass ~ Treatment
##   Res.Df    RSS Df Sum of Sq      F    Pr(>F)
## 1    397   9804
## 2    398 164536 -1   -154732 6265.7 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Questo rafforza l'osservazione tratta da fig. 12.1 per cui le due specie A e B non hanno risposto al trattamento sperimentale. Continuiamo il processo di selezione confrontando `lm.additive` e `lm.only.species`:

```
anova(lm.additive, lm.only.species)
```

```
## Analysis of Variance Table
##
## Model 1: body.mass ~ Species + Treatment
## Model 2: body.mass ~ Species
```

```
##   Res.Df    RSS Df Sum of Sq      F Pr(>F)
## 1    397 9803.9
## 2    398 9811.4 -1   -7.5211 0.3046 0.5813
```

I modelli non differiscono significativamente. Dal momento che l'inclusione dell'effetto del trattamento "dieta" non migliora significativamente il potere esplicativo del modello possiamo considerare il modello `lm.only.species`, con la sola identità di specie come predittore, il migliore tra i due.

Quale è il modello migliore tra `lm.only.species` e `lm.only.Treatment`? Confrontiamoli entrambi con il modello nullo:

```
anova(lm.only.Treatment, lm.null)
```

```
## Analysis of Variance Table
##
## Model 1: body.mass ~ Treatment
## Model 2: body.mass ~ 1
##   Res.Df    RSS Df Sum of Sq      F Pr(>F)
## 1    398 164536
## 2    399 164544 -1   -7.5211 0.0182 0.8928
```

Non c'è differenza statisticamente significativa tra `lm.only.Treatment` e `lm.null`. Confrontiamo ora `lm.only.species` e `lm.null`:

```
anova(lm.only.species, lm.null)
```

```
## Analysis of Variance Table
##
## Model 1: body.mass ~ Species
## Model 2: body.mass ~ 1
##   Res.Df    RSS Df Sum of Sq      F    Pr(>F)
## 1    398   9811
## 2    399 164544 -1   -154732 6276.7 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Il modello che include l'identità di specie come predittore è significativamente migliore del modello nullo nel descrivere la variabilità osservata. Il confronto dei modelli tramite AIC conferma il risultato dei test F sui residui a coppie:

```
AIC(lm.interactive, lm.additive, lm.only.species, lm.only.Treatment, lm.null)
```

```
##                   df      AIC
## lm.interactive     5 2423.439
## lm.additive        4 2422.779
## lm.only.species    3 2421.086
## lm.only.Treatment  3 3548.920
## lm.null            2 3546.938
```

I modelli `lm.only.Treatment` e `lm.null` sono nettamente peggiori degli altri tre, con un ΔAIC > 1000 (ben maggiore delle sei unità suggerite da Harrison et al. come valore-soglia). I modelli

interattivo, additivo, e con solo "specie" come predittore hanno valori di AIC molto simili, per cui ha senso considerare il modello con minore complessità (quello con il minor numero di parametri) come il migliore tra i tre: il "vincitore" tra i modelli a confronto è dunque `lm.only.species`. Tra i modelli a confronto, `lm.only.species` è quello che spiega la maggior porzione di variabilità osservata con il minor numero di parametri.

Estraiamo la frazione di variabilità dei dati spiegata dal modello `lm.only.species`:

```
summary(lm.only.species)$adj.r.squared
```

```
## [1] 0.9402221
```

Il modello descrive il 94% della variabilità osservata. In questo caso ho usato il segno `$` per estrarre da `summary(lm.only.species)` solo il frammento di informazione relativo all'R^2. Per esplorare la struttura di tutte le informazioni contenute in `summary(lm.only.species)` potete usare la funzione `str()`: `str(summary(lm.only.species))`.

Usiamo ora `emmeans` per estrarre i parametri del modello:

```
emmeans(lm.only.species, # modello di interesse
        pairwise ~ Species, # predittori per cui fare stime
        adjust = "bonferroni" # correzione per test multipli
        )
```

```
## $emmeans
##  Species emmean    SE  df lower.CL upper.CL
##  A         69.9 0.351 398     69.2     70.6
##  B         30.5 0.351 398     29.8     31.2
##
## Confidence level used: 0.95
##
## $contrasts
##  contrast estimate    SE  df t.ratio p.value
##  A - B        39.3 0.497 398  79.226 <0.0001
```

12.3 Comunicazione

Possiamo comunicare i dettagli di questo studio come segue:

Ci siamo posti l'obiettivo di studiare l'effetto di due tipi di dieta, Dieta 1 e Dieta 2, sulla massa corporea di individui appartenenti alle specie Specie A e Specie B.

Metodi. Abbiamo ottenuto misure di massa corporea per 100 individui per ciascuna combinazione dei livelli di identità di specie e dieta sperimentale per un totale di 400 misurazioni. Abbiamo confrontato cinque modelli lineari alternativi: un modello con un effetto interattivo del tipo di dieta e dell'identità di specie sulla massa corporea; un modello di tipo additivo; i modelli con solo dieta o identità di specie come predittori di massa corporea; il modello nullo. Abbiamo svolto i confronti tra modelli con dei test F sulle loro variabilità residue, nonchè in base al loro indice di Akaike (AIC). Abbiamo svolto tutte le analisi con R 4.5.2 (R Core Team, 2025).

Abbiamo definito i modelli lineari con la funzione `lm()` e li abbiamo confrontati con le funzioni `anova()` ed `AIC()`.

Risultati. Il modello con la sola identità di specie come predittore è risultato il migliore tra quelli a confronto. I trattamenti sperimentali (Dieta 1 e Dieta 2) non hanno prodotto differenze significative nelle masse corporee di nessuna delle due specie, mentre l'identità di specie spiega il 94% della variabilità osservata nella massa corporea degli individui appartenenti al campione di studio. Gli individui di Specie A e Specie B hanno masse corporee medie rispettivamente di 69.86 Kg (SE=0.35) e 30.52 Kg (SE=0.35), a prescindere dal tipo di dieta sperimentale a cui sono stati sottoposti. Le masse corporee individuali differiscono significativamente tra le due specie ($df=398$, $t=79.226$, $p < 0.0001$).

12.4 Appendice: grafici diagnostici

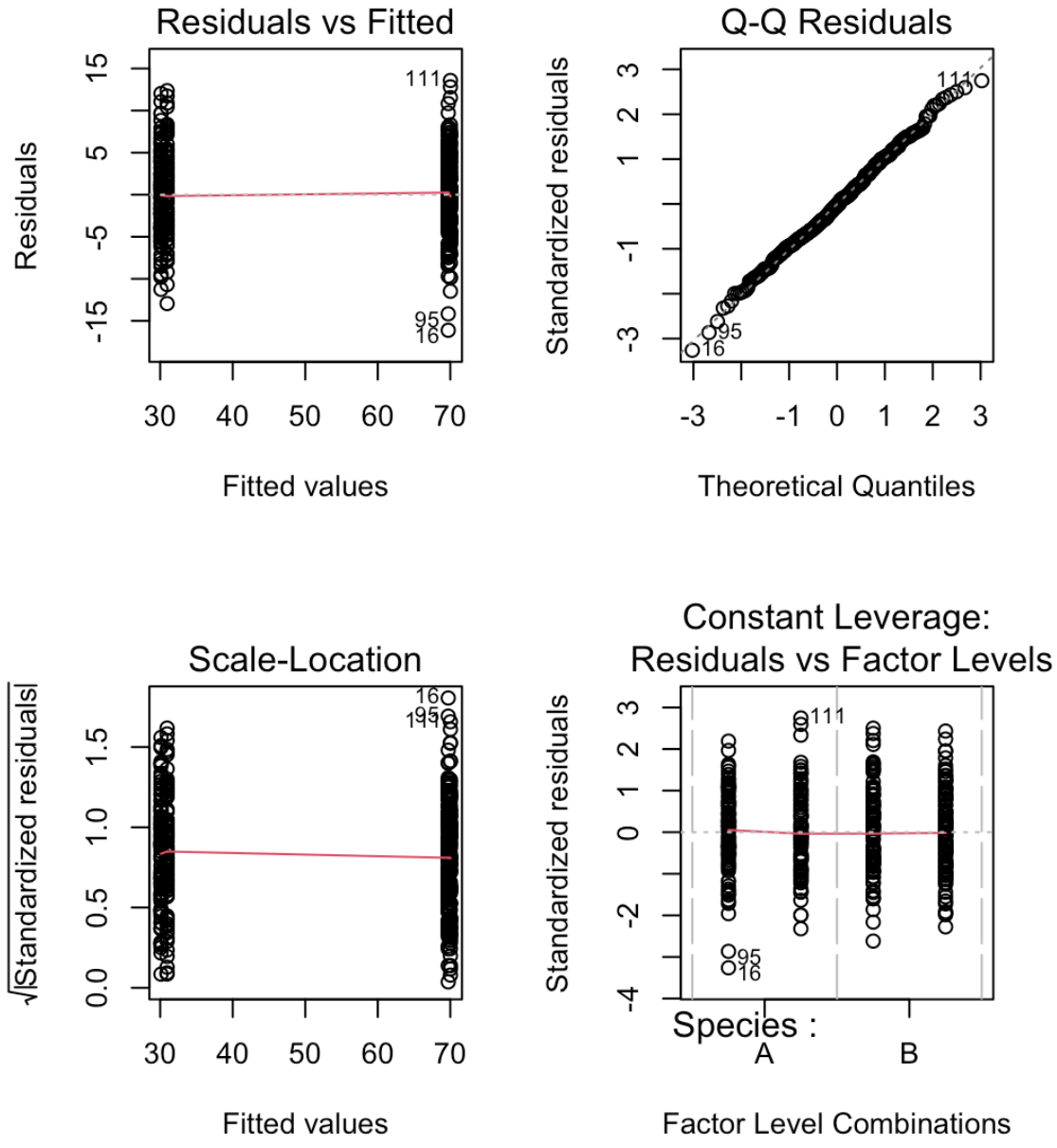

Figura 12.3: Grafici diagnostici per `lm.interactive`.

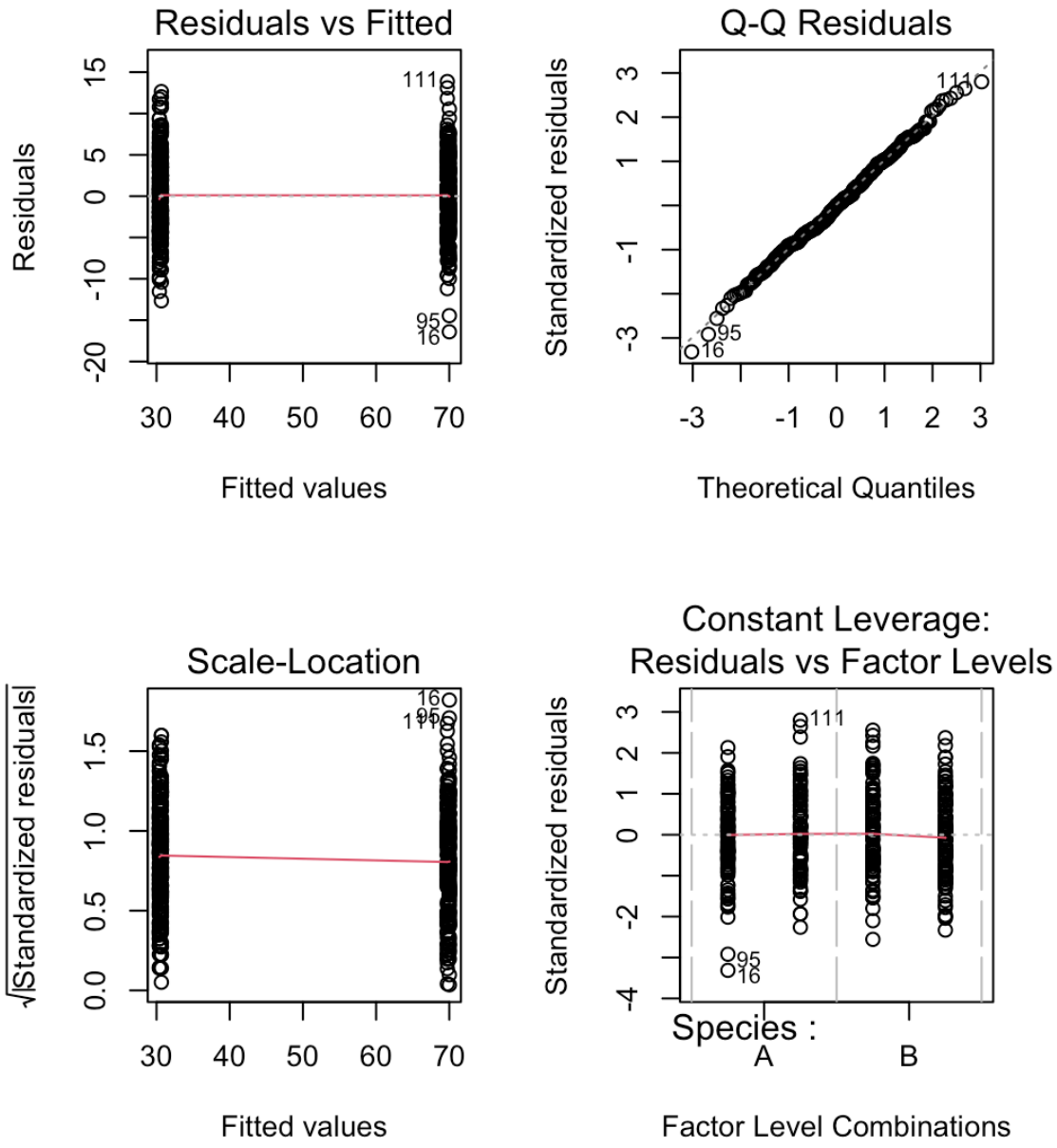

Figura 12.4: Grafici diagnostici per `lm.additive`.

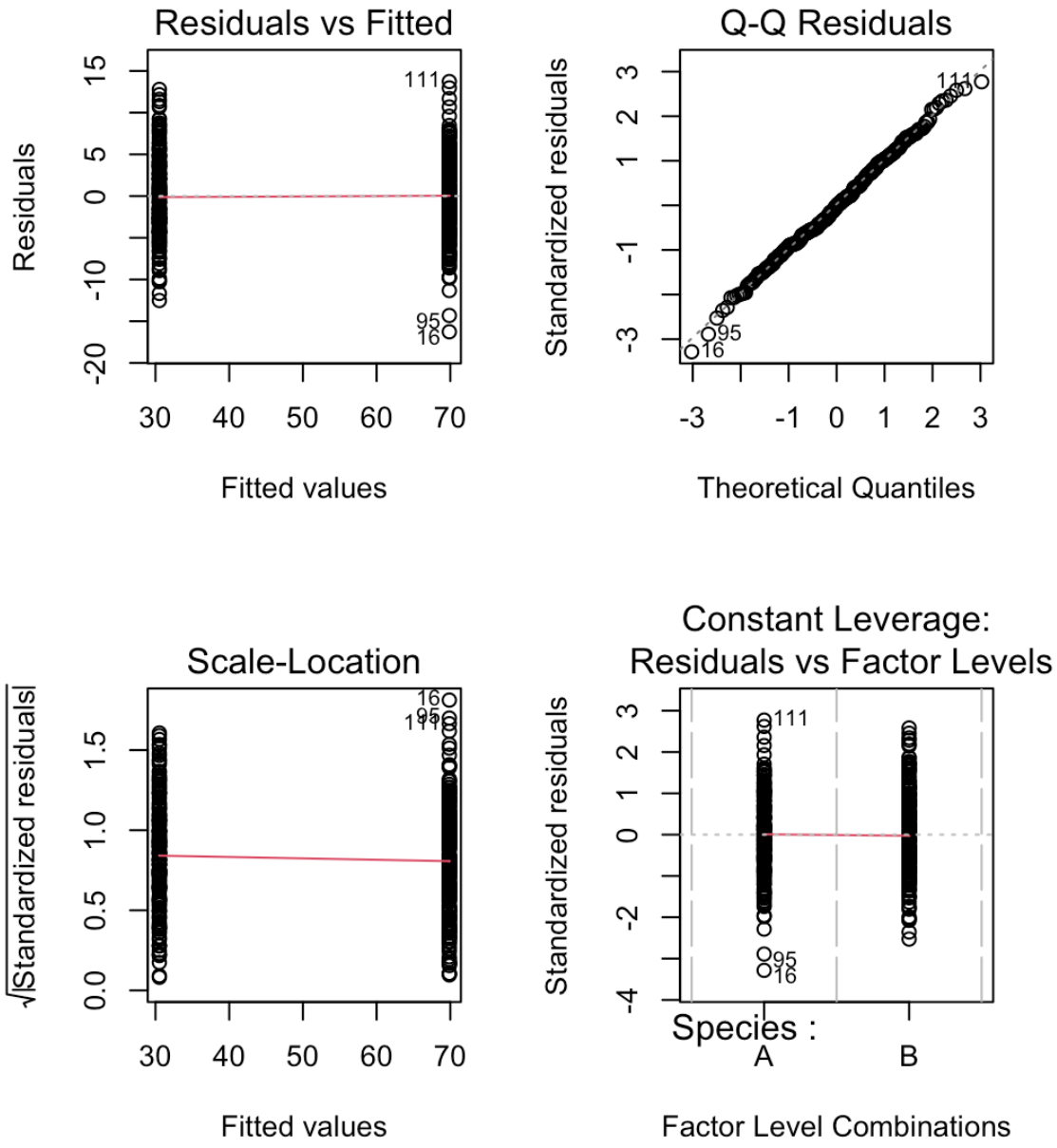

Figura 12.5: Grafici diagnostici per `lm.only.species`.

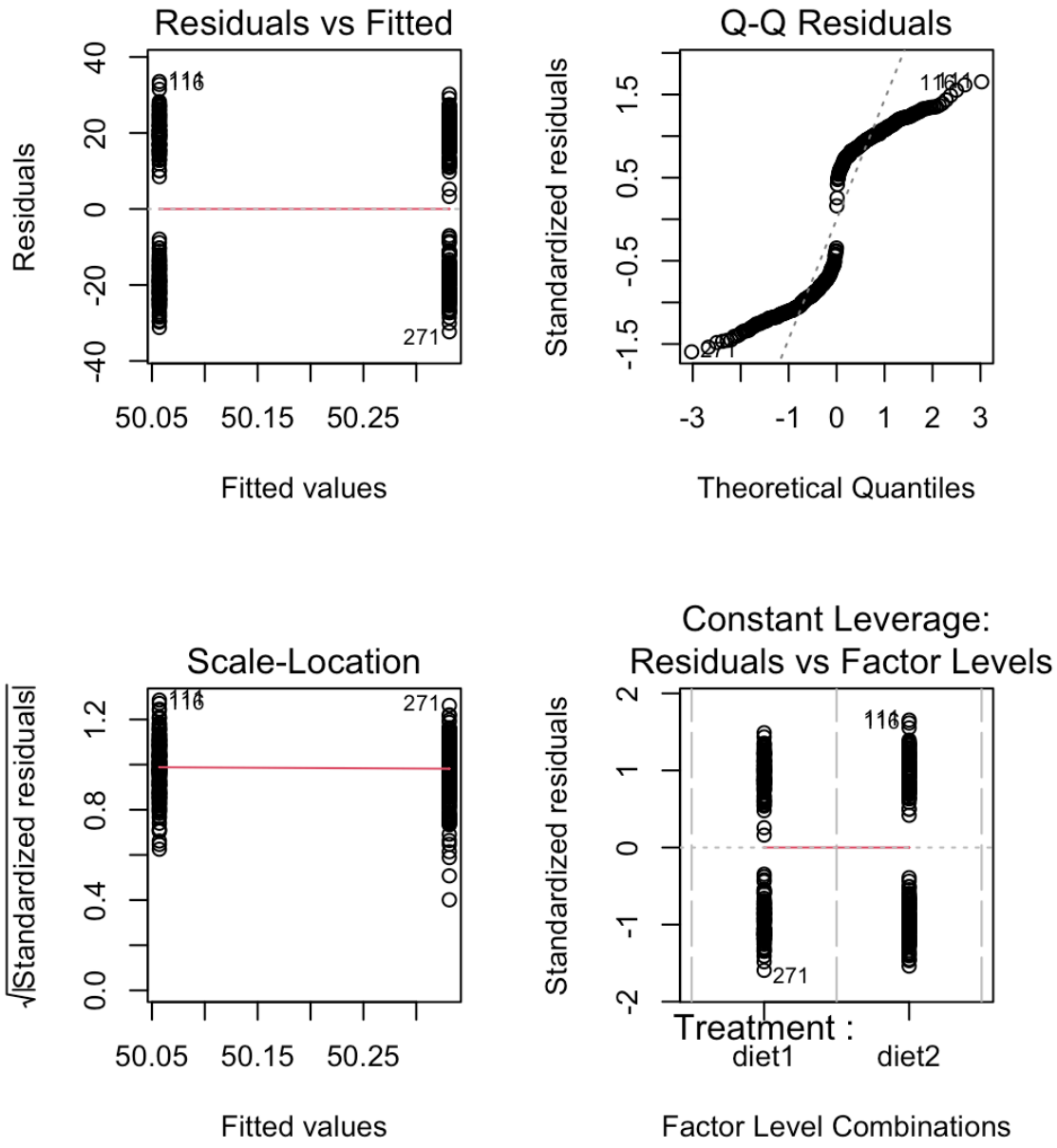

Figura 12.6: Grafici diagnostici per `lm.only.Treatment`.

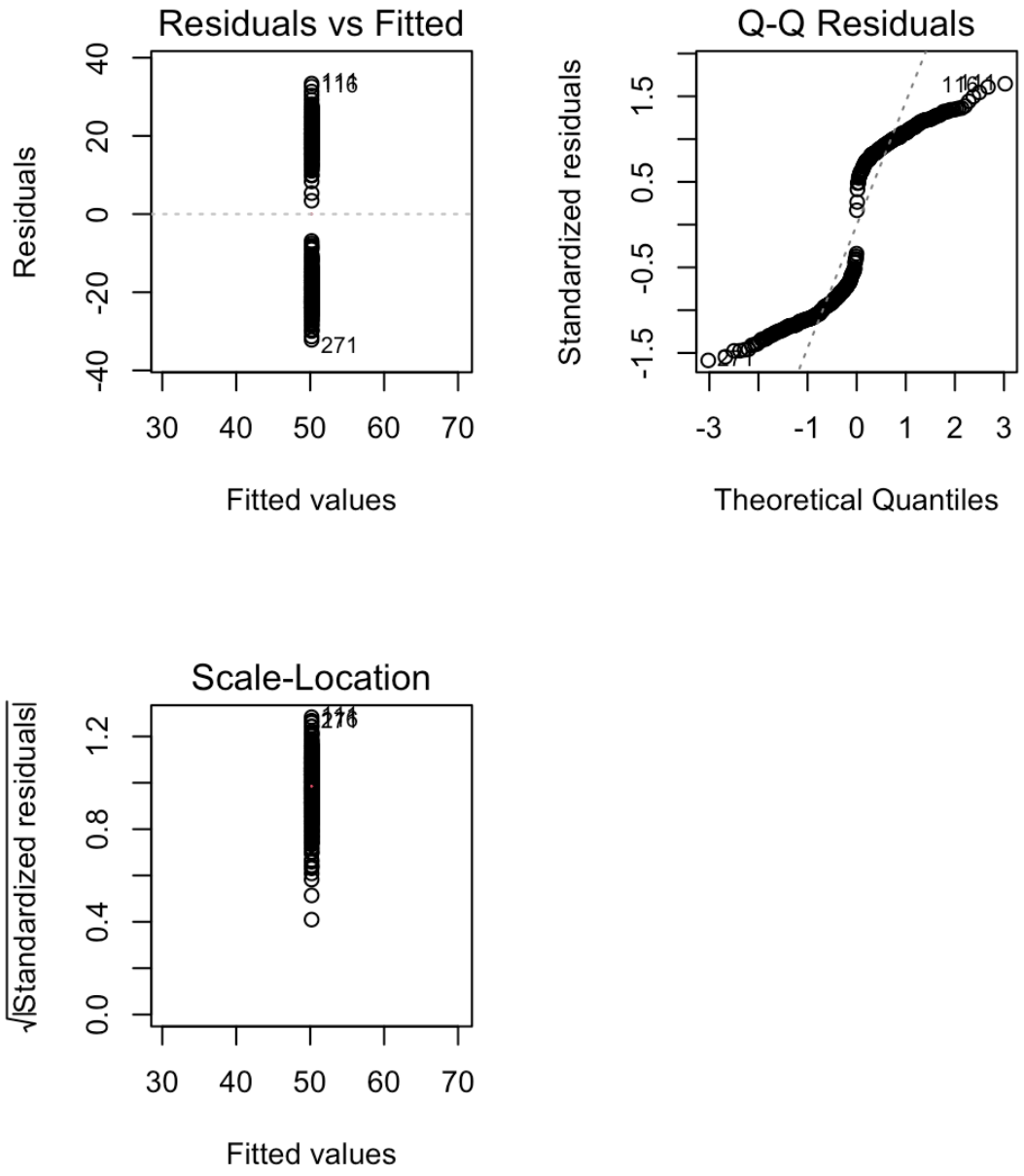

Figura 12.7: Grafici diagnostici per `lm.null`.

13 Il caso di due predittori continui: regressione lineare multipla

Lo studio della correlazione tra una variabile di interesse e due o più predittori continui è noto come regressione lineare multipla. Esamineremo il caso della regressione lineare multipla con due predittori continui, che può essere visualizzato in uno spazio tridimensionale (fig. 13.1, 13.2, 13.4). L'approccio che stiamo per esaminare si applica anche a modelli di regressione lineare multipla con più di due predittori.

13.1 Scenari possibili

Come nel caso degli scenari "ANCOVA" e "ANOVA a due vie", nel caso della regressione multipla con due predittori di interesse ci sono cinque possibili modelli lineari da confrontare: il modello nullo (fig. 13.1a), i modelli in cui solo uno dei due predittori è correlato con la variabile di risposta (fig. 13.1b), il modello additivo (fig. 13.1c), e quello con interazione (fig. 13.1d).

Immaginiamo di avere una variabile di risposta y e due possibili predittori di interesse, x e z. Nel caso del modello nullo H_0, rappresentato in fig. 13.1a, il modello lineare che descrive la relazione tra la variabile di risposta y ed i predittori x e z è il seguente:

$$y_i = a + \epsilon_i \qquad (13.1)$$

La costante a è stimata dalla media di tutte le osservazioni y_i.

Nel caso in cui y vari in relazione di solo un predittore, ad esempio x, la loro relazione è illustrata in figura 13.1b e descritta dal modello lineare di una retta di regressione:

$$y_i = a + b \cdot x_i + \epsilon_i \qquad (13.2)$$

Nel caso in cui y vari linearmente al variare sia di x, sia di z, la loro relazione è illustrata in figura 13.1c e descritta dal modello:

$$y_i = a + b \cdot x_i + c \cdot z_i + \epsilon_i \qquad (13.3)$$

Infine, nel caso in cui ci sia un effetto interattivo di x e z su y, la relazione tra i predittori e la variabile di risposta è illustrata in figura 13.1d e descritta dal modello:

$$y_i = a + b \cdot x_i + c \cdot z_i + d \cdot x_i \cdot z_i + \epsilon_i \qquad (13.4)$$

Come sempre, l'obiettivo della nostra analisi sarà quello di individuare quale di questi modelli descriva al meglio i nostri dati e di stimarne i parametri. L'interpretazione dei parametri del modello di regressione lineare multipla è simile a quanto visto nel caso della regressione lineare semplice. La differenza è che in questo caso, avendo due predittori invece di uno, la loro correlazione con la variabile di risposta non è rappresentata da una retta di regressione ma da una "superficie di regressione" (fig. 13.1). Nel caso di modelli senza interazione, la correlazione tra le tre variabili è rappresentata da un piano più o meno inclinato (fig. 13.1a-c). In presenza di un effetto interattivo tra i predittori la "superficie di regressione" è tanto più distorta quanto maggiore è la magnitudine dell'interazione (fig. 13.1d).

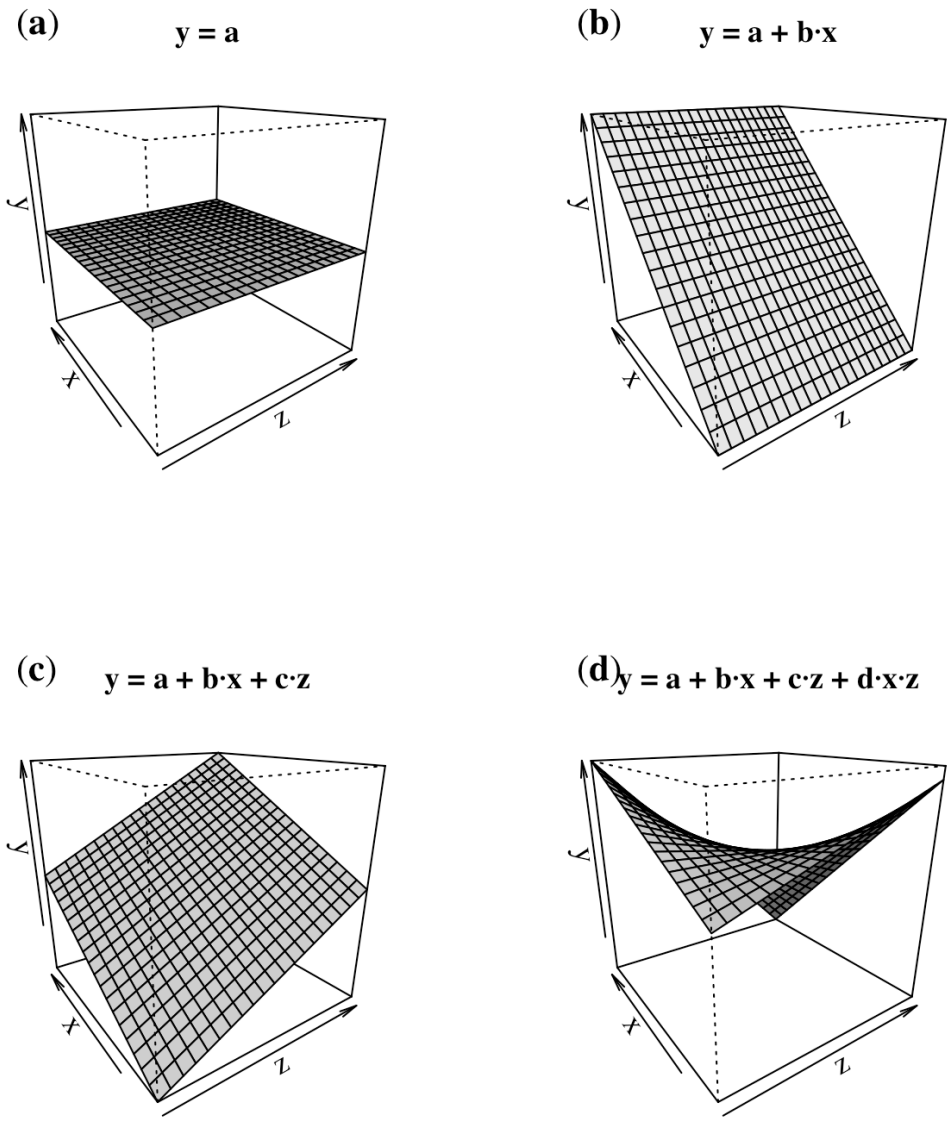

Figura 13.1: Alcuni dei possibili scenari di correlazione tra la variabile di risposta y e due predittori continui x e z.

13.2 Esempio di regressione lineare multipla

Supponiamo di voler studiare come varia la taglia di un'alga (`alga.length`) al variare della profondità a cui cresce (`depth`) e della disponibilità di nutrienti (`nutrients`). Per fare ciò è necessario ottenere misure della taglia algale in corrispondenza di varie combinazioni di valori di profondità e di disponibilità di nutrienti. Il dataset contenuto nel file `multi_reg_data.csv` rappresenta un possibile risultato di questo studio ipotetico.

Prepariamo la sessione di lavoro per analizzare i dati:

```
rm(list = ls())  # ripulisce la memoria di R

# carica il pacchetto 'pacman', che contiene la funzione p_load():
if (!require(pacman)) install.packages("pacman")
library(pacman)
p_load(emmeans)
p_load(scatterplot3d)  # per fare grafici a dispersione tridimensionali
```

Definiamo la directory di lavoro e carichiamo i dati in R (li trovate a https://github.com/marcoplebani85/datasets/):

```
setwd("~/Desktop/Corso_R")
# Carichiamo il file con i dati:
df <- read.csv("my_data/multi_reg_data.csv")
str(df)

## 'data.frame':    170 obs. of  3 variables:
##  $ alga.length: num  125 134 150 182 110 ...
##  $ depth      : int  2 3 4 5 6 7 8 9 10 11 ...
##  $ nutrients  : num  280 385 581 910 218 ...
```

Visualizziamo i dati usando la funzione `scatterplot3d()` contenuta nel pacchetto omonimo (Ligges e Mächler, 2003):

```
par(family="Times")
grafico3d <- scatterplot3d(x = df$depth,
                           y = df$nutrients,
                           z = df$alga.length, # assi
                  # etichette assi:
                  xlab="Profondità [m]",
                  ylab="Nutrienti [mg/L]",
                  zlab="Lunghezza algale [cm]",
                  pch=16,
                  # per evidenziare tridimensionalità:
                  highlight.3d=T,
                  # limiti degli assi:
                  xlim=c(0, max(df$depth)),
                  ylim=c(0, max(df$nutrients)),
                  zlim=c(50, max(df$alga.length)),
                  angle=40, # angolo di rotazione
                  type="p", # punti. Alternative: "l","h"
```

```
                  main="" # titolo (vuoto)
                  )
```

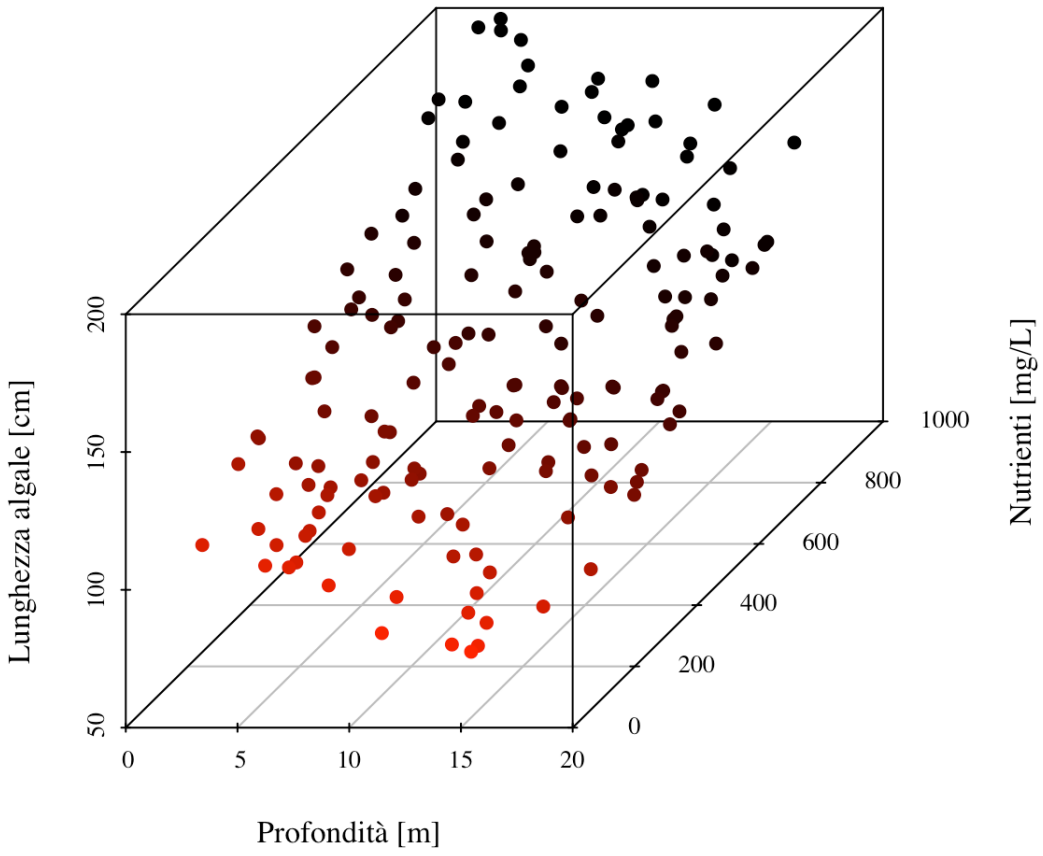

Figura 13.2: Grafico a dispersione tridimensionale.

La figura 13.2 suggerisce che la lunghezza algale abbia una correlazione lineare sia con la profondità, siaq con l'abbondanza di nutrienti. Un modello additivo sembrerebbe adeguato a descrivere i dati. Verifichiamolo comparando i modelli nullo, con singolo predittore, additivo, e con interazione:

```
lm.full <- lm(alga.length ~ depth * nutrients, data = df)
lm.additive <- lm(alga.length ~ depth + nutrients, data = df)
lm.depth <- lm(alga.length ~ depth, data = df)
lm.nutrients <- lm(alga.length ~ nutrients, data = df)
lm0 <- lm(alga.length ~ 1, data = df)
```

13.3 Controllo delle assunzioni

I grafici diagnostici indicano che i residui dei modelli `lm.full` e `lm.additive` non mostrano nulla di sospetto (fig. 13.5-13.6, in appendice a questa sezione), mentre quelli dei modelli `lm.nutrients` ed `lm.depth` non seguono una distribuzione normale (fig. 13.7-13.8, in appendice a questa sezione).

13.4 Collinearità

Nel caso in cui due o più predittori di un modello lineare siano continui è importante verificare che non ci sia collinearità tra di essi, ovvero che essi varino in maniera indipendente l'uno dall'altro. La collinearità tra predittori non è necessariamente un problema se il modello ha scopo predittivo; al contrario, se l'obiettivo del modello è di tipo esplicativo, la non-indipendenza dei predittori rende arduo discernere gli effetti dei predittori sulla variabile di risposta. Si può verificare la collinearità tra predittori continui con test quantitativi, ad esempio usando la funzione `cor.test()` per calcolare l'indice di correlazione di Spearman e se esso sia significativamente diverso da zero:

```
cor.test(df$nutrients, y = df$depth)

##
##  Pearson's product-moment correlation
##
## data:  df$nutrients and df$depth
## t = -0.16457, df = 168, p-value = 0.8695
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.1628989  0.1380822
## sample estimates:
##         cor
## -0.01269596
```

A questo si può aggiungere una valutazione visiva (fig. 13.3):

```
scatter.smooth(x=df$nutrients, y=df$depth,
    ylab="depth", xlab="nutrients",
    lpars = list(col = "red", lwd = 1, lty = 1)
    )
```

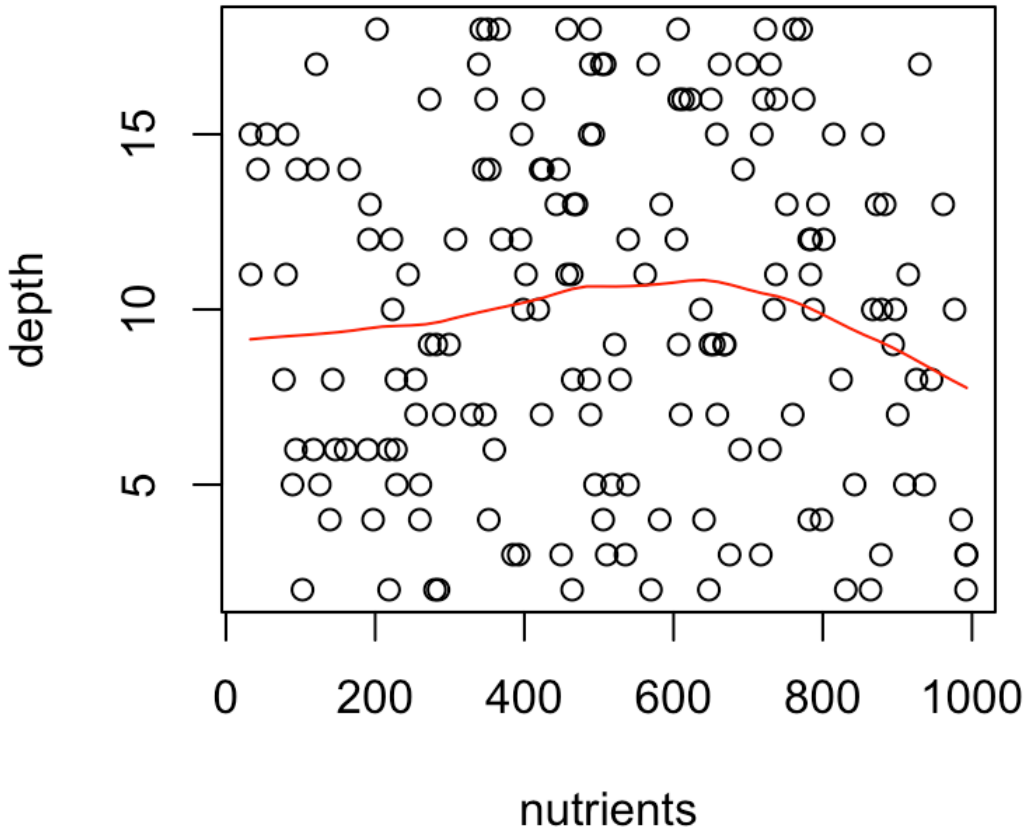

Figura 13.3: Grafico diagnostico per valutare visivamente se ci sia collinearità tra i predittori. La linea rossa è una curva che approssima l'andamento locale medio dei punti.

La funzione `scatter.smooth()` produce un grafico a dispersione, aggiungendovi una linea di "smoothing" che aiuta ad evidenziare eventuali relazioni (lineari o meno) tra le variabili. In questo caso non c'è una correlazione lineare significativa tra `nutrients` e `depth`.

13.5 Selezione del modello

Confrontiamo ora i modelli interattivo ed additivo con l'ormai familiare test F sulle loro variabilità residue:

```
anova(lm.full, lm.additive)

## Analysis of Variance Table
##
## Model 1: alga.length ~ depth * nutrients
```

```
## Model 2: alga.length ~ depth + nutrients
##   Res.Df    RSS Df Sum of Sq     F Pr(>F)
## 1    166 362.40
## 2    167 363.16 -1  -0.75977 0.348  0.556
```

I modelli `lm.full` e `lm.additive` non differiscono significativamente: l'aggiunta di un termine di interazione non riduce significativamente la variabilità residua del modello additivo. Il modello additivo risulta il migliore dei due perchè spiega una quantità di variabilità statisticamente non distinguibile da quella spiegata dal modello interattivo, e lo fa con un parametro in meno.

Abbiamo già appurato che i modelli `lm.depth` e `lm.nutrients` non rispettano le assunzioni dei modelli lineari (fig. 13.7-13.8, in appendice a questa sezione), per cui non sono buoni modelli per descrivere i dati osservati. Possiamo comunque confrontarli con il modello additivo, per eccesso di zelo:

```
anova(lm.additive, lm.depth)
```

```
## Analysis of Variance Table
##
## Model 1: alga.length ~ depth + nutrients
## Model 2: alga.length ~ depth
##   Res.Df    RSS Df Sum of Sq     F    Pr(>F)
## 1    167    363
## 2    168 117017 -1   -116654 53644 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
anova(lm.additive, lm.nutrients)
```

```
## Analysis of Variance Table
##
## Model 1: alga.length ~ depth + nutrients
## Model 2: alga.length ~ nutrients
##   Res.Df     RSS Df Sum of Sq      F    Pr(>F)
## 1    167   363.2
## 2    168 16929.8 -1    -16567 7618.2 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Il modello additivo `lm.additive` ha una quantità di variabilità residua significativemente minore di quanta non ne abbiano `lm.nutrients` e `lm.depth`. Il modello additivo resta il migliore tra quelli a confronto per descrivere i dati osservati.

Il confronto tra modelli tramite AIC giunge alla medesima conclusione:

```
AIC(lm.full,
    lm.additive,
    lm.depth,
    lm.nutrients,
    lm0
    )
```

```
##                df      AIC
## lm.full         5  621.120
## lm.additive     4  619.476
## lm.depth        3 1599.266
## lm.nutrients    3 1270.614
## lm0             2 1621.217
```

Il modello `lm.additive` è quello dei cinque candidati che meglio approssima i dati osservati con il minor numero di parametri.

Vediamo ora i parametri del modello:

```
summary(lm.additive)$coef
```

```
##                 Estimate   Std. Error   t value      Pr(>|t|)
## (Intercept) 100.0158534 0.3418577713 292.56569 2.865405e-228
## depth        -2.0152169 0.0230884493 -87.28247 2.952862e-141
## nutrients     0.1002383 0.0004327871 231.61103 2.290408e-211
```

Nel caso dei modelli di regressione multipla l' *output* di `summary()` è di immediata interpretazione, come nel caso dei modelli di regressione semplice. L'equazione della "superficie di regressione" secondo `lm.additive` è:

$$\widehat{lunghezza}.media = 100 - 2 \cdot profondita + 0.1 \cdot nutrienti \qquad (13.5)$$

Secondo il modello la lunghezza algale si riduce di due centimetri per ogni aumento di un metro di profondità, ed aumenta di 0.1 centimetri per ogni aumento della concentrazione di nutrienti pari ad un mg/L.

Possiamo aggiungere superfici piatte (corrispondenti a modelli senza interazioni) ai grafici prodotti da `scatterplot3d()` con il codice seguente (il risultato è mostrato in fig. 13.4):

```
grafico3d$plane3d(lm.additive, lty.box = "solid")
```

Nota: è necessario dare un nome allo scatterplot tridimensionale prodotto da `scatterplot3d()` per potervi aggiungere il piano di regressione. In questo caso l'ho chiamato `grafico3d`. Per aggiungere il piano a `grafico3d` è necessario usare l'inusuale sintassi `grafico3d$plane3d(...)`.

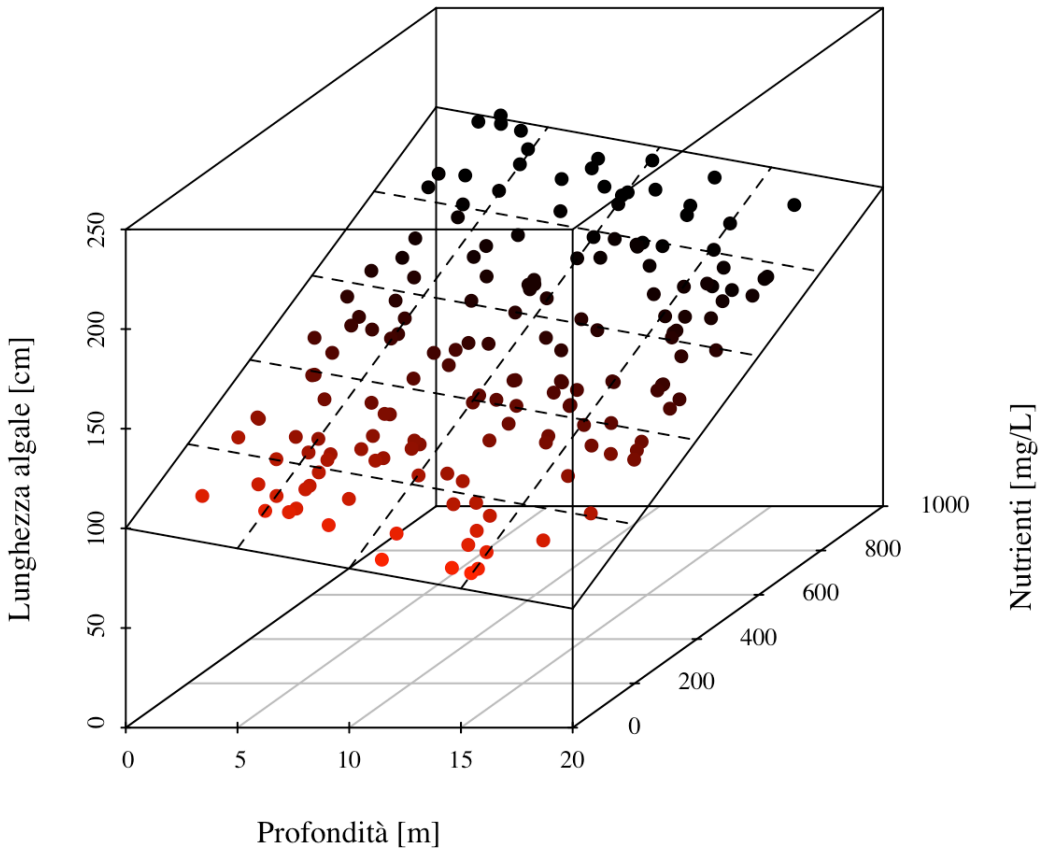

Figura 13.4: Grafico a dispersione tridimensionale dei dati e delle predizioni del modello additivo `lm.additive`.

13.6 Bonus: scatterplot 3D interattivo

Il codice seguente permette di creare un grafico a dispersione interattivo usando i pacchetti `car` (Fox e Weisberg, 2019) e `rgl` (Murdoch et al., 2023):

```
p_load(rgl)
p_load(car)
scatter3d(x = df$depth,
          y = df$nutrients,
          z = df$alga.length, # variabili
          groups=NULL,
          grid = F,
          fit = "linear"
          # oppure: "quadratic", "smooth", "additive"
          )
```

Provatelo! Il grafico può essere ruotato sui tre assi, permettendo di visualizzare i dati e le possibili interazioni tra variabili più facilmente rispetto ai grafici prodotti da `scatterplot3d()`.

13.7 Appendice: grafici diagnostici

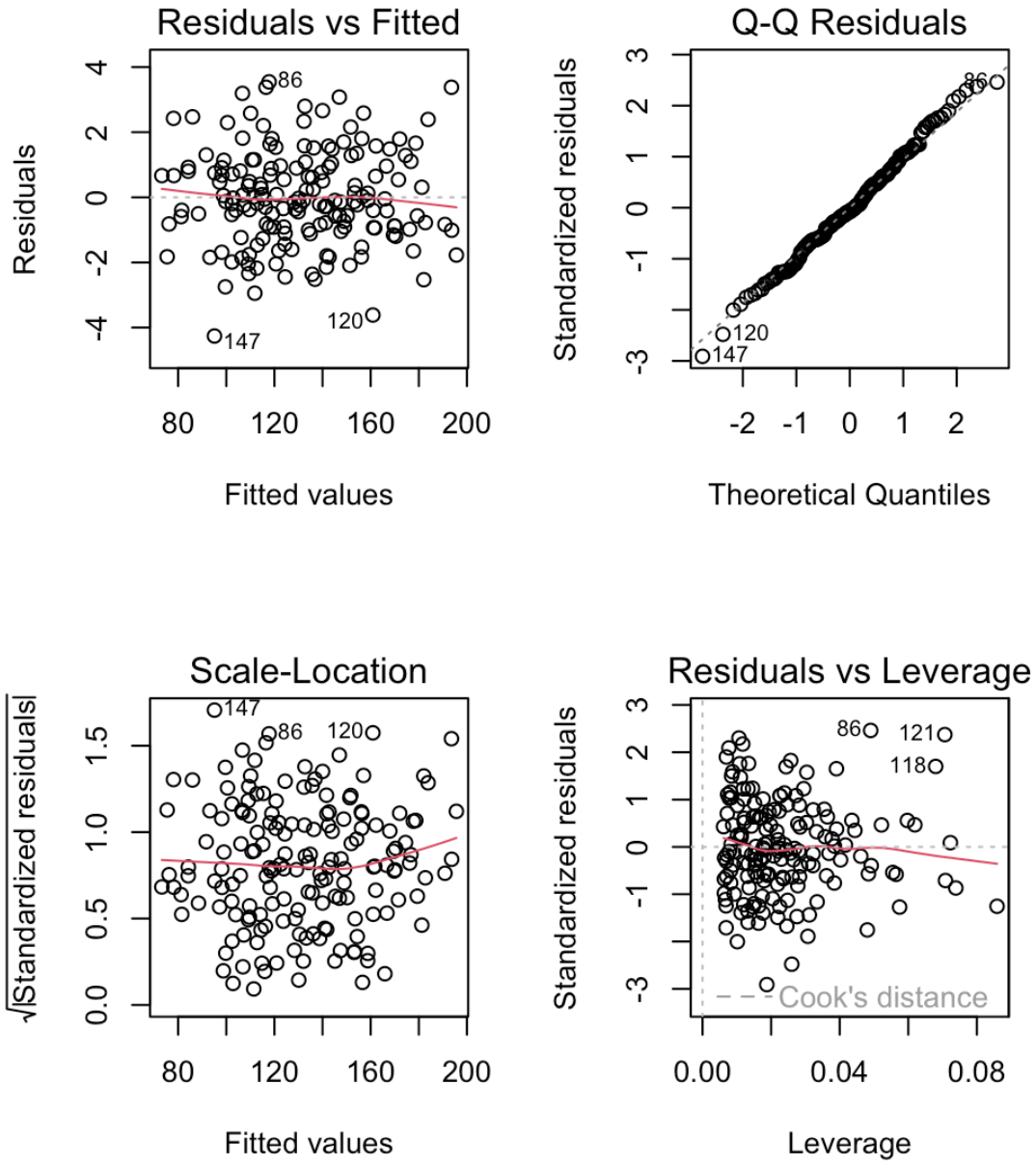

Figura 13.5: Grafici diagnostici per lm.full.

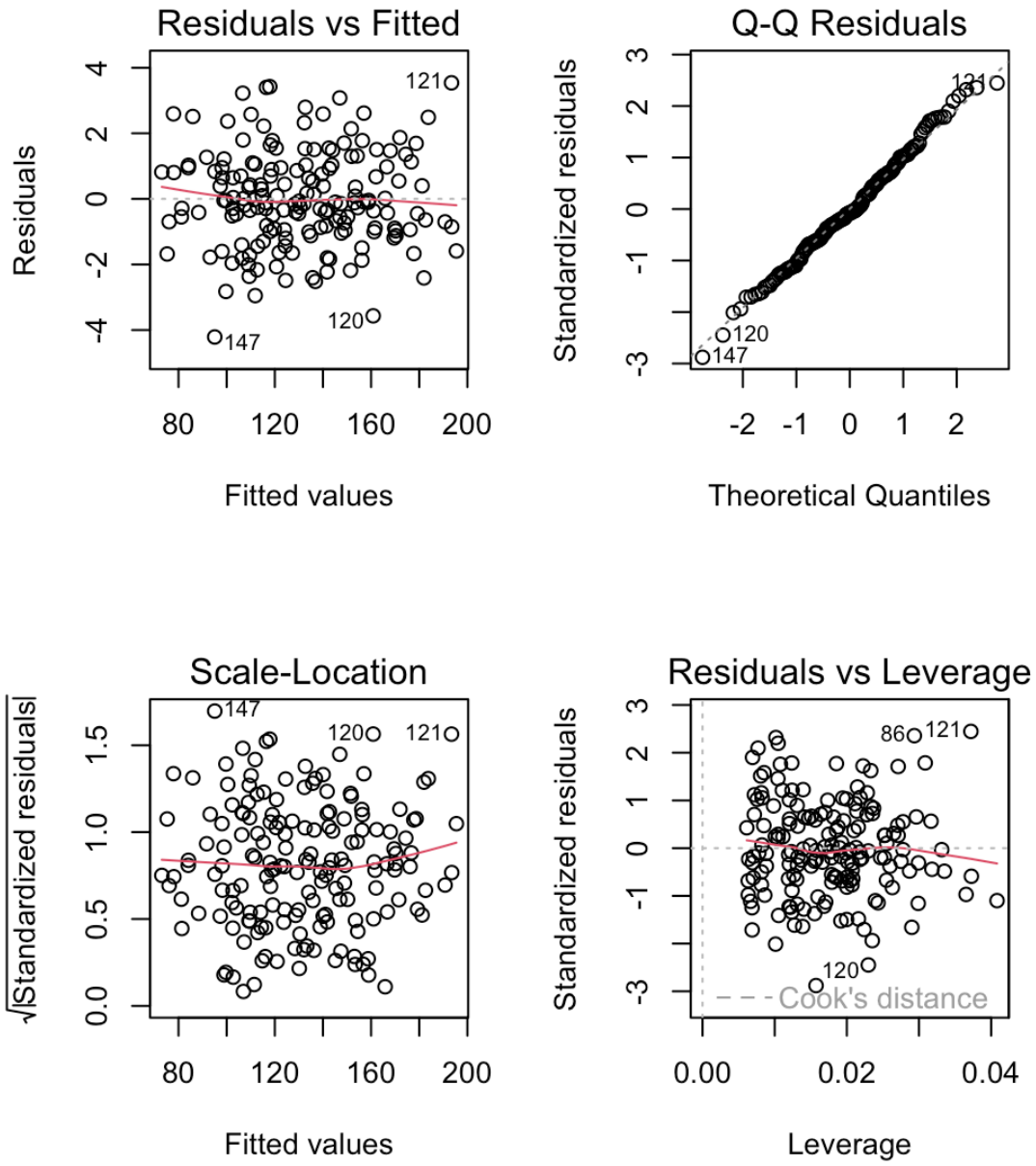

Figura 13.6: Grafici diagnostici per `lm.additive`.

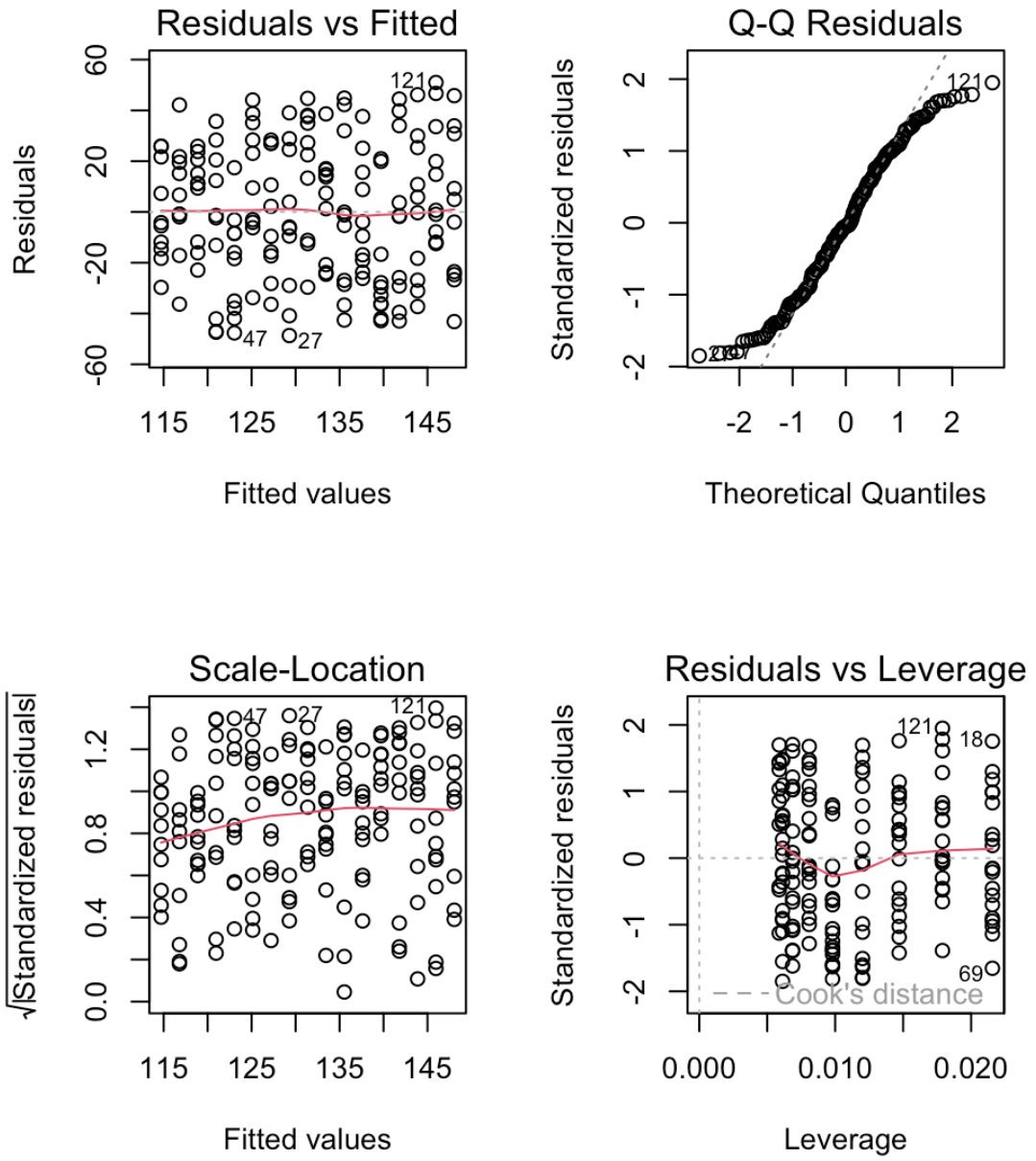

Figura 13.7: Grafici diagnostici per `lm.depth`. *Il grafico quantile-quantile indica non-normalità dei residui.*

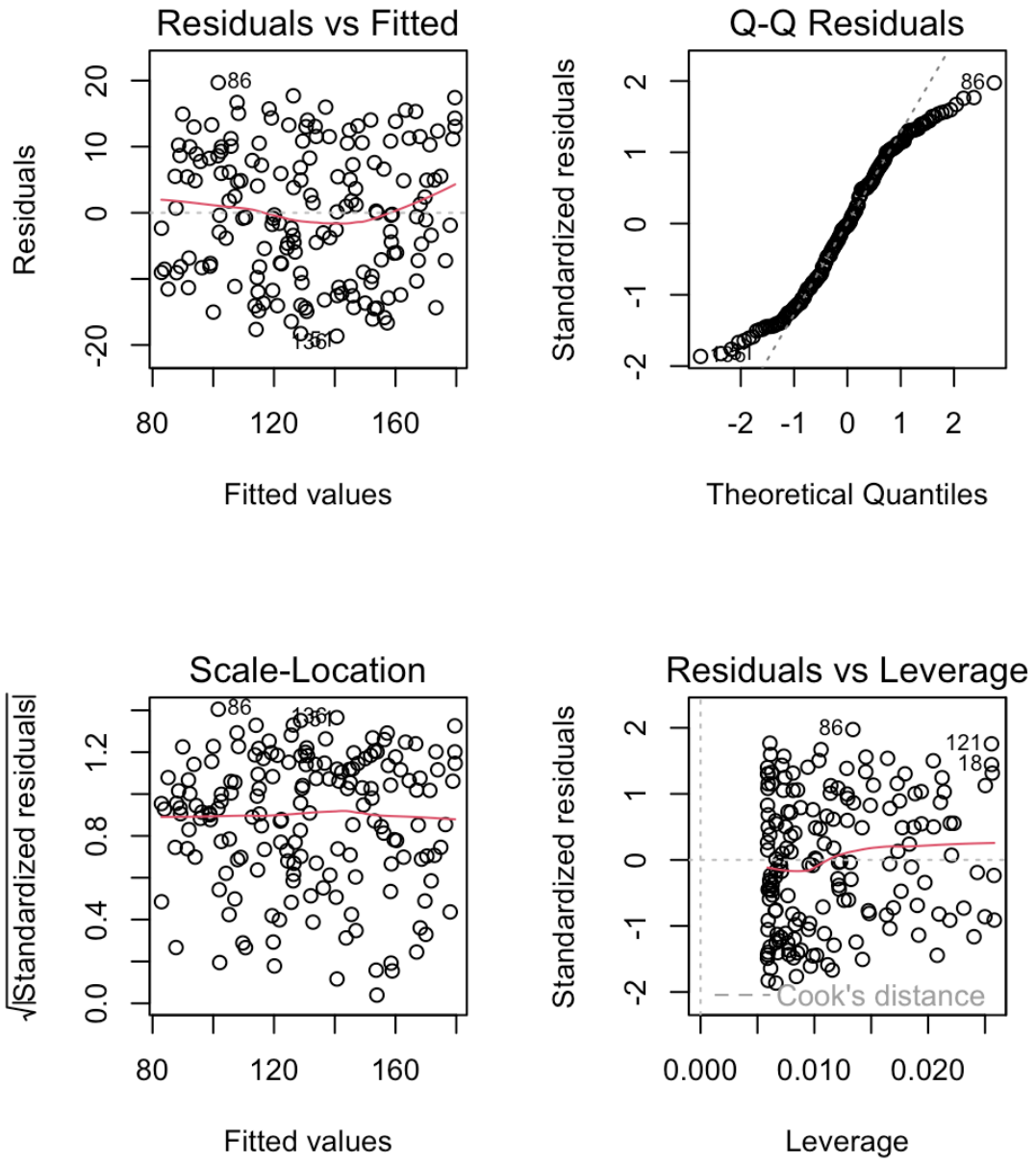

Figura 13.8: Grafici diagnostici per `lm.nutrients`*. Il grafico quantile-quantile indica non-normalità dei residui.*

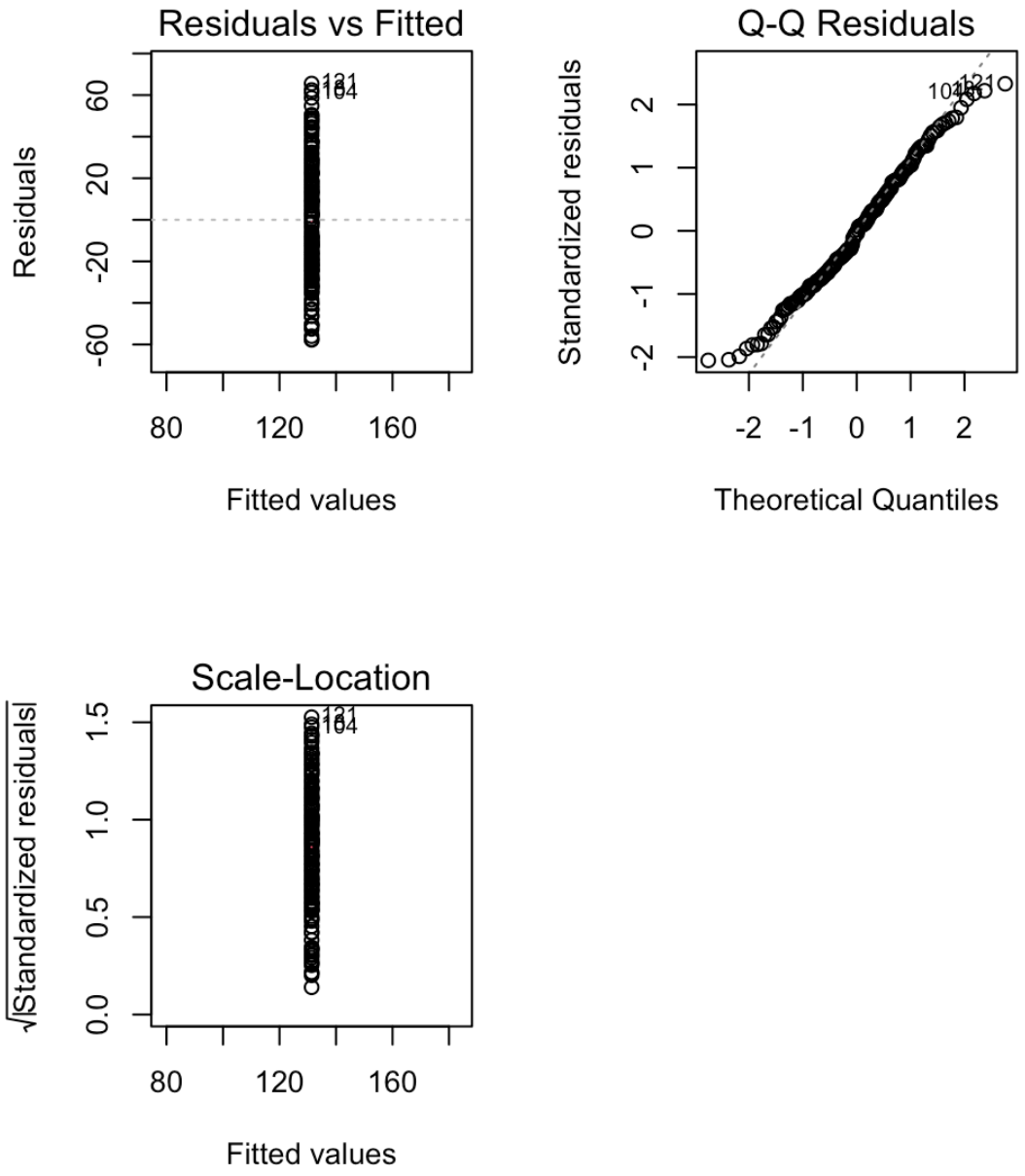

Figura 13.9: Grafici diagnostici per lm0. *Il grafico quantile-quantile è sospetto: non-normalità dei residui?*

14 Modelli lineari con più di due predittori

Questa sezione è stata rimossa dall'edizione gratuita di questo manuale. Potete acquistare l'edizione senza tagli qui:

- https://www.lulu.com/it/shop/marco-plebani/analisi-dei-dati-con-r-ed-rstudio/ebook/product-yvkp2ng.html (e-book)
- https://www.lulu.com/it/shop/marco-plebani/analisi-dei-dati-con-r-ed-rstudio/paperback/product-je6k8y6.html (in formato cartaceo)

Buona lettura!

- l'autore

15 Violazioni delle assunzioni dei modelli e come affrontarle

Finora abbiamo analizzato dati che si prestavano all'analisi tramite modelli lineari senza mostrare la minima violazione delle assunzioni degli stessi. Nei prossimi capitoli vedremo:

- come identificare e gestire la presenza di valori estremi o *outliers* (capitolo 16);
- come analizzare dati che mostrano correlazioni non-lineari tra una variabile di risposta ed un predittore continuo (regressione polinomiale: capitolo 17; modelli non-lineari: 18);
- come gestire dati per i quali i residui di un modello lineare violano l'assunzione di normalità (capitolo 19);
- come analizzare dati non-indipendenti (capitolo 20).

16 Valori estremi *(outliers)*

La presenza di valori estremi, o *outliers*, all'interno di un dataset può alterare sia le stime dei parametri dei modelli, sia i risultati e l'affidabilità dei test statistici. Gli *outliers* presentano due problemi: come identificarli, e come gestirli.

I grafici diagnostici prodotti da R per i modelli lineari permettono di individuare possibili *outliers* tramite esame visivo e tramite un indice noto come **distanza di Cook**. Come già spiegato nella sezione 6, la distanza di Cook *D* per una certa osservazione corrisponde alla somma delle differenze tra i valori stimati dal modello in base all'intero dataset ed i valori stimati escludendo l'osservazione in questione dal dataset. Si può pensare alla distanza di Cook di una osservazione come una misura dell'"effetto-leva" di quella osservazione sulla stima dei parametri del modello.

Vediamo un esempio pratico. Nel capitolo sulla regressione lineare (sez. 8) abbiamo studiato la variabilità della lunghezza di un'alga a diverse profondità. Supponiamo che i dati raccolti in quell'occasione fossero stati lievemente differenti, ovvero che fossero come quelli riportati nel file *outlier_example.csv* (lo trovate a https://github.com/marcoplebani85/datasets/):

```
rm(list = ls())  # ripulisce la memoria di R
# definiamo la working directory:
setwd("~/Desktop/Corso_R")
# carichiamo il file di dati:
df <- read.csv("my_data/outlier_example.csv")
str(df)

## 'data.frame':    85 obs. of  2 variables:
##  $ depth      : int  2 3 4 5 6 7 8 9 10 11 ...
##  $ alga.length: num  5.79 7.64 23.1 11.07 10.23 ...
```

Visualizzando i dati notiamo che una osservazione (indicata dalla freccia in figura 16.1) si discosta da tutte le altre:

```
# rappresentazione dei dati:
par(bty="L", family="Times")
plot(alga.length ~ depth, data=df,
     # personalizzazione dei nomi degli assi:
     xlab="Profondità [m s.l.m.]",
     ylab="Lunghezza algale [cm]",
     xlim=c(0,20), # limiti degli assi
     ylim=c(0,25),
     pch=21
     )

#freccia per indicare il potenziale outlier:
arrows(x0=8, y0=23, x1 = 5, y1 = 23, length=0.15, lwd=2, col="black")
```

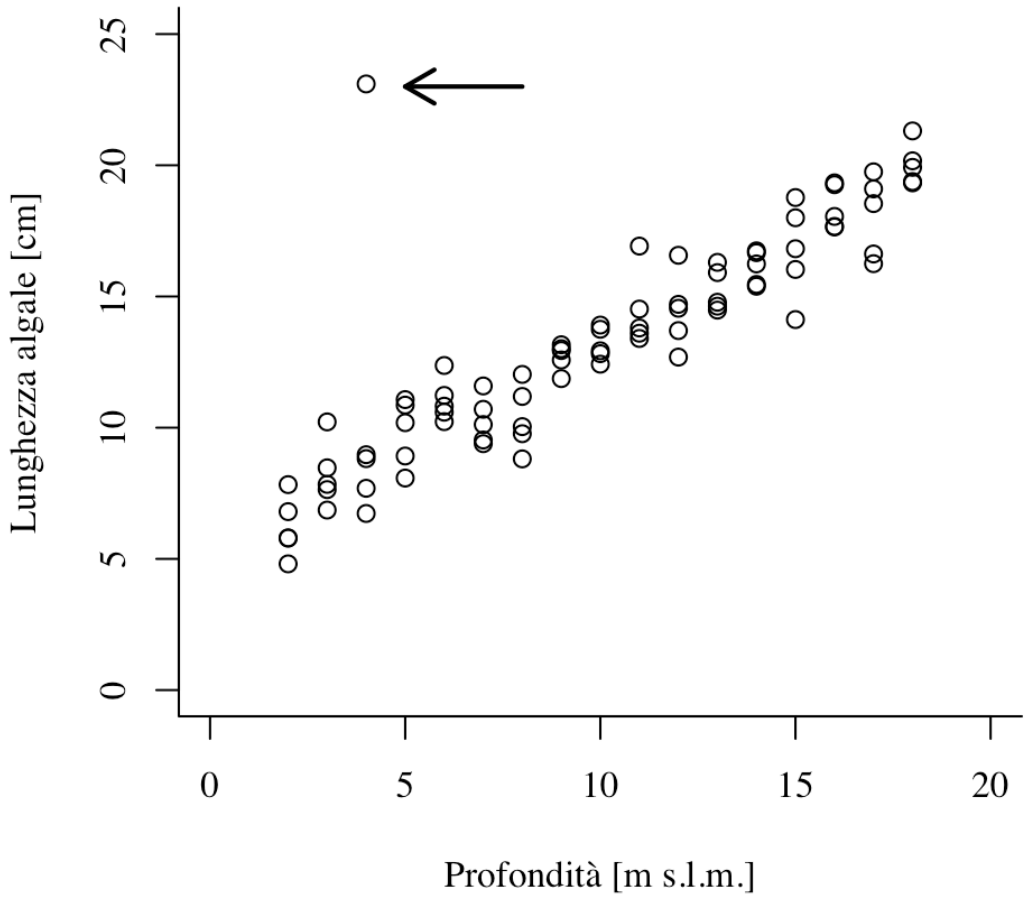

Figura 16.1: Lunghezza algale misurata a diverse profondità. La freccia indica un potenziale outlier.

Definiamo un modello di regressione lineare tra lunghezza algale e profondità:

```
lm1 <- lm(alga.length ~ depth, data = df)
```

Prima di procedere a stimare i parametri del modello e a valutarne la significatività statistica dobbiamo ispezionare i grafici diagnostici del modello appena creato:

```
par(mfrow = c(2, 2))
plot(lm1)
```

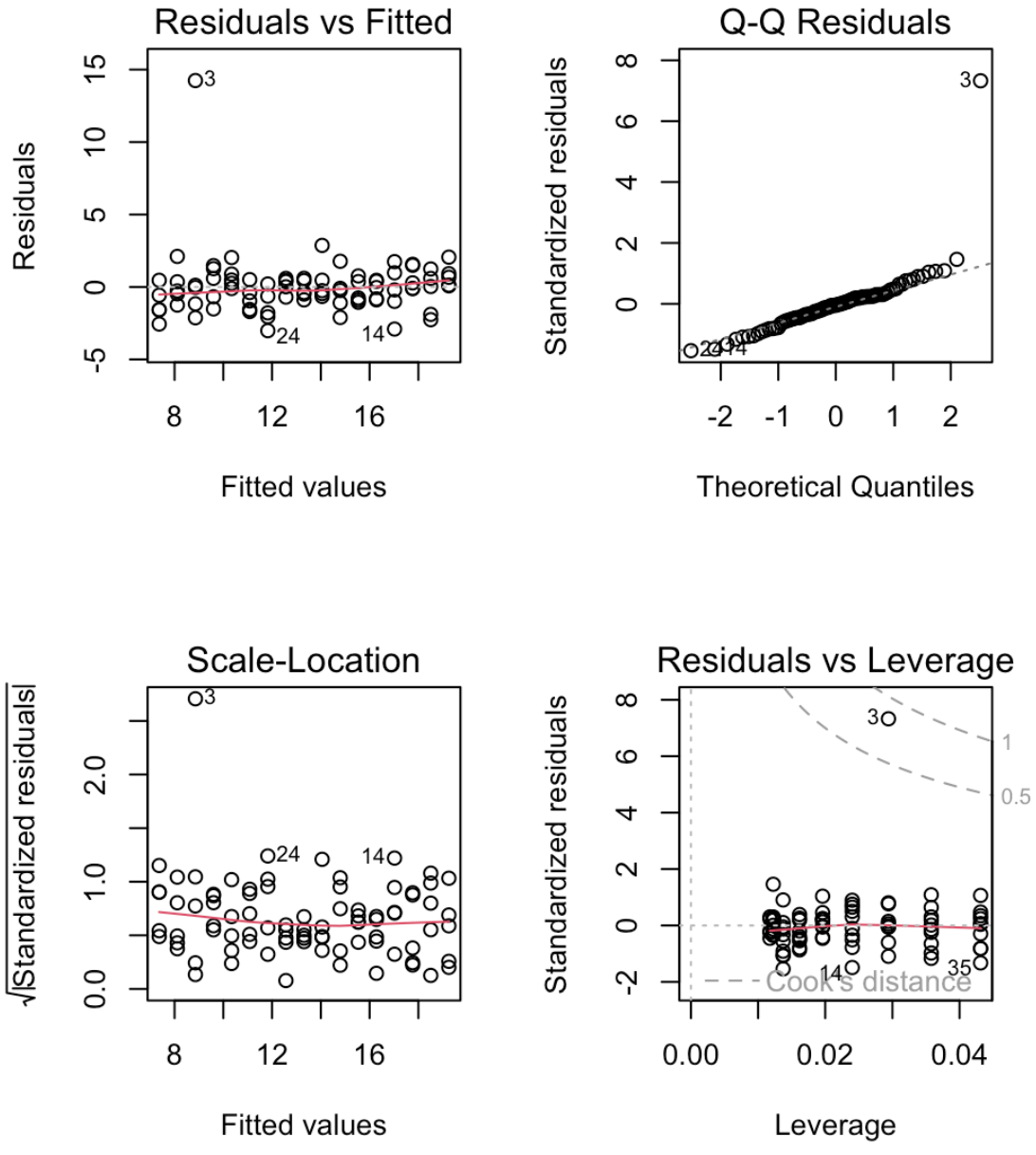

Figura 16.2: Grafici diagnostici per il modello lm1.

Osservando i grafici diagnostici (fig. 16.2) notiamo che l'osservazione n. 3, ovvero quella riportata nella terza riga del nostro dataset, si discosta da tutte le altre. In particolare il grafico *Residuals vs Leverage* mostra che la distanza di Cook *D* per la terza osservazione è compresa tra 0.5 ed 1. Non c'è un valore-soglia di *D* universalmente accettato per considerare un valore un *outlier,* e l'identificazione degli *outliers* tende ad avere un elemento di soggettività. La gestione degli *outliers* dipende inoltre dalla loro origine: un *outlier* potrebbe essere risultato di un errore di misurazione o di campionamento, oppure potrebbe dovuto alla normale variabilità della popolazione in studio. Per questi motivi, la gestione degli *outliers* richiede cautela e trasparenza.

Un possibile modo di procedere consiste nel rappresentare i possibili *outliers* in maniera chiara ed evidente nelle figure, rimuovendoli quindi dall'analisi quantitativa dei dati. Per escludere l'osservazione contenuta nel terzo rigo del dataset possiamo procedere come segue:

```
df2 <- df[-3, ]
```

In R le parentesi quadre permettono di selezionare specifiche righe e colonne di una matrice o un `data.frame`: a sinistra della virgola tra parentesi si indicano le righe, a destra le colonne. Il codice qui sopra sta dicendo a R: "rimuovi la riga 3 da `df`, preserva tutte le colonne". Il risultato dei modelli di regressione calcolati sul dataset completo e su quello senza *outlier* sono mostrati in figura 16.3:

```
# rappresentazione dei dati:
par(mfrow=c(1, 1), bty="L", family="Times")
plot(alga.length ~ depth, data=df,
     xlab="Profondità [m]", # personalizzazione dei nomi degli assi
     ylab="Lunghezza algale [cm]",
     xlim=c(0,20), # limiti degli assi
     ylim=c(0,25),
     pch=21
     )

#freccia per indicare il potenziale outlier:
arrows(x0=8, y0=23, x1 = 5, y1 = 23, length=0.15, lwd=2)

# fit del modello di regressione su df2:
lm1 <- lm(alga.length ~ depth, data = df)
# retta di regressione stimata senza l'outlier:
abline(lm1, lwd=2, lty="dashed")

# fit del modello di regressione su df2:
lm2 <- lm(alga.length ~ depth, data = df2)
# retta di regressione stimata senza l'outlier:
abline(lm2, lwd=2)
```

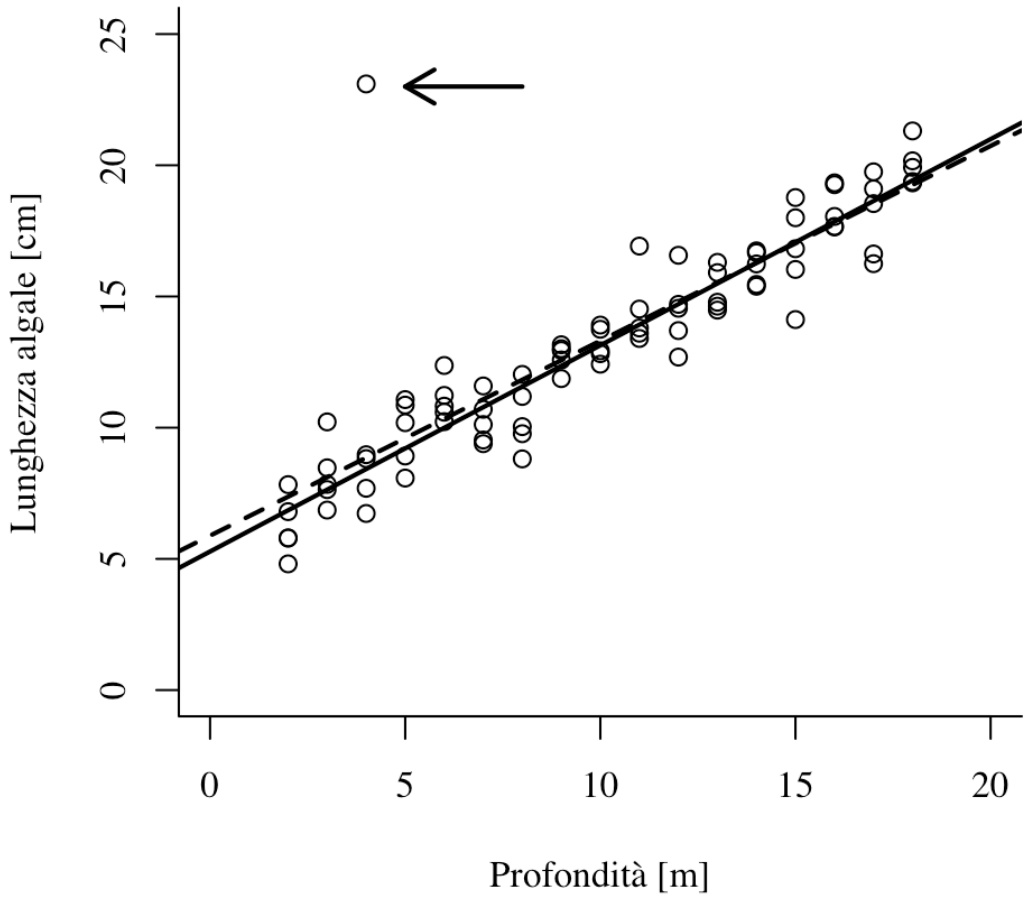

Figura 16.3: Lunghezza algale misurata a diverse profondità. La freccia indica un potenziale `outlier`*. La retta tratteggiata corrisponde alla retta di regressione calcolata sull'intero dataset. La retta continua corrisponde alla retta di regressione calcolata escludendo il potenziale* `outlier` *dal dataset.*

È cruciale descrivere in modo chiaro ed esplicito modo in cui sono gestiti gli *outliers*. In un ipotetico articolo riporteremmo la scelta operata e ne discuteremmo le possibili implicazioni. Se considerassimo l' *outlier* come il probabile riflesso della naturale variabilità della popolazione in studio, questo fatto e le sue possibili implicazioni dovrebbero essere discussi esplicitamente.

Altri possibili approcci analitici in presenza di *outliers*, non trattati in questo libro, sono:

- l'uso di metodi di analisi non-parametrici;
- l'uso di cosiddetti "metodi di regressione robusti", così chiamati in quanto scarsamente influenzati dalla presenza di *outliers*. Si vedano, ad esempio, la funzione `rlm()` presente nel pacchetto `car` e le referenze bibliografiche riportate nella pagina di aiuto della funzione stessa.

17 Regressione polinomiale

La relazione tra una variabile di risposta ed un predittore continuo non è sempre rappresentabile con una retta. Un possibile approccio per modellizzare questo tipo di correlazioni consiste nell'usare equazioni polinomiali.

17.1 Esempio

Vediamo un esempio di relazione non rettilinea tra una variabile di risposta `y` ed un predittore continuo `x`. Cominciamo come nostro solito col preparare la sessione di lavoro e col caricare i pacchetti che prevediamo ci serviranno:

```
rm(list = ls())  # ripulisce la memoria di R

# carica 'pacman', che contiene la funzione p_load():
if (!require(pacman)) install.packages("pacman")
library(pacman)
p_load(emmeans)
p_load(ggplot2)
p_load(readxl)  # per caricare dati da file Excel
```

Carichiamo i dati, che in questo caso sono salvati nel file `non_linear_datasets.xlsx` (lo trovate a https://github.com/marcoplebani85/datasets/) in R:

```
# definiamo la directory di lavoro. In R:
setwd("~/Desktop/Corso_R")
# In RSTUDIO: menu -> Session -> Set working directory
df <- as.data.frame(
  readxl::read_excel("my_data/non_linear_datasets.xlsx", sheet=1)
  )
```

La funzione `read_excel()` del pacchetto `readxl` (Wickham e Bryan, 2023) permette di aprire file `.xlsx` in R. L'argomento "`sheet=`" permette di scegliere il foglio di lavoro da cui attingere i dati. Il dataset consiste di valori numerici organizzati in 155 righe e due colonne:

```
str(df)

## 'data.frame':    155 obs. of  2 variables:
##  $ x: num  10 11 12 13 14 15 16 17 18 19 ...
##  $ y: num  328 294 383 401 484 ...
```

Visualizziamo i dati:

```
par(bty = "L", family = "Times")
plot(y ~ x, data = df)
```

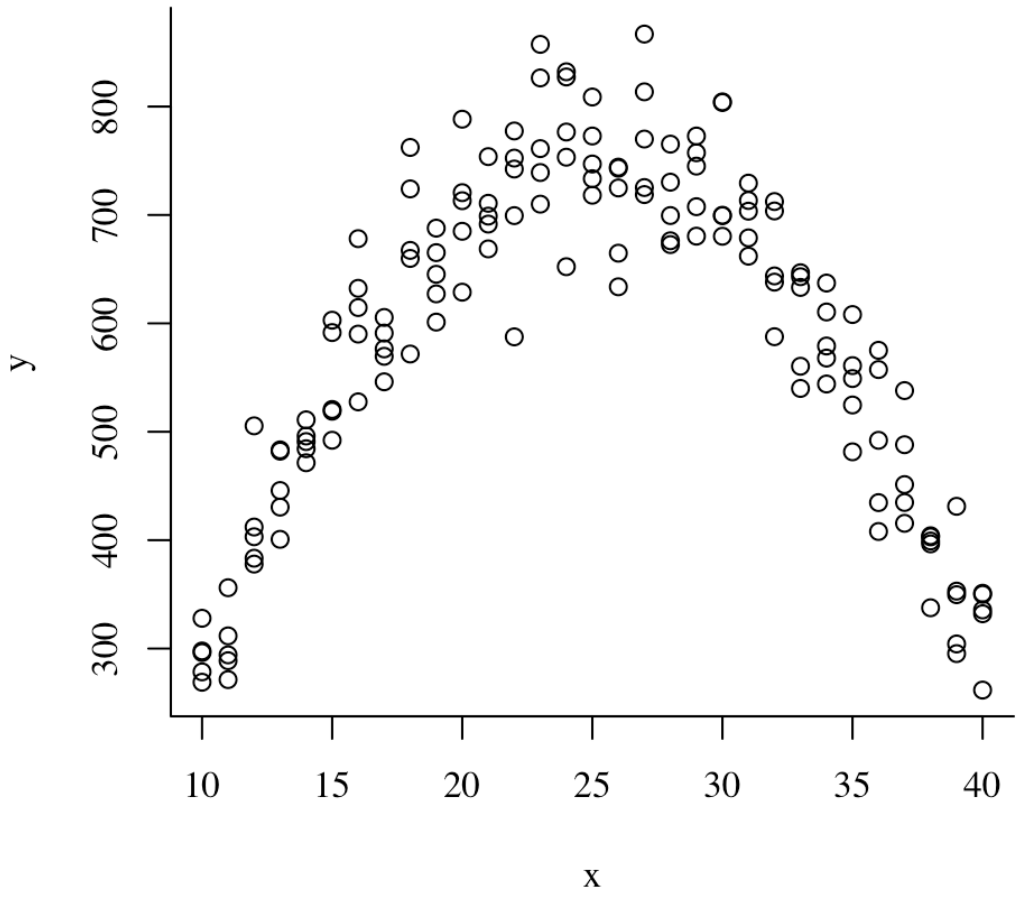

Figura 17.1: Andamento di y rispetto ad x.

La figura 17.1 mostra un'ovvia correlazione non-rettilinea tra `y` ed `x`. Un modello di regressione lineare (`lm1`) non è in grado di catturare questa correlazione, come possiamo notare dai corrispondenti grafici diagnostici (fig. 17.3):

Dal momento che la relazione tra `y` ed `x` appare curvilinea (fig. 17.1), un polinomio di secondo grado dovrebbe approssimare bene i dati. Per definire regressioni polinomiali in R si può usare la funzione `poly()` all'interno di `lm()`:

```
lm.poly <- lm(y ~ poly(x, degree = 2, raw = T), data = df)
```

È importante specificare l'argomento "`raw=T`" all'interno di `poly()`. Questo fa sì che la funzione stimi i parametri del polinomio in maniera non-ortogonale, facilitandone l'interpretabilità. Il modello `lm.poly` corrisponde al polinomio di secondo grado:

$$y_i = a + b \cdot x_i + c \cdot x_i^2 + \epsilon_i \qquad (17.1)$$

I grafici diagnostici del modello `lm.poly` (fig. 17.4) indicano che il modello rispetta le assunzioni dei modelli lineari.

Il confronto dei due modelli indica che `lm.poly` descrive i dati molto meglio di `lm1`:

```
AIC(lm.poly, lm1)

##         df      AIC
## lm.poly  4 1655.768
## lm1      3 2011.330

anova(lm.poly, lm1)

## Analysis of Variance Table
##
## Model 1: y ~ poly(x, degree = 2, raw = T)
## Model 2: y ~ x
##   Res.Df     RSS Df Sum of Sq      F    Pr(>F)
## 1    152  375610
## 2    153 3772181 -1  -3396571 1374.5 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Possiamo ottenere i coefficienti di `lm.poly` con `summary()`, come abbiamo fatto per la regressione lineare e per la regressione multipla:

```
summary(lm.poly)

##
## Call:
## lm(formula = y ~ poly(x, degree = 2, raw = T), data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -145.267  -27.400   -5.518   30.497  121.026
##
## Coefficients:
##                                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)                   -555.33398   32.68446  -16.99   <2e-16 ***
## poly(x, degree = 2, raw = T)1  104.13746    2.82988   36.80   <2e-16 ***
## poly(x, degree = 2, raw = T)2   -2.07204    0.05589  -37.07   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 49.71 on 152 degrees of freedom
## Multiple R-squared:  0.9005, Adjusted R-squared:  0.8992
## F-statistic:   688 on 2 and 152 DF,  p-value: < 2.2e-16
```

Usiamo ora `ggplot()` per rappresentare sia i dati sia le predizioni del modello `lm.poly`(fig. 17.2):

```
ggplot(df, aes(x=x, y=y)) + # informazioni base
  geom_point(shape=1, size=2) +
  theme_classic() +
  # scelta del carattere di testo da usare:
  theme(text = element_text(family="Times", size=15)) +
  # predizioni del modello:
    geom_smooth(method = "lm", formula = y ~ poly(x,2),
                color="black", linewidth=0.5)
```

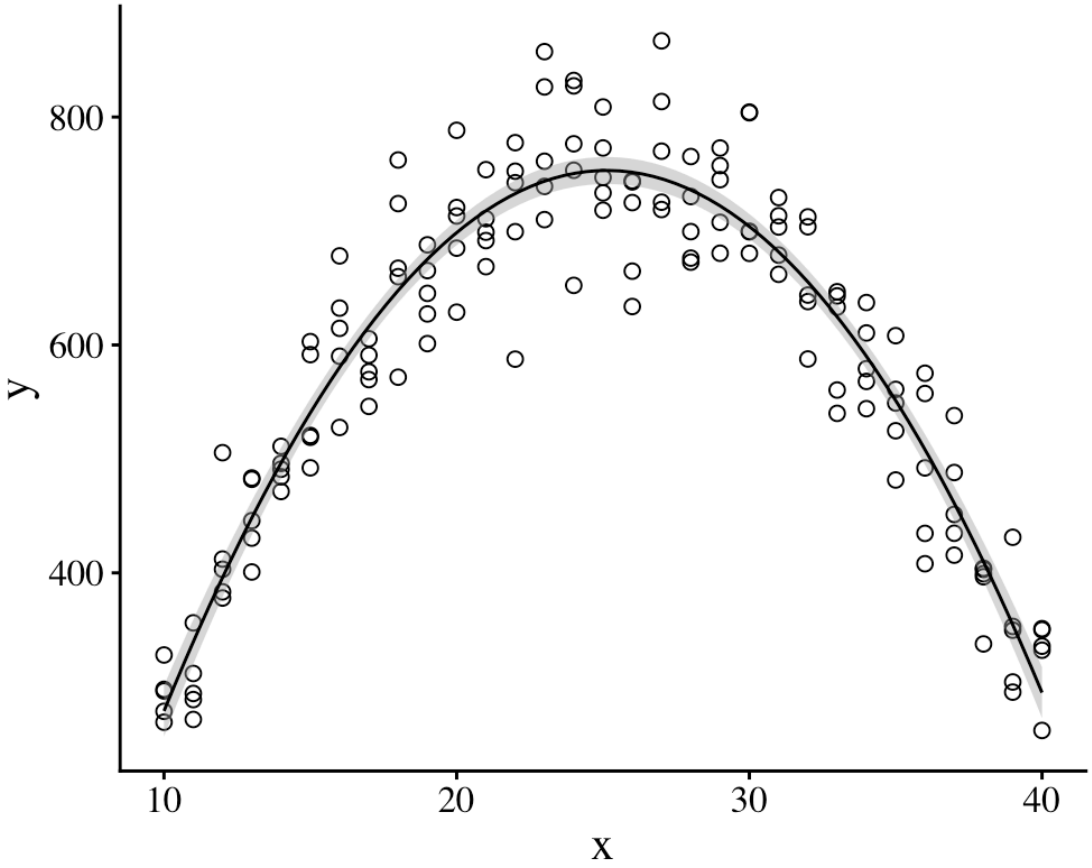

Figura 17.2: Dati e predizioni del modello di regressione polinomiale. L'area grigia rappresenta l'intervallo di confidenza del modello.

La funzione `geom_smooth()` permette di indicare la formula dei modelli da rappresentare.

Nota: un modello lineare è così chiamato perchè descrive la variabilità osservata nella variabile di risposta come la somma *lineare* degli effetti dei predittori. I modelli di regressione lineare e quelli di regressione polinomiale sono entrambi modelli lineari. Il modello di regressione polinomiale riportato nell'equazione (17.1), pur descrivendo una correlazione non-rettilinea tra y ed x, è un modello lineare in quanto descrive la variabilità media di y come la somma *lineare* degli effetti di a, x, ed x^2.

17.1.1 Appendice: grafici diagnostici

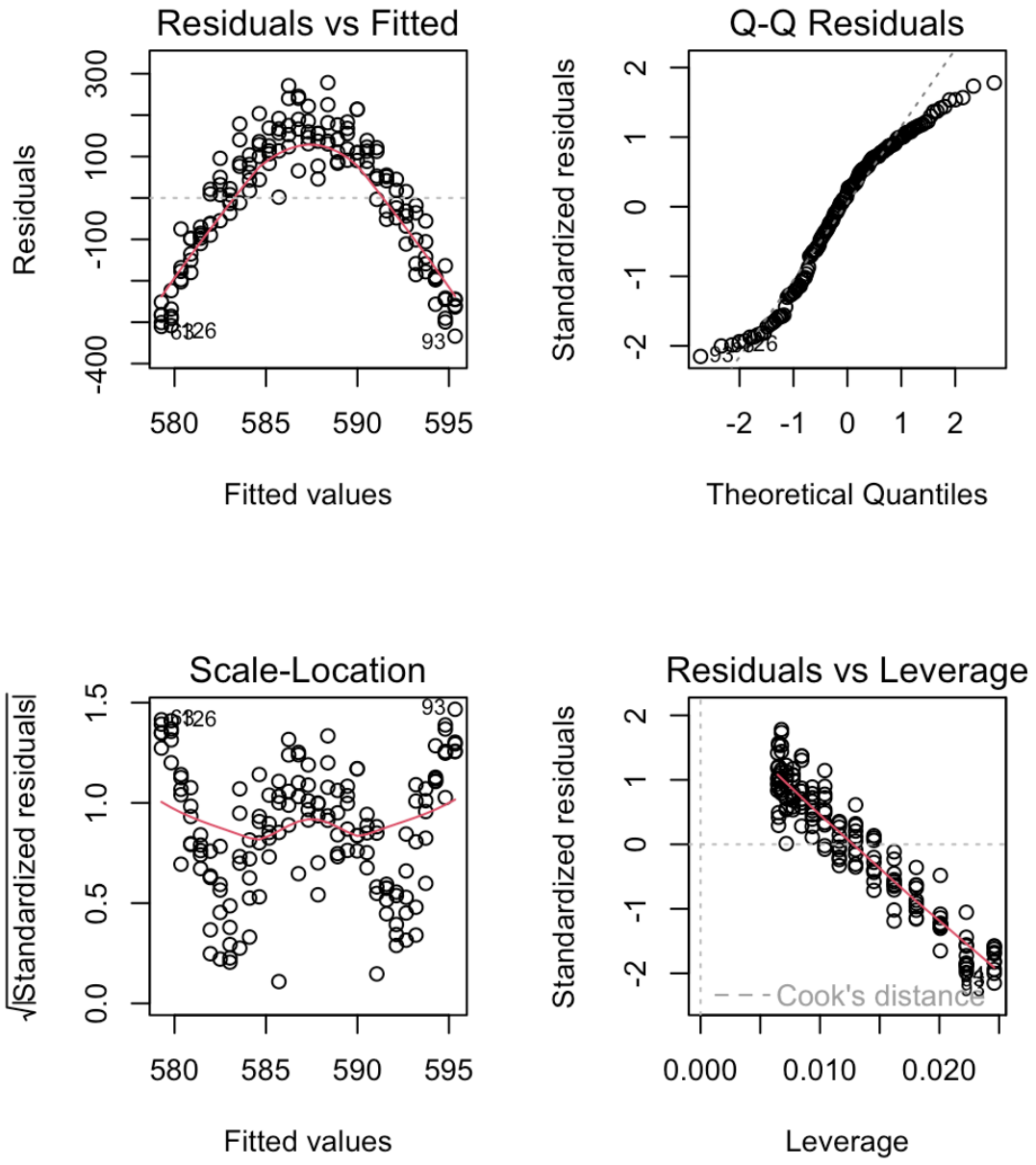

Figura 17.3: Grafici diagnostici per il modello lm1.

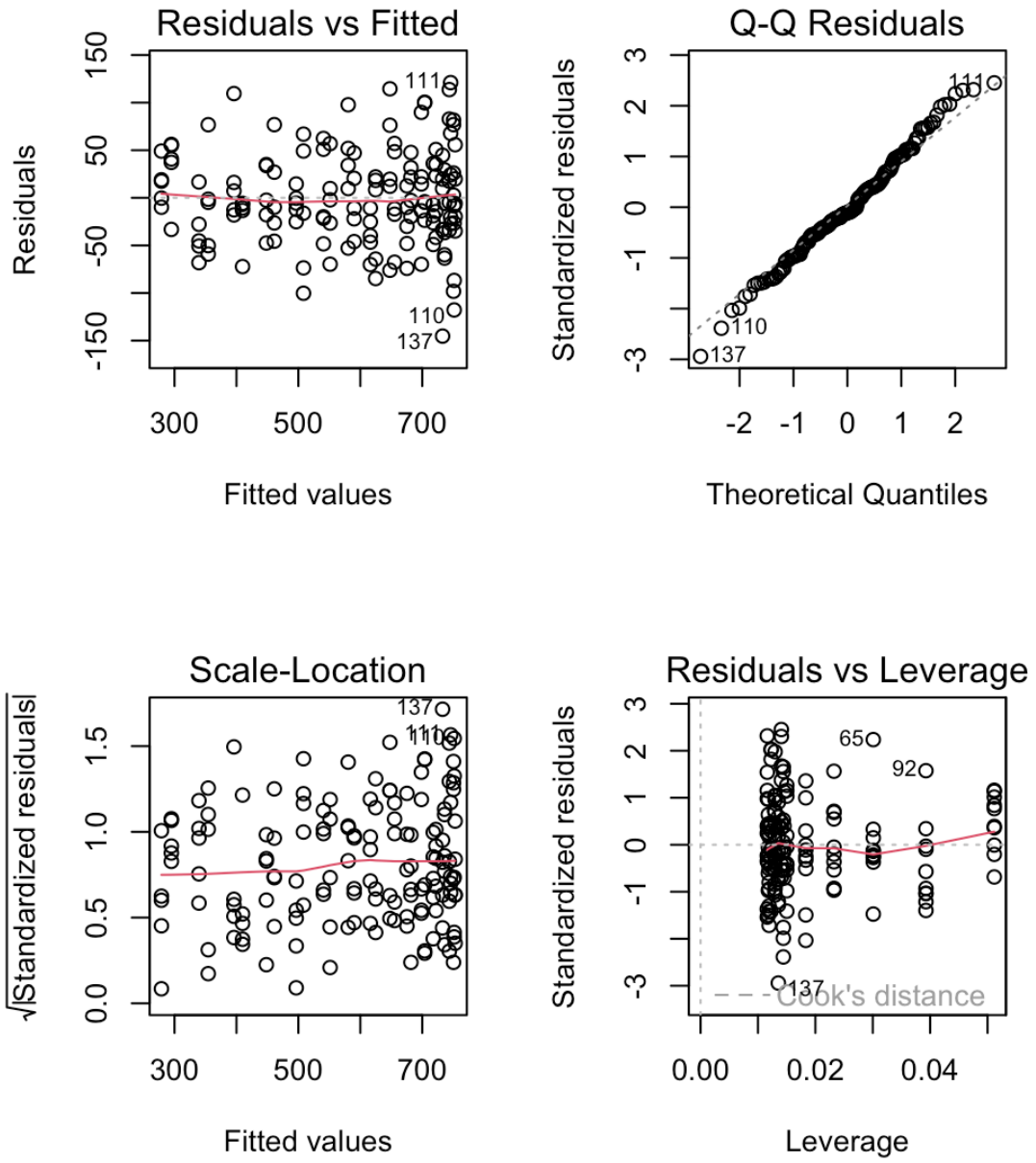

Figura 17.4: Grafici diagnostici per il modello `lm.poly`.

17.2 Regressione polinomiale ed *overfitting*

Questa sezione è stata rimossa dall'edizione gratuita di questo manuale. Potete acquistare l'edizione senza tagli qui:

- https://www.lulu.com/it/shop/marco-plebani/analisi-dei-dati-con-r-ed-rstudio/ebook/product-yvkp2ng.html (e-book)
- https://www.lulu.com/it/shop/marco-plebani/analisi-dei-dati-con-r-ed-rstudio/paperback/product-je6k8y6.html (in formato cartaceo)

Buona lettura!

- l'autore

18 Modelli non lineari con `nls()`

A volte la relazione tra predittori e variabili di risposta non può essere adeguatamente descritta da un modello lineare, ovvero dalla semplice somma degli effetti dei predittori sulla variabile di risposta. In questi casi può essere necessario ricorrere a modelli non-lineari. In R si possono implementare modelli non-lineari tramite la funzione `nls()`, che permette di definire modelli non-lineari e di stimarne i parametri. Vediamone due esempi.

18.1 Esempio 1

Supponiamo di avere misurato l'altezza e la biomassa di una specie arborea e di voler studiare la relazione tra le due variabili. I dati sono nel foglio 3 del file *non_linear_datasets.xlsx* (lo trovate a https://github.com/marcoplebani85/datasets/).

```
rm(list=ls()) # ripulisce la memoria di R
options(scipen=9999) # evita che R esprima numeri come potenze di 10
# carichiamo il pacchetto "pacman", che contiene la funzione p_load():
if (!require(pacman)) install.packages('pacman'); library(pacman)

# altri pacchetti:
p_load(emmeans)
p_load(ggplot2)
p_load(readxl)  # per caricare dati da file .xlsx
p_load(nlstools) # grafici diagnostici per nls()
p_load(nls.multstart) # extra options per nls()

# definiamo la directory di lavoro. In R:
setwd("~/Desktop/Corso_R")
# In RSTUDIO: menu -> Session -> Set working directory

# Carichiamo i dati in R:
df <- as.data.frame(
  readxl::read_excel("my_data/non_linear_datasets.xlsx", sheet=3)
  )
```

Visualizziamo i dati contenuti in `df` (fig. 18.1):

```
par(bty="L", family="Times")
plot(biomass ~ tree.height, data = df,
     xlab="Altezza [m]", ylab="Biomassa [Kg]")
```

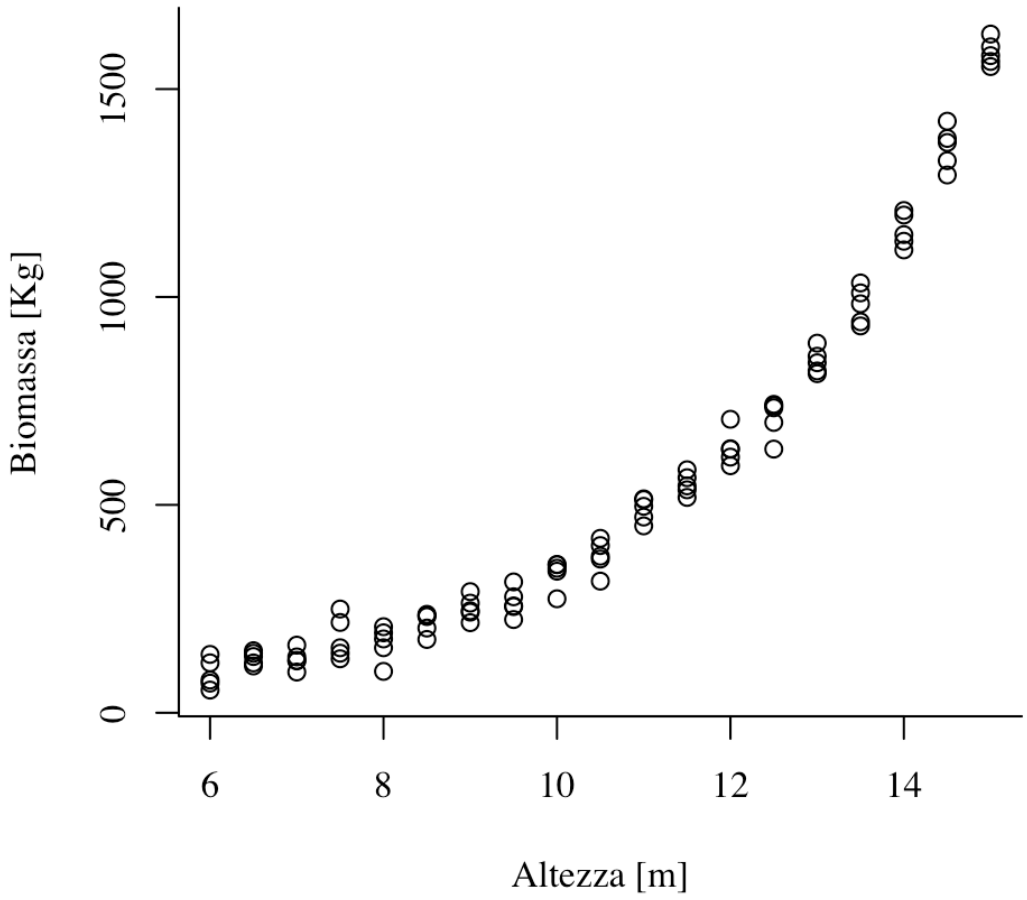

Figura 18.1: Relazione tra la biomassa e l'altezza degli alberi.

La relazione tra `tree.height` ed `biomass` è curvilinea (fig. 18.1), e potrebbe essere ben descritta da una parabola (ovvero un polinomio di secondo grado) oppure da un modello esponenziale[7]. Di seguito definiremo e confronteremo:

- il modello nullo;
- un modello di regressione lineare;
- un modello di regressione polinomiale;
- un modello esponenziale.

[7] Si tratta di dati simulati. Non me ne vogliano i botanici se questi dati non rispecchiano andamenti osservati in natura.

Sappiamo già definire i primi tre modelli con `lm()`:

```
lm0 <- lm(biomass ~ 1, data = df)
lm1 <- lm(biomass ~ tree.height, data = df)
lm.poly <- lm(biomass ~ poly(tree.height, degree = 2, raw = T), data = df)
```

Definiamo ora un modello esponenziale con intercetta 0 usando la funzione `as.formula()`:

```
exponential <- as.formula(biomass ~ 0 + b * exp(tree.height * c))
```

Usiamo ora la funzione `nls()` per stimare i parametri *b* e *c* del modello `exponential` sulla base dei dati contenuti in `df`:

```
nlm1 <-nls(exponential, # nome del modello
    data=df,
    # Aiutiamo l'algoritmo a trovare le stime giuste
    # dandogli dei punti di partenza:
    start = list(b=2, c = 1)
    )
```

I grafici diagnostici per `lm0` ed `lm1` mostrano che i due modelli sono chiaramente inadeguati a descrivere i dati (fig. 18.4 e 18.5, in appendice al capitolo). I grafici diagnostici per `lm.poly` indicano che il modello non descrive al meglio la non-linearità della correlazione tra le variabili (grafico *Residuals vs Fitted* in fig. 18.6) e c'è una certa eteroscedasticità nei suoi residui (grafico *Scale-Location* in fig. 18.6).

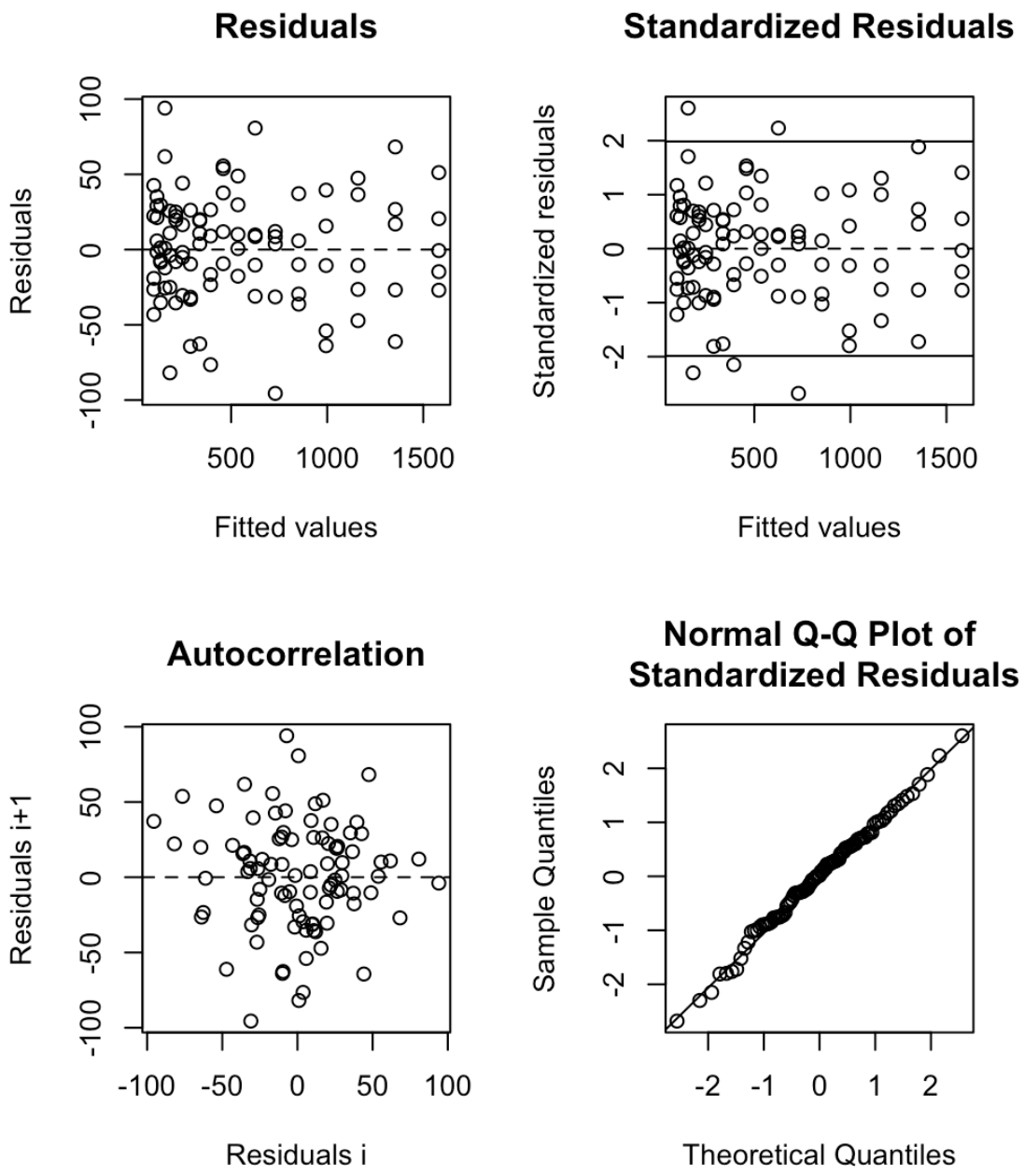

Figura 18.2: Grafici diagnostici per nlm1.

Le assunzioni dei modelli non-lineari sono simili a quelle dei modelli lineari: indipendenza delle osservazioni; indipendenza, normalità ed omoscedasticità dei residui. Per ottenere i grafici diagnostici per i modelli non lineari usiamo la funzione `nlsResiduals()` dal pacchetto `nlstools` (Baty et al.,2015):

```
plot(nlsResiduals(nlm1), which = 0)

# 'which = 0' mostra i quattro grafici nella stessa figura
```

Il terzo pannello in fig. 18.2 ("Autocorrelation") permette di valutare l'eventuale autocorrelazione della variabile di risposta, cioè se l'i-esima osservazione sia più o meno simile alla $(i - l)$-esima osservazione di quanto ci si aspetti per effetto del caso. L'autocorrelazione di una variabile è una forma di non-indipendenza delle osservazioni per quella variabile. Il valore di l è il "lag", o ritardo, ovvero la distanza tra i residui a confronto. Per una valutazione approfondita dell'autocorrelazione bisognerebbe esaminare più di un *lag*, ma l=1 fornisce un'utile prima approssimazione. Il grafico "Autocorrelation" mostra la correlazione tra i residui relativi a ciascuna osservazione e i residui relativi all'osservazione immediatamente precedente.

I grafici diagnostici del modello esponenziale `nlm1` indicano che i suoi residui seguono una distribuzione normale; non c'è autocorrelazione tra le osservazioni per lag=1; il modello cattura l'andamento non-lineare della correlazione tra `biomass` e `tree.height` (fig. 18.2).

Confrontiamo gli AIC di `nlm1` ed `lm.poly`:

```
AIC(lm.poly, nlm1)
```

```
##         df       AIC
## lm.poly  4 1023.4784
## nlm1     3  953.7677
```

Il confronto degli AIC dei due modelli indica che i dati sono meglio descritti da `nlm1`, cioè da un modello esponenziale. Otteniamo le predizioni di entrambi i modelli in modo da poterle confrontare:

```
# sequenza di tree.height da fornire a predict():
predictions <- data.frame(tree.height = seq(0, 16, by = 0.01))
# predizioni del modello esponenziale:
predictions$nlm1 <- predict(nlm1, newdata = predictions)
# predizioni del modello polinomiale:
predictions$lm.poly <- predict(lm.poly, newdata = predictions)
```

Visualizziamo le predizioni di entrambi i modelli insieme ai dati:

```
par(bty="L", family="Times")
plot(biomass ~ tree.height,
     data = df,
     xlim=c(0,16),
     cex=2.5, cex.axis=1.2, cex.lab =1.2,
     pch=16, col=grey(level=0.4, alpha=0.3),
     xlab="Altezza [m]",
     ylab="Biomassa [Kg]",
     las=1 # cambia l'orientamento dei valori degli assi
)

# predizioni del modello esponenziale:
points(x=predictions$tree.height,
       y=predictions$nlm1,
       type="l", lwd=2, col="black")
```

```r
# predizioni del modello polinomiale:
points(x=predictions$tree.height,
       y=predictions$lm.poly,
       type="l", lwd=3, col="red", lty="dotted")

# legenda:
legend("topleft", # posizione
       bty="n", # no contorno
       cex=1.2,
       lwd=2,
       title="", # titolo (vuoto)
       legend=c("modello esponenziale", "modello polinomiale"), # legenda
       lty=c("solid", "dotted"), # tipo di linea in legenda
       col=c("black", "red") # colore linea in legenda
       )
```

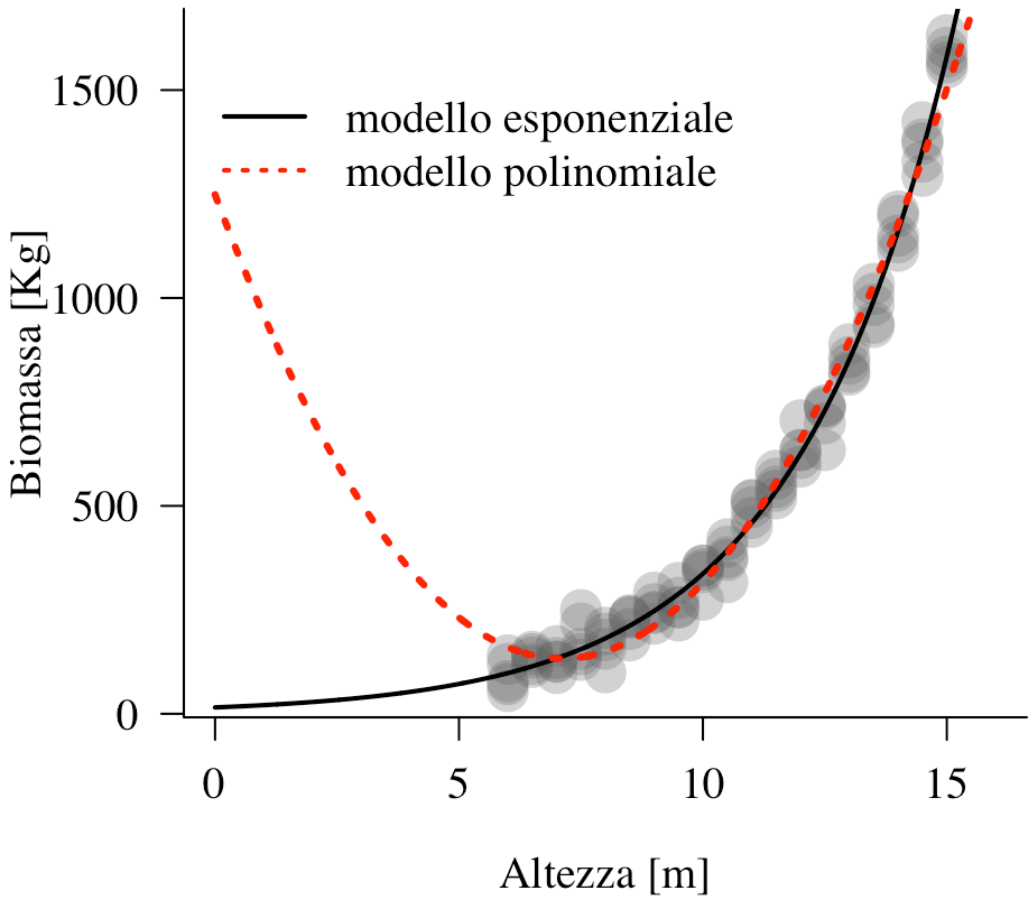

Figura 18.3: Dati e predizioni dei modelli esponenziale e polinomiale.

Non solo il modello esponenziale approssima i dati osservati meglio del modello polinomiale, ma produce predizioni sensate anche al di fuori dell'intervallo dei valori osservati: il modello esponenziale predice una correlazione positiva tra la biomassa degli alberi e la loro altezza anche al di fuori dei valori osservati di `tree.height`; al contrario, il modello polinomiale suggerisce anche che, per altezze inferiori ai ~7 m, la biomassa degli alberi aumenti al diminuire della loro altezza, una predizione in ovvio contrasto con l'evidenza biologica (fig. 18.3).

18.1.1 Appendice: grafici diagnostici

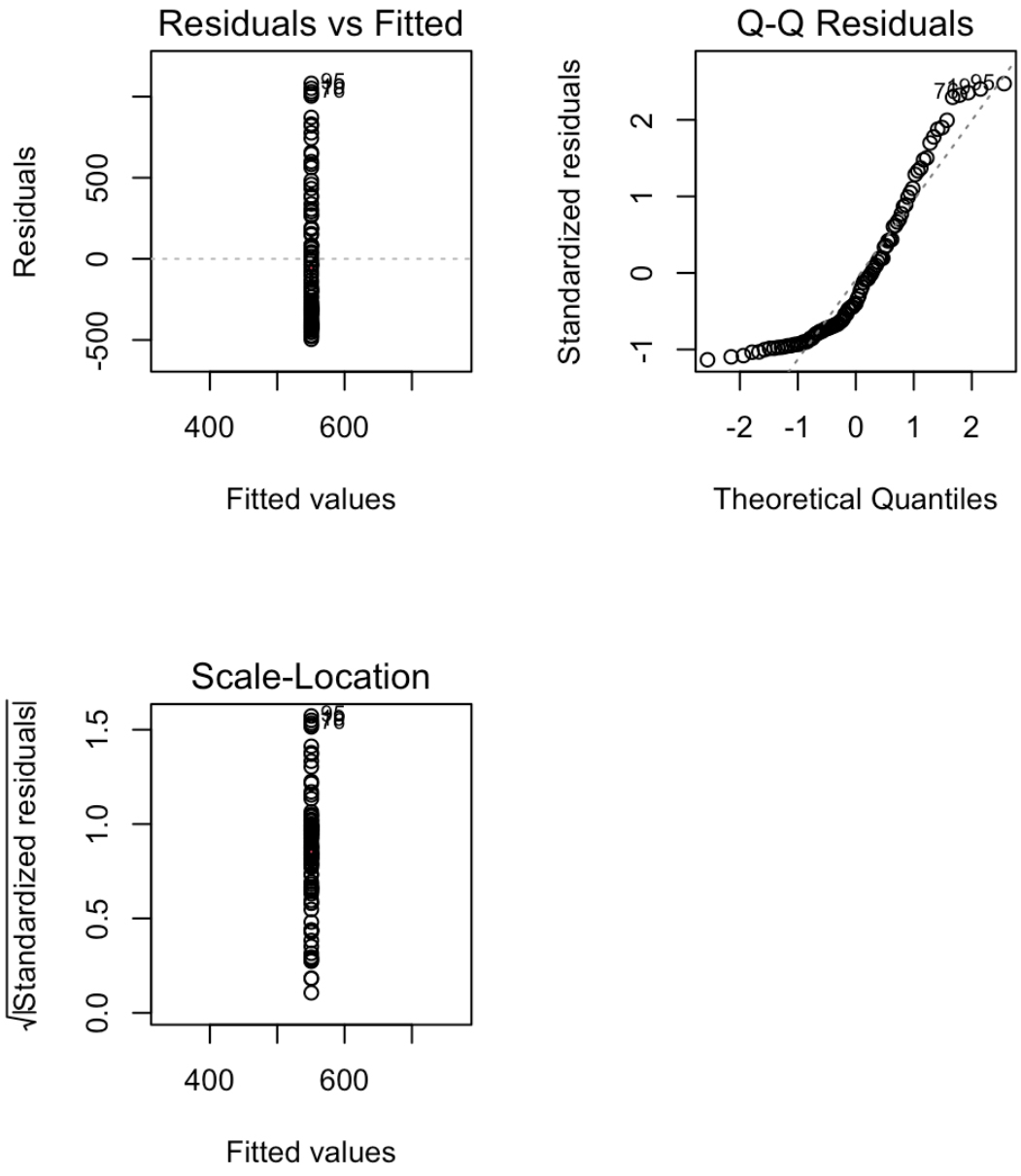

Figura 18.4: Grafici diagnostici per lm0.

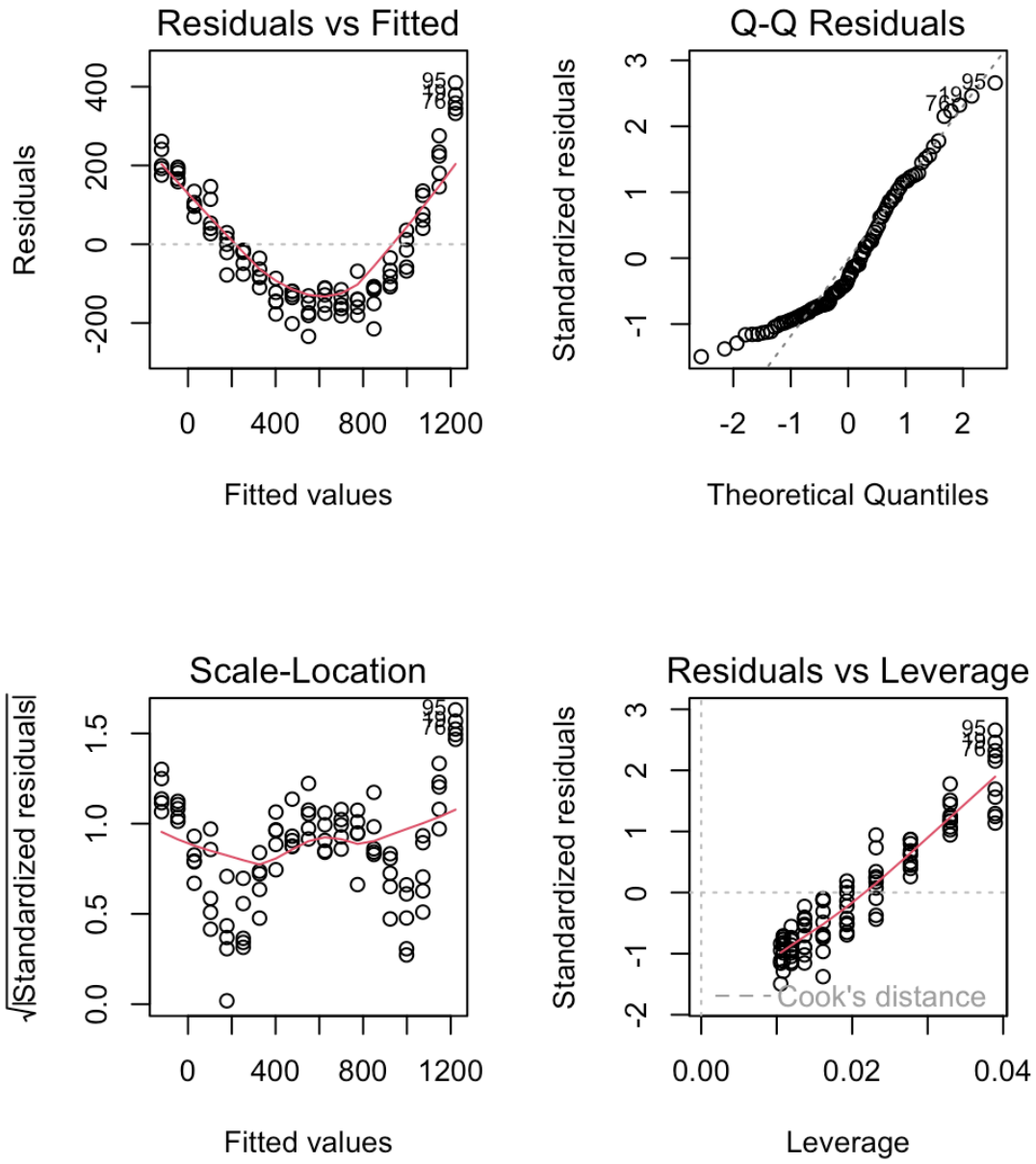

Figura 18.5: Grafici diagnostici per lm1.

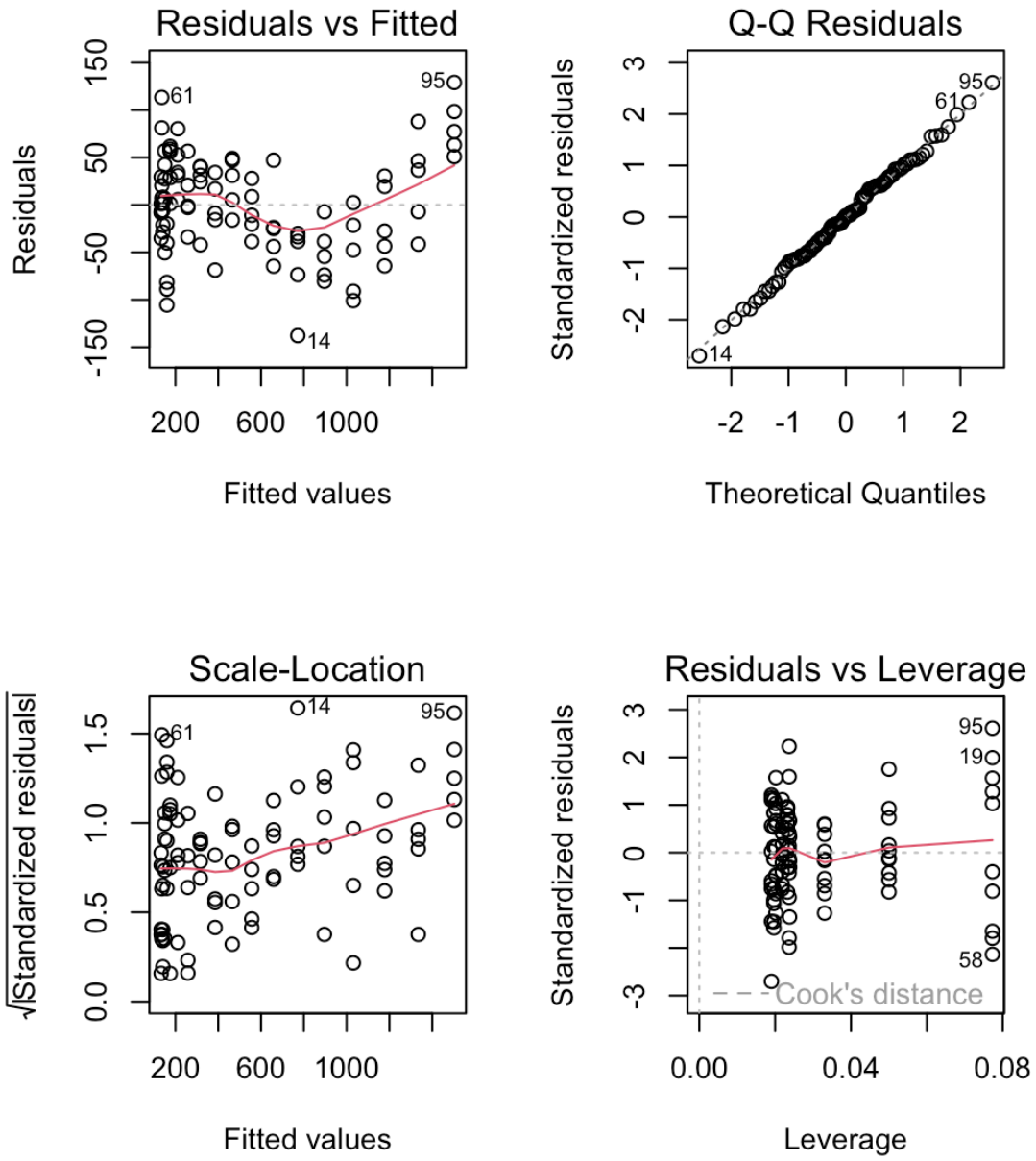

Figura 18.6: Grafici diagnostici per `lm.poly`.

18.2 Esempio 2

Questa sezione è stata rimossa dall'edizione gratuita di questo manuale. Potete acquistare l'edizione senza tagli qui:

- https://www.lulu.com/it/shop/marco-plebani/analisi-dei-dati-con-r-ed-rstudio/ebook/product-yvkp2ng.html (e-book)
- https://www.lulu.com/it/shop/marco-plebani/analisi-dei-dati-con-r-ed-rstudio/paperback/product-je6k8y6.html (in formato cartaceo)

Buona lettura!

- l'autore

19 Violazioni delle assunzioni sui residui

Tutti i modelli che abbiamo utilizzato finora si basano sulle assunzioni di normalità ed omoscedasticità dei residui, ovvero prevedono che i residui del modello siano distribuiti attorno ai valori stimati secondo una distribuzione normale (normalità), e che la varianza dei residui attorno ai valori stimati sia costante per tutti i valori stimati (omoscedasticità). Finora abbiamo avuto la fortuna di avere a che fare con dati che, una volta stimati dai modelli, non producessero violazioni delle assunzioni dei modelli stessi. Spesso però i residui di un modello non seguono una distribuzione normale, oppure non rispettano l'assunzione di omoscedasticità. Spesso queste violazioni delle assunzioni dei modelli lineari sono associate a variabili di risposta che, per loro natura, non hanno relazioni lineari con eventuali predittori. Un approccio diffuso è stato, per lungo tempo, quello di trasformare la variabile di risposta in modo da linearizzare la sua relazione con i predittori. Questo approccio, però, non risolve sempre il problema della non-normalità e/o dell'eteroscedasticità dei residui del modello. Un approccio più moderno consiste nel ricorrere ai **modelli lineari generalizzati,** che sono l'oggetto di questo capitolo.

19.1 La funzione `glm()` per modelli lineari generalizzati

I modelli lineari generalizzati possono essere definiti in R tramite la funzione `glm()`. La funzione `glm()` permette di specificare la forma della distribuzione dei residui tramite l'argomento `family=`. La funzione `glm()` permette inoltre di indicare una *link function*, o funzione di collegamento, che istruisce `glm()` sul tipo di trasformazione da operare per linearizzare le predizioni del modello e normalizzarne i corrispondenti intervalli di confidenza.

19.2 Esempio 1: dati di conteggio come variabile di risposta

Supponiamo di voler studiare come varia il numero di uova deposte dalle femmine riproduttive di una specie di pesce al variare della loro massa corporea (fig. 19.1). Carichiamo i dati contenuti nel file *"count_data.csv"*, che trovate a https://github.com/marcoplebani85/datasets/:

```
rm(list = ls())  # ripulisce la memoria di R
options(scipen = 9999)  # evita di esprimere numeri come potenze di 10

# carica il pacchetto 'pacman', che contiene la funzione p_load():
if (!require(pacman)) install.packages("pacman")
library(pacman)

# altri pacchetti:
p_load(emmeans)
p_load(ggplot2)
p_load(DHARMa)
# DHARMa produce test e grafici diagnostici per GLM e mixed-effect models
p_load(performance)
# le funzioni di performance permettono di calcolare vari indici di qualità d
ei
# modelli
```

```
# definiamo la directory di lavoro. In R:
setwd("~/Desktop/Corso_R")
# In RSTUDIO: menu -> Session -> Set working directory

# Carichiamo i dati in R:
df <- read.csv("my_data/count_data.csv")
str(df)
```

```
## 'data.frame':    124 obs. of  2 variables:
##  $ body.mass: num  5 5.5 6 6.5 7 7.5 8 8.5 9 9.5 ...
##  $ n.progeny: int  3 0 5 1 3 12 4 7 14 7 ...
```

Il predittore `body.mass` è una variabile continua, mentre la variabile di risposta, `n.progeny`, consiste in dati di conteggio. Visualizziamo i dati (fig. 19.1) con il seguente codice:

```
par(mfrow=c(1,2),
    # tipo di "scatola" da tracciare attorno ai grafici:
    bty="L",
    family="Times" # font type
    )
plot(n.progeny ~ body.mass, data=df,
     # tipo di punto, colore, dimensione:
     pch=19, col=grey(0.5, alpha=0.5), cex=1.5,
     # etichette assi:
     ylab="Numero di uova deposte",
     xlab="Massa corporea [g]"
     )
mtext(expression(paste("(", bold(a),")")),
      side = 3, line = 0.3, adj = 0, family="Times")
plot(log(n.progeny + 1) ~ body.mass, data=df,
     pch=19, col=grey(0.5, alpha=0.5), cex=1.5,
     ylab="log(Numero di uova deposte + 1)",
     xlab="Massa corporea [g]"
     )
mtext(expression(paste("(", bold(b),")")),
      side = 3, line = 0.3, adj = 0, family="Times")
```

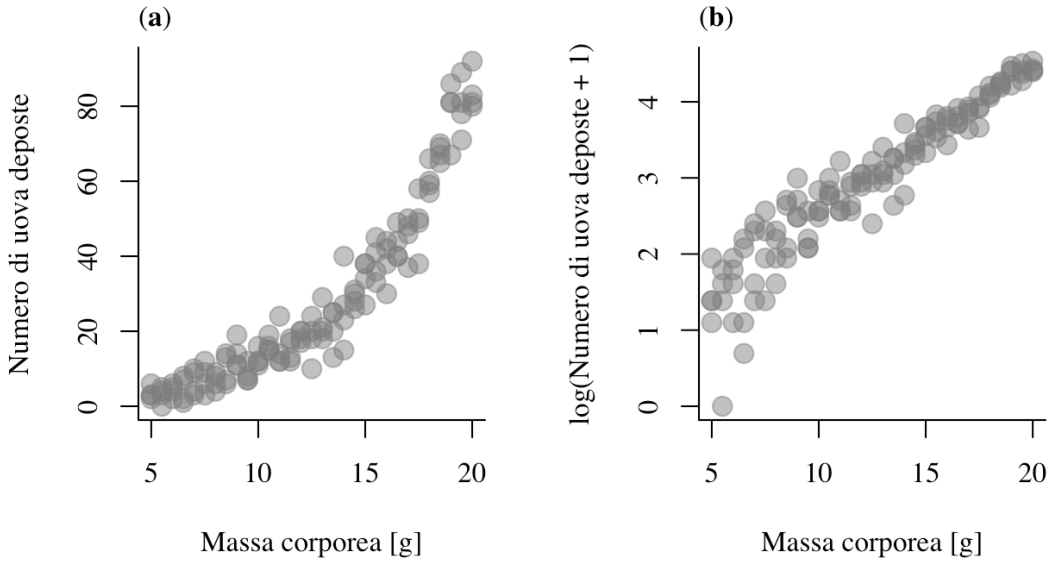

Figura 19.1: Numero di uova deposte in relazione alla massa corporea della madre (a), e gli stessi dati rappresentati usando il logaritmo del numero di uova deposte + 1 (b).

Un modello di regressione lineare è evidentemente inadeguato a descrivere la relazione tra `n.progeny` e `body.mass`, che sembra essere esponenziale (fig. 19.1a).

La soluzione tradizionale per analizzare questo tipo di dati sarebbe quello di log-trasformare i dati (fig. 19.1b), per poi stimare i parametri del modello lineare:

$$log(\widehat{\text{uova deposte}}_i) = a + b \cdot \text{massa corporea}_i \qquad (19.1)$$

Se proviamo a svolgere questa operazione in R, però, emerge subito un problema:

```
lm.log <- lm(log(n.progeny) ~ body.mass, data = df)

## Error in `lm.fit()`:
## ! NA/NaN/Inf in 'y'
```

Per valori di `n.progeny` pari a 0 (cioè: zero uova deposte) il corrispondente logaritmo naturale equivale a *-infinito*. In questi casi la "soluzione" tradizionale è quella di aumentare la variabile di risposta di un valore arbitrariamente piccolo (di solito una unità) prima di log-trasformarla. Facciamolo:

```
lm.log <- lm(log(n.progeny + 1) ~ body.mass, data = df)
# grafici diagnostici:
par(mfrow = c(2, 2))
plot(lm.log)
```

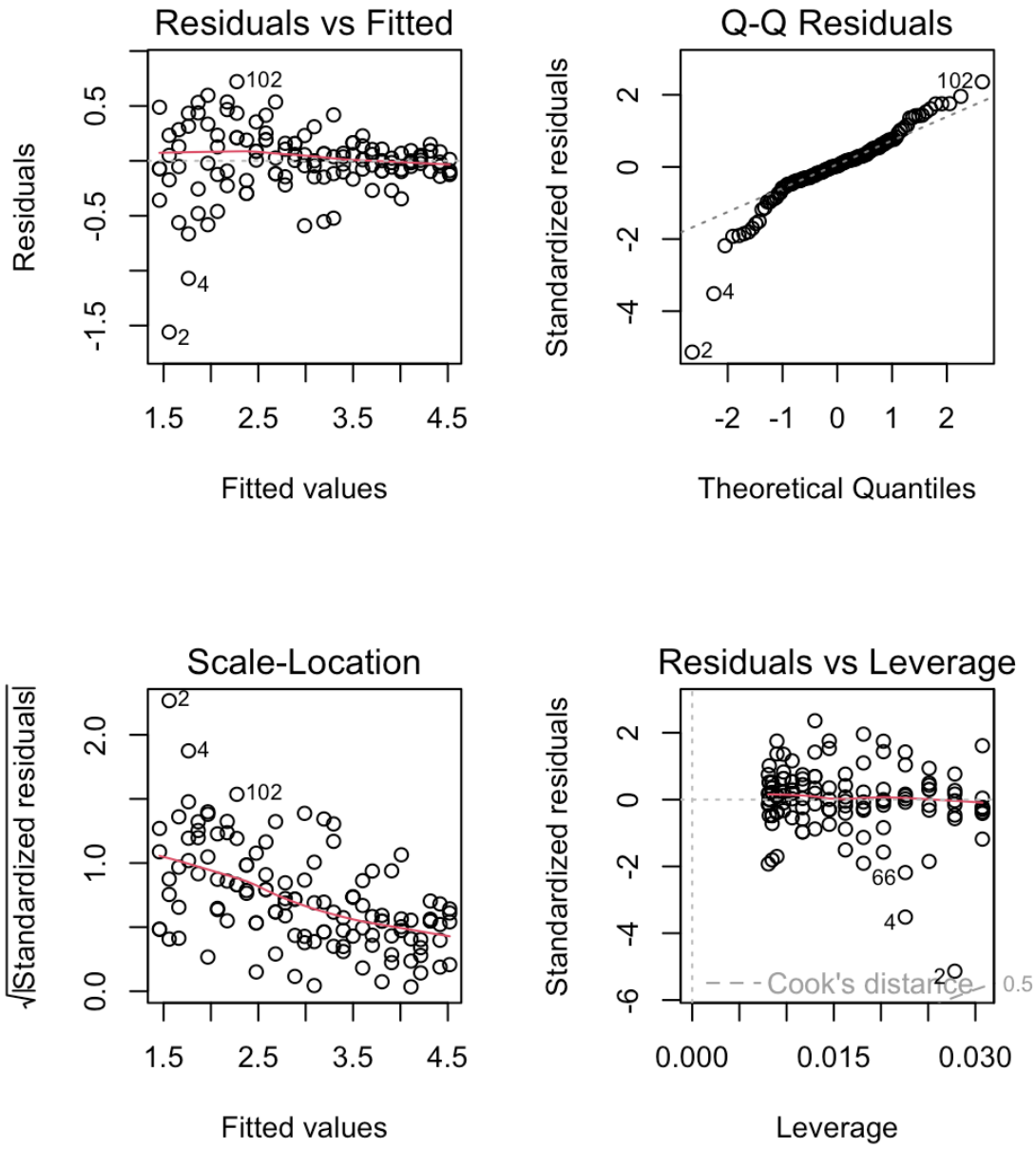

Figura 19.2: Grafici diagnostici per lm.log.

Esaminando i grafici diagnostici di `lm.log` (fig. 19.2) emerge un altro limite di questo approccio: anche se la relazione tra `log(n.progeny)` e `body.mass` è lineare, log-trasformare la variabile di risposta non risolve i problemi di non-normalità ed eteroscedasticità dei residui del modello.

Un altro approccio potrebbe essere quello di descrivere i dati non trasformati con il seguente modello esponenziale usando la funzione `nls()`:

$$\text{uova } \widehat{\text{deposte}}_i = e^{a+b\cdot\text{massa corporea}_i} \qquad (19.2)$$

Questo approccio descrive bene la relazione esponenziale tra numero di uova deposte e massa corporea della madre, ma non risolve i problemi di non-normalità ed eteroscedasticità dei residui del modello. La funzione `nls()` assume che i residui del modello siano omoscedastici e distribuiti secondo una distribuzione normale, ma questa assunzione non è rispettata dai residui del modello (vi invito a verificarlo usando la funzione `nlsResiduals()` dal pacchetto `nlstools` introdotta in sez. 18.1).

Ci serve uno strumento che permetta di approssimare le relazioni non lineari più comuni, ma che permetta anche di tenere conto della non-normalità e/o eteroscedasticità dei residui attorno alle stime medie del modello. Questo strumento è rappresentato dai modelli lineari generalizzati, comunemente implementati in R tramite la funzione `glm()`:

```
glm1 <- glm(n.progeny ~ body.mass, data=df,
            family=poisson(link="log"))
```

Esaminiamo la sintassi usata da `glm()`. I primi due argomenti della funzione `glm()` ci sono già familiari in quanto identici a quelli usati nella funzione `lm()`: con la formula `n.progeny ~ body.mass` definiamo un modello secondo il quale `n.progeny` varia in funzione di `body.mass`; l'argomento `data=` permette di indicare i dati sulla base dei quali vogliamo stimare i parametri del modello. La grande novità della funzione `glm()` è l'argomento `family=`, che ci permette di specificare:

- il tipo di trasformazione da operare *sulle predizioni del modello* (non sui dati) per linearizzarle;
- il tipo di distribuzione da usare per approssimare la forma della distribuzione dei residui.

Potete ottenere la lista delle distribuzioni disponibili per `glm()` digitando `?family` nel terminale. Nel nostro caso, mi aspetto che una distribuzione Poissoniana sia una buona approssimazione dei residui del modello, dato che la variabile di risposta è rappresentata da dati di conteggio e la variabilità dei valori osservati sembra aumentare all'aumentare del loro valor medio (fig. 19.2). Tutte le funzioni che si possono fornire all'argomento `family=` includono l'argomento `link=`, che permette di indicare una *link function* o "funzione di collegamento". La funzione di collegamento istruisce `glm()` sul tipo di trasformazione da operare per linearizzare le predizioni del modello. In questo caso ho scelto una trasformazione logaritmica, adatta a linearizzare la relazione esponenziale tra `n.progeny` e `body.mass`.

In breve: il modello `glm1` descrive i dati con il modello descritto in Eq. (19.2), ma assume che i residui del modello seguano una distribuzione poissoniana anziche gaussiana.

19.2.1 Grafici diagnostici col pacchetto `DHARMa`

Verifichiamo ora che le assunzioni del modello lineare generalizzato `glm1` siano verificate. I grafici diagnostici che abbiamo usato finora per i modelli lineari sono orientati a verificare che la distribuzione attesa dei residui sia di tipo gaussiano ed omoscedastico, dunque non si prestano all'uso con i modelli lineari generalizzati. Nel caso dei modelli lineari generalizzati *sappiamo già* che i loro residui sono non-normali ed eteroscedastici. L'assunzione da verificare è dunque che la forma della distribuzione osservata nei residui del modello rispecchi quella indicata in `family=`. Nel caso dei modelli lineari generalizzati possiamo ottenere grafici diagnostici usando il pacchetto `DHARMa` (Hartig 2022). Le funzioni del pacchetto `DHARMa` permettono di confrontare

la distribuzione dei residui osservati con quelli prodotti da simulazioni secondo la distribuzione specificata nel modello stesso:

- la funzione `simulateResiduals()` produce, tramite simulazioni numeriche, i residui attesi per un dato modello;
- per ottenere un grafico quantile-quantile possiamo usare la funzione `plotQQunif()`, che confronta i residui osservati con quelli attesi in base alle simulazioni numeriche (fig. 19.3a);
- per visualizzare l'andamento dei residui prodotti dalle simulazioni rispetto ai valori medi stimati dal modello possiamo usare la funzione `plotResiduals()`. Questo grafico mostra tre linee corrispondenti al primo, secondo, e terzo quartile dei residui; queste linee dovrebbero essere rette e tutte all'incirca parallele all'asse delle ascisse (fig. 19.3b).

Produciamo dunque i grafici diagnostici per `glm1`:

```
# simulazione dei residui attesi:
glm1Resids <- simulateResiduals(glm1)
par(mfrow = c(1, 2))
# confronto tra residui osservati e quelli attesi secondo una distribuzione
# poissoniana:
plotQQunif(glm1Resids)
mtext(text = "(a)", side = 3, adj = 0, line = 3)
# andamento dei residui simulati rispetto ai valori medi stimati dal modello:
plotResiduals(glm1Resids, quantreg = T)
mtext(text = "(b)", side = 3, adj = 0, line = 3)
```

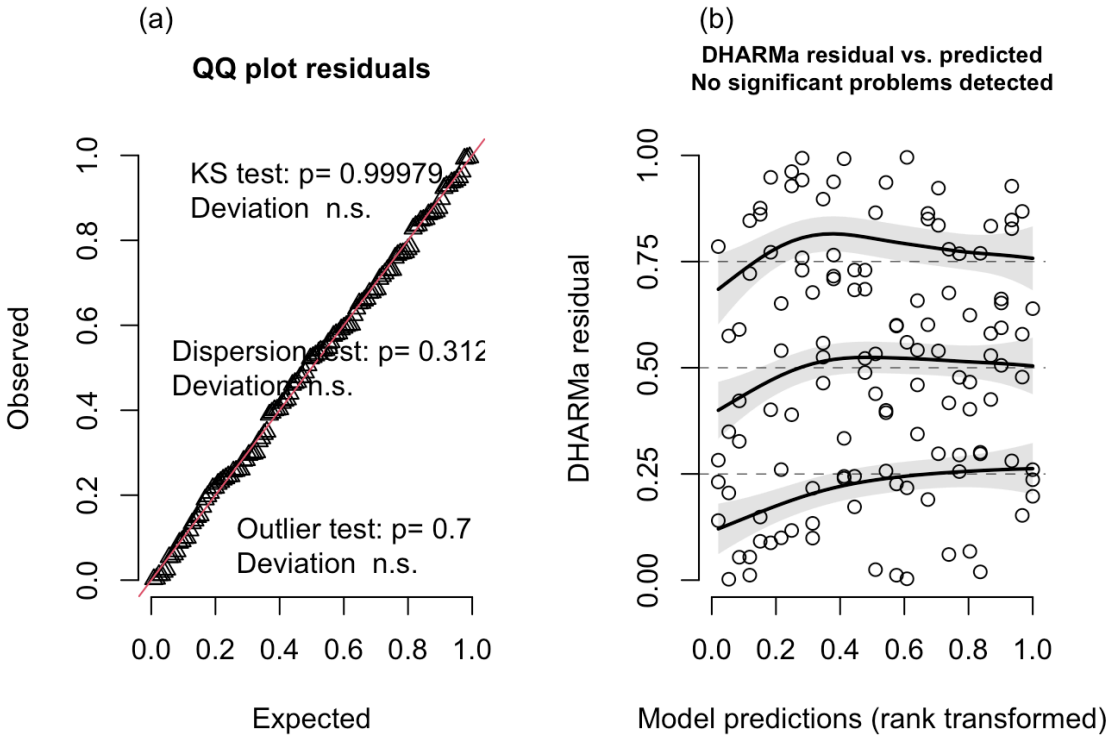

Figura 19.3: Grafici diagnostici per `glm1`.

Il *Q-Q plot* prodotto da `DHARMa` fornisce anche i risultati del test di Kolmogorov–Smirnov (KS) sulla deviazione dei residui dalla distribuzione attesa, un test sulla dispersione dei residui (che approfondiremo nella sezione 19.2.2), ed un test sulla presenza di *outliers* (prodotto dalla funzione `testOutliers()`) (fig. 19.3a). Nel caso in cui il test sulla presenza di *outliers* fosse significativo, l'elenco dei potenziali outliers può essere ottenuto con la funzione `outliers()`.

I grafici diagnostici per `glm1` indicano che il modello cattura la non-linearità della relazione tra `n.progeny` e `body.mass`, e che la distribuzione scelta (quella poissoniana) è adeguata a descrivere la distribuzione dei residui del modello (fig. 19.3).

19.2.2 Controllo della dispersione dei residui

Nel caso dei modelli lineari generalizzati bisogna anche verificare che la relazione tra media e varianza coincida con quella attesa secondo la distribuzione specificata. Nel caso di una distribuzione poissoniana dei residui media e varianza dovrebbero coincidere, nel qual caso la *dispersione* dei residui dovrebbe essere all'incirca 1. Il pacchetto `DHARMa` include un test non-parametrico della dispersione dei residui tramite la funzione `testDispersion()`, il cui grado di significatività è incluso di default nel grafico quantile-quantile prodotto da `plotQQunif()`. Nel caso di `glm1` il test non-parametrico prodotto da`DHARMa` non indica una dispersione dei residui significativamente diversa da quella attesa per una distribuzione poissoniana (fig. 19.3a). Se ci fosse sovra-dispersione *(overdispersion)* le stime dei parametri del modello avrebbero un grado di incertezza apparentemente minore di quanto i dati in realtà non consentano. Al contrario, un modello affetto da sotto-dispersione *(underdispersion)* porterebbe a sovrastimare l'incertezza dei parametri, rendendo di conseguenza i risultati dei test statistici sul modello più conservativi. Nel

nostro esempio, la dispersione dei residui non differisce significativamente da quella attesa secondo una distribuzione poissoniana. Se avessimo rilevato una significativa sovra- o sotto-dispersione dei residui, una possibile soluzione sarebbe stato ri-fittare il modello usando `family="quasipoisson"` anzichè `family="poisson"`. I modelli che assumono una distribuzione quasi-poissoniana dei dati assumono che il rapporto tra la loro media e varianza sia esattamente quello osservato, anzichè essere pari ad 1 come per le distribuzioni poissoniane vere e proprie.

19.2.3 Confronto tra modelli

Per valutare se `glm1` differisce significativamente dal modello nullo basta modificare lievemente il metodo di confronto dei modelli che abbiamo già usato per i modelli lineari:

```
# modello nullo:
glm0 <- glm(n.progeny ~ 1, data = df, family = poisson(link = "log"))

# confronto tra la variabilità residua di glm1 e glm0:
anova(glm1, glm0, test = "Chisq")

## Analysis of Deviance Table
##
## Model 1: n.progeny ~ body.mass
## Model 2: n.progeny ~ 1
##   Resid. Df Resid. Dev Df Deviance              Pr(>Chi)
## 1       122     134.19
## 2       123    2459.97 -1  -2325.8 < 0.00000000000000022 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Nel caso dei *glm*, `anova()` svolge una *Analysis of Deviance* invece della *Analysis of Variance* a noi familiare. Si tratta di strumenti statistici differenti per raggiungere lo stesso obiettivo: operare un confronto statistico tra la variabilità residua dei modelli di interesse. Nel caso di modelli con dispersione dei residui nota, quali `family=poisson` e `family=binomial`, il test raccomandato per confrontare la variabilità residua dei modelli è un Chi-quadro (vedere la sezione *Details* di `?anova.glm`).

Il modello `glm1` ha una variabilità residua significativamente minore di quella del modello nullo `glm0`. Possiamo dunque rigettare il modello nullo.

Esaminiamo `glm1` con `summary()`:

```
summary(glm1)

##
## Call:
## glm(formula = n.progeny ~ body.mass, family = poisson(link = "log"),
##     data = df)
##
## Coefficients:
##             Estimate Std. Error z value             Pr(>|z|)
```

```
## (Intercept) 0.436927   0.076656    5.70          0.000000012 ***
## body.mass   0.202433   0.004671   43.33 < 0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 2459.97  on 123  degrees of freedom
## Residual deviance:  134.19  on 122  degrees of freedom
## AIC: 726.37
##
## Number of Fisher Scoring iterations: 4
```

I parametri stimati per il modello sono ~0.44 e ~0.20. Il modello è dunque:

$$\widehat{\text{uova deposte}}_i = e^{0.44+0.20 \cdot \text{massa corporea}_i} \qquad (19.3)$$

Ovvero, nella sua forma linearizzata:

$$log(\widehat{\text{uova deposte}}_i) = 0.44 + 0.20 \cdot \text{massa corporea}_i \qquad (19.4)$$

In questo caso `summary()` non fornisce una stima di R^2. Il motivo è che l'R^2 è basato sulla stima dei parametri di un modello tramite il metodo dei minimi quadrati, mentre la funzione `glm()` usa un approccio chiamato *"maximum likelihood method"* (lo vedremo più in dettaglio in sez. 20.3.1). È comunque possibile stimare la frazione di variabilità spiegata dal modello tramite vari indici di pseudo-R^2. Il pacchetto `performance` (Lüdecke et al., 2021) raccoglie funzioni per calcolare vari indici di qualità dei modelli, inclusi vari indici di pseudo-R^2. Calcoliamo lo pseudo-R^2 di McFadden (1973) con la funzione `r2_mcfadden()`:

```
performance::r2_mcfadden(glm1)
```

```
## # R2 for Generalized Linear Regression
##        R2: 0.763
##   adj. R2: 0.762
```

Secondo questa stima, il modello `glm1` spiega il 76.2% della variabilità osservata nel numero di uova deposte (`n.progeny`).

19.2.4 Predizioni del modello usando `emmeans`

Possiamo usare `emmeans()` per ottenere sia le predizioni medie del modello, sia gli intervalli di confidenza ad esse associate direttamente sulla scala originale della variabile di risposta:

```
emmeans(glm1,
        specs = ~body.mass,
        at=list(body.mass=c(5,10,15,20)),
        type="response")
```

```
##  body.mass  rate    SE  df asymp.LCL asymp.UCL
##          5  4.26 0.230 Inf      3.83      4.74
##         10 11.72 0.384 Inf     10.99     12.50
```

```
##         15 32.25 0.564 Inf     31.16     33.37
##         20 88.73 2.230 Inf     84.47     93.21
##
## Confidence level used: 0.95
## Intervals are back-transformed from the log scale
```

L'argomento `type=` permette di specificare se vogliamo predizioni e intervalli di confidenza calcolati sulla scala originale della variabile di risposta oppure su quella trasformata (logaritmica).

Il prodotto di `emmeans()` comprende le previsioni del modello (`rate`) ed i limiti inferiore (`asymp.LCL`) e superiore (`asymp.UCL`) dell'intervallo di confidenza al 95% attorno alle predizioni del modello.

19.2.5 Rappresentazione grafica usando `ggplot()`

Rappresentiamo i dati con `ggplot()` e le predizioni del modello con `geom_smooth()` (fig. 19.4):

```
ggplot(df, aes(body.mass, n.progeny)) +
  geom_point(shape=19, color=grey(level=0.5, alpha=0.5), size=3) +
  geom_smooth(method = "glm",
              method.args=list(family = "poisson"),
              se=T, # int. conf. al 95%
              color="black") +
  labs(x = "Massa corporea [g]", y = "Numero di uova deposte") +
  theme_classic() +
  theme(text = element_text(family="Times", size=13))
```

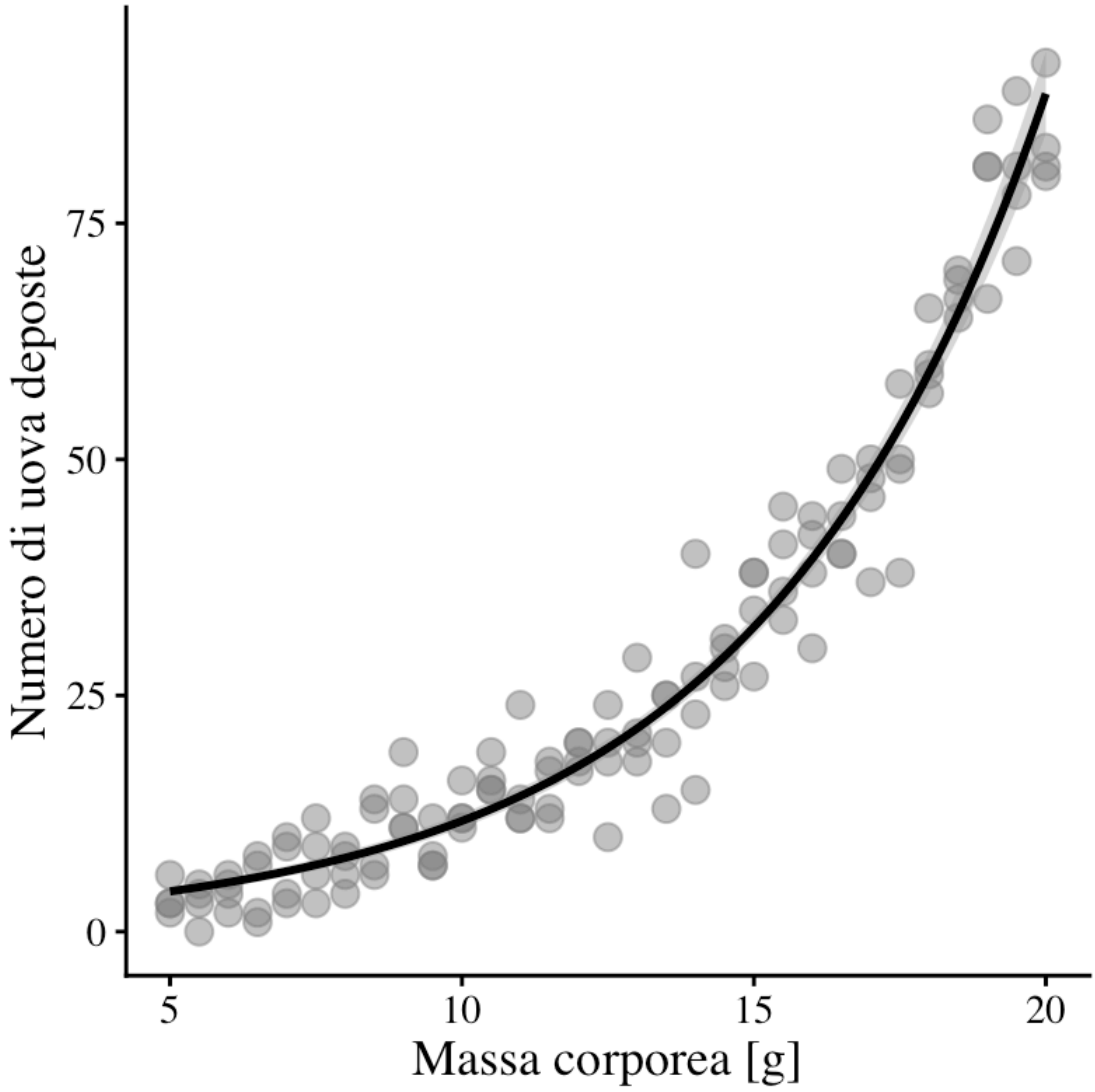

Figura 19.4: Valori osservati e valori attesi secondo il modello glm1.

Nel caso dei modelli lineari generalizzati gli argomenti quali `family =`, ecc. vanno indicati in `geom_smooth()` usando `method.args=list(...)`.

19.2.6 Comunicazione

Questa sezione è stata rimossa dall'edizione gratuita di questo manuale. Potete acquistare l'edizione senza tagli qui:

- https://www.lulu.com/it/shop/marco-plebani/analisi-dei-dati-con-r-ed-rstudio/ebook/product-yvkp2ng.html (e-book)
- https://www.lulu.com/it/shop/marco-plebani/analisi-dei-dati-con-r-ed-rstudio/paperback/product-je6k8y6.html (in formato cartaceo)

Buona lettura!

- l'autore

19.3 Esempio 2: un altro esempio con dati di conteggio

Questa sezione è stata rimossa dall'edizione gratuita di questo manuale. Potete acquistare l'edizione senza tagli qui:

- https://www.lulu.com/it/shop/marco-plebani/analisi-dei-dati-con-r-ed-rstudio/ebook/product-yvkp2ng.html (e-book)
- https://www.lulu.com/it/shop/marco-plebani/analisi-dei-dati-con-r-ed-rstudio/paperback/product-je6k8y6.html (in formato cartaceo)

Buona lettura!

- l'autore

19.4 Esempio 3: …Ancora un esempio con dati di conteggio

Questa sezione è stata rimossa dall'edizione gratuita di questo manuale. Potete acquistare l'edizione senza tagli qui:

- https://www.lulu.com/it/shop/marco-plebani/analisi-dei-dati-con-r-ed-rstudio/ebook/product-yvkp2ng.html (e-book)
- https://www.lulu.com/it/shop/marco-plebani/analisi-dei-dati-con-r-ed-rstudio/paperback/product-je6k8y6.html (in formato cartaceo)

Buona lettura!

- l'autore

19.5 Esempio 4: dati binomiali (e.g. presenza/assenza) come variabile di risposta

Le variabili di risposta binomiali consistono nel verificarsi o meno di un evento, ad esempio: vivo/morto, presenza/assenza, successo/fallimento. Se chiamiamo π la probabilità attesa del verificarsi di un evento di interesse, il valore di π può assumere solo valori compresi tra 0 ed 1. Per riflettere ciò, la relazione tra π e ciascuna combinazione dei predittori viene di solito descritta con modelli logistici. La probabilità media stimata $\hat{\pi}$ associata a ciascun valore di un generico predittore continuo X può essere dunque calcolata con l'equazione logistica:

$$\hat{\pi} = \frac{e^{a+X \cdot b}}{1 + e^{a+X \cdot b}} \qquad (19.6)$$

Nella sua forma linearizzata, l'eq. (19.6) corrisponde a:

$$\log\left(\frac{\hat{\pi}}{1-\hat{\pi}}\right) = a + X \cdot b \qquad (19.7)$$

Il rapporto $\frac{\pi}{1-\pi}$ è chiamato *odds ratio*, e il logaritmo della *odds ratio* è detto $logit(\pi)$.

Per modellizzare correttamente questo tipo di variabili bisogna anche descrivere correttamente la distribuzione dei residui di una variabile binomiale attorno alle predizioni di un modello logistico. A questo scopo, una distribuzione binomiale è un buon punto di partenza.

Vediamo ora come implementare un modello logistico con distribuzione binomiale dei residui nella funzione `glm()`. Supponiamo di avere raccolto dati sulla sopravvivenza di una specie di albero esposta a vari livelli di stress idrico, ovvero carenza d'acqua. I dati sono salvati nel file *stress_death.csv* (disponibile a https://github.com/marcoplebani85/datasets/). Vediamoli:

```
rm(list = ls())  # ripulisce la memoria di R
# carica il pacchetto 'pacman', che contiene la funzione p_load():
if (!require(pacman)) install.packages("pacman")
library(pacman)
# altri pacchetti necessari:
p_load(emmeans)
p_load(ggplot2)
p_load(DHARMa)
p_load(performance)
# definiamo la directory di lavoro:
setwd("~/Desktop/Corso_R")
# Carichiamo i dati in R:
df <- read.csv("my_data/stress_death.csv")
str(df)

## 'data.frame':    90 obs. of  2 variables:
##  $ stress.per100: int  0 20 40 50 70 90 0 20 40 50 ...
##  $ death.01     : int  0 0 0 1 1 1 1 0 1 1 ...
```

Il predittore `stress.per100` consiste di sei livelli di stress idrico espressi in percentuale. Per ciascun livello del predittore sono stati studiati 15 alberi, la cui sopravvivenza nel periodo dello

studio è registrata nella colonna `death.01`: le morti sono registrate come 1, le sopravvivenze come 0. Rappresentiamo i dati:

```
par(bty="L", family="Times")
plot(jitter(death.01, 0.1) ~ jitter(stress.per100),
    data= df,
    pch=21, cex=2, # tipo di punto da usare
    bg=grey(0.7, alpha=0.5), # colore
    xlab="Stress idrico [%]",
    ylab="Mortalità",
    xlim=c(0,100)
    )
```

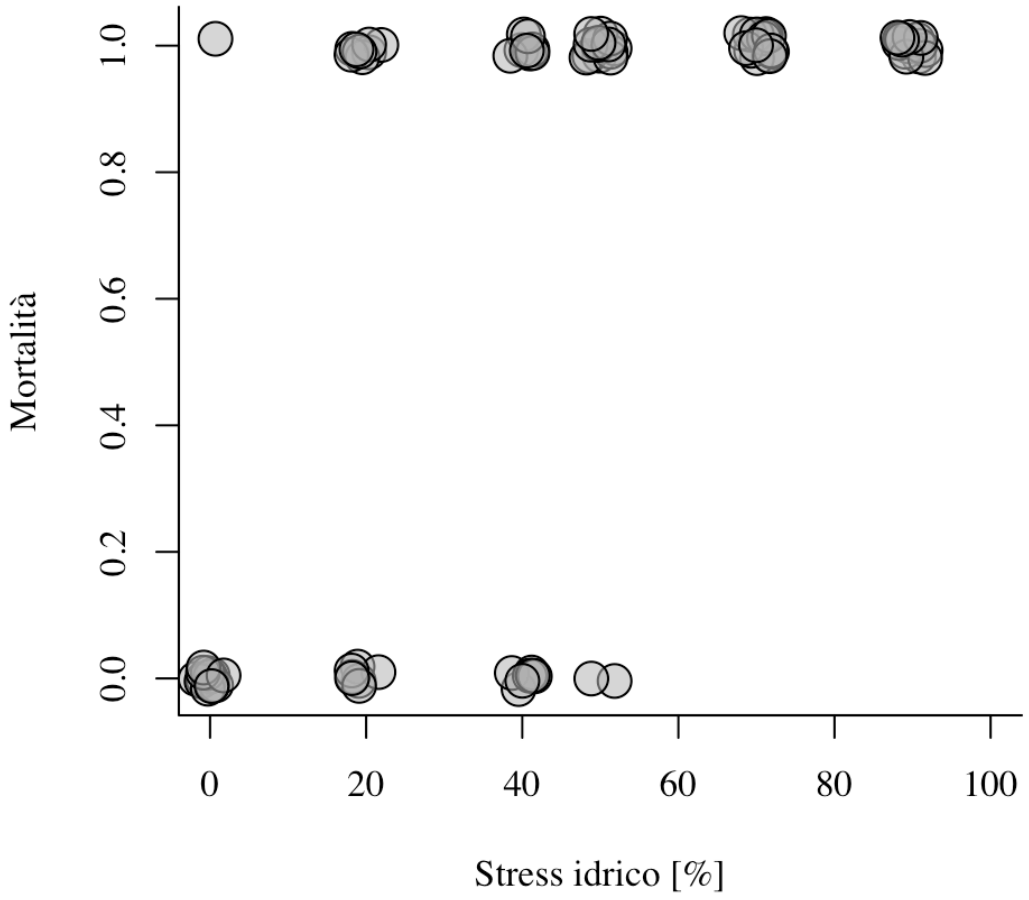

Figura 19.11: Mortalità degli alberi in relazione allo stress idrico (0: vivo; 1: morto). Un pizzico di rumore artificiale è stato aggiunto alle coordinate delle misurazioni per evitare che si sovrapponessero completamente.

Ho usato la funzione `jitter()` ("tremarella") in fig. 19.11 per introdurre un pizzico di rumore artificiale lungo entrambi gli assi, in modo da evitare che le osservazioni si sovrapponessero completamente. La funzione `grey()` permette di colorare i punti in grigio definendone il grado di scurezza e trasparenza. La figura 19.11 suggerisce che la probabilità di morte di un albero aumenti all'aumentare dello stress idrico. Usiamo `glm()` per implementare una regressione logistica con distribuzione binomiale dei residui basata sui dati contenuti in `df`:

```
glm.binom1 <- glm(death.01 ~ stress.per100,
                  data= df,
                  family=binomial(link="logit")
                  )
```

Come descritto all'inizio di questa sezione, la funzione di collegamento *"logit"* è usata per linearizzare le predizioni dell'equazione logistica.

Otteniamo i grafici diagnostici per `glm.binom1`:

```
glm.binom1Resids <- simulateResiduals(glm.binom1)
par(mfrow = c(1, 2))
plotQQunif(glm.binom1Resids)
mtext(text = "(a)", side = 3, adj = 0, line = 3)
plotResiduals(glm.binom1Resids, quantreg = T)
mtext(text = "(b)", side = 3, adj = 0, line = 3)
```

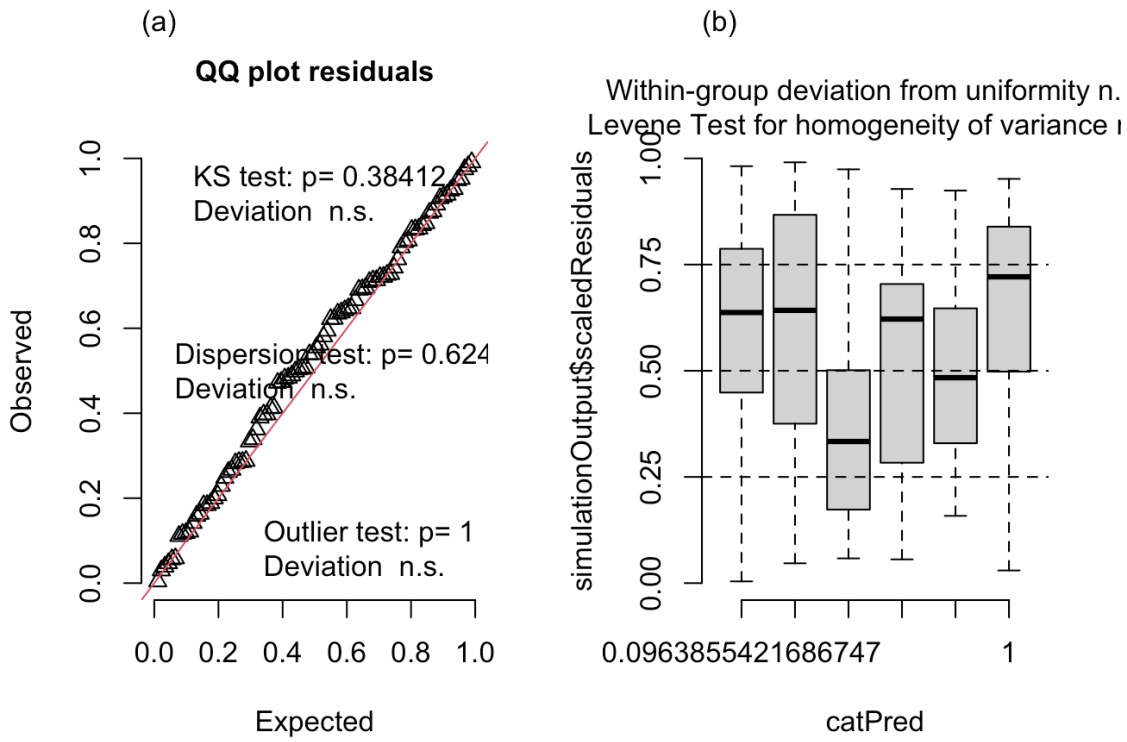

Figura 19.12: Grafici diagnostici per glm.binom1.

I grafici diagnostici di `glm.binom1` non indicano violazioni delle assunzioni (fig. 19.12). Similmente ai modelli che assumono una distribuzione poissoniana dei residui, anche i modelli per dati binomiali assumono che ci sia un rapporto costante tra la media μ e varianza σ^2 dei residui: nel caso della distribuzione binomiale, $\sigma^2 = \mu \cdot (1 - \pi)$. Come abbiamo già visto nelle sezioni precedenti (sezione 19.2.2) è possibile verificare che il rapporto atteso tra media e varianza sia rispettato usando il test di dispersione fornito da `DHARMa`. Nel nostro caso il test non-parametrico di dispersione dei residui indica che essa non differisce significativamente da quella attesa (fig. 19.12a). Se avessimo riscontrato sovradispersione dei residui avremmo potuto adeguare il nostro modello, ad esempio usando `family=quasibinomial`.

Dal confronto tra `glm.binom1` ed il corrispondente modello nullo vediamo che `glm.binom1` è quello con la variabilità residua significativamente inferiore:

```
glm.binom0 <- glm(death.01 ~ 1, data= df,
    family=binomial(link="logit")
    )

anova(glm.binom1, glm.binom0, test="Chisq")
```

```
## Analysis of Deviance Table
##
## Model 1: death.01 ~ stress.per100
## Model 2: death.01 ~ 1
##   Resid. Df Resid. Dev Df Deviance          Pr(>Chi)
## 1        88     66.602
## 2        89    114.573 -1  -47.971 0.000000000004326 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Possiamo dunque rigettare il modello nullo.

Calcoliamo lo pseudo-R^2 per `glm.binom1`:

```
performance::r2(glm.binom1)
```

```
## # R2 for Logistic Regression
##   Tjur's R2: 0.458
```

Il modello spiega il 45.8% della variabilità osservata nella mortalità degli alberi sotto esame.

19.5.1 Predizioni del modello

I parametri stimati per il modello `glm.binom1 sono:

```
coef(glm.binom1)
```

```
##   (Intercept) stress.per100
##   -2.09413763    0.07645954
```

La mortalità media associata a ciascun valore di stress idrico può essere dunque stimata con l'equazione logistica:

$$\widehat{mortalita} = \frac{e^{-2.1+Stress \cdot 0.08}}{1 + e^{-2.1+Stress \cdot 0.08}} \qquad (19.8)$$

Otteniamo le predizioni del modello con `emmeans`:

```
pred.df <- emmeans(glm.binom1, # modello
    specs = ~stress.per100, # predittori da considerare
    # valori del predittore per i quali ottenere predizioni:
    at=list(stress.per100=0:100),
    type="response" # scala delle predizioni
    )
head(pred.df)
```

```
##  stress.per100  prob     SE  df asymp.LCL asymp.UCL
##              0 0.110 0.0576 Inf    0.0373     0.281
##              1 0.117 0.0597 Inf    0.0412     0.291
##              2 0.126 0.0618 Inf    0.0455     0.302
##              3 0.134 0.0638 Inf    0.0501     0.313
##              4 0.143 0.0659 Inf    0.0552     0.324
##              5 0.153 0.0678 Inf    0.0607     0.335
##
## Confidence level used: 0.95
## Intervals are back-transformed from the logit scale
```

Vediamo come rappresentare le predizioni di `glm.binom1` con le funzioni del pacchetto `ggeffects` (Lüdecke, 2018). Il pacchetto `ggeffects` include le funzioni `ggpredict()` e `ggemmeans()`, che si basano sulle funzioni `predict()` ed `emmeans()`. Il vantaggio delle funzioni di `ggeffects` è che producono stime formattate in modo tale da poter essere usate agevolmente per produrre grafici: la funzione `plot()` del pacchetto `ggeffect` produce grafici delle stime calcolate da `ggpredict()` e `ggemmeans()` usando la sintassi di `ggplot2`.

Usiamo `ggemmeans()` per ottenere le stime medie di mortalità secondo il modello `glm.binom1` ed i corrispondenti intervalli di confidenza. La funzione richiede che siano specificati il nome del modello e l'elenco dei predittori da prendere in considerazione:

```
p_load(ggeffects)
glm.binom1.ggpred <- ggemmeans(glm.binom1, terms = c("stress.per100"))
```

Per rappresentare le predizioni di `ggemmeans()` basta passarle come argomento alla funzione `plot()` del pacchetto `ggeffects`:

```
plot(glm.binom1.ggpred)
```

Il risultato è in fig. 19.13a. La funzione `ggemmeans()` calcola la mortalità attesa ed i rispettivi intervalli di confidenza solo per i valori osservati dei predittori. Per aumentare il numero di valori dei predittori per i quali stimare le predizioni, la funzione richiede una sintassi inusuale: il numero di stime va indicato in formato "`[n=...]`" all'interno dell'argomento `terms=`. In questo caso ho scelto di ottenere predizioni per 50 valori equidistanti di `stress.per100`:

```
glm.binom1.ggpred <- ggpredict(glm.binom1, terms = c("stress.per100 [n=50]"))
plot(glm.binom1.ggpred)
```

Il risultato è in fig. 19.13b. I grafici in fig. 19.13a-b mostrano solo le predizioni del modello. Per aggiungere le osservazioni al grafico delle predizioni di `ggemmeans()` è sufficiente aggiungere l'argomento `show_data=T`:

```
plot(glm.binom1.ggpred, show_data=T, # predizioni e osservazioni
     dot_alpha = 0.1, dot_size=4) +
  labs(y="Mortalità",
       x="Stress idrico [%]",
       title="") +
  theme_classic() +
  theme(text = element_text(family="Times", size=13))
```

Il risultato è in fig. 19.13c.

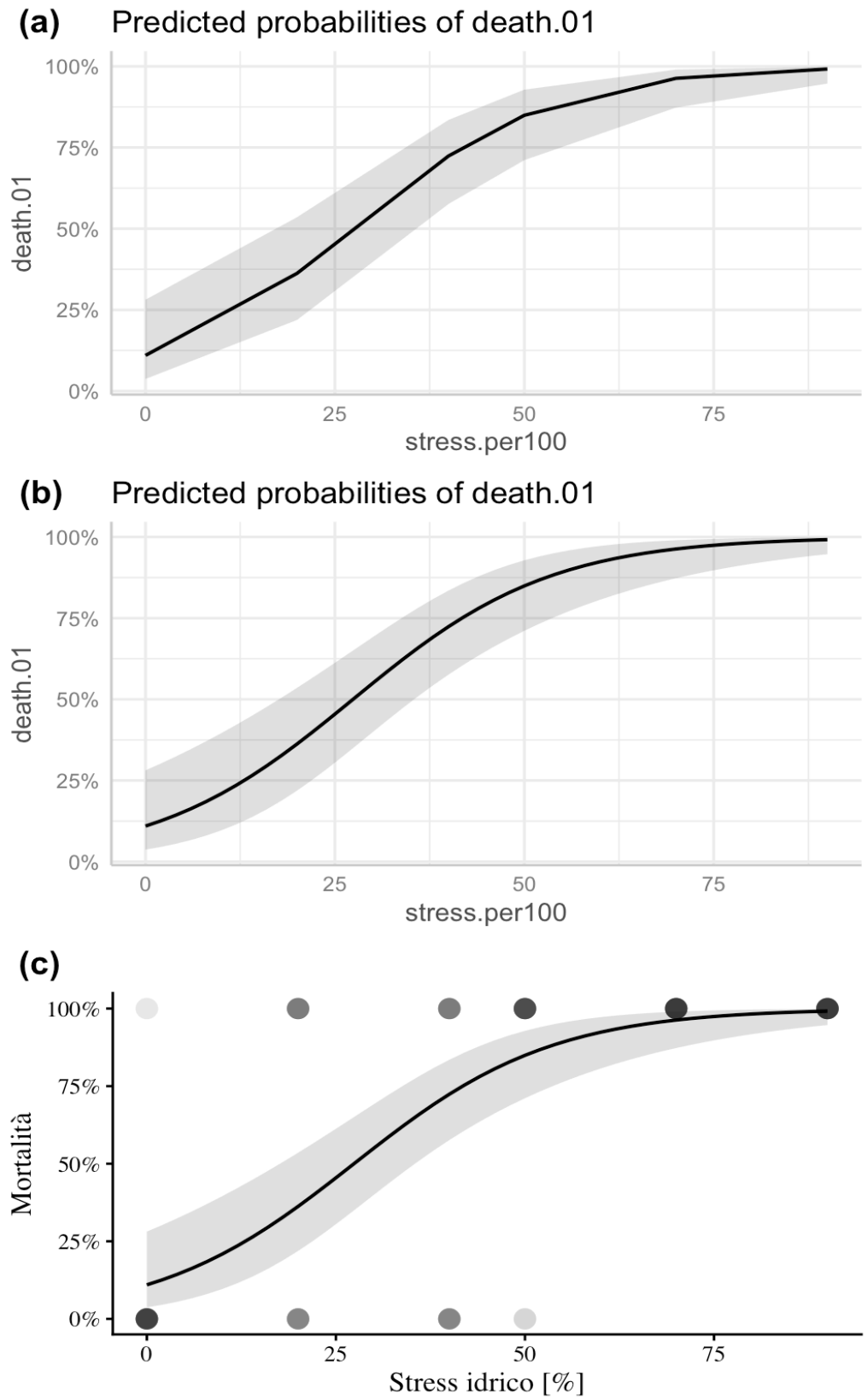

Figura 19.13: Tre rappresentazioni delle predizioni del modello $glm.binom1$.

19.6 Esempio 5: proporzioni come variabile di risposta

Questa sezione è stata rimossa dall'edizione gratuita di questo manuale. Potete acquistare l'edizione senza tagli qui:

- https://www.lulu.com/it/shop/marco-plebani/analisi-dei-dati-con-r-ed-rstudio/ebook/product-yvkp2ng.html (e-book)
- https://www.lulu.com/it/shop/marco-plebani/analisi-dei-dati-con-r-ed-rstudio/paperback/product-je6k8y6.html (in formato cartaceo)

Buona lettura!

- l'autore

19.7 Note

In queste sezioni abbiamo esaminato alcune applicazioni comuni dei modelli lineari generalizzati. Nello spirito di questo libro, che si propone come una *introduzione* all'analisi dei dati in R, ho preferito concentrarmi su pochi casi esemplificativi. Altre situazioni che possono essere studiate con modelli lineari generalizzati, non trattate in questo libro, sono:

- studi con densità di popolazione come variabile di risposta;
- studi in cui la variabile di risposta è un indice percentuale, come ad esempio la percentuale di territorio deputata ad un certo uso;
- studi in cui la variabile di risposta è il tempo trascorso prima del verificarsi di un certo evento (ad esempio: *"survival studies"*);
- l'analisi di dataset *"zero-inflated"*, ovvero dataset in cui la variabile di risposta è reppresentata da eventi rari o da una variabile di conteggio che si discosta raramente da zero.

La lista potrebbe continuare. I manuali di Zuur et al. (citati in *Bibliografia*) sono dei buoni testi di riferimento per approfondire l'applicazione dei modelli lineari generalizzati.

20 Violazione di indipendenza delle osservazioni: i modelli misti

Come abbiamo visto, i modelli lineari si basano su una serie di assunzioni: linearità della relazione tra predittori e variabile di risposta, distribuzione normale (i.e. gaussiana) dei residui, omoscedasticità dei residui, e indipendenza delle osservazioni. Abbiamo già esaminato approcci alternativi ai modelli lineari per i casi in cui le assunzioni di linearità, normalità dei residui, e/o omoscedasticità non siano rispettate (capitoli 18 e 19). In questo capitolo fornirò una breve introduzione ai *modelli misti* o modelli a effetti misti (in inglese: *mixed models* o *mixed-effect models*), che permettono di prendere in considerazione la *non-indipendenza delle osservazioni* e di modellizzarla esplicitamente. Questi modelli sono chiamati anche modelli gerarchizzati o multilivello *(hierarchical models* o *multilevel models)*.

20.1 L'importanza del disegno sperimentale

In tutti gli esempi esaminati fin'ora abbiamo dato per scontato che tutti i dati fossero indipendenti tra loro. Nella realtà, l'indipendenza delle osservazioni non è affatto scontata: al di là delle variabili al centro dei nostri interrogativi sperimentali ci sono numerosissime variabili "di sfondo" che potrebbero rendere i dati raccolti più simili tra loro di quanto non ci si aspetterebbe per puro effetto del caso. Ad esempio: nel capitolo 6 abbiamo confrontato la massa corporea di due specie chiedendoci se l'identità di specie fosse un predittore significativo della massa corporea degli individui. Altre variabili che possono avere un effetto sulla massa corporea sono: l'età degli individui misurati, il loro genere sessuale, la loro dieta, il loro grado di parentela… Se avessimo misurato solo o prevalentemente individui maschi nelle due specie i nostri risultati sarebbero stati poco rappresentativi: ci avrebbero informato solo sulle differenze di massa corporea tra i *maschi* delle due specie. Se avessimo misurato solo o prevalentemente maschi nella specie A e solo o prevalentemente femmine nella specie B, questo avrebbe creato confusione tra il possibile effetto dell'identità di specie ed il possibile effetto del genere sessuale sulla massa degli individui misurati.

Se, nello studio di cui al capitolo 6, avessimo misurato la massa corporea di individui imparentati tra loro, è probabile che essi avrebbero avuto masse corporee più simili tra loro di quanto non sarebbero state per puro effetto del caso: il nostro campione avrebbe quindi avuto una variabilità minore di quella che avremmo osservato tra individui non imparentati tra loro, fornendoci una stima falsata della variabilità della loro popolazione di provenienza.

Tra le possibili fonti di non-indipendenza delle misurazioni va anche considerato l'effetto sistematico che potrebbe essere introdotto dagli sperimentatori e/o da chi raccoglie i dati: ad esempio, alcune delle persone che si occupano di raccogliere i dati di uno studio potrebbero tendere ad arrotondare le misurazioni per eccesso mentre altre potrebbero farlo per difetto.

Per gestire il "rumore di fondo" generato da queste "variabili di sfondo" è importante definire un adeguato disegno sperimentale o, nel caso di uno studio osservativo, un adeguato disegno di raccolta dei dati.

Gli obiettivi principali di un adeguato disegno sperimentale o di raccolta dei dati sono:

- identificare dei gruppi che siano accomunati dal medesimo livello di ciascuno dei nostri predittori;
- ottenere un grado di replicazione sufficiente a stimare la variabilità della popolazione da cui è originato il campione;
- "randomizzare" l'assegnazione delle singole repliche ai gruppi sperimentali/di studio, cioè assegnarle in modo casuale a ciascun gruppo in modo da evitare che i nostri dati siano strutturati in base a una o più variabili "di sfondo". La sovrapposizione parziale o totale tra la struttura dei dati generata dai predittori di interesse e la struttura generata da variabili "confondenti" rende infatti i loro effetti difficili da scindere.

A volte non è possibile evitare che i dati siano strutturati secondo altre variabili oltre a quelle di interesse. In questi casi si può ricorrere ai modelli misti. Nel gergo dei modelli misti, le variabili di interesse in uno studio sono chiamate "effetti fissi", mentre le variabili di sfondo inserite nel modello per separarne l'effetto da quello delle variabili di interesse sono chiamate "effetti random".

20.2 Modelli misti con i pacchetti `lme4`, `lmerTest`, e `glmmTMB`

In R esistono vari pacchetti e funzioni per sviluppare e testare modelli misti. Eccone alcuni:

- il pacchetto `lme4` (Bates et al., 2015) permette di produrre modelli lineari misti gaussiani con la funzione `lmer()`, modelli lineari misti generalizzati (i.e. non-gaussiani) con la funzione `glmer()`, e modelli misti non-lineari con la funzione `nlmer()`;
- il pacchetto `lmerTest` (Kuznetsova et al., 2017) include una funzione `lmer()` basata sull'omonima funzione del pacchetto `lme4`. La differenza principale tra le funzioni `lmer()` dei due pacchetti è che, al contrario di `lme4::lmer()`, la funzione `lmer()` di `lmerTest` fornisce valori (approssimati) dei *p-values* per le stime dei parametri dei modelli;
- il pacchetto `glmmTMB` (Brooks et al., 2017) contiene la funzione omonima `glmmTMB()`, che permette di definire ed analizzare modelli lineari (similmente ad `lm()`), modelli lineari generalizzati (similmente a `glm()`), e modelli lineari misti (similmente a `lmer()` e `glmer()`). A differenza di `lme4`, `glmmTMB` non permette di sviluppare modelli non-lineari. In compenso, `glmmTMB` include una più vasta gamma di distribuzioni non-gaussiane, e permette inoltre di definire strutture di correlazione spaziali e/o temporali per le osservazioni.

Le funzioni di `lme4`, `lmerTest`, e `glmmTMB` usano una sintassi molto simile tra loro.

20.3 Esempio 1: un modello misto con predittore categorico ed intercette "random"

Immaginiamo uno studio sugli effetti di tre fertilizzanti - chiamiamoli A, B, e C - sui raccolti di grano. Lo studio coinvolge un team internazionale di ricercatori e ricercatrici, che hanno riprodotto l'esperimento in nove paesi. In ciascun paese gli sperimentatori hanno delimitato trenta campi sperimentali di 10x10 m, assegnandone casualmente dieci al trattamento con

ciascun fertilizzante. Potremmo essere tentati di analizzare i dati seguendo il protocollo di analisi sviluppato nel capitolo 7 per testare il seguente modello lineare:

$$yield_i = \mu + f_k \cdot Fert_k + \epsilon_i \qquad (20.1)$$

Dove:

- $yield_i$ è il raccolto di grano nell'i-esimo quadrato sperimentale;
- μ è la media di tutte le osservazioni a prescindere del fertilizzante usato;
- f_k è lo scostamento medio da μ corrispondente al$k$-esimo fertilizzante $Fert_k$ a cui è esposto l'i-esimo quadrato sperimentale;
- ϵ_i rappresenta lo scostamento di ciascun valore $yield_i$ dalle predizioni medie del modello.

Il modello in Eq. (20.1) presuppone che tutte le osservazioni siano indipendenti. Noi sappiamo però che le osservazioni provengono da paesi differenti, e ciò potrebbe essere una fonte di disomogeneità e non-indipendenza nei dati: ovvero, le osservazioni raccolte in ciascun paese potrebbero essere più simili tra loro di quanto non ci si aspetti per puro effetto del caso. Le cause di ciò possono essere varie, ad esempio: differenze climatiche tra i paesi, differenze edafiche, differenze nell'attuazione del protocollo sperimentale… Si tratta di fattori al di fuori del controllo delle sperimentatrici e degli sperimentatori, ma che fanno capo al paese a cui appartiene ciascuna osservazione. Con i modelli misti possiamo tenere conto del paese di provenienza di ciascuna misurazione in modo da "ripulire" il segnale di interesse, ovvero l'eventuale effetto del trattamento sperimentale, da una parte del "rumore di fondo" causato dalla variabilità tra paesi.

Nel nostro esempio, con un modello misto possiamo stimare l'effetto medio di ciascun fertilizzante sui raccolti *in generale*, a prescindere dal paese. è inoltre possibile stimare la porzione di variabilità nei raccolti associata al fattore "fertilizzante", quella associata al fattore "paese", e quella residua. Per quanto riguarda il fattore "paese", il modello misto non stima gli effetti specifici di ciascun paese sui raccolti, ma solo la porzione di variabilità nei raccolti associata al fattore "paese". Questo risulta in un notevole risparmio nel numero di parametri del modello.

Nel contesto dei modelli misti, i predittori che sono al centro del nostro interrogativo di ricerca sono chiamati *fixed effects* o *effetti fissi*; le variabili che rappresentano potenziali fonti di disomogeneità e non-indipendenza nei dati sono invece chiamati *random effects* o *effetti random*. Nel nostro caso l'effetto del tipo di fertilizzante è la parte fissa del modello *(fixed effect)*, mentre l'effetto del paese ne è la parte random *(random effect)*. I modelli misti stimano i parametri per la parte fissa del modello nonchè il suo contributo alla variabilità totale osservata nella variabile di risposta. Quanto alla parte random del modello, i modelli misti si limitano a stimare il loro contributo alla variabilità totale osservata nella variabile di risposta. In altre parole, i modelli misti non stimano i parametri associati a ciascun livello degli effetti random, ma solo la varianza ad essi associata.

Importante: affinchè sia possibile stimare la varianza associata ai fattori random è necessario che essi abbiano un numero adeguato di livelli. Vari autori suggeriscono un numero minimo di cinque livelli (Harrison et al., 2018).

I modelli misti assumono:

- che i residui per ciascun livello della parte random del modello siano distribuiti secondo una distribuzione normale, siano omoscedastici, e siano indipendenti tra loro;
- che i parametri associati alla parte random del modello provengano da una distribuzione normale.

I dati relativi all'esempio descritto all'inizio di questa sezione sono disponibili nel file `mixed_effect_data1.csv` all'indirizzo internet https://github.com/marcoplebani85/datasets/. Possiamo scaricare il file e farlo leggere a R come sappiamo già fare, oppure possiamo far sì che R legga direttamente il file online usando la funzione `getURL()` del pacchetto `RCurl` (Lang, 2023). Di seguito, dopo aver caricato i pacchetti che ci serviranno in questo capitolo, trovate una dimostrazione dell'uso di `getURL()`.

```
rm(list = ls())  # ripulisce la memoria di R
# carica il pacchetto 'pacman', che contiene la funzione p_load():
if (!require(pacman)) install.packages("pacman")
library(pacman)
# altri pacchetti:
p_load(doBy)
p_load(emmeans)
p_load(ggplot2)
# per rappresentare le predizioni dei modelli misti in ggplot2:
p_load(ggeffects)
# per produrre test e grafici diagnostici per GLM e mixed-effect models:
p_load(DHARMa, performance, see)
p_load(lme4, lmerTest, glmmTMB)  # per fittare modelli misti
p_load(MuMIn)  # per stimare lo pseudo-R-quadro dei modelli misti
p_load(RCurl)

# definiamo la directory di lavoro:
setwd("~/Desktop/Corso_R")

# Carichiamo i dati in R:
dd <- read.csv(text = RCurl::getURL(
  "https://raw.githubusercontent.com/marcoplebani85/datasets/master/mixed_eff
ect_data1.csv")
  )
# getURL() permette di accedere a dati online
```

Visualizziamo i dati:

```
# calcoliamo media e deviazione standard per ciascun trattamento in ciascun p
aese:
dd_smr <- summaryBy(yield ~ fertilizer * country, data=dd, FUN=c(mean, sd))
# cambiamo i nomi delle colonne di dd_smr:
names(dd_smr) <- c("fertilizer", "country", "yield", "sd")

# rappresentiamo graficamente dd_smr:
ggplot(data=dd_smr, aes(x=fertilizer, y=yield, fill=country)) +
  geom_bar(stat="identity", color="black", # istogramma
           position=position_dodge()) +
  geom_errorbar(aes(ymin=yield-sd, ymax=yield+sd), # barre d'errore
                width=.2, position=position_dodge(.9)) +
    # etichettiamo gli assi:
  labs(x = "Tipo di fertilizzante", y = "Raccolto [Kg]") +
    scale_fill_grey(name = "Paese") + theme_classic() +
  theme(text = element_text(family="Times", size=15))
```

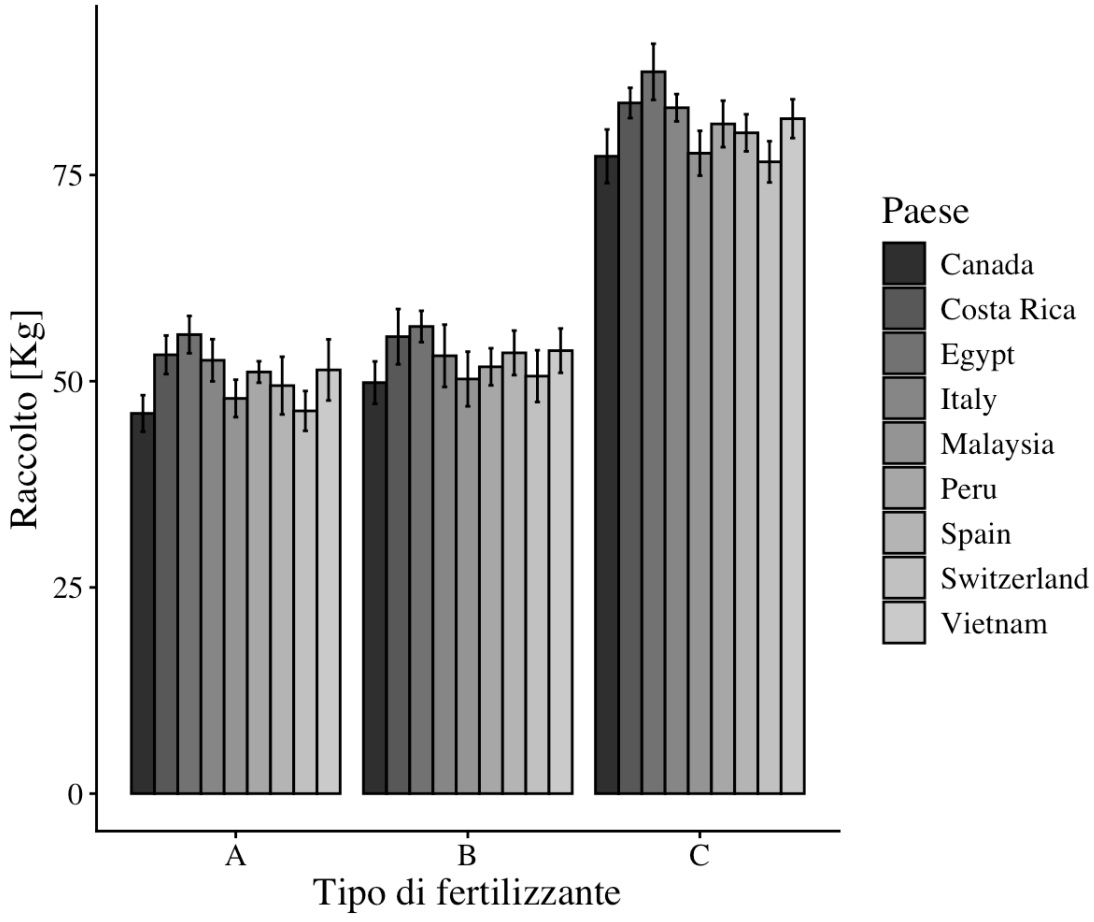

Figura 20.1: Dati e predizioni per ciascun trattamento in ciascun paese.

Esaminando fig. 20.1 si può notare che i raccolti dei quadrati sperimentali fertilizzati con fertilizzanti di tipo A e B hanno una resa di circa 50 Kg, mentre i quadrati fertilizzati con fertilizzante di tipo C hanno una resa di circa 80 Kg. Si può inoltre notare che i raccolti ottenuti in ciascun paese mostrano un andamento caratteristico, al di là del tipo di fertilizzante usato: ad esempio, i raccolti medi ottenuti nei quadrati sperimentali dell'esperimento canadese sono mediamente più bassi che in tutti gli altri paesi, mentre quelli dell'esperimento svolto in Egitto sono mediamente più alti che in tutti gli altri paesi.

Definiamo un modello misto con `lmer()` dal pacchetto `lmerTest`:

```
mem1 <- lmerTest::lmer(yield ~ fertilizer + (1 | country),
                       data = dd,
                       REML=F)
```

La componente `yield ~ fertilizer` rappresenta la parte "fissa" del modello: "`yield` varia in funzione di `fertilizer`". La componente `(1 | country)` rappresenta la parte "random" del modello, e sta ad indicare che l'intercetta della relazione tra `yield` e `fertilizer` può variare in modo specifico per ciascun paese.

20.3.1 REML o ML?

L'argomento "`REML=`" merita un breve approfondimento. Per stimare i parametri dei modelli misti e gli intervalli di confidenza ad essi associati si usano algoritmi di ricerca basati sul concetto di *maximum likelihood*. Questi metodi sono usati anche per i modelli lineari generalizzati, come accennato in sez. 19.2.3. La *likelihood* di un modello per un certo set di parametri corrisponde alla probabilità di osservare i dati osservati assumendo che il modello ed il set di parametri in questione siano esatti. Gli algoritmi di parametrizzazione dei modelli misti vanno alla ricerca dei valori dei parametri del modello per i quali la *likelihood* di osservare i dati osservati è massima[8]. La *likelihood* può essere calcolata in vari modi: la procedura di *maximum likelihood* "classica", ML, stima i parametri dei fattori fissi e di quelli random in maniera non indipendente; la procedura di *maximum likelihood* "ristretta", REML, stima i parametri dei fattori random indipendentemente da quelli fissi. La procedura REML ottiene stime della varianza dei fattori random più affidabili di quanto non faccia la procedura ML ma, al contrario di quest'ultima, non permette di confrontare affidabilmente i modelli con metodi basati sulla *likelihood* (come l'AIC) (Zuur et al. 2009). La procedura raccomandata è dunque la seguente:

1. per svolgere confronti e selezione tra modelli si raccomanda di definire i modelli e di stimarne i parametri usando la procedura ML (fissando l'argomento `REML=FALSE`);
2. una volta individuato il modello misto che descrive al meglio i dati, si consiglia di ri-fittare il modello in questione usando la procedura REML (fissando l'argomento `REML=TRUE`) per estrarne i parametri ed i relativi intervalli di confidenza.

[8] In realtà, per facilità di computo, gli algoritmi di parametrizzazione vanno alla ricerca del set di parametri che minimizza il negativo della log-likelihood.

20.3.2 Grafici diagnostici

Vediamo i grafici diagnostici prodotti da `DHARMa` per il modello misto `mem1`:

```
# grafici diagnostici usando le funzioni di DHARMa:
mem1Resids <- simulateResiduals(mem1, n = 3000, use.u = F)
par(mfrow = c(1, 2))
plotQQunif(mem1Resids)
mtext(text = "(a)", side = 3, adj = 0, line = 3)
plotResiduals(mem1Resids, quantreg = T)
mtext(text = "(b)", side = 3, adj = 0, line = 3)
```

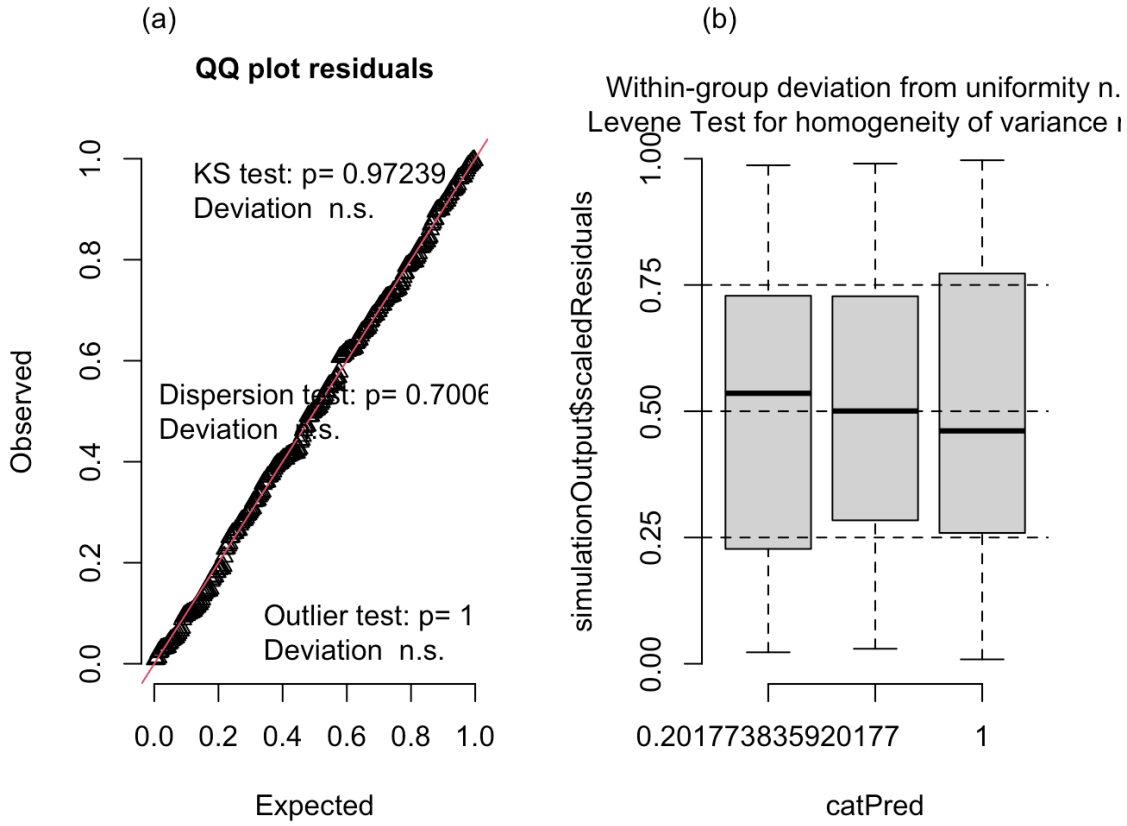

Figura 20.2: Grafici diagnostici per `mem1` *usando il pacchetto DHARMa.*

Si possono produrre grafici diagnostici per i modelli misti anche usando il pacchetto `performance`:

```
performance::check_model(mem1,
  check=c("qq", "linearity", "homogeneity", "outliers", "reqq"))
```

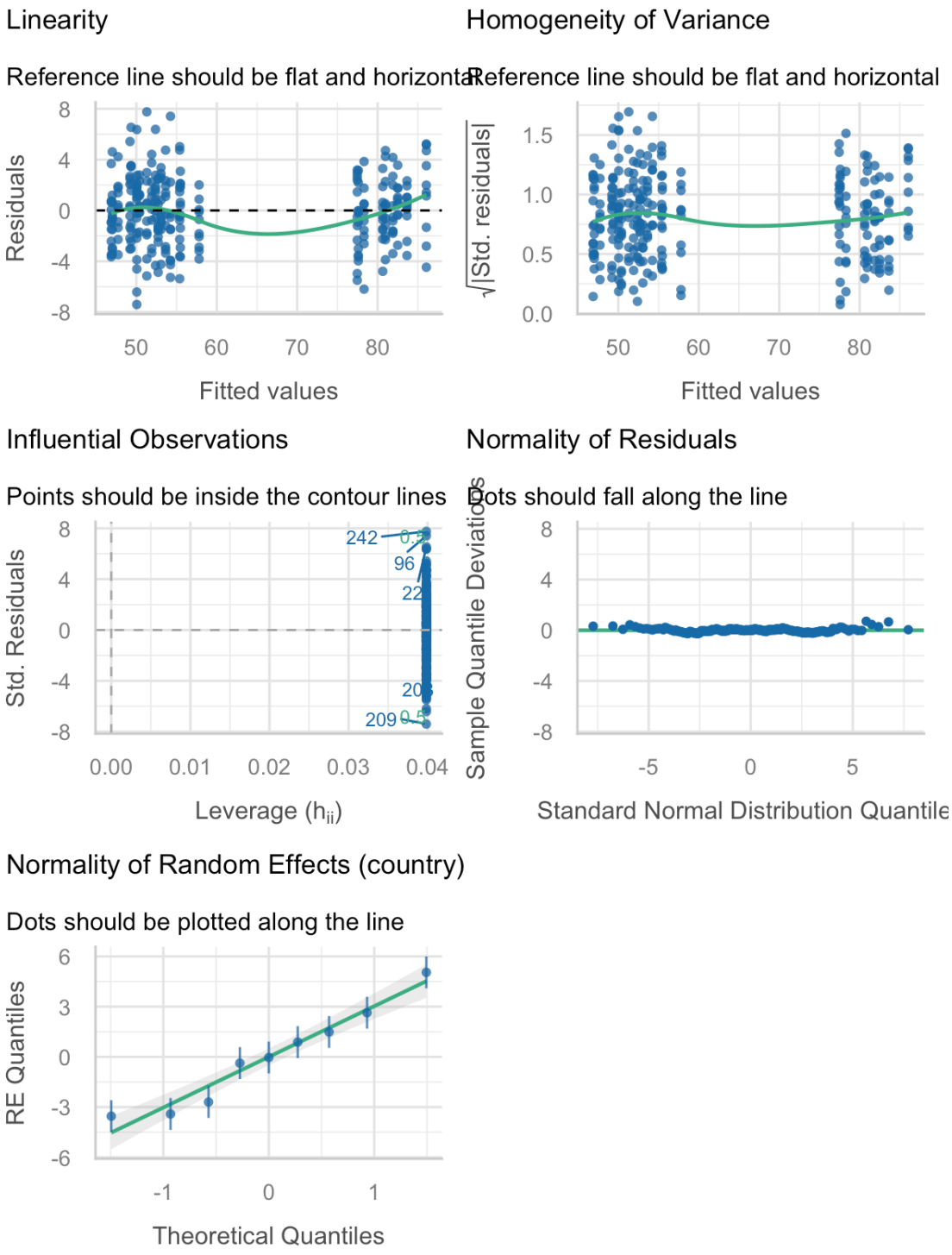

Figura 20.3: Grafici diagnostici per `mem1` *usando il pacchetto* `performance`.

C'è una importante differenza tra i grafici diagnostici prodotti da `performance` e quelli prodotti da `DHARMa`: il grafico quantile-quantile prodotto da `performance` confronta sempre i quantili osservati con quelli attesi *secondo una distribuzione normale*; il grafico quantile-quantile prodotto da `DHARMa` invece confronta i quantili osservati con quelli attesi secondo la distribuzione

dei residui specificata nel modello, per cui si presta anche al controllo delle assunzioni per modelli misti generalizzati (sezione 20.6). I grafici diagnostici forniti dal pacchetto `performance` non includono i risultati dei test statistici per normalità, dispersione, e omoscedasticità forniti dai grafici di `DHARMa`, ma `performance` permette di ottenerli con funzioni specifiche. Ecco alcuni esempi dei test statistici disponibili nel pacchetto `performance`:

```
check_heteroscedasticity(mem1)

## OK: Error variance appears to be homoscedastic (p = 0.427).

check_homogeneity(mem1)

## OK: There is not clear evidence for different variances across groups (Bar
tlett Test, p = 0.315).

check_outliers(mem1)

## OK: No outliers detected.
## - Based on the following method and threshold: cook (0.5).
## - For variable: (Whole model)
```

I grafici diagnostici ed i test prodotti da `DHARMa` (20.2) e da `performance` (20.3) indicano che ci non sono problemi di violazioni delle assunzioni dei modelli lineari misti.

20.3.3 Selezione del modello

Confrontiamo ora `mem1`, che include l'effetto fisso di `fertilizer` e quello random di `(1 | country)`, con i seguenti modelli semplificati:

```
# modello senza effetti fissi:
mem_ranef <- lmerTest::lmer(yield ~ 1 + (1 | country), data = dd, REML = F)
# modello nullo:
lm0 <- lm(yield ~ 1, data = dd)
```

Abbiamo fittato i modelli `mem1` e `mem_ranef` con il metodo ML (`REML=F`) in modo da poterli confrontare tramite i loro AIC (`lm0` non è un modello misto, per cui il problema della scelta tra REML e ML non si pone):

```
# confronto tramite AIC:
AIC(mem1, mem_ranef, lm0)

##           df      AIC
## mem1       5 1344.578
## mem_ranef  3 2213.647
## lm0        2 2211.734
```

Gli AIC suggeriscono che il modello `mem1` sia il migliore tra quelli a confronto.

20.3.4 Estrazione dei parametri, ecc.

Per estrarre i parametri del modello di interesse procediamo per prima cosa a ri-fittarlo usando il metodo REML (`REML=T`):

```
mem1reml <- lmerTest::lmer(yield ~ fertilizer + (1 | country),
                      data = dd,
                      REML=T)
```

Vediamone il *summary*:

```
summary(mem1reml)
```

```
## Linear mixed model fit by REML. t-tests use Satterthwaite's method [
## lmerModLmerTest]
## Formula: yield ~ fertilizer + (1 | country)
##    Data: dd
##
## REML criterion at convergence: 1333
##
## Scaled residuals:
##      Min       1Q   Median       3Q      Max
## -2.72549 -0.64388  0.00756  0.68393  2.85857
##
## Random effects:
##  Groups   Name        Variance Std.Dev.
##  country  (Intercept) 8.676    2.946
##  Residual             7.364    2.714
## Number of obs: 270, groups:  country, 9
##
## Fixed effects:
##             Estimate Std. Error       df t value Pr(>|t|)
## (Intercept)  50.4181     1.0227   8.9038  49.300 3.66e-12 ***
## fertilizerB   2.3307     0.4045 259.0000   5.761 2.36e-08 ***
## fertilizerC  30.5888     0.4045 259.0000  75.616  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) frtlzB
## fertilizerB -0.198
## fertilizerC -0.198  0.500
```

La struttura delle informazioni prodotte da `summary()` per i modelli misti ci è per lo più familiare. La novità è che, in questo caso, abbiamo dei parametri stimati per la parte fissa del modello e per quella random. Le stime dei parametri per la parte random sono stime di variabilità: nel caso del modello `mem1reml` ci indicano che la deviazione standard osservata nei dati associata a "paese" *(country)* è pari a ~2.9 Kg, mentre la deviazione standard dei residui è di circa 2.7 Kg.

Per ottenere stime dei parametri per la parte fissa del modello possiamo ricorrere al fidato pacchetto `emmeans` come già fatto nel capitolo 7:

```
emmeans(mem1reml, pairwise ~ fertilizer, adjust = "bonferroni")
```

```
## $emmeans
##  fertilizer emmean   SE  df lower.CL upper.CL
##  A            50.4 1.02 8.9     48.1     52.7
##  B            52.7 1.02 8.9     50.4     55.1
##  C            81.0 1.02 8.9     78.7     83.3
##
## Degrees-of-freedom method: kenward-roger
## Confidence level used: 0.95
##
## $contrasts
##  contrast estimate    SE  df t.ratio p.value
##  A - B       -2.33 0.405 259  -5.761 <0.0001
##  A - C      -30.59 0.405 259 -75.616 <0.0001
##  B - C      -28.26 0.405 259 -69.854 <0.0001
##
## Degrees-of-freedom method: kenward-roger
## P value adjustment: bonferroni method for 3 tests
```

Possiamo infine calcolare gli pseudo-R^2 per il modello con la funzione `r2()` del pacchetto `performance` (o, se preferite, con la funzione `r.squaredGLMM()` del pacchetto `MuMIn`):

```
performance::r2(mem1reml)
```

```
## # R2 for Mixed Models
##
##   Conditional R2: 0.965
##      Marginal R2: 0.924
```

L'R^2 "condizionale" rappresenta la stima della frazione di variabilità totale spiegata dal modello. L'R^2 "marginale" rappresenta invece la stima della porzione di variabilità spiegata dalla sola parte fissa del modello.

20.3.5 Visualizzazione delle predizioni del modello

Possiamo usare le funzioni del pacchetto `ggeffect` per rappresentare graficamente le predizioni dei modelli misti, similmente a come abbiamo già fatto nell'ambito dei modelli lineari generalizzati (sez. 19.6.1). La funzione `ggemmeans()` si basa su `emmeans()` rendendone le stime *"ggplot-friendly"*:

```
predizioni <- ggemmeans(model = mem1reml, terms = c("fertilizer"))
```

Queste predizioni possono essere rese graficamente come segue (fig. 20.4):

```
plot(predizioni) + # predizioni
  # fronzoli:
  labs(x="Tipo di fertilizzante",
       y="Raccolto (Kg)",
       title="") +
  theme_classic() +
  theme(text = element_text(family="Times", size=15))
```

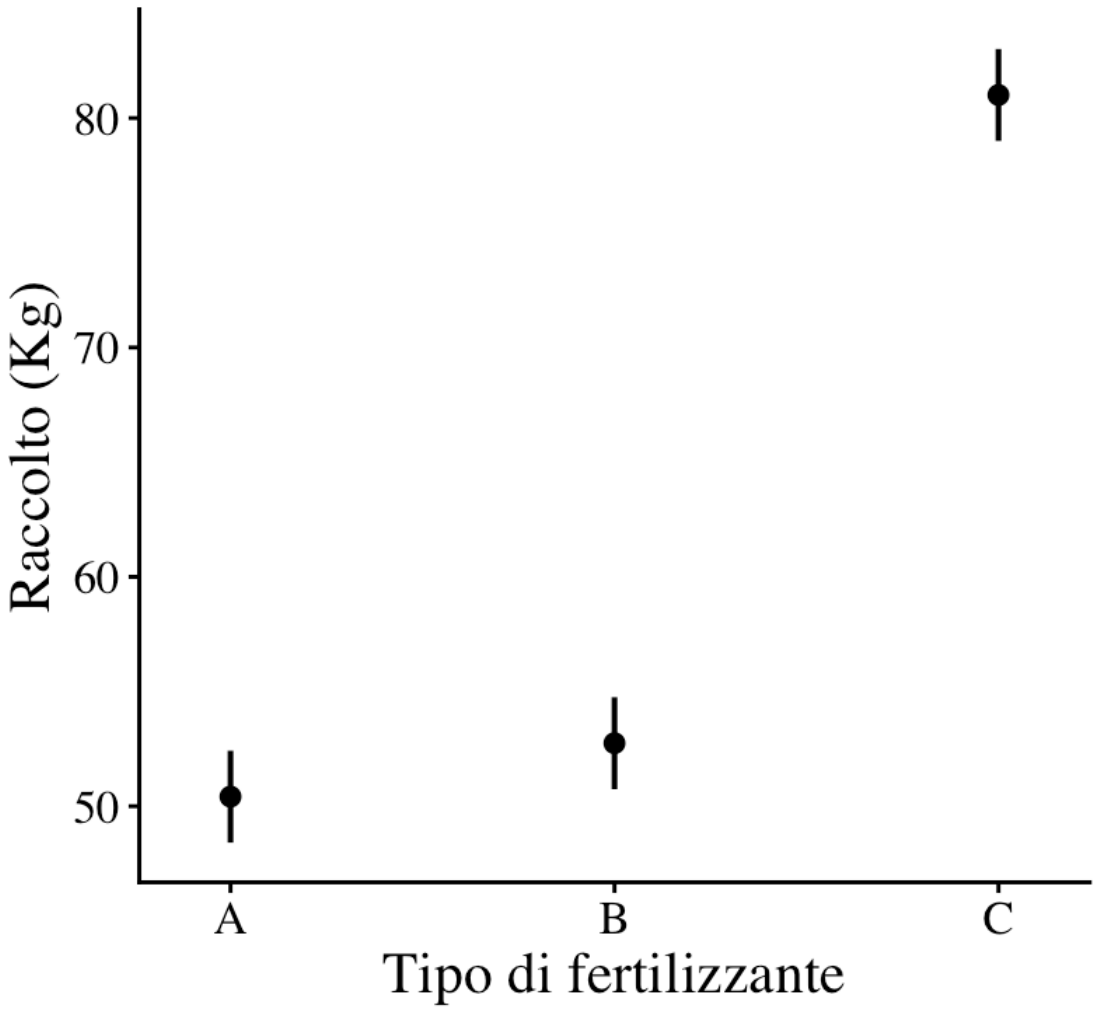

Figura 20.4: Predizioni per `mem1reml`.

20.4 Esempio 2: un modello misto con predittore continuo ed intercette "random"

Questa sezione è stata rimossa dall'edizione gratuita di questo manuale. Potete acquistare l'edizione senza tagli qui:

- https://www.lulu.com/it/shop/marco-plebani/analisi-dei-dati-con-r-ed-rstudio/ebook/product-yvkp2ng.html (e-book)
- https://www.lulu.com/it/shop/marco-plebani/analisi-dei-dati-con-r-ed-rstudio/paperback/product-je6k8y6.html (in formato cartaceo)

Buona lettura!

- l'autore

20.5 Esempio 3: un modello misto con intercette e pendenze “random”

Questa sezione è stata rimossa dall’edizione gratuita di questo manuale. Potete acquistare l’edizione senza tagli qui:

- https://www.lulu.com/it/shop/marco-plebani/analisi-dei-dati-con-r-ed-rstudio/ebook/product-yvkp2ng.html (e-book)
- https://www.lulu.com/it/shop/marco-plebani/analisi-dei-dati-con-r-ed-rstudio/paperback/product-je6k8y6.html (in formato cartaceo)

Buona lettura!

- l’autore

20.6 Esempio 4: un modello misto generalizzato

Per questo esempio useremo un estratto del dataset "RIKZ" presentato da Janssen e Mulder (2005) e reso popolare da Zuur et al. (2007), nella versione semplificata distribuita da Luštrik e Stachelek (2015). Si tratta di dati sulla biodiversità bentonica (numero di specie presenti) delle spiagge olandesi in relazione all'altezza del punto di campionamento rispetto al livello medio della marea ("NAP") e al grado di esposizione del punto di campionamento agli elementi ("Exposure"). I dati sono contenuti nel file `RIKZ.csv` all'indirizzo https://github.com/marcoplebani85/datasets/ e sono mostrati in fig. 20.13.

Prepariamo la sessione di lavoro:

```
rm(list = ls())  # ripulisce la memoria di R
# carica il pacchetto 'pacman', che contiene la funzione p_load():
if (!require(pacman)) install.packages("pacman")
library(pacman)
# altri pacchetti:
p_load(doBy)
p_load(emmeans)
p_load(ggplot2)
# per rappresentare le predizioni dei modelli misti in ggplot2:
p_load(ggeffects)
# per produrre test e grafici diagnostici per GLM e mixed-effect
# models:
p_load(DHARMa, performance, see)
p_load(lme4, lmerTest, glmmTMB)  # per fittare modelli misti
p_load(MuMIn)  # per stimare l'R-quadro dei modelli misti
p_load(RCurl)
```

Carichiamo i dati in R:

```
RIKZ <- read.csv(text = RCurl::getURL(
  "https://raw.githubusercontent.com/marcoplebani85/datasets/master/RIKZ.csv
"))
# getURL() permette di accedere a dati online
str(RIKZ)
# "Exposure" in realtà è un fattore:
RIKZ$Exposure <- as.factor(RIKZ$Exposure)
```

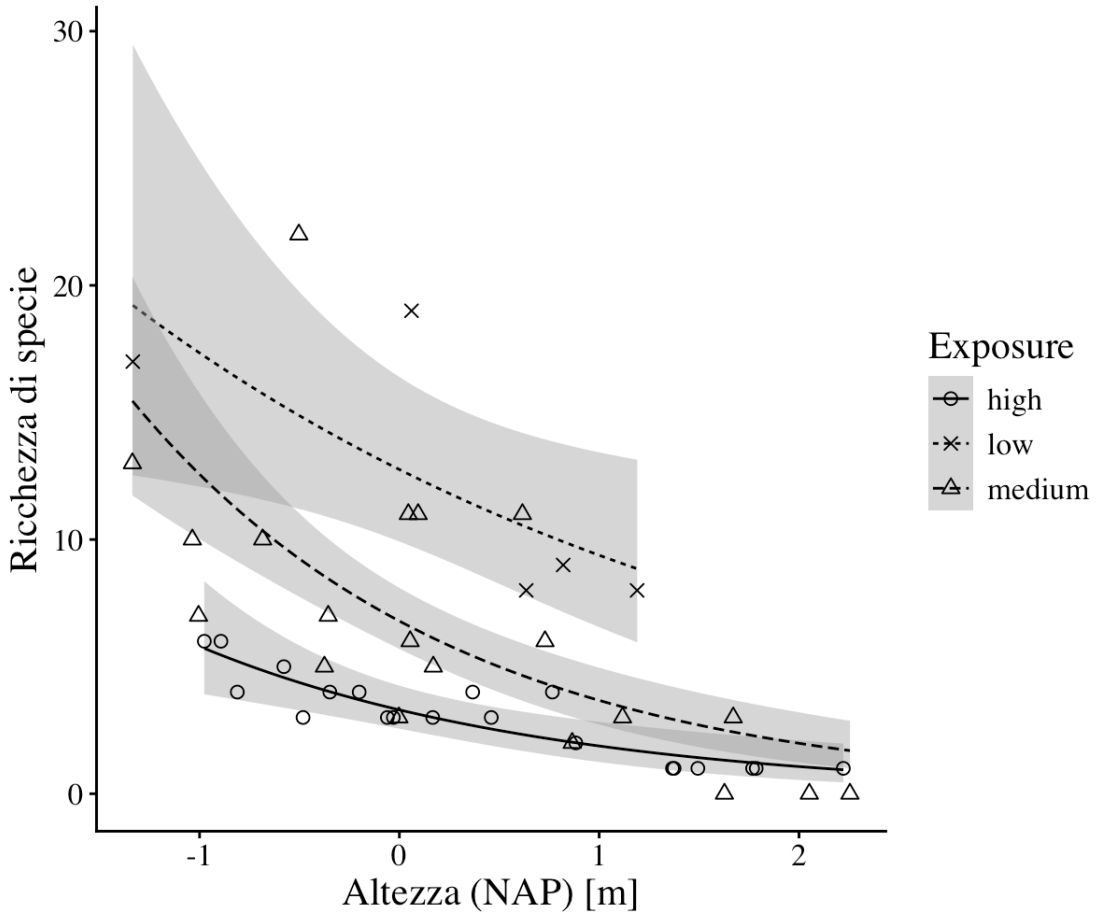

Figura 20.13: Dati e predizioni per ciascun livello di Exposure in base a modelli lineari generalizzati con distribuzione poissoniana dei residui.

Se volessimo studiare l'effetto dell'altezza del punto di campionamento ("NAP", continuo) e del grado di esposizione ("Exposure", fattoriale) sul numero di specie presenti, in prima approssimazione potremmo implementare ed esaminare il seguente modello lineare generalizzato:

```
glm1 <- glm(Richness ~ NAP*Exposure,
            family = "poisson",
            data = RIKZ)
```

Le predizioni del modello `glm1`, rappresentate in fig. 20.13, suggeriscono che la biodiversità diminuisca all'aumentare dell'altezza del punto di campionamento rispetto al livello medio della marea e all'aumentare del grado di esposizione. Questo approccio modellistico ha un limite: non tiene conto del fatto che i dati del dataset "RIKZ" potrebbero non essere indipendenti, bensì strutturati al livello di spiaggia ("Beach"). È infatti lecito supporre che le osservazioni provenienti dalla stessa spiaggia siano più simili tra loro che non rispetto a quelle compiute su altre spiagge semplicemente perchè si trovano nel medesimo ambiente, a prescindere dai valori di "NAP" e di "Exposure" a cui sono esposti. Per questa ragione una analisi con un modello lineare generalizzato potrebbe essere fuorviante. Per ovviare a questo limite possiamo usare un modello generalizzato di tipo misto indicando "Beach" come fattore random. Implementiamolo usando la funzione `glmmTMB()` del pacchetto omonimo:

```
mem1 <- glmmTMB(Richness ~ NAP*Exposure + (1 | Beach),
                family = "poisson",
                data = RIKZ,
                REML = F)
```

La funzione `glmmTMB()` usa in larga parte la stessa sintassi che abbiamo già incontrato in `glm()` ed in `lmer()`. La componente fissa del modello è `Richness ~ NAP*Exposure`: il modello prevede che `Richness` varii in dipendenza di `NAP` (continuo) ed `Exposure` (categorico), e che i parametri della dipendenza tra `Richness` e `NAP` possano differire in ciascun livello di `Exposure`. La componente random del modello è `(1 | Beach)`, che sta ad indicare che ciascuna spiaggia è caratterizzata da una sua intercetta - in altre parole, la `Richness` media varia di spiaggia in spiaggia. Valutiamo il rispetto delle assunzioni per il modello misto `mem1` con i grafici diagnostici prodotti da `DHARMa`:

```
# grafici diagnostici usando le funzioni di DHARMa:
mem1Resids <- simulateResiduals(mem1, n = 3000, use.u = F)
par(mfrow = c(1, 2))
plotQQunif(mem1Resids)
mtext(text = "(a)", side = 3, adj = 0, line = 3)
plotResiduals(mem1Resids, quantreg = T)
mtext(text = "(b)", side = 3, adj = 0, line = 3)
```

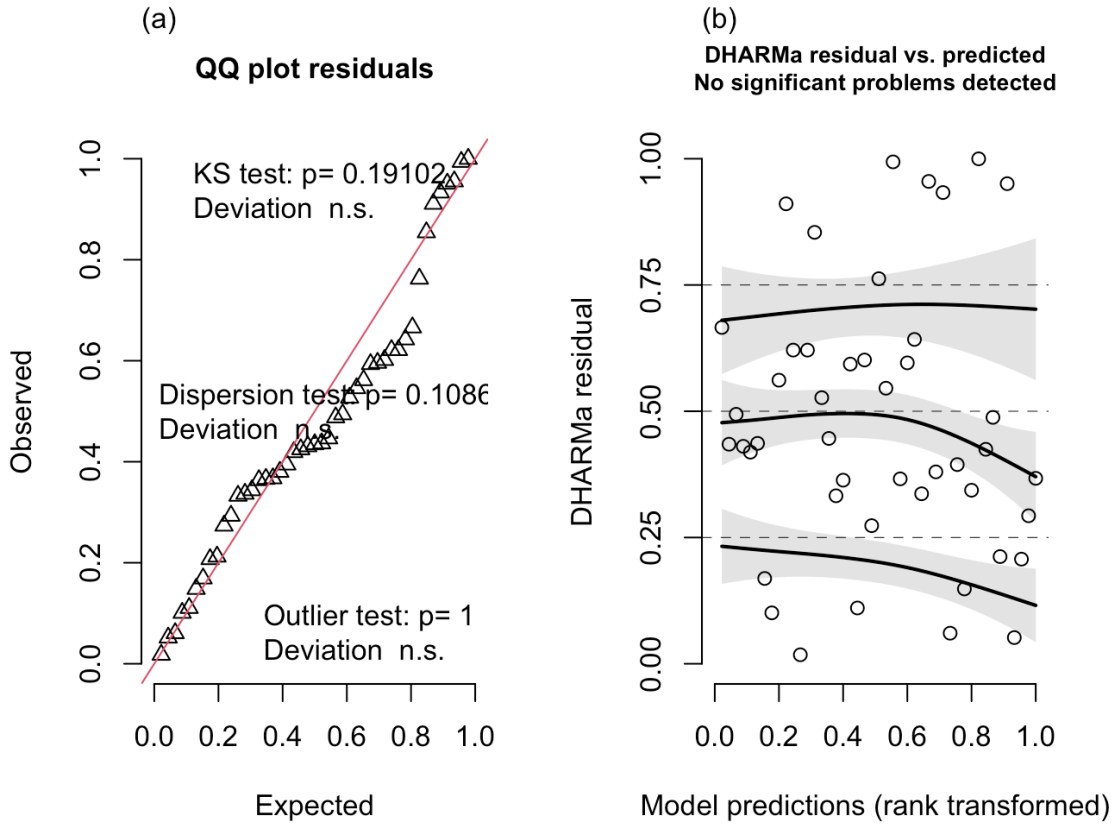

Figura 20.14: Grafici diagnostici per `mem1`.

20.6.1 Selezione del modello migliore

Confrontiamo `mem1` con i seguenti modelli alternativi, ciascuno dei quali rappresenta una versione più o meno semplificata di `mem1` stesso:

```
mem1_additivo <- glmmTMB(Richness ~ NAP+Exposure + (1 | Beach),
                family = "poisson",
                data = RIKZ, REML = F)
mem1_NAP <- glmmTMB(Richness ~ NAP + (1 | Beach),
                family = "poisson",
                data = RIKZ, REML = F)
mem1_Exposure <- glmmTMB(Richness ~ Exposure + (1 | Beach),
                family = "poisson",
                data = RIKZ, REML = F)
mem1_ranef <- glmmTMB(Richness ~ 1 + (1 | Beach),
                family = "poisson",
                data = RIKZ, REML = F)
glm0 <- glmmTMB(Richness ~ 1,
                family = "poisson",
                data = RIKZ, REML = F)
# confronto tramite AIC:
AIC(mem1, mem1_additivo, mem1_NAP, mem1_Exposure, mem1_ranef, glm0)

##                df      AIC
## mem1            7 210.4972
## mem1_additivo   5 210.0085
## mem1_NAP        3 220.7843
## mem1_Exposure   4 259.1488
## mem1_ranef      2 265.8775
## glm0            1 323.7514
```

I modelli `mem1` (con interazione tra predittori) e `mem1_additivo` (senza interazione) hanno i valori di AIC più bassi tra quelli dei modelli a confronto. I loro valori di AIC che differiscono per meno di una unità, per cui possiamo considerare il più semplice dei due modelli, ovvero `mem1_additivo`, come il migliore tra i modelli a confronto.

20.6.2 Estrazione dei parametri, ecc.

Per estrarre i parametri della parte fissa del modello di interesse procediamo per prima cosa a rifittarlo usando il metodo REML (`REML = T`):

```
mem1additivoREML <- glmmTMB(Richness ~ NAP+Exposure + (1 | Beach),
                family = "poisson",
                data = RIKZ, REML = T)
summary(mem1additivoREML)
```

```
##  Family: poisson  ( log )
## Formula:          Richness ~ NAP + Exposure + (1 | Beach)
## Data: RIKZ
##
##       AIC       BIC    logLik -2*log(L)  df.resid
##     219.2     228.3    -104.6     209.2        40
##
## Random effects:
##
## Conditional model:
##  Groups Name        Variance Std.Dev.
##  Beach  (Intercept) 0.04122  0.203
## Number of obs: 45, groups:  Beach, 9
##
## Conditional model:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)     1.19193    0.16502   7.223 5.08e-13 ***
## NAP            -0.50223    0.07306  -6.875 6.21e-12 ***
## Exposurelow     1.33718    0.29163   4.585 4.54e-06 ***
## Exposuremedium  0.72450    0.21345   3.394 0.000688 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

La funzione `summary()` riporta i parametri del modello:

$$Richness_i = e^{a+b \cdot NAP_i+\epsilon_i}, \epsilon_i \sim Poiss(\widehat{Richness}_{NAP_i}) \qquad (20.2)$$

Per ciascuno dei tre livelli di `Exposure`. Il parametro b = -0.50223 è in comune per i tre livelli di `Exposure`, dato che `mem1reml` è un modello di tipo additivo. Nel caso di un modello di tipo interattivo avremmo potuto calcolare le stime di pendenza ai vari gradi di esposizione, i rispettivi intervalli di confidenza, ecc. con la funzione `emtrends()` del pacchetto `emmeans`, come già visto nei capitoli precedenti.

Le stime di intercetta fornite da `summary()` sono sulla scala della funzione *link*; inoltre sono fornite, come da consuetudine, come differenze rispetto ad una intercetta di riferimento.

Per ottenere le stime di intercetta sulla scala non trasformata (ovvero come numero di specie) e in termini assoluti (anzichè in relazione ad una intercetta di riferimento) usiamo `emmeans`:

```
emmeans(mem1additivoREML,
            specs = ~ NAP + Exposure,
            at = list(NAP = 0),
            type = "response"
            )
```

```
##  NAP Exposure  rate    SE  df asymp.LCL asymp.UCL
##    0 high      3.29 0.543 Inf      2.38      4.55
##    0 low      12.54 3.020 Inf      7.83     20.09
##    0 medium    6.80 0.923 Inf      5.21      8.87
##
## Confidence level used: 0.95
## Intervals are back-transformed from the log scale
```

Il numero medio di specie attese al livello medio di marea (NAP = 0) per `Exposure` bassa, media, ed alta è rispettivamente 12.54, 6.80, e 3.29.

La ricchezza di specie media attesa per ciascun valore di `NAP` ed `Exposure` è dettata dai modelli:

$$\widehat{Richness}_{i,low} = e^{12.5-0.5 \cdot NAP_i} \quad (20.3)$$

$$\widehat{Richness}_{i,medium} = e^{6.8-0.5 \cdot NAP_i} \quad (20.4)$$

$$\widehat{Richness}_{i,high} = e^{3.3-0.5 \cdot NAP_i} \quad (20.5)$$

Usiamo ora `ggemmeans()` per calcolare le previsioni del modello `mem1additivoREML` in modo da poterle rappresentare agevolmente:

```
predicted.richness <- ggemmeans(mem1additivoREML, terms = c("NAP", "Exposure
"))
```

Per visualizzare le predizioni del modello con `ggpredict()`, mostrate in figura 20.15, basta eseguire:

```
plot(predicted.richness, # predizioni
     show_data=T, # osservazioni
     alpha=0.2, # trasparenze
     colors=c("#E7B800", "#00AFBB", "#FC4E07")) +
# fronzoli
labs(x = "Altezza (NAP) [m]",
       y = "Ricchezza di specie",
       title = "") +
  theme_classic() +
  theme(text = element_text(family = "Times", size = 14))
```

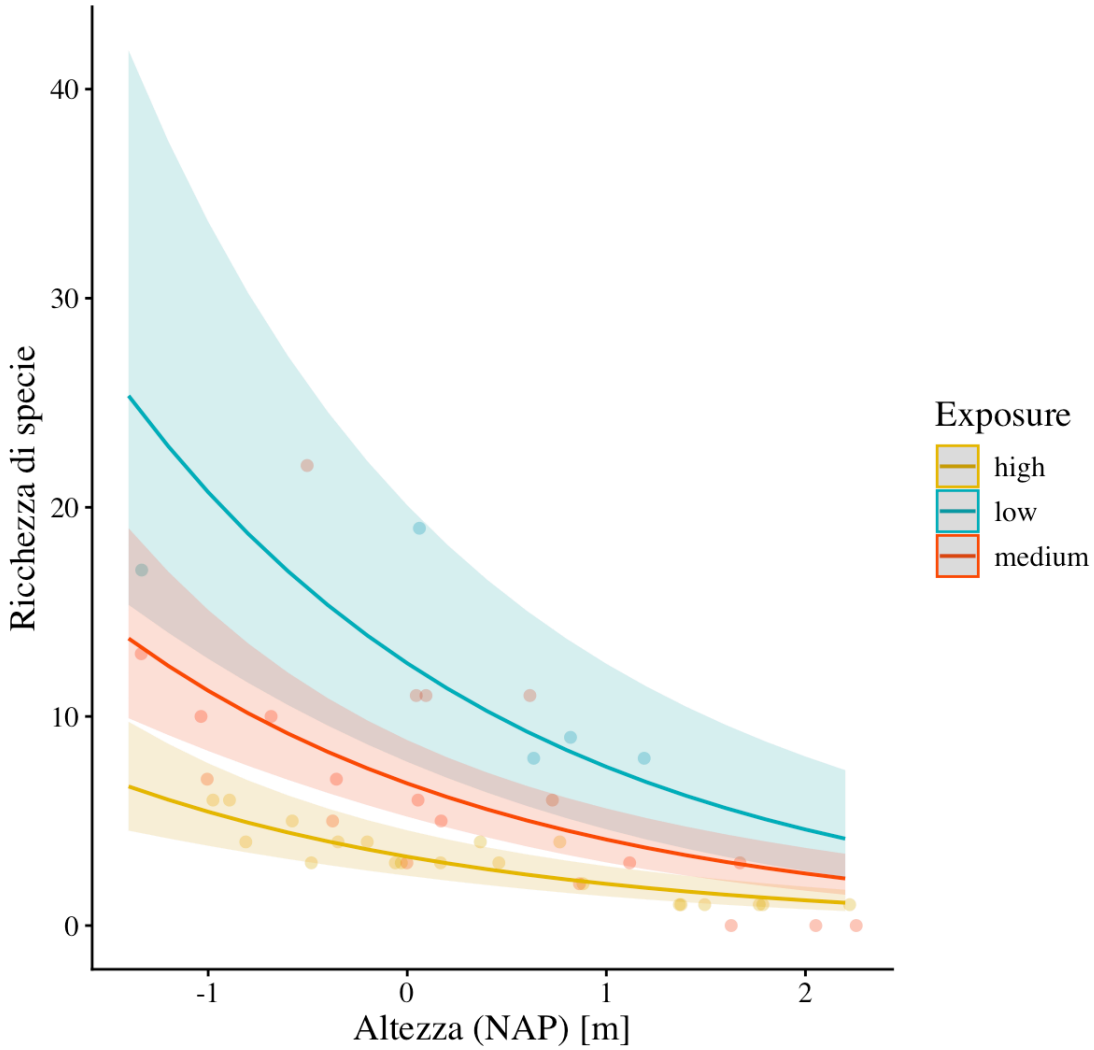

Figura 20.15: Predizioni del modello `mem1additivoREML`.

21 “Data massaging”

Con il capitolo sui modelli ad effetti misti abbiamo concluso la trattazione degli argomenti strettamente “analitici” di questo libro. In tutti gli esempi che abbiamo trattato abbiamo avuto a che fare con dataset già pronti per essere analizzati, ma nella realtà è spesso necessario un lavoro di pulizia e formattazione dei dati prima di poterli analizzare. Il processo di formattazione dei dataset in preparazione alle analisi è talvolta descritto come “data massaging”, ovvero il “massaggio” dei dati. In questo breve capitolo riporterò alcuni esempi di situazioni tipiche e come affrontarle.

Per cominciare, creiamo un dataset “minimo” di esempio:

```
rm(list=ls()) # ripulisce la memoria di R
dd <- data.frame(Lettera = c("B","B","A","A","C","C"),
                 x1 = runif(n=6, min=0, max=1), # valori casuali tra 0 e 1
                 x2 = runif(n=6, min=0.2, max=1.5)
                 )
dd
```

```
##   Lettera         x1        x2
## 1       B 0.06672103 0.3551431
## 2       B 0.48636461 0.4549017
## 3       A 0.66614263 0.5379975
## 4       A 0.77442318 0.4470598
## 5       C 0.64632866 0.6610715
## 6       C 0.37887038 0.6796479
```

21.1 Ordinamento alfanumerico

La funzione `order()` permette di ordinare un dataset alfanumericamente in base a una o più colonne. Supponiamo di voler ordinare il dataset che abbiamo appena creato, `dd`, in ordine alfabetico in base alla colonna `Lettera`:

```
dd.ordered <- with(dd, dd[order(Lettera), ])
dd.ordered
```

```
##   Lettera         x1        x2
## 3       A 0.66614263 0.5379975
## 4       A 0.77442318 0.4470598
## 1       B 0.06672103 0.3551431
## 2       B 0.48636461 0.4549017
## 5       C 0.64632866 0.6610715
## 6       C 0.37887038 0.6796479
```

21.2 Subsetting

La pratica del *subsetting* consiste nell’ottenere dei sottoinsiemi, o *subset*, di un certo dataset. Questo può essere svolto con una moltitudine di funzioni e pacchetti a seconda della necessità e delle preferenze personali. Di seguito ne vedremo alcuni esempi.

Per selezionare righe e/o colonne specifiche di un dataset esistono vari metodi. Si possono selezionare singole colonne usando il segno `$` come segue:

```
dd$x1
```

```
## [1] 0.06672103 0.48636461 0.66614263 0.77442318 0.64632866 0.37887038
```

Si possono indicare le righe e le colonne di interesse tra parentesi quadre, separandole con una virgola, secondo il formato `dataset[righe, colonne]`. Per selezionare tutte le righe ma solo alcune colonne si usa il formato `dataset[, colonne]`; vice versa, per selezionare tutte le colonne ma solo alcune righe si usa il formato `dataset[righe,]`.

Vediamo alcuni esempi.

Per selezionare le righe 1, 2, e 4:

```
dd[c(1, 2, 4), ]
```

```
##   Lettera         x1        x2
## 1       B 0.06672103 0.3551431
## 2       B 0.48636461 0.4549017
## 4       A 0.77442318 0.4470598
```

Per selezionare le colonne 1 e 3:

```
dd[, c(1, 3)]
```

```
##   Lettera        x2
## 1       B 0.3551431
## 2       B 0.4549017
## 3       A 0.5379975
## 4       A 0.4470598
## 5       C 0.6610715
## 6       C 0.6796479
```

Per selezionare le righe 1, 2, e 4 delle colonne 1 e 3:

```
dd[c(1, 2, 4), c(1, 3)]
```

```
##   Lettera        x2
## 1       B 0.3551431
## 2       B 0.4549017
## 4       A 0.4470598
```

Le colonne possono anche essere selezionate indicandole per nome:

```
dd[c(1, 2, 4), c("Lettera", "x1")]
```

```
##   Lettera         x1
## 1       B 0.06672103
## 2       B 0.48636461
## 4       A 0.77442318
```

Quest'ultima è una pratica migliore rispetto alla precedente perchè è meno prona a confusione.

La funzione `subset()` è un'altra opzione semplice e versatile:

```
dd1 <- subset(dd, subset = Lettera %in% c("A", "B"), select = -c(x1))
dd1
```

```
##   Lettera        x2
## 1       B 0.3551431
## 2       B 0.4549017
## 3       A 0.5379975
## 4       A 0.4470598
```

Consideriamo la sintassi di `subset()` nel dettaglio:

- l'argomento `subset=` permette di elencare una serie di condizioni logiche da imporre al sottoinsieme di dati. Nell'esempio ho usato `subset=` per richiedere il sottoinsieme di righe di `dd` aventi come valori della colonna `Lettera` la lettera `"A"` oppure `"B"`. L'operatore `%in%` permette di specificare i valori desiderati di un elemento (la colonna `Lettera` nell'esempio) in base a quelli elencati in un altro elemento (il vettore `c("A", "B")` nell'esempio);
- l'argomento `select=` permette di elencare le colonne desiderate o indesiderate; nel secondo caso è sufficiente anteporre un segno "-" all'elenco delle colonne (come nell'esempio).

È possibile aggiungere più di una condizione logica all'argomento `subset=` usando i simboli "&" oppure "|". Il simbolo "&" impone che sia le condizioni a sinistra del simbolo, sia quelle a destra di esso siano rispettate. Il simbolo "|" significa "oppure": richiede che *almeno* una delle condizioni a sinistra e a destra del simbolo siano rispettate. Ad esempio:

```
subset(dd, subset = Lettera %in% c("A", "B") & x2 < 1, select = -c(x1))
```

```
##   Lettera        x2
## 1       B 0.3551431
## 2       B 0.4549017
## 3       A 0.5379975
## 4       A 0.4470598
```

21.3 Unire datasets

Supponiamo di voler unire due dataset. Se le loro colonne coincidono in numero, nome, ed ordine li possiamo unire verticalmente con la funzione `rbind()`. Ad esempio, creiamo il dataset `ee`:

```
ee <- data.frame(Lettera = c("D","D"),
                 x1 = runif(n=2, min=0, max=1), # valori casuali tra 0 e 1
                 x2 = runif(n=2, min=0.5, max=2)
                 )
```

```
ee

##   Lettera        x1       x2
## 1       D 0.9599815 1.446608
## 2       D 0.6320184 1.099596
```

Usiamo `rbind()` per incolonnare i dataset `dd` ed `ee` in un unico dataset, che chiamerò `dd2`:

```
dd2 <- rbind(dd, ee)
dd2

##   Lettera         x1        x2
## 1       B 0.06672103 0.3551431
## 2       B 0.48636461 0.4549017
## 3       A 0.66614263 0.5379975
## 4       A 0.77442318 0.4470598
## 5       C 0.64632866 0.6610715
## 6       C 0.37887038 0.6796479
## 7       D 0.95998152 1.4466084
## 8       D 0.63201839 1.0995956
```

Per unire colonne o dataset affiancandoli anzichè incolonnandoli si può usare la funzione `cbind()`. Ad esempio, creiamo il vettore `x3`:

```
x3 <- runif(n = 2, min = 0, max = 1)
x3

## [1] 0.2677668 0.9947570
```

Il vettore `x3` ha due elementi al suo interno. Supponiamo di volerlo aggiungere come una nuova colonna al dataset `ee`, creando il nuovo dataset `ee2`:

```
ee2 <- cbind(ee, x3)
ee2

##   Lettera        x1       x2        x3
## 1       D 0.9599815 1.446608 0.2677668
## 2       D 0.6320184 1.099596 0.9947570
```

Naturalmente bisogna sempre assicurarsi che queste unioni abbiano senso, ovvero che i valori in ciascuna riga da unire siano ordinati in modo coerente. Per evitare il rischio di confusione appena citato, la cosa più sicura è assicurarsi che le righe dei dataset da unire abbiano delle "etichette" univoche, in modo che queste etichette si possano usare come guida logica per unire i dataset senza confusioni. Consideriamo i seguenti dataset:

```
gg <- data.frame(Lettera = c("A", "A", "B", "B"), Repliche = c("rep1", "rep2",
 "rep1",
    "rep2"), x1 = runif(n = 4, min = 0, max = 1)  # valori casuali tra 0 e 1
)
```

```
gg
```

```
##   Lettera Repliche        x1
## 1       A     rep1 0.4341751
## 2       A     rep2 0.4164858
## 3       B     rep1 0.6633822
## 4       B     rep2 0.2739113
```

```
ff <- data.frame(Lettera = c("B", "B", "A", "A"), Repliche = c("rep2", "rep1",
 "rep2",
    "rep1"), x2 = runif(n = 4, min = 0, max = 1)  # valori casuali tra 0 e 1
)
ff
```

```
##   Lettera Repliche         x2
## 1       B     rep2 0.06354999
## 2       B     rep1 0.53494022
## 3       A     rep2 0.94198048
## 4       A     rep1 0.95548448
```

Gli elementi delle colonne `Lettera` e `Repliche` non hanno lo stesso ordine nei dataset `gg` ed `ff`. La funzione `merge()` ci permette di "fondere" i due dataset usando le colonne `Lettera` e `Repliche` come "etichette" in modo da associare i valori delle colonne dei due dataset in modo univoco ai valori di `Lettera` e `Repliche`:

```
mm <- merge(gg, ff, by = c("Lettera", "Repliche"), all = T)
mm
```

```
##   Lettera Repliche        x1         x2
## 1       A     rep1 0.4341751 0.95548448
## 2       A     rep2 0.4164858 0.94198048
## 3       B     rep1 0.6633822 0.53494022
## 4       B     rep2 0.2739113 0.06354999
```

L'argomento `all=TRUE` fa sì che, nel caso in cui la corrispondenza tra le righe dei dataset da unire sia solo parziale, il dataset risultante dall'unione di due dataset conservi tutte le righe del dataset originale. Vediamone un esempio. Il seguente dataset indica se le lettere A, B, C, e D sono vocali o consonanti:

```
vv <- data.frame(Lettera = c("A", "B", "C", "D"), Tipo = c("vocale", "consona
nte",
    "consonante", "consonante"))
vv
```

```
##   Lettera       Tipo
## 1       A     vocale
## 2       B consonante
## 3       C consonante
## 4       D consonante
```

Uniamo il dataset vv, appena creato, con il dataset ff che abbiamo creato in precedenza:

```
hh <- merge(ff, vv, by = c("Lettera"), all = T)
hh
```

```
##   Lettera Repliche         x2       Tipo
## 1       A     rep2 0.94198048     vocale
## 2       A     rep1 0.95548448     vocale
## 3       B     rep2 0.06354999 consonante
## 4       B     rep1 0.53494022 consonante
## 5       C     <NA>         NA consonante
## 6       D     <NA>         NA consonante
```

Il dataset hh deriva dalla fusione di ff e vv. Le celle per le quali non ci sono informazioni disponibili riportano la sigla "NA", acronimo di *"not available"*.

21.4 Cambiare il formato dei dati da "lungo" a "largo" e vice versa

Ho già accennato alla differenza tra i formati "lungo" e "largo" dei dataset nella sezione 6. I nomi dei due formati si riferiscono alla disposizione della variabile di risposta: nel formato "lungo" i valori della variabile di risposta sono disposti in colonna; in quello "largo" sono in riga. Tutte le analisi che abbiamo esaminato in questo libro richiedono dati in formato "lungo". Esistono altre tecniche di analisi, come quelle multivariate (non trattate in questo manuale) che richiedono dati in formato "largo". Vedremo ora come trasformare i dataset da un formato all'altro usando le funzioni `pivot_longer()` e `pivot_wider()` del pacchetto `tidyr`.

Usiamo il dataset gg come esempio di dataset in formato lungo. Possiamo "allargarlo" con `pivot_wider()`:

```
library(pacman)
p_load(tidyr)
# cambiamo il formato di gg da lungo a largo:
gg_large <- pivot_wider(data = gg, id_cols = c("Lettera"), names_from = "Repl
iche",
    values_from = "x1")
gg_large
```

```
## # A tibble: 2 × 3
##   Lettera  rep1  rep2
##   <chr>   <dbl> <dbl>
## 1 A       0.434 0.416
## 2 B       0.663 0.274
```

Vediamo in dettaglio gli argomenti della funzione `pivot_wider()`:

- `id_cols=` permette di indicare la colonna da usare per identificare ciascun rigo nel dataset che stiamo generando;
- `names_from=` permette di scegliere la colonna da cui prendere i nomi di ciascuna delle colonne del dataset che stiamo generando;
- `values_from=` permette di indicare la colonna contenente i valori da usare per popolare le celle del dataset che stiamo generando.

Le funzioni del pacchetto `tidyr` producono dei dataset di classe "tibble". Si tratta di oggetti simili a quelli di classe "data.frame" e possono essere usati tali e quali o trasformati in "data.frame" con la funzione `as.data.frame()`.

Andiamo ora a trasformare il formato di `gg_large` da "largo" a "lungo" con `pivot_longer()`:

```
# cambiamo il formato di gg da lungo a largo:
gg_long <- pivot_longer(data = gg_large, cols = c("rep1", "rep2"), names_to =
 "Repliche",
    values_to = "valore")
gg_long
```

```
## # A tibble: 4 × 3
##   Lettera Repliche valore
##   <chr>   <chr>     <dbl>
## 1 A       rep1      0.434
## 2 A       rep2      0.416
## 3 B       rep1      0.663
## 4 B       rep2      0.274
```

Esaminiamo gli argomenti della funzione `pivot_longer()`:

- `cols=` permette di elencare i nomi delle colonne che vogliamo incolonnare;
- `names_to=` permette di scegliere il nome della colonna creata con `cols=`;
- `values_to=` permette di scegliere il nome della colonna che ospiterà i valori della variabile di risposta.

Con i metodi e le funzioni descritte sopra, unite ad un po' di inventiva, riuscirete a gestire le situazioni più comuni di formattazione, riorganizzazione e pulizia dei dataset.

22 Come simulare dati realistici in R

La gran maggioranza dei dataset usati in questo manuale contengono dati generati al computer (fanno eccezione quelli dell'esempio presentato in sezione 20.6). Ho scelto di utilizzare dati artificiali perchè questo mi ha permesso di creare scenari *ad hoc*, evidenziando con facilità degli aspetti specifici dell'analisi dei dati. Al fine di creare dati "sintetici" realistici ho utilizzato funzioni di R che permettono di ottenere dati provenienti da distribuzioni di frequenza con caratteristiche definite dall'utente.

Produrre dati simulati è un ottimo esercizio per capire il comportamento delle distribuzioni di frequenza e per comprendere le fonti di variabilità che influiscono sui valori assunti da una variabile. Al di fuori di un contesto didattico, generare dati tramite simulazioni può essere utile nella fase di *design* di uno studio o di un esperimento:

- obbliga a produrre un modello del sistema di studio, favorendone una comprensione quantitativa;
- aiuta a scegliere gli strumenti analitici che saranno necessari ad analizzare i dati reali *prima* di averli raccolti;
- aiuta ad individuare problemi di design sperimentale *prima* di avere svolto l'esperimento, dunque quando si è ancora in tempo a correggerli;
- permette di valutare le conseguenze della dimensione campionaria (il numero di dati a disposizione) sul risultato di una analisi statistica.

In questo capitolo vedremo alcuni esempi di simulazioni di dati in R. Potete divertirvi a riprodurre questi esempi, modificarli, e rappresentarli graficamente con gli strumenti descritti nelle sezioni precedenti.

22.1 Simulazione di variabili di risposta categoriche

Si possono estrarre valori da una popolazione caratterizzata da una distribuzione Gaussiana normale con la funzione `rnorm()`. Ad esempio, per ottenere 10 valori da una popolazione "sintetica" con media 5 e deviazione standard pari a 1 si esegue:

```
rm(list = ls())  # ripulisce la memoria di R
rnorm(n = 10, mean = 0, sd = 1)

##  [1] -2.365257889 -1.308065747 -0.335002169 -0.004971704 -0.476619782
##  [6] -0.818769860  1.424845936 -0.309981162  1.072883086  0.321861856
```

Possiamo dunque riprodurre il dataset esaminato nel capitolo 6 come segue:

```
dd <- data.frame(body.mass = c(rnorm(n = 50, mean = 100, sd = 1), rnorm(n = 5
0, mean = 110,
    sd = 1)), Species = as.factor(rep(c("A", "B"), each = 50)))
```

Vi invito a visualizzarli con `plot(body.mass~Species, data=dd)` e a confrontare il risultato con quello in figura 6.1.

22.2 Simulazione di una relazione lineare tra variabili continue

Creiamo un dataset contenente un predittore continuo e delle etichette per identificare le future repliche. Cominciamo col creare i rispettivi vettori:

```
x1 <- c(10:20)
reps <- paste("rep", 1:5)
```

Passiamo ora i vettori così creati alla funzione `expand.grid()`, che produce un `data.frame` contenente tutte le possibili combinazioni dei valori di `x1` e `reps`:

```
df <- expand.grid(predittore = x1, repliche = reps)
```

A questo punto aggiungiamo una colonna contenente le predizioni medie per ciascun valore del predittore applicando l'equazione di una retta:

```
df$risposta_media <- 10 + 3 * df$predittore
```

Infine simuliamo le predizioni osservate per ciascuna replica aggiungendo a ciascuna predizione media uno scostamento ϵ_i proveniente da una distribuzione normale con media 0 e deviazione standard `sd` di nostra scelta (ad esempio `sd=3`):

```
df$risposta <- df$risposta_media + rnorm(n = length(df[, 1]), mean = 0, sd =
3)
```

Abbiamo così creato un dataset in cui la variabile di risposta varia linearmente al variare del predittore, che consiste di una variabile continua.

22.3 Simulazione di una risposta a predittori continui e categorici

Nel caso della simulazione di una risposta ad un predittore continuo e ad uno fattoriale è necessario ripetere la simulazione descritta nella sezione 22.2 per tante volte quanti sono i livelli del predittore fattoriale, assegnando a ciascun livello un suo set di parametri (intercetta e coefficiente angolare).

Cominciamo col creare un dataset per i predittori:

```
x1 <- c(1, 10, 20)
reps <- paste("rep", 1:3)
x2 <- c("A", "B")
df <- expand.grid(predittore1 = x1, predittore2 = x2, repliche = reps)
```

Definiamo ora i valori di intercetta e di coefficiente angolare (la "pendenza") per ciascuno dei due livelli di `df$predittore2`. Si può procedere in vari modi; in questo caso ho scelto di usare la funzione `ifelse()`:

```
df$intercetta <- ifelse(df$predittore2 == "A", 25, 30)
df$pendenza <- ifelse(df$predittore2 == "A", 3, 0)
```

A questo punto possiamo calcolare i valori medi della variabile di risposta:

```
df$risposta_media <- df$intercetta + df$pendenza * df$predittore1
```

Infine simuliamo le predizioni osservate per ciascuna replica aggiungendo a ciascuna predizione media uno scostamento ϵ_i:

```
df$risposta <- df$risposta_media + rnorm(n = length(df[, 1]), mean = 0, sd =
3)
```

22.4 Simulazione di una risposta esponenziale con distribuzione poissoniana

Cominciamo col creare il dataset dei predittori, come di consueto:

```
x1 <- c(10:20)
reps <- paste("rep", 1:5)
df <- expand.grid(predittore = x1, repliche = reps)
```

In questo caso vogliamo ottenere i valori medi della variabile di risposta da una equazione esponenziale:

```
df$risposta_media <- exp(1 + 0.2 * df$predittore)
```

Infine produciamo le osservazioni simulate usando la funzione `rpois()`:

```
df$risposta <- rpois(length(df[, 1]), lambda = df$risposta_media)
```

La funzione `rpois()` produce valori secondo una distribuzione poissoniana con media pari ai valori medi attesi per la variabile di risposta (ricordiamo che in una distribuzione di Poisson la media e la varianza hanno lo stesso valore λ).

22.5 Simulazione di dati multilivello (1)

Vediamo come produrre un dataset simile a quello usato nel capitolo 20. Creiamo il dataset contenente informazioni su predittori e repliche:

```
fertilizzanti <- LETTERS[1:2]
# Immaginiamo che lo studio sia stato replicato in vari paesi (random effect):
paesi <- c("Canada", "Costa Rica", "Egitto", "Grecia", "Italia", "Mongolia",
"Portogallo")
df <- expand.grid(Fertilizzante = fertilizzanti, Paese = paesi, rep = 1:5)
df$unique_ID <- paste(df$Fertilizzante, df$Paese, df$rep, sep = "_")
df <- df[with(df, order(Fertilizzante, Paese)), ]
```

Simuliamo l'effetto dei *fixed effects*:

```
df$fixef <- ifelse(df$Fertilizzante == "A", 50, 80)
```

Aggiungiamo l'effetto dei *random effects* secondo un modello ad "intercetta random":

```
# creiamo un data.frame contenente i random effects:
re <- data.frame(Paese = unique(df$Paese))
re$ranef <- rnorm(n = length(re$Paese), mean = 0, sd = 3.5)
# 'fondiamo' re e df:
dd <- merge(df, re)
# calcoliamo l'effetto medio di fixed + random effect:
dd$raccolto_medio = dd$fixef + dd$ranef
```

Aggiungiamo degli scostamenti residui dai valori attesi a livello delle singole osservazioni:

```
dd$raccolto <- dd$raccolto_medio + rnorm(n = length(dd[, 1]), mean = 0, sd =
2.5)
# arrotondiamo al secondo decimale:
dd$raccolto <- round(dd$raccolto, 2)
```

22.6 Simulazione di dati multilivello (2)

Vediamo come creare un dataset simile a quello usato nel capitolo 20.5. Creiamo il dataset contenente informazioni su predittori e repliche:

```
Localita <- LETTERS[1:8]
rep <- paste("rep", 1:10, sep = "_")
Profondita <- 1:15
df <- expand.grid(Localita = Localita, Profondita = Profondita, rep = rep)
# diamo un nome univoco a ciascun campione:
df$unique_ID <- paste("loc", df$Localita, "depth", df$Profondita, df$rep, sep
 = "_")
```

Aggiungiamo l'effetto dei *random effects*:

```
re <- data.frame(Localita = unique(df$Localita))
re$ri <- rnorm(n = length(re$Localita), mean = 100, sd = 20)
re$rs <- rnorm(n = length(re$Localita), mean = 3, sd = 1)
dd <- merge(df, re)
# calcoliamo l'effetto medio di fixed + random effect:
dd$lunghezza_media <- dd$ri + dd$rs * dd$Profondita
```

In questo caso i *random effects* interessano sia l'intercetta, sia il coefficiente angolare: ovvero, ciascuna località ha un'intercetta ed una pendenza caratteristiche. È importante notare che tutti i valori di intercetta provengono da un'unica "popolazione" di intercette distribuite in modo gaussiano; allo stesso modo, i valori di pendenza provengono da un'unica "popolazione" di pendenze distribuite in modo gaussiano. Questa rappresenta una delle assunzioni dei modelli lineari a effetto misto.[9]

[9] Nel caso dei modelli lineari a effetto misto *generalizzati* si assume che i valori dei parametri provengano da una popolazione di parametri distribuita secondo la distribuzione specificata nel modello.

Aggiungiamo infine degli scostamenti residui dai valori attesi a livello delle singole osservazioni:

```
dd$lunghezza_alga <- dd$lunghezza_media + rnorm(n = length(dd[, 1]), mean = 0,
 sd = 5)
# arrotondiamo al secondo decimale:
dd$lunghezza_alga <- round(dd$lunghezza_alga, 2)
# mettiamo le righe in ordine alfanumerico:
dd <- with(dd, dd[order(Localita, Profondita, rep), ])
```

23 Conclusioni

Siamo giunti alla fine di questo manuale. Spero di essere riuscito a fornirvi le basi per iniziare il vostro viaggio nel mondo dell'analisi dei dati. Se ho centrato il mio obiettivo, questo manuale dovrebbe offrirvi un armamentario piuttosto vasto di strumenti per l'analisi statistica, ma è lungi dall'essere omnicomprensivo. Inoltre, al fine di mantenere il manuale in un formato agevole e conciso, mi sono limitato ad accennare a molti dettagli importanti senza troppo approfondirli. La bibliografia riportata in coda a questo libro vi offre una serie di spunti di lettura per approfondire gli argomenti introdotti in questo mio *vademecum*, sia nei loro aspetti teorici sia nella loro implementazione pratica.

Quali potrebbero essere i passi successivi della vostra formazione nell'analisi dei dati?

A seconda dei vostri interessi e delle vostre necessità di ricerca, ci sono numerosi altri ambiti statistici perseguibili in R, ad esempio: l'analisi multivariata; analisi genetiche e filogenetiche; analisi di sistemi geografici (GIS); analisi di serie storiche…

Al di là delle sue applicazioni statistiche ed analitiche R è un linguaggio di programmazione che permette di automatizzare operazioni e procedure. Ad esempio, imparare a scrivere semplici funzioni e a programmare comandi ricorsivi - tramite la famiglia di funzioni `apply()` oppure i cosiddetti *for-loops* - può essere utile per automatizzare il vostro processo di lavoro.

Buon proseguimento!

Marco

Bergamo, 4 gennaio 2026

24 Ringraziamenti

Cara lettrice, caro lettore, ti ringrazio per avermi seguito fino a questo punto e spero che questo manuale sia stato di tuo gradimento oltre che utile.

Voglio spendere qualche parola di apprezzamento verso quanti mi hanno aiutato a sviluppare le conoscenze che ho presentato in questo libro.

Lisandro Benedetti-Cecchi ed i membri del suo gruppo di ricerca sono stati i primi a fornirmi una comprensione pratica della statistica al di là di formule e teoremi.

Devo molto ad Owen Petchey per avermi aiutato a costruire solide basi nell'ambito della statistica applicata e dell'analisi dei dati. All'inizio del mio dottorato Owen mi chiese di controllare le bozze della prima edizione del suo libro *"Getting Started with R - An Introduction for Biologists"* (scritto a quattro mani con Andrew Beckerman). Quell'esercizio ebbe effetti radicali sulla mia capacità di svolgere e comprendere le analisi statistiche, nonchè sulla mia fiducia nelle mie capacità di saper imparare ad usare R, che fino ad allora mi era sembrato uno strumento affascinante ma criptico.

Ringrazio Andy Hector e vari membri del suo gruppo di ricerca all'università di Zurigo nei primi anni 2010 per i loro insegnamenti sui modelli lineari generalizzati e sui modelli lineari misti.

Ringrazio tutti gli studenti, le studentesse, le colleghe ed i colleghi che, con i loro suggerimenti e le loro domande, hanno contribuito a farmi crescere sia come ricercatore sia come insegnante. Ringrazio in particolare Grysie M. Waiguchu per avermi ingaggiato per una serie di lezioni private nel 2021. La preparazione del materiale didattico per quelle lezioni mi portò ad una generale rivalutazione critica tanto del mio metodo di insegnamento quanto del mio processo di analisi dei dati. Gli script e le presentazioni che preparai in quell'occasione sono stati il germe che ha portato prima ai miei corsi on-line *"First steps in data analysis with R"*[10] ed *"Analisi dei dati e statistica applicata in R ed RStudio"*[11], poi a questo libro.

La prima stesura di questo libro è nata dalle dispense da me preparate per un corso di analisi statistica tenuto nel 2023 a Danilo Russo, Luca Cistrone, Flavia de Benedetta e Vincenzo Meola, che ringrazio per l'entusiasmo mostrato e per gli spunti di riflessione forniti.

Questo libro è stato compilato in RStudio e generato in R con il pacchetto "bookdown" di Yihui Xie et al. (2025).

Gli eventuali errori e refusi presenti in questo libro sono di mia sola responsabilità. Se doveste trovarne, vi sarei grato se me li comunicaste scrivendo a contact@marcoplebani.com.

[10] https://bit.ly/DataAnalysisCourse

[11] https://bit.ly/AnalisiDatiR

25 Script di esempio

Questa sezione è stata rimossa dall'edizione gratuita di questo manuale. Potete acquistare l'edizione senza tagli qui:

- https://www.lulu.com/it/shop/marco-plebani/analisi-dei-dati-con-r-ed-rstudio/ebook/product-yvkp2ng.html (e-book)
- https://www.lulu.com/it/shop/marco-plebani/analisi-dei-dati-con-r-ed-rstudio/paperback/product-je6k8y6.html (in formato cartaceo)

Buona lettura!

- l'autore

26 Bibliografia

26.1 R e pacchetti di R utilizzati

- Bartoń K. (2023), *MuMIn: Multi-Model Inference*. R package version 1.47.5.
- Bates D., Mächler M., Bolker B., Walker S. (2015). *Fitting Linear Mixed-Effects Models Using lme4.* Journal of Statistical Software, 67(1), 1–48. doi:10.18637/jss.v067.i01.
- Baty F., Ritz C., Charles S., Brutsche M., Flandrois J., Delignette-Muller M. (2015), *A Toolbox for Nonlinear Regression in R: The Package nlstools.* Journal of Statistical Software, 66(5), 1–21. https://www.jstatsoft.org/article/view/v066i05
- Brooks M.E., Kristensen K., van Benthem K.J., Magnusson A., Berg C.W., Nielsen A., Skaug H.J., Maechler M., Bolker B.M. (2017), *glmmTMB Balances Speed and Flexibility Among Packages for Zero-inflated Generalized Linear Mixed Modeling.* The R Journal, 9(2), 378–400. https://doi.org/10.32614/RJ-2017-066
- Fox J., Weisberg S. (2019), *An R Companion to Applied Regression, Third edition.* Sage, Thousand Oaks CA. https://socialsciences.mcmaster.ca/jfox/Books/Companion
- Hartig F. (2022), *DHARMa: Residual Diagnostics for Hierarchical (Multi-Level / Mixed) Regression Models.* R package version 0.4.6. https://florianhartig.github.io/DHARMa
- Højsgaard S., Halekoh U. (2023), doBy: Groupwise Statistics, LSmeans, Linear Estimates, Utilities. Package version 4.6.16.
- Kassambara A. (2023), ggpubr: 'ggplot2' Based Publication Ready Plots. R package version 0.6.0.
- Kuznetsova A., Brockhoff P.B., Christensen R.H.B. (2017), *lmerTest Package: Tests in Linear Mixed Effects Models.* Journal of Statistical Software, 82(13), 1–26. https://doi.org/10.18637/jss.v082.i13
- Lang D.T. (2023), *RCurl.* R package version 1.98-1.12
- Lenth R. (2023), emmeans: Estimated Marginal Means, aka Least-Squares Means. R package version 1.8.6.
- Ligges U., Mächler M. (2003), *Scatterplot3d - an R Package for Visualizing Multivariate Data.* Journal of Statistical Software, 8(11), 1–20. https://doi.org/10.18637/jss.v008.i11
- Lüdecke D. (2018), *ggeffects: Tidy Data Frames of Marginal Effects from Regression Models.* Journal of Open Source Software, 3(26), 772. https://doi.org/10.21105/joss.00772
- Lüdecke et al. (2021), *performance: An R Package for Assessment, Comparison and Testing of Statistical Models.* Journal of Open Source Software, 6(60), 3139. https://doi.org/10.21105/joss.03139
- Luštrik R. & Joseph Stachelek J. (2015) *Package Accompanying 'Mixed Effects Models and Extensions in Ecology with R'.* R package AED version 1.5. https://github.com/romunov/AED/
- Murdoch D. et al. (2023), *rgl: 3D Visualization Using OpenGL.* R package version 1.1.3.
- Padfield D., Matheson G. (2020), nls.multstart: Robust Non-Linear Regression using AIC Scores. R package version 1.2.0.

- R Core Team (2025), *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. https://www.R-project.org
- Rinker T.W., Kurkiewicz D. (2018), *pacman: Package Management for R.* R package version 0.5.
- Wickham H. (2011), *The Split-Apply-Combine Strategy for Data Analysis.* Journal of Statistical Software, 40(1), 1–29. https://www.jstatsoft.org/v40/i01/
- Wickham H. (2016), *ggplot2: Elegant Graphics for Data Analysis.* Springer-Verlag New York. ISBN 978-3-319-24277-4, https://ggplot2.tidyverse.org
- Wickham H., Bryan J. (2023), *readxl: Read Excel Files.* https://readxl.tidyverse.org
- Wickham H., Hester J., Chang W., Bryan J. (2022). *devtools: Tools to Make Developing R Packages Easier.* R package version 2.4.5. https://devtools.r-lib.org
- Wilke C.O. (2022), *ggridges: Ridgeline Plots in 'ggplot2'.* R package version 0.5.4. https://wilkelab.org/ggridges
- Yihui Xie (2025), *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version version 0.44. https://yihui.org/bookdown/

26.2 Letteratura citata o consigliata

- Akaike H. (1985), *Prediction and entropy.* In: *A Celebration of Statistics*, curated by Atkinson, A. C. and Fienberg, S. E. Springer, pp. 1–24.
- Beckerman A.P., Petchey O.L. (2012), Getting Started with R: An Introduction for Biologists. OUP Oxford.
- Bolker M.M., Brooks M.E., Clark C.J., Geange S.W., Poulsen J.R., Stevens M.H.H., and White J.S. (2009). *Generalized Linear Mixed Models: A Practical Guide for Ecology and Evolution.* Trends in Ecology & Evolution 24, no. 3 (March 1, 2009): 127–35. https://doi.org/10.1016/j.tree.2008.10.008
- Benedetti-Cecchi L. (2004), *Experimental design and hypothesis testing in ecology.* In: Biologia Marina Mediterranea, 11 (Suppl. 1): 407-455.
- Gotelli N.J. and Ellison A.M. (2012), *A Primer of Ecological Statistics.* Second Edition. Oxford University Press.
- Harrison X.A., Donaldson L., Correa-Cano M.E., Evans J., Fisher D.N., Goodwin C.E.D., Robinson B.S., Hodgson D.J., Inger R. (2018), *A Brief Introduction to Mixed Effects Modelling and Multi-Model Inference in Ecology.* PeerJ 6 (May 23, 2018): e4794. https://doi.org/10.7717/peerj.4794
- Janssen, G. and Mulder, S. (2005), Zonation of macrofauna across sandy beaches and surf zones along the Dutch coast. Oceanologia 47(2): 265-282.
- McFadden, D. (1973). *Conditional logit analysis of qualitative choice behavior.* In: Zarembka, P., Ed., Frontiers in Econometrics, Academic Press, 105-142.
- Underwood, A.J. (1996), *Experiments in Ecology: Their Logical Design and Interpretation Using Analysis of Variance.* Cambridge: Cambridge University Press. https://doi.org/10.1017/CBO9780511806407

- Schielzeth H., Nakagawa S. (2013), *Nested by Design: Model Fitting and Interpretation in a Mixed Model Era.* Methods in Ecology and Evolution 4, no. 1 (2013): 14–24. https://doi.org/10.1111/j.2041-210x.2012.00251.x
- Scholer F. *Anova – Type I/II/III SS explained.* https://archive.fo/ZqRTy
- Zuur A.F., Hilbe J.M. and Ieno E.N. (2013), A Beginner's Guide to GLM and GLMM with R: A Frequentist and Bayesian Perspective for Ecologists. Highland Statistics Ltd.
- Zuur, A.F., Ieno E.N., and Elphick C.S. (2010), *A Protocol for Data Exploration to Avoid Common Statistical Problems.* Methods in Ecology and Evolution 1, no. 1 (2010): 3–14. https://doi.org/10.1111/j.2041-210X.2009.00001.x
- Zuur, A.F., Ieno, E.N., and Smith, G.M. (2007), *Analysing ecological data.* Springer Science & Business Media
- Zuur, A.F., Ieno E.N., Walker N., Saveliev A.A., and Smith G.M. (2009), *Mixed Effects Models and Extensions in Ecology with R.* Statistics for Biology and Health. Springer-Verlag, New York. https://doi.org/10.1007/978-0-387-87458-6
- Zuur A.F., Saveliev A.A., Ieno E.N. (2012), *Zero Inflated Models and Generalized Linear Mixed Models with R.* Highland Statistics Ltd.

27 L’autore

Marco Plebani ha studiato all’Università di Pisa conseguendo una laurea triennale in Scienze Ecologiche e della Biodiversità ed una laurea specialistica in Biologia Marina. Ha poi conseguito un dottorato in Ecologia presso l’Università di Zurigo con una tesi sugli effetti della temperatura sulle dinamiche ecologiche dei microrganismi d’acqua dolce. Successivamente si è trasferito in Sud Africa per studiare le relazioni ecologiche tra piante ed impollinatori ed i loro effetti sulle loro reciproche traiettorie evolutive.

L’autore ha accumulato più di dieci anni di esperienza nella pratica e l’insegnamento della biostatistica e dell’analisi dei dati, collaborando a progetti di ricerca su vari argomenti, tra cui: dinamiche di popolazione, struttura di comunità ecologiche, biodiversità, evoluzione e coevoluzione, e biogeografia.

www.marcoplebani.com