

WoRking with data frames

Ottavia M. Epifania, Ph.D

Lezione di Dottorato @Università Cattolica del Sacro Cuore (MI)

8-9 Giugno 2023

Table of contents

① Importa i dati

② Lavora con i dati

③ Esporta i dati

Table of Contents

① Importa i dati

② Lavora con i dati

③ Esporta i dati

Superato lo scoglio dell'importazione dei dati è tutta in discesa (kind of)

Diversi tipi di formati:

- .csv comma separated value → sono tra i più usati, “universali” (nonché i miei preferiti)
- .tab o .dat plain text, li potete creare anche con il blocco note del computer, molto comodi
- .xls o .xlsx molto comuni, servono pacchetti esterni
- .sav servono pacchetti esterni

.CSV

comma separated value → i separatori di colonna sono le virgole “,”

Nonostante siano comma separated value nei computer in italiano sono “;”, cosa che ovviamente genera non poca confusione

Il comando di base per leggere i .csv:

```
read.csv(file, header = TRUE, sep = ",", quote = "\"",  
         dec = ".", ...)
```

file: Il nome del file (se serve anche la sua directory)

header = TRUE: La prima riga contiene i nomi delle variabili

sep = ",": I separatori delle colonne sono le virgole

dec = ".": Il separatore dei decimali

.csv in Italia

Due opzioni:

- ① usare la funzione `read.csv = sep()` settando `sep = ";"` e `dec=","`
- ② usare la funzione `read.csv2()` → cambiano i default per cui `sep = ";"` e `dec = ","`

read.csv() in pratica

Come si chiama il file?

```
dir("data") # elenca tutti gli oggetti che sono all'interno
```

```
[1] "babies.tab" "benessere.csv" "benessereScores.csv"  
[4] "CioccoRazzaBuilt.dat" "database_benessere.xls"  
"datiBenessere.xlsx"  
[7] "score.tab"
```

Voglio importare il data set benessere.csv e assegnarlo all'oggetto data:

```
data = read.csv("data/benessere.csv",  
                header = TRUE,  
                sep = ",", dec = ".")
```

Ha funzionato?

Si!

```
head(data)
```

	benessere	stipendio	genere
1	5	1461.0983	m
2	7	1132.3637	f
3	7	1675.9004	m
4	2	328.9587	f
5	6	1370.0146	m
6	5	954.3540	f

Ha funzionato?

No

```
head(data.2)
```

```
benessere.stipendio.genere
```

```
1      5,1461.09828023079,m
2      7,1132.36368361099,f
3      7,1675.90040479853,m
4      2,328.958701913838,f
5      6,1370.01460952768,m
6      5,954.354030915469,f
```

.tab o .dat

Il comando di base per leggere i .tab o .dat:

```
read.table(file, header = FALSE, sep = "", quote = "\"'",  
           dec = ".", ...)
```

file: Il nome del file (se serve anche la sua directory)

header = FALSE: La prima riga contiene i nomi delle variabili (letto di default)

sep = "": I separatori delle colonne sono inferiti dal file

dec = ".": Il separatore dei decimali

read.table() in pratica I

```
tab_data = read.table("data/babies.tab")  
head(tab_data)
```

	id	genere	peso	altezza
1	baby1	f	7.424646	62.07722
2	baby2	m	7.442727	58.18877
3	baby3	f	9.512598	84.52737
4	baby4	f	11.306349	85.13573
5	baby5	m	9.345165	75.23783
6	baby6	m	5.411290	46.80163

read.table() in pratica II

```
dat_data = read.table("data/CioccoRazzaBuilt.dat",  
                      header = TRUE, sep = "\t")  
head(dat_data)
```

```
date time build subject blocknum blockcode trialnum  
1 121318 09:55 3.0.6.0 1 1 consenso 1  
2 121318 09:55 3.0.6.0 1 2 fame 1  
3 121318 09:55 3.0.6.0 1 6 BuiltLatteFondente2nd 1  
4 121318 09:55 3.0.6.0 1 6 BuiltLatteFondente2nd 2  
5 121318 09:55 3.0.6.0 1 6 BuiltLatteFondente2nd 4  
6 121318 09:55 3.0.6.0 1 6 BuiltLatteFondente2nd 6  
trialcode  
1 consenso  
2 fame  
3 reminder  
4 Builtlatterright  
5 Builtfondenteleft  
6 Builtfondenteleft  
response correct
```

File excel

Servono aiuti esterni (un pacchetto apposito):

`install.packages("readxl")` va digitato e fatto correre una volta nella console

```
library("readxl") # rende disponibile il pacchetto nell'ambiente
```

La funzione:

```
read_excel(path, sheet = NULL, range = NULL, col_names = TRUE,
            col_types = NULL, ...)
```

`path`: Il nome del file (se serve anche la sua directory)

`sheet = NULL` dà la possibilità di specificare il foglio specifico del file excel

`range = NULL`: Il range specifico di celle da leggere (e.g., B0:B13)

`col_names = TRUE`: La prima riga contiene i nomi delle variabili

read_excel() in pratica

```
benessere = read_excel("data/datiBenessere.xlsx")
head(benessere)
```

```
# A tibble: 6 x 19
  ID età genere frat item1 item2 item3 item4 item5 au1 au2
au3 au4
<dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
<dbl> <dbl> <dbl>
1 1 16 1 0 1 2 4 3 4 5 4 5 2
2 2 21 1 1 2 2 3 4 3 5 4 5 2
3 3 28 2 4 2 3 5 1 2 4 4 4 4
4 4 15 2 2 3 4 2 2 2 4 5 5 3
5 5 23 1 3 4 5 1 5 2 3 2 3 2
6 6 31 1 2 2 3 2 1 2 5 1 5 1
# i 6 more variables: au5 <dbl>, au6 <dbl>, au7 <dbl>, au8
<dbl>, au9 <dbl>,
# au10 <dbl>
```

File .sav

Aiuti esterni:

```
install.packages("foreign") oppure  
install.packages(foreign)
```

Table of Contents

① Importa i dati

② Lavora con i dati

③ Esporta i dati

Disclaimer

Presento solo le opzioni disponibili con base-R.

Si ottengono le stesse cose che si otterrebbero con tidyverse

La logica di tidyverse è un po' diversa, ma si ottiene lo stesso risultato con più codice

Sorting (Riordinare)

```
babies
```

```

      id genere      peso  altezza
1  baby1      m  9.858097 68.83238
2  baby2      f  8.128073 72.30143
3  baby3      f 11.347780 74.75981
....

```

```
order():
```

Ordine crescente

```
babies[order(babies$peso), ]
```

```

      id genere      peso  altezza
5  baby5      m  5.547482 50.95574
10 baby10      f  6.899776 71.39348
2  baby2      f  8.128073 72.30143
....

```

```
babies[order(babies$peso,
             decreasing = TRUE), ]
```

```

      id genere      peso  altezza
7  baby7      m 15.253421 89.57014
4  baby4      f 15.246266 87.40951
3  baby3      f 11.347780 74.75981
....

```

Si può ordinare anche considerando più variabili:





```
babies[order(babies$peso, babies$altezza,
             decreasing = TRUE), ]
```

	id	genere	peso	altezza
7	baby7	m	15.253421	89.57014
4	baby4	f	15.246266	87.40951
3	baby3	f	11.347780	74.75981
6	baby6	m	10.005233	70.75147
1	baby1	m	9.858097	68.83238
9	baby9	m	8.767017	65.63333
8	baby8	f	8.638523	83.48096
2	baby2	f	8.128073	72.30143
10	baby10	f	6.899776	71.39348
5	baby5	m	5.547482	50.95574

Wide format

Siamo abituati ad avere i dati in formato wide, ovvero matrici $p \times v$ dove $p = 1, \dots, P$ partecipanti e $v = 1, \dots, V$ variabili

Il numero di righe è uguale a P (numero dei partecipanti) e il numero di colonne è uguale a V (numero di variabili):

Respondent	Condition A		Condition B	
	RT 	Accuracy 	RT 	Accuracy 
$p1$	520	1	420	0
$p2$	320	0	620	0
$p3$	720	1	520	1







Long format

I dati organizzati in long format stanno prendendo sempre più piede

I software per la somministrazione di esperimenti (e.g., Inquisit, e-prime, PsychopPy) forniscono i risultati in long format

In riga si trovano le singole osservazioni. Ogni partecipante ha tante righe quante sono le osservazioni (i.e., i trial o le domande a cui ha risposto)

Il numero totale di righe è il prodotto tra P e il numero di trial di cui è composto l'esperimento

Respondent	Condition	Stimulus	RT	Accuracy
$p1$	A		520	1
$p1$	B		420	0
$p2$	A		320	0
$p2$	B		620	0
$p3$	A		720	1
$p3$	B		520	1

Long to wide

Da qui (long format)

Indometh # Long format

```

      Subject time conc
1          1 0.25 1.50
2          1 0.50 0.94
3          1 0.75 0.78
4          1 1.00 0.48
5          1 1.25 0.37
6          1 2.00 0.19
...

```

A qui (wide format):

```

      Subject conc.0.25 conc.0.5 conc.0.75 conc.1 conc.1.25 conc.2 conc.3 conc.4
1          1          1.50      0.94      0.78      0.48      0.37      0.19      0.12      0.08
12         2          2.03      1.63      0.71      0.70      0.64      0.36      0.32      0.19
23         3          2.72      1.49      1.16      0.80      0.80      0.39      0.22      0.13
34         4          1.85      1.39      1.02      0.89      0.59      0.40      0.16      0.08
45         5          2.05      1.04      0.81      0.39      0.30      0.23      0.13      0.05

```

reshape()

Per girare il data set si utilizza la funzione reshape

Sempre meglio sapere come sono fatti i dati per evitare sorprese:

```
levels(Indometh$Subject) # quanti soggetti ho?
```

```
[1] "1" "4" "2" "5" "6" "3"
```

```
table(Indometh$Subject) # quante osservazioni ho per soggetto?
```

```
  1  4  2  5  6  3
11 11 11 11 11 11
```

```
nrow(Indometh) # quante righe ha il mio data set?
```

```
[1] 66
```

Ci aspettiamo un data frame a 6 righe e con 11 colonne + la colonna con gli id dei soggetti:

```
# From long to wide
df.w <- reshape(Indometh, v.names = "conc", timevar = "time",
  idvar = "Subject", direction = "wide")
```

Le aspettative sono rispettate?

```
nrow(df.w) == length(levels(Indometh$Subject)) # ho sei righe?
```

```
[1] TRUE
```

```
sum(grepl("conc", colnames(df.w))) # ho 11 colonne per le 11 variabili?
```

```
[1] 11
```


Facendo prima...

	Subject	conc.0.25	conc.0.5	conc.0.75	conc.1	conc.1.25	conc.2	conc.3	con
1	1	1.50	0.94	0.78	0.48	0.37	0.19	0.12	0
12	2	2.03	1.63	0.71	0.70	0.64	0.36	0.32	0
23	3	2.72	1.49	1.16	0.80	0.80	0.39	0.22	0
34	4	1.85	1.39	1.02	0.89	0.59	0.40	0.16	0
45	5	2.05	1.04	0.81	0.39	0.30	0.23	0.13	0
56	6	2.31	1.44	1.03	0.84	0.64	0.42	0.24	0
	conc.5	conc.6	conc.8						
								

Wide to long

```
# From wide to long
df.l <- reshape(df.w, varying = list(2:12), v.names = "conc",
  idvar = "Subject", times = names(df.w)[-1], direction = "long")
```

	Subject	time	conc
1.conc.0.25	1	conc.0.25	1.50
2.conc.0.25	2	conc.0.25	2.03
3.conc.0.25	3	conc.0.25	2.72
....			

Unire data set

Se i data set hanno lo stesso numero di colonne **con lo stesso nome**

```
all_data = rbind(data, data1, data2, ...)
```

Se i data set hanno lo stesso numero di righe:

```
all_data = cbind(data, data1, data2, ...)
```

Se i data set hanno numeri diversi di righe e/o colonne, ma hanno almeno una caratteristica in comune (e.g., id) → `merge()`

```
all_data = merge(data1, data2,  
                  by = "ID")
```

All'argomento `by` si può passare anche un vettore di character che indica secondo quale variabile vogliamo che vengano uniti i data set

merge(): Un esempio

```
babies
```

	id	genere	peso	altezza
1	baby1	m	9.858097	68.83238
2	baby2	f	8.128073	72.30143
3	baby3	f	11.347780	74.75981
...				

merge(): Un esempio

babies

	id	genere	peso	altezza
1	baby1	m	9.858097	68.83238
2	baby2	f	8.128073	72.30143
3	baby3	f	11.347780	74.75981
....				

baby_detail

	id	termine	apgar	genere
1	baby1	yes	8	m
2	baby2	yes	8	f
3	baby3	yes	9	f
....				

`merge(babies, baby_detail)`

	id	genere	peso	altezza	termine	apgar
1	baby1	m	9.858097	68.83238	yes	8
2	baby10	f	6.899776	71.39348	no	3
3	baby2	f	8.128073	72.30143	yes	8
4	baby3	f	11.347780	74.75981	yes	9
5	baby4	f	15.246266	87.40951	yes	6
6	baby5	m	5.547482	50.95574	yes	8
7	baby6	m	10.005233	70.75147	yes	5
8	baby7	m	15.253421	89.57014	yes	3
9	baby8	f	8.638523	83.48096	no	6
10	baby9	m	8.767017	65.63333	no	4

Attenzione! data set con id diversi

	id	termine	apgar	genere
1	baby1	yes	8	m
2	baby2	yes	8	f
3	baby3	yes	9	f
4	baby4	yes	6	f
5	baby11	yes	8	m
6	baby12	yes	5	m
7	baby13	yes	3	m
8	baby14	no	6	f
9	baby15	no	4	m
10	baby16	no	3	f

Attenzione! data set con id diversi

```

      id termine apgar genere
1  baby1      yes    8      m
2  baby2      yes    8      f
3  baby3      yes    9      f
4  baby4      yes    6      f
5 baby11      yes    8      m
6 baby12      yes    5      m
7 baby13      yes    3      m
8 baby14      no     6      f
9 baby15      no     4      m
10 baby16     no     3      f

```

```
merge(babies, new_baby)
```

```

      id genere      peso  altezza termine apgar
1 baby1      m  9.858097  68.83238      yes    8
2 baby2      f  8.128073  72.30143      yes    8
3 baby3      f 11.347780  74.75981      yes    9
4 baby4      f 15.246266  87.40951      yes    6

```


Oppure

```
merge(babies, new_baby,
      all = T)
```

	id	genere	peso	altezza	termine	apgar
1	baby1	m	9.858097	68.83238	yes	8
2	baby10	f	6.899776	71.39348	<NA>	NA
3	baby11	m	NA	NA	yes	8
4	baby12	m	NA	NA	yes	5
5	baby13	m	NA	NA	yes	3
6	baby14	f	NA	NA	no	6
7	baby15	m	NA	NA	no	4
8	baby16	f	NA	NA	no	3
9	baby2	f	8.128073	72.30143	yes	8
10	baby3	f	11.347780	74.75981	yes	9
11	baby4	f	15.246266	87.40951	yes	6
12	baby5	m	5.547482	50.95574	<NA>	NA
13	baby6	m	10.005233	70.75147	<NA>	NA
14	baby7	m	15.253421	89.57014	<NA>	NA
15	baby8	f	8.638523	83.48096	<NA>	NA
16	baby9	m	8.767017	65.63333	<NA>	NA

Aggiungere una nuova variabile

Basta utilizzare il \$ e creare una nuova variabile:

```
babies$termmine <- sample(rep(c("si", "no"),
                               c(nrow(babies)/2, nrow(babies)/2)))
babies
```

	id	genere	peso	altezza	termmine
1	baby1	m	9.858097	68.83238	si
2	baby2	f	8.128073	72.30143	no
3	baby3	f	11.347780	74.75981	si
4	baby4	f	15.246266	87.40951	no
5	baby5	m	5.547482	50.95574	si
....					

Per eliminare una colonna:

```
babies[, -3]
```

	id	genere	altezza	termmine
1	baby1	m	68.83238	si
2	baby2	f	72.30143	no
3	baby3	f	74.75981	si

```
babies$peso = NULL
babies
```

	id	genere	altezza	termmine
1	baby1	m	68.83238	si
2	baby2	f	72.30143	no
3	baby3	f	74.75981	si

Creare nuove variabili “condizionate”

```
ifelse(test, vero, falso)
```

Ad esempio:

```
ifelse(babies$altezza > mean(babies$altezza), # test  
      "alti", # se vero  
      "bassi") # se falso
```

```
[1] "bassi" "bassi" "alti"  "alti"  "bassi" "bassi" "alti"  
[10] "bassi"
```

Check

	id	genere	altezza	termmine	altezza_media	altezza_dicotomico
1	baby1	m	68.83238	si	73.50883	bassi
2	baby2	f	72.30143	no	73.50883	bassi
3	baby3	f	74.75981	si	73.50883	alti
4	baby4	f	87.40951	no	73.50883	alti
5	baby5	m	50.95574	si	73.50883	bassi
6	baby6	m	70.75147	no	73.50883	bassi
7	baby7	m	89.57014	no	73.50883	alti
8	baby8	f	83.48096	si	73.50883	alti
9	baby9	m	65.63333	no	73.50883	bassi
10	baby10	f	71.39348	si	73.50883	bassi

Calcolare i punteggi di scala

Assumendo di avere il data set in formato wide:

- Sommando attraverso le variabili (colonne) → punteggio di scala per le persone
rowSums() (oppure *rowMean()* se si vuole usare la media)
- Sommando attraverso le persone (righe) → punteggi sugli item
colSums() (oppure *colMean()* se si vuole usare la media)

Dati benessere

```
library(readxl)
benessere = read_xlsx("data/datiBenessere.xlsx")

head(benessere,2)
```

```
# A tibble: 2 x 19
      ID età genere frat item1 item2 item3 item4 item5 au1 au
  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1     1    16     1     0     1     2     4     3     4     5
2     2    21     1     1     2     2     3     4     3     5
# i 6 more variables: au5 <dbl>, au6 <dbl>, au7 <dbl>, au8 <dbl>, a
# au10 <dbl>
```

Se proviamo ad applicare rowSums() :

```
rowSums(benessere)
```

```
[1] 65 79 88 74 75 83 78 79 85 87 80 88 83 84 92 1
[19] 79 83 112 105 98 86 94 94 120 108 98 100 107 111 113 1
[37] 100 108 107 126 126 116 116 110 117 117 106 111 122 123 129 1
```

Condizionare ai nomi delle variabili

è necessario che le variabili abbiano una radice comune:

```
colnames(benessere)
```

```
[1] "ID" "età" "genere" "frat" "item1" "item2"  
"item3" "item4"  
[9] "item5" "au1" "au2" "au3" "au4" "au5" "au6"  
"au7"  
[17] "au8" "au9" "au10"
```

Le colonne con la radice `item` sono gli item che misurano il benessere, quelli con radice `au` misurano un'altra variabile

Le funzioni `grep()` e `grepl()` permettono di filtrare gli oggetti sulla base di una regular expression (regex)

```
grep("regex", vettore)
```

Il funzionamento è il medesimo, differiscono nel loro risultato

```
(my_vector = colnames(benessere))
```

```
[1] "ID"      "età"      "genere"  "frat"     "item1"    "item2"    "item3"    "item4"
[9] "item5"    "au1"      "au2"     "au3"     "au4"      "au5"      "au6"      "au7"
[17] "au8"      "au9"      "au10"
```

`grep()`

```
grep("au", my_vector)
```

```
[1] 10 11 12 13 14 15 16 17 18 19
```

`grepl()`

```
grepl("au", my_vector)
```

```
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE TRUE TR
[13] TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```


Tornando al calcolo di scala

Bisogna condizionare `rowSums()` a calcolare le somme solo per certe colonne:

```
cat(rowSums(benessere[, grep("item", colnames(benessere))])[1:15],
```

```
14 14 13 13 17 10 16 16 9 13 14 16 17 12 9 ...
```

Si può assegnare a una nuova variabile nel data frame di partenza:

```
benessere$score_ben = rowSums(benessere[, grep("item", colnames(ben
```

Your turn!



- Creare una variabile dicotomica dal data set benessere partendo dalla variabile `frat`
0 fratelli → *figlio_unico*
1+ → *sibling*
- Calcolate il punteggio di scala della scala composta dagli item `au` e assegnatelo a una nuova variabile `score_au`
- create un nuovo data set `score` che contenga `ID`, `età`, `genere`, `score_ben` e `score_au`

Table of Contents

1 Importa i dati

2 Lavora con i dati

3 Esporta i dati

File (csv) o text:

```
write.table(data, # il data frame
            file = "mydata.txt", # il nome che si vuol dare +
            header = TRUE,
            sep = "\t",
            ....)
```

L'ambiente di R:

```
save(dat, file = "exp1_data.rda") # salva un oggetto specifico
save(file = "the_earth.rda")      # save the environment
load("the_earth.rda")             # re-importa l'environment
```

Your turn!



- Esportate il data set benessere con i due score calcolati e aggiunti come nuove variabili in fondo in formato csv
- Esportate il data set score in formato tab