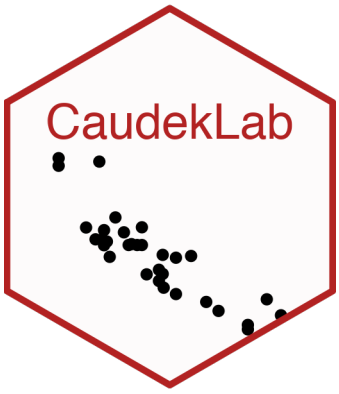
Psicometria

Corrado Caudek

2021-11-30

Questo documento è stato realizzato con:

- LaTeX e la classe memoir (http://www.ctan.org/pkg/memoir);
- R (http://www.r-project.org/) e RStudio (http://www.rstudio.com/);
- bookdown (http://bookdown.org/) e memoiR (https://ericmarcon.github.io/memoiR/).



Nel blog della mia pagina personale sono forniti alcuni approfondimenti degli argomenti qui trattati.

https://ccaudek.github.io/caudeklab/

Indice

Data della versione presente: Novembre 30, 2021.

Prefazione

Data Science per psicologi contiene il materiale delle lezioni dell'insegnamento di *Psicometria B000286* (A.A. 2021/2022) rivolto agli studenti del primo anno del Corso di Laurea in Scienze e Tecniche Psicologiche dell'Università degli Studi di Firenze.

L'insegnamento di Psicometria si propone di fornire agli studenti un'introduzione all'analisi dei dati in psicologia. Le conoscenze/competenze che verranno sviluppate in questo insegnamento sono quelle della *Data science*, ovvero le conoscenze/competenze che si pongono all'intersezione tra statistica (ovvero, richiedono la capacità di comprendere teoremi statistici) e informatica (ovvero, richiedono la capacità di sapere utilizzare un software).

La psicologia e la Data Science

> It's worth noting, before getting started, that this material is hard. If you find yourself confused at any point, you are normal. Any sense of confusion you feel is just your brain correctly calibrating to the subject matter. Over time, confusion is replaced by comprehension [...] — Richard McElreath

Sembra sensato spendere due parole su un tema che è importante per gli studenti: quello indicato dal titolo di questo Capitolo. È ovvio che agli studenti di psicologia la statistica non piace. Se piacesse, forse studierebbero Data Science e non psicologia; ma non lo fanno. Di conseguenza, gli studenti di psicologia si chiedono: "perché dobbiamo perdere tanto tempo a studiare queste cose quando in realtà quello che ci interessa è tutt'altro?'' Questa è una bella domanda.

C'è una ragione molto semplice che dovrebbe farci capire perché la Data Science è così importante per la psicologia. Infatti, a ben pensarci, la psicologia è una disciplina intrinsecamente statistica, se per statistica intendiamo quella disciplina che studia la variazione delle caratteristiche degli individui nella popolazione. La psicologia studia *gli individui* ed è proprio la variabilità inter- e intra-individuale ciò che vogliamo descrivere e, in certi casi, predire. In questo senso, la psicologia è molto diversa dall'ingegneria, per esempio. Le proprietà di un determinato ponte sotto certe condizioni, ad esempio, sono molto simili a quelle di un altro ponte, sotto le medesime condizioni. Quindi, per un ingegnere la statistica è poco importante: le proprietà dei materiali sono unicamente dipendenti dalla loro composizione e restano costanti. Ma lo stesso non può dirsi degli individui: ogni individuo è unico e cambia nel tempo. E le variazioni tra gli individui, e di un individuo nel tempo, sono l'oggetto di studio proprio della psicologia: è dunque chiaro che i problemi che la psicologia si pone sono molto diversi da quelli affrontati, per esempio, dagli ingegneri. Questa è la ragione per cui abbiamo tanto bisogno della *data science* in psicologia: perché la *data science* ci consente di descrivere la variazione e il cambiamento. E queste sono appunto le caratteristiche di base dei fenomeni psicologici.

Sono sicuro che, leggendo queste righe, a molti studenti sarà venuta in mente la seguente domanda: perché non chiediamo a qualche esperto di fare il "lavoro sporco" (ovvero le analisi statistiche) per noi, mentre noi (gli psicologi) ci occupiamo solo di ciò che ci interessa, ovvero dei problemi psicologici slegati dai dettagli "tecnici" della *data science*? La risposta a questa domanda è che non è possibile progettare uno studio psico-

logico sensato senza avere almeno una comprensione rudimentale della *data science*. Le tematiche della *data science* non possono essere ignorate né dai ricercatori in psicologia né da coloro che svolgono la professione di psicologo al di fuori dell'Università. Infatti, anche i professionisti al di fuori dall'università non possono fare a meno di leggere la letteratura psicologica più recente: il continuo aggiornamento delle conoscenze è infatti richiesto dalla deontologia della professione. Ma per potere fare questo è necessario conoscere un bel po' di *data science*! Basta aprire a caso una rivista specialistica di psicologia per rendersi conto di quanto ciò sia vero: gli articoli che riportano i risultati delle ricerche psicologiche sono zeppi di analisi statistiche e di modelli formali. E la comprensione della letteratura psicologica rappresenta un requisito minimo nel bagaglio professionale dello psicologo.

Le considerazioni precedenti cercano di chiarire il seguente punto: la *data science* non è qualcosa da studiare a malincuore, in un singolo insegnamento universitario, per poi poterla tranquillamente dimenticare. Nel bene e nel male, gli psicologi usano gli strumenti della *data science* in tantissimi ambiti della loro attività professionale: in particolare quando costruiscono, somministrano e interpretano i test psicometrici. È dunque chiaro che possedere delle solide basi di *data science* è un tassello imprescindibile del bagaglio professionale dello psicologo. In questo insegnamento verrano trattati i temi base della *data science* e verrà adottato un punto di vista bayesiano, che corrisponde all'approccio più recente e sempre più diffuso in psicologia.

Come studiare

> I know quite certainly that I myself have no special talent. Curiosity, obsession and dogged endurance, combined with self-criticism, have brought me to my ideas. — Albert Einstein

Il giusto metodo di studio per prepararsi all'esame di Psicometria è quello di seguire attivamente le lezioni, assimilare i concetti via via che essi vengono presentati e verificare in autonomia le procedure presentate a lezione. Incoraggio gli studenti a farmi domande per chiarire ciò che non è stato capito appieno. Incoraggio gli studenti a utilizzare i forum attivi su Moodle e, soprattutto, a svolgere gli esercizi proposti su Moodle. I problemi forniti su Moodle rappresentano il livello di difficoltà richiesto per superare l'esame e consentono allo studente di comprendere se le competenze sviluppate fino a quel punto sono sufficienti rispetto alle richieste dell'esame.

La prima fase dello studio, che è sicuramente individuale, è quella in cui è necessario acquisire le conoscenze teoriche relative ai problemi che saranno presentati all'esame. La seconda fase di studio, che può essere facilitata da scambi con altri e da incontri di gruppo, porta ad acquisire la capacità di applicare le conoscenze: è necessario capire come usare un software (R) per applicare i concetti statistici alla specifica situazione del problema che si vuole risolvere. Le due fasi non sono però separate: il saper fare molto spesso ci aiuta a capire meglio.

Sviluppare un metodo di studio efficace

> Memorization is not learning. — Richard Phillips Feynman

Avendo insegnato molte volte in passato un corso introduttivo di analisi dei dati ho notato nel corso degli anni che gli studenti con l'atteggiamento mentale che descriverò qui sotto generalmente ottengono ottimi risultati. Alcuni studenti sviluppano naturalmente questo approccio allo studio, ma altri hanno bisogno di fare uno sforzo per maturarlo. Fornisco qui sotto una breve descrizione del "metodo di studio' " che, nella mia esperienza, è il più efficace per affrontare le richieste di questo insegnamento (Burger & Starbird, 2012).

- Dedicate un tempo sufficiente al materiale di base, apparentemente facile; assicuratevi di averlo capito bene. Cercate le lacune nella vostra comprensione. Leggere presentazioni diverse dello stesso materiale (in libri o articoli diversi) può fornire nuove intuizioni.

- Gli errori che facciamo sono i nostri migliori maestri. Istintivamente cerchiamo di dimenticare subito i nostri errori. Ma il miglior modo di imparare è apprendere dagli errori che commettiamo. In questo senso, una soluzione corretta è meno utile di una soluzione sbagliata. Quando commettiamo un errore questo ci fornisce un'informazione importante: ci fa capire qual è il materiale di studio sul quale dobbiamo ritornare e che dobbiamo capire meglio.

- C'è ovviamente un aspetto "psicologico" nello studio. Quando un esercizio o problema ci sembra incomprensibile, la cosa migliore da fare è dire: "mi arrendo", "non ho idea di cosa fare!". Questo ci rilassa: ci siamo già arresi, quindi non abbiamo niente da perdere, non dobbiamo più preoccuparci. Ma non dobbiamo fermarci qui. Le cose "migliori" che faccio (se ci sono) le faccio quando non ho voglia di lavorare. Alle volte, quando c'è qualcosa che non so fare e non ho idea di come affontare, mi dico: "oggi non ho proprio voglia di fare fatica", non ho voglia di mettermi nello stato mentale per cui "in 10 minuti devo risolvere il problema perché dopo devo fare altre cose". Però ho voglia di *divertirmi* con quel problema e allora mi dedico a qualche aspetto "marginale" del problema, che so come affrontare, oppure considero l'aspetto più difficile del problema, quello che non so come risolvere, ma invece di cercare di risolverlo, guardo come altre persone hanno affrontato problemi simili, opppure lo stesso problema in un altro contesto. Non mi pongo l'obiettivo "risolvi il problema in 10 minuti", ma invece quello di farmi un'idea "generale" del problema, o quello di capire un caso più specifico e più semplice del problema. Senza nessuna pressione. Infatti, in quel momento ho deciso di non lavorare (ovvero, di non fare fatica). Va benissimo se "parto per la tangente", ovvero se mi metto a leggere del materiale che sembra avere poco a che fare con il problema centrale (le nostre intuizioni e la nostra curiosità solitamente ci indirizzano sulla strada giusta). Quando faccio così, molto spesso trovo la soluzione del problema che mi ero posto e, paradossalmente, la trovo in un tempo minore di quello che, in precedenza, avevo dedicato a "lavorare" al problema. Allora perché non faccio sempre così? C'è ovviamente l'aspetto dei "10 minuti" che non è sempre facile da dimenticare. Sotto pressione, possiamo solo agire in maniera automatica, ovvero possiamo solo applicare qualcosa che già sappiamo fare. Ma se dobbiamo imparare qualcosa di nuovo, la pressione è un impedimento.

- È utile farsi da soli delle domande sugli argomenti trattati, senza limitarsi a cercare di risolvere gli esercizi che vengono assegnati. Quando studio qualcosa mi viene in mente: "se questo è vero, allora deve succedere quest'altra cosa". Allora verifico se questo è vero, di solito con una simulazione. Se i risultati della simulazione sono quelli che mi aspetto, allora vuol dire che ho capito. Se i risultati sono diversi da quelli che mi aspettavo, allora mi rendo conto di non avere capito e ritorno indietro a studiare con più attenzione la teoria che pensavo di avere capito – e ovviamente mi rendo conto che c'era un aspetto che avevo frainteso. Questo tipo di verifica è qualcosa che dobbiamo fare da soli, in prima persona: nessun altro può fare questo al posto nostro.

- Non aspettatevi di capire tutto la prima volta che incontrate un argomento nuovo.[1] È utile farsi una nota mentalmente delle lacune nella vostra comprensione e tornare su di esse in seguito per carcare di colmarle. L'atteggiamento naturale, quando non capiamo i dettagli di qualcosa, è quello di pensare: "non importa, ho capito

[1] Ricordatevi inoltre che gli individui tendono a sottostimare la propria capacità di apprendere (Horn & Loewenstein, 2021).

in maniera approssimativa questo punto, non devo preoccuparmi del resto". Ma in realtà non è vero: se la nostra comprensione è superficiale, quando il problema verrà presentato in una nuova forma, non riusciremo a risolverlo. Per cui i dubbi che ci vengono quando studiamo qualcosa sono il nostro alleato più prezioso: ci dicono esattamente quali sono gli aspetti che dobbiamo approfondire per potere migliorare la nostra preparazione.

- È utile sviluppare una visione d'insieme degli argomenti trattati, capire l'obiettivo generale che si vuole raggiungere e avere chiaro il contributo che i vari pezzi di informazione forniscono al raggiungimento di tale obiettivo. Questa organizzazione mentale del materiale di studio facilita la comprensione. È estremamente utile creare degli schemi di ciò che si sta studiando. Non aspettate che sia io a fornirvi un riepilogo di ciò che dovete imparare: sviluppate da soli tali schemi e tali riassunti.
- Tutti noi dobbiamo imparare l'arte di trovare le informazioni, non solo nel caso di questo insegnamento. Quando vi trovate di fronte a qualcosa che non capite, o ottenete un oscuro messaggio di errore da un software, ricordatevi: "Google is your friend".

Corrado Caudek

Febbraio 2022

0.1 Manipolazione dei dati

Motivazione

Si chiamano "dati grezzi" quelli che provengono dal mondo circostanze, i dati raccolti per mezzo degli strumenti usati negli esperimenti, per mezzo di interviste, di questionari, ecc. Questi dati (chiamati *dataset*) raramente vengono forniti con una struttura logica precisa. Per potere elaborarli mediante dei software dobbiamo prima trasformarli in maniera tale che abbiano una struttura logica organizzata. La struttura che solitamente si utilizza è quella tabellare (matrice dei dati), ovvero si dispongono i dati in una tabella nella quale a ciascuna riga corrisponde ad un'osservazione e ciascuna colonna corrisponde ad una variabile rilevata. In `R` una tale struttura è chiamata *data frame*.

Utilizzando i pacchetti del `tidyverse` (`tidyverse` è un insieme, o *bundle*, di pacchetti `R`), le operazioni di trasformazione dei dati risultano molto semplificate. Nel `tidyverse` i data frame vengono leggermente modificati e si chiamano `tibble`. Per la manipolazione dei dati vengono usati i seguenti pacchetti del `tidyverse`:

- `dplyr`
- `tidyr` (tibbles, dataframe e tabelle)
- `stringr` (stringhe)

Il pacchetto `dplyr` (al momento uno dei pacchetti più famosi e utilizzati per la gestione dei dati) offre una serie di funzionalità che consentono di eseguire le operazioni più comuni di manipolazione dei dati in maniera più semplice rispetto a quanto succeda quando usiamo le funzioni base di `R`.

Trattamento dei dati con `dplyr`

Il pacchetto `dplyr` include sei funzioni base: `filter()`, `select()`, `mutate()`, `arrange()`, `group_by()` e `summarise()`. Queste sei funzioni costituiscono i *verbi* del linguaggio di manipolazione dei dati. A questi sei verbi si aggiunge il pipe `%>%` che serve a concatenare più operazioni. In particolare, considerando una matrice osservazioni per variabili, `select()` e `mutate()` si occupano di organizzare le variabili, `filter()` e `arrange()` i casi, e `group_by()` e `summarise()` i gruppi.

Per introdurre le funzionalità di `dplyr`, utilizzeremo i dati `msleep` forniti dal pacchetto `ggplot2`. Tali dati descrivono le ore di sonno medie di 83 specie di mammiferi (Savage et al., 2007). Carichiamo il *boundle* `tidyverse` (che contiene `ggplot2`) e leggiamo nella memoria di lavoro l'oggetto `msleep`:

```
library("tidyverse")
data(msleep)
dim(msleep)
#> [1] 83 11
```

Operatore pipe

Prima di presentare le funzionalità di `dplyr`, introduciamo l'operatore pipe `%>%` del pacchetto `magrittr` – ma ora presente anche in base `R` nella versione `|>`. L'operatore pipe, `%>%` o `|>`, serve a concatenare varie funzioni insieme, in modo da inserire un'operazione dietro l'altra. Una spiegazione intuitiva dell'operatore pipe è stata fornita in un tweet di `@andrewheiss`. Consideriamo la seguente istruzione in pseudo-codice `R`:

```
leave_house(get_dressed(get_out_of_bed(wake_up(me, time = "8:00"), side = "correct"),
pants = TRUE, shirt = TRUE), car = TRUE, bike = FALSE)
```

Il listato precedente descrive una serie di (pseudo) funzioni concatenate, le quali costituiscono gli argomenti di altre funzioni. Scritto così, il codice è molto difficile da capire. Possiamo però ottenere lo stesso risultato utilizzando l'operatore pipe che facilita la leggibilità del codice:

```
me %>%
  wake_up(time = "8:00") %>%
  get_out_of_bed(side = "correct") %>%
  get_dressed(pants = TRUE, shirt = TRUE) %>%
  leave_house(car = TRUE, bike = FALSE)
```

In questa seconda versione del (pseudo) codice `R` si capisce molto meglio ciò che vogliamo fare. Il `tibble me` viene passato alla funzione `wake_up()`. La funzione `wake_up()` ha come argomento l'ora del giorno: `time = "8:00"`. Una volta "svegliati" (wake up) dobbiamo scendere dal letto. Quindi l'output di `wake_up()` viene passato alla funzione `get_out_of_bed()` la quale ha come argomento `side = "correct"` perché vogliamo scendere dal letto dalla parte giusta. E così via.

Questo pseudo-codice chiarisce il significato dell'operatore pipe. L'operatore `%>%` viene utilizzato quando abbiamo una serie di funzioni concatenate. Per concatenazione di funzioni si intende una serie di funzioni nelle quali l'output di una funzione costituisce l'input della funzione successiva. L'operatore pipe è "syntactic sugar" per una serie di chiamate di funzioni concatenate, ovvero, detto in altre parole, consente di definire la concatenazione tra una serie di funzioni nelle quali il risultato (output) di una funzione viene utilizzato come l'input di una funzione successiva.

Estrarre una singola colonna con `pull()`

Ritorniamo ora all'esempio precedente. Iniziamo a trasformare il data frame `msleep` in un `tibble` (che è identico ad un data frame ma viene stampato sulla console in un modo diverso):

```
msleep <- tibble(msleep)
```

Estraiamo da `msleep` la variabile `sleep_total` usando il verbo `pull()`:

```
msleep %>%
  pull(sleep_total)
#>  [1] 12.1 17.0 14.4 14.9  4.0 14.4  8.7  7.0 10.1  3.0  5.3  9.4 10.0 12.5 10.3
#> [16]  8.3  9.1 17.4  5.3 18.0  3.9 19.7  2.9  3.1 10.1 10.9 14.9 12.5  9.8  1.9
#> [31]  2.7  6.2  6.3  8.0  9.5  3.3 19.4 10.1 14.2 14.3 12.8 12.5 19.9 14.6 11.0
#> [46]  7.7 14.5  8.4  3.8  9.7 15.8 10.4 13.5  9.4 10.3 11.0 11.5 13.7  3.5  5.6
#> [61] 11.1 18.1  5.4 13.0  8.7  9.6  8.4 11.3 10.6 16.6 13.8 15.9 12.8  9.1  8.6
#> [76] 15.8  4.4 15.6  8.9  5.2  6.3 12.5  9.8
```

Selezionare più colonne con `select()`

Se vogliamo selezionare da `msleep` un insieme di variabili, ad esempio `name`, `vore` e `sleep_total`, possiamo usare il verbo `select()`:

```
dt <- msleep %>%
  dplyr::select(name, vore, sleep_total)
dt
#> # A tibble: 83 x 3
#>   name                      vore  sleep_total
#>   <chr>                     <chr>       <dbl>
```

```
#> 1 Cheetah                    carni       12.1
#> 2 Owl monkey                 omni        17
#> 3 Mountain beaver            herbi       14.4
#> 4 Greater short-tailed shrew omni        14.9
#> 5 Cow                        herbi        4
#> 6 Three-toed sloth           herbi       14.4
#> # ... with 77 more rows
```

laddove la sequenza di istruzioni precedenti significa che abbiamo passato `msleep` alla funzione `select()` contenuta nel pacchetto `dplyr` e l'output di `select()` è stato salvato (usando l'operatore di assegnazione, ←) nell'oggetto `dt`. Alla funzione `select()` abbiamo passato gli argomenti `name`, `vore` e `sleep_total`.

Filtrare le osservazioni (righe) con `filter()`

Il verbo `filter()` consente di selezionare da un `tibble` un sottoinsieme di righe (osservazioni). Per esempio, possiamo selezionare tutte le osservazioni nella variabile `vore` contrassegnate come `carni` (ovvero, tutti i carnivori):

```
dt %>%
  dplyr::filter(vore == "carni")
#> # A tibble: 19 x 3
#>   name                 vore  sleep_total
#>   <chr>                <chr>       <dbl>
#> 1 Cheetah              carni        12.1
#> 2 Northern fur seal    carni         8.7
#> 3 Dog                  carni        10.1
#> 4 Long-nosed armadillo carni        17.4
#> 5 Domestic cat         carni        12.5
#> 6 Pilot whale          carni         2.7
#> # ... with 13 more rows
```

Per utilizzare il verbo `filter()` in modo efficace è neccessario usare gli operatori relazionali (Tabella 0.1) e gli operatori logici (Tabella 0.2) di `R`. Per un approfondimento, si veda il Capitolo Comparisons di *R for Data Science*.

Tabella 0.1: *Operatori relazionali.*

uguale	==
diverso	!=
minore	<
maggiore	>
minore o uguale	<=
maggiore o uguale	>=

Tabella 0.2: *Operatori logici.*

AND	&
OR	|
NOT	!

Creare una nuova variabile con `mutate()`

Talvolta vogliamo creare una nuova variabile, per esempio, sommando o dividendo due variabili, oppure calcolandone la media. A questo scopo si usa il verbo `mutate()`. Per esempio, se vogliamo esprimere i valori di `sleep_total` in minuti, moltiplichiamo per 60:

```
dt %>%
  mutate(
    sleep_minutes = sleep_total * 60
  ) %>%
  dplyr::select(sleep_total, sleep_minutes)
#> # A tibble: 83 x 2
#>   sleep_total sleep_minutes
#>         <dbl>         <dbl>
#> 1        12.1           726
#> 2        17            1020
#> 3        14.4           864
#> 4        14.9           894
#> 5         4             240
#> 6        14.4           864
#> # ... with 77 more rows
```

Ordinare i dati con `arrange()`

Il verbo `arrange()` ordina i dati in base ai valori di una o più variabili. Per esempio, possiamo ordinare la variabile `sleep_total` dal valore più alto al più basso in questo modo:

```
dt %>%
  arrange(
    desc(sleep_total)
  )
#> # A tibble: 83 x 3
#>   name                  vore    sleep_total
#>   <chr>                 <chr>         <dbl>
#> 1 Little brown bat      insecti        19.9
#> 2 Big brown bat         insecti        19.7
#> 3 Thick-tailed opposum  carni          19.4
#> 4 Giant armadillo       insecti        18.1
#> 5 North American Opossum omni          18
#> 6 Long-nosed armadillo  carni          17.4
#> # ... with 77 more rows
```

Raggruppare i dati con `group_by()`

Il verbo `group_by()` raggruppa insieme i valori in base a una o più variabili. Lo vedremo in uso in seguito insieme a `summarise()`.

Sommario dei dati con `summarise()`

Il verbo `summarise()` collassa il dataset in una singola riga dove viene riportato il risultato della statistica richiesta. Per esempio, la media del tempo totale del sonno è

```
dt %>%
  summarise(
    m_sleep = mean(sleep_total, na.rm = TRUE)
```

```
  )
#> # A tibble: 1 x 1
#>   m_sleep
#>     <dbl>
#> 1    10.4
```

Operazioni raggruppate

Sopra abbiamo visto come i mammiferi considerati dormano, in media, 10.4 ore al giorno. Troviamo ora il sonno medio in funzione di `vore`:

```
dt %>%
  group_by(vore) %>%
  summarise(
    m_sleep = mean(sleep_total, na.rm = TRUE),
    n = n()
  )
#> # A tibble: 5 x 3
#>   vore    m_sleep     n
#>   <chr>     <dbl> <int>
#> 1 carni     10.4     19
#> 2 herbi      9.51    32
#> 3 insecti   14.9      5
#> 4 omni      10.9     20
#> 5 <NA>      10.2      7
```

Si noti che, nel caso di 7 osservazioni, il valore di `vore` non era specificato. Per tali osservazioni, dunque, la classe di appartenenza è `NA`.

Applicare una funzione su più colonne: `across()`

È spesso utile eseguire la stessa operazione su più colonne, ma copiare e incollare è sia noioso che soggetto a errori:

```
df %>%
  group_by(g1, g2) %>%
  summarise(a = mean(a), b = mean(b), c = mean(c), d = mean(d))
```

In tali circostanze è possibile usare la funzione `across()` che consente di riscrivere il codice precedente in modo più succinto:

```
df %>%
  group_by(g1, g2) %>%
  summarise(across(a:d, mean))
```

Per i dati presenti, ad esempio, possiamo avere:

```
msleep %>%
  group_by(vore) %>%
  summarise(across(starts_with("sleep"), ~ mean(.x, na.rm = TRUE)))
#> # A tibble: 5 x 4
#>   vore    sleep_total sleep_rem sleep_cycle
#>   <chr>         <dbl>     <dbl>       <dbl>
#> 1 carni         10.4       2.29       0.373
#> 2 herbi          9.51      1.37       0.418
```

```
#> 3 insecti       14.9       3.52       0.161
#> 4 omni          10.9       1.96       0.592
#> 5 <NA>          10.2       1.88       0.183
```

Dati categoriali in R

Consideriamo una variabile che descrive il genere e include le categorie `male`, `female` e `non-conforming`. In R, ci sono due modi per memorizzare queste informazioni. Uno è usare la classe *character strings* e l'altro è usare la classe *factor*. Non ci addentrimo qui nelle sottigliezze di questa distinzione, motivata in gran parte per le necessità della programmazione con le funzioni di `tidyverse`. Per gli scopi di questo insegnamento sarà sufficiente codificare le variabili qualitative usando la classe *factor*. Una volta codificati i dati qualitativi utilizzando la classe *factor*, si pongono spesso due problemi:

1. modificare le etichette dei livelli (ovvero, le modalità) di un fattore,
2. riordinare i livelli di un fattore.

Modificare le etichette dei livelli di un fattore

Esaminiamo l'esempio seguente.

```
f_1 <- c("old_3", "old_4", "old_1", "old_1", "old_2")
f_1 <- factor(f_1)
y <- 1:5
df <- tibble(f_1, y)
df
#> # A tibble: 5 x 2
#>   f_1       y
#>   <fct> <int>
#> 1 old_3     1
#> 2 old_4     2
#> 3 old_1     3
#> 4 old_1     4
#> 5 old_2     5
```

Supponiamo ora di volere che i livelli del fattore `f_1` abbiano le etichette `new_1`, `new_2`, ecc. Per ottenere questo risultato usiamo la funzione `forcats::fct_recode()`:

```
df <- df %>%
  mutate(f_1 =
    forcats::fct_recode(
      f_1,
      "new_poco" = "old_1",
      "new_medio" = "old_2",
      "new_tanto" = "old_3",
      "new_massimo" = "old_4"
      )
   )
df
#> # A tibble: 5 x 2
#>   f_1             y
#>   <fct>       <int>
#> 1 new_tanto       1
#> 2 new_massimo     2
#> 3 new_poco        3
```

```
#> 4 new_poco        4
#> 5 new_medio       5
```

Riordinare i livelli di un fattore

Spesso i livelli dei fattori hanno un ordinamento naturale. Quindi, gli utenti devono avere un modo per imporre l'ordine desiderato sulla codifica delle loro variabili qualitative. Se per qualche motivo vogliamo ordinare i livelli `f_1` in ordine inverso, ad esempio, possiamo procedere nel modo seguente.

```
df$f_1 <- factor(df$f_1,
  levels = c(
    "new_massimo", "new_tanto", "new_medio", "new_poco"
  )
)
summary(df$f_1)
#> new_massimo   new_tanto   new_medio    new_poco
#>           1           1           1           2
```

Per approfondire le problematiche della manipolazione di variabili qualitative in `R`, si veda McNamara e Horton (2018).

Creare grafici con `ggplot2()`

Il pacchetto `ggplot2()` è un potente strumento per rappresentare graficamente i dati. Le iniziali del nome, `gg`, si riferiscono alla "Grammar of Graphics", che è un modo di pensare le figure come una serie di layer stratificati. Originariamente descritta da Wilkinson (2012), la grammatica dei grafici è stata aggiornata e applicata in `R` da Hadley Wickham, il creatore del pacchetto.

La funzione da cui si parte per inizializzare un grafico è `ggplot()`. La funzione `ggplot()` richiede due argomenti. Il primo è l'oggetto di tipo data.frame che contiene i dati da visualizzare – in alternativa al primo argomento, un dataframe può essere passato a `ggplot()` mediante l'operatore pipe. Il secondo è una particolare lista che viene generata dalla funzione `aes()`, la quale determina l'aspetto (*aesthetic*) del grafico. La funzione `aes()` richiede necessariamente di specificare "x" e "y", ovvero i nomi delle colonne del data.frame che è stato utilizzato quale primo argomento di `ggplot()` (o che è stato passato da pipe), le quali rappresentano le variabili da porre rispettivamente sugli assi orizzontale e verticale.

La definizione della tipologia di grafico e i vari parametri sono poi definiti successivamente, aggiungendo all'oggetto creato da `ggplot()` tutte le componenti necessarie. Saranno quindi altre funzioni, come `geom_bar()`, `geom_line()` o `geom_point()` a occuparsi di aggiungere al livello di base barre, linee, punti, e così via. Infine, tramite altre funzioni, ad esempio `labs()`, sarà possibile definire i dettagli più fini.

Gli elementi grafici (bare, punti, segmenti, ...) usati da `ggplot2` sono chiamati `geoms`. Mediante queste funzioni è possibile costruire diverse tipologie di grafici:

- `geom_bar()`: crea un layer con delle barre;
- `geom_point()`: crea un layer con dei punti (diagramma a dispersione);
- `geom_line()`: crea un layer con una linea retta;
- `geom_histogram()`: crea un layer con un istogramma;
- `geom_boxplot()`: crea un layer con un box-plot;
- `geom_errorbar()`: crea un layer con barre che rappresentano intervalli di confidenza;

- `geom_hline()` e `geom_vline()` : crea un layer con una linea orizzontale o verticale definita dall'utente.

Un comando generico ha la seguente forma:

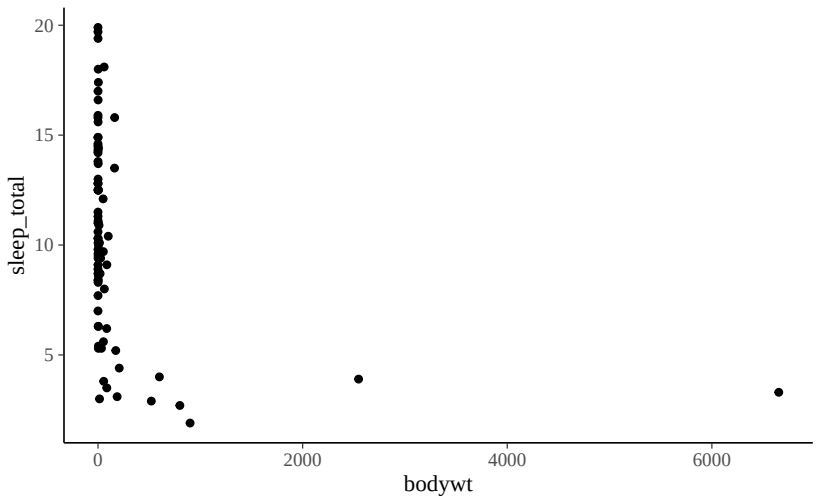
```
my_graph <- my_data %>%
  ggplot(aes(x_var, y_var)) +
  geom_...()
```

La prima volta che si usa il pacchetto `ggplot2` è necessario installarlo. Per fare questo possiamo installare `tidyverse` che, oltre a caricare `ggplot2`, carica anche altre utili funzioni per l'analisi dei dati. Ogni volta che si inizia una sessione R è necessario attivare i pacchetti che si vogliono usare, ma non è necessario istallarli una nuova volta. Se è necessario specificare il pacchetto nel quale è contenuta la funzione che vogliamo utilizzare, usiamo la sintassi `package::function()`. Per esempio, l'istruzione `ggplot2::ggplot()` rende esplicito che stiamo usando la funzione `ggplot()` contenuta nel pacchetto `ggplot2`.

Diagramma a dispersione

Consideriamo nuovamenti i dati contenuti nel `tibble msleep` e poniamoci il problema di rappresentare graficamente la relazione tra il numero medio di ore di sonno giornaliero (`sleep_total`) e il peso dell'animale (`bodywt`). Usando le impostazioni di default di `ggplot2`, con le istruzioni seguenti, otteniamo il grafico fornito dalla figura seguente.

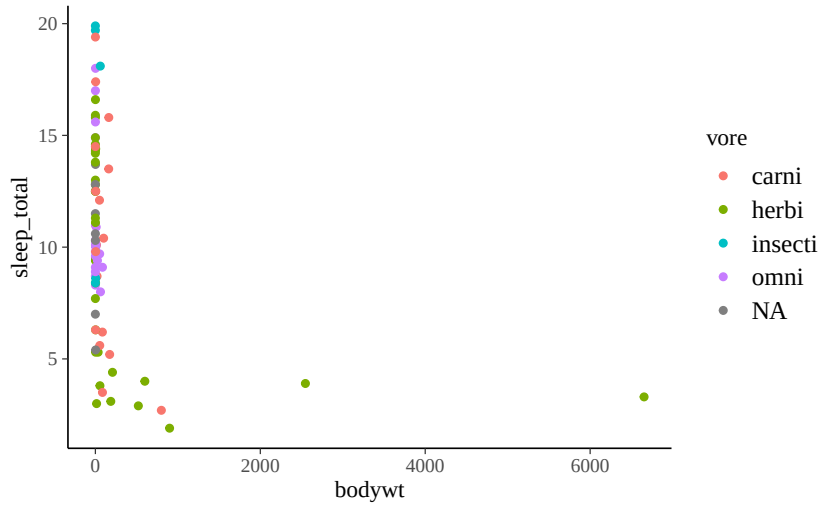
```
data(msleep)
p <- msleep %>%
  ggplot(
    aes(x = bodywt, y = sleep_total)
  ) +
  geom_point()
print(p)
```



Coloriamo ora in maniera diversa i punti che rappresentano animali carnivori, erbivori, ecc.

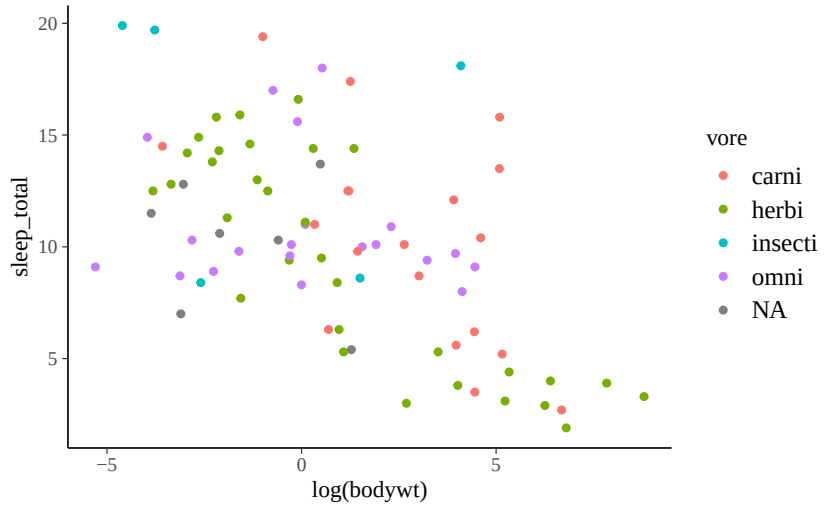
```
p <- msleep %>%
  ggplot(
    aes(x = bodywt, y = sleep_total, col = vore)
  ) +
```

```
  geom_point()
print(p)
```



È chiaro, senza fare alcuna analisi statistica, che la relazione tra le due variabili non è lineare. Trasformando in maniera logaritmica i valori dell'asse x la relazione si linearizza.

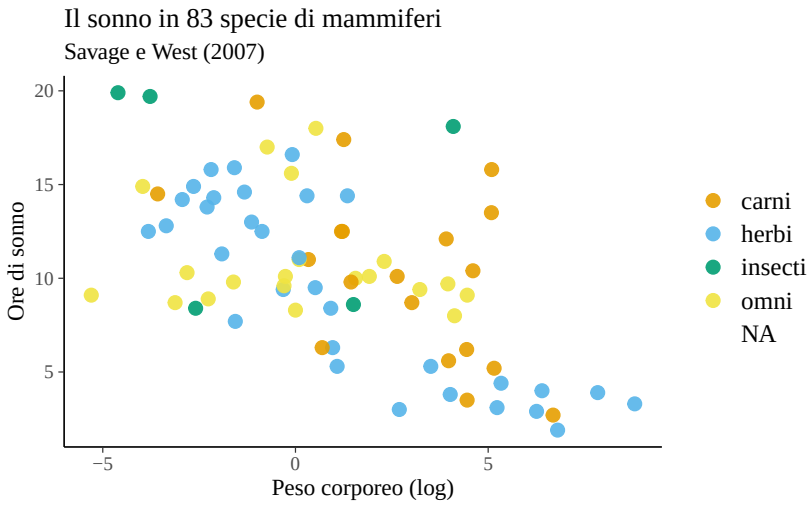
```
p <- msleep %>%
  ggplot(
    aes(x = log(bodywt), y = sleep_total, col = vore)
  ) +
  geom_point()
print(p)
```



Infine, aggiustiamo il "tema" del grafico (si noti l'utilizzo di una tavolozza di colori adatta ai daltonici), aggiungiamo le etichette sugli assi e il titolo.

```
msleep %>%
  ggplot(
    aes(x = log(bodywt), y = sleep_total, col = vore)
  ) +
```

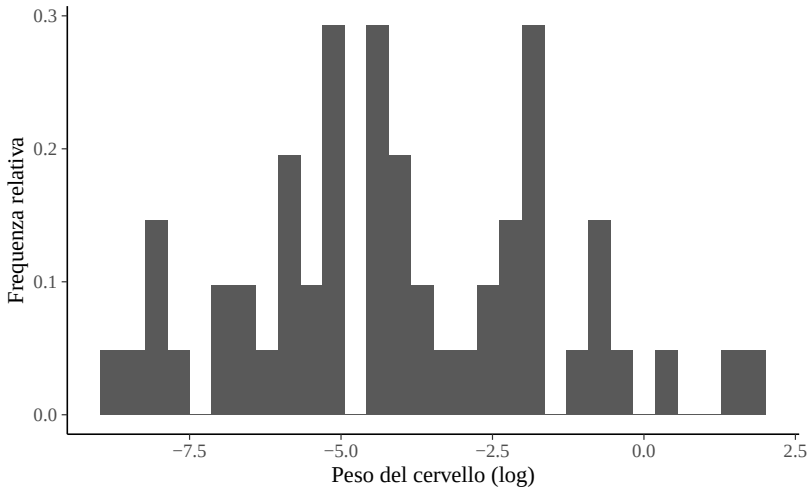
```
  geom_point(size = 3) +
  scale_color_okabe_ito(name = "vore", alpha = .9) +
  theme(legend.title = element_blank()) +
  labs(
    x = "Peso corporeo (log)",
    y = "Ore di sonno",
    title = "Il sonno in 83 specie di mammiferi",
    subtitle = "Savage e West (2007)"
  )
```



Istogramma

Creiamo ora un istogramma che rappresenta la distribuzione del (logaritmo del) peso medio del cervello delle 83 specie di mammiferi considerate da Savage e West (2007). L'argomento `aes(y = ..density..)` in `geom_histogram()` produce le frequenze relative. L'opzione di default (senza questo argomento) porta `ggplot()` a rappresentare le frequenze assolute.

```
msleep %>%
  ggplot(
    aes(log(brainwt))
  ) +
  geom_histogram(aes(y = ..density..)) +
  labs(
    x = "Peso del cervello (log)",
    y = "Frequenza relativa"
  ) +
  theme(legend.title = element_blank())
```



Scrivere il codice in R con stile

Uno stile di programmazione è un insieme di regole per la gestione dell'indentazione dei blocchi di codice, per la creazione dei nomi dei file e delle variabili e per le convenzioni tipografiche che vengono usate. Scrivere il codice in R con stile consente di creare listati più leggibili e semplici da modificare, minimizza la possibilità di errore, e consente correzioni e modifiche più rapide. Vi sono molteplici stili di programmazione che possono essere utilizzati dall'utente, anche se è bene attenersi a quelle che sono le convenzioni maggiormente diffuse, allo scopo di favorire la comunicazione. In ogni caso, l'importante è di essere coerenti, ovvero di adottare le stesse convenzioni in tutte le parti del codice che si scrive. Ad esempio, se si sceglie di usare lo stile `snake_case` per il nome composto di una variabile (es., `personality_trait`), non è appropriato usare lo stile *lower Camel case* per un'altra variabile (es., `socialStatus`). Dato che questo argomento è stato trattato ampiamente in varie sedi, mi limito qui a rimandare ad uno stile di programmazione molto popolare, quello proposto da Hadley Wickham, il creatore di `tidyverse`. Potete trovare maggiori informazioni al seguente link: http://style.tidyverse.org/.

Bibliografia

Burger, E. B. & Starbird, M. (2012). *The 5 elements of effective thinking*. Princeton University Press. (Cit. a p. viii).

Horn, S. & Loewenstein, G. (2021). Underestimating Learning by Doing. *Available at SSRN 3941441* (cit. a p. ix).

McNamara, A. & Horton, N. J. (2018). Wrangling categorical data in R. *The American Statistician*, *72*(1), 97–104 (cit. a p. 7).

Savage, V. M., Allen, A. P., Brown, J. H., Gillooly, J. F., Herman, A. B., Woodruff, W. H. & West, G. B. (2007). Scaling of number, size, and metabolic rate of cells with body size in mammals. *Proceedings of the National Academy of Sciences*, *104*(11), 4718–4723 (cit. a p. 1).

Savage, V. M. & West, G. B. (2007). A quantitative, theoretical framework for understanding mammalian sleep. *Proceedings of the National Academy of Sciences*, *104*(3), 1051–1056 (cit. a p. 10).

Wilkinson, L. (2012). The grammar of graphics. In *Handbook of computational statistics* (pp. 375–414). Springer. (Cit. a p. 7).

Elenco delle figure

Abstract This document contains the material of the lessons of Psicometria B000286 (2021/2022) aimed at students of the first year of the Degree Course in Psychological Sciences and Techniques of the University of Florence, Italy.

Keywords Data science, Bayesian statistics.