

2.4_dataframe

Strutture Dati

- vettori
- fattori
- liste
- matrici
- array
- ***dataframe***

Dataframe

Il dataframe è la struttura più “complessa”, utile e potente di R. Da un punto di vista intuitivo è un foglio excel mentre da un punto di vista di R è una tipologia di lista con alcune caratteristiche/restrizioni

- ogni elemento della lista è un **vettore** con un **nome associato** (aka una colonna)
- ogni lista/colonna deve avere lo stesso numero di elementi
- di conseguenza ogni riga ha lo stesso numero di elementi (**struttura rettangolare**)

Creazione

Nella maggior parte dei casi vi capiterà d'importare (lo vedremo più avanti) più che creare dei dataframe, ma è importante prima capire come funzionano e come crearli/manipolarli.

```
1 # Creo un dataframe con 3 colonne
2 my_df = data.frame( col1 = 1:4, col2 = letters[1:4],
3                     col3 = rnorm(4))
4 my_df
```

	col1	col2	col3
1	1	a	-0.7643168
2	2	b	-0.9250629
3	3	c	2.4530594
4	4	d	-0.9047478

Attributi

Il **dataframe** ha sia gli attributi della lista ovvero i *names* ma anche gli attributi della matrice ovvero le *dimensioni* (righe e colonne)

```
1 typeof(my_df)
```

```
[1] "list"
```

```
1 attributes(my_df)
```

```
$names
```

```
[1] "col1" "col2" "col3"
```

```
$class
```

```
[1] "data.frame"
```

```
$row.names
```

```
[1] 1 2 3 4
```

```
1 dim(my_df)
```

```
[1] 4 3
```

Come per le altre strutture che abbiamo visto, possiamo utilizzare le funzioni **names()**, **dim()**, **nrow()**, **ncol()**... per ottenere informazioni sulle caratteristiche del dataframe. La funzione più utile è **str()** poichè ci restituisce una veloce overview della struttura del dataframe: dimensioni, tipi di variabili,...

```
1 str(iris) # iris è un dataset presente di default in R (?iris)
```

```
'data.frame':  150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species      : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1
1 1 ...
```

Indicizzazione

Si posso utilizzare sia le parentesi quadre `[]` che il simbolo del dollaro `$`

```
1 my_df[1] # estraggo un data.frame 5x1
```

```
col1
```

```
1    1
2    2
3    3
4    4
```

```
1 my_df[[1]]# estraggo la prima colonna del data.frame
```

```
[1] 1 2 3 4
```

```
1 my_df$col1# estraggo la prima colonna del data.frame
```

```
[1] 1 2 3 4
```

```
1 my_df[1,1]# estraggo il primo elemento della prima colonna del data.frame
```

```
[1] 1
```

Indicizzazione - Operatori relazionali

Una delle operazioni più comuni che dovrete affrontare sarà sicuramente quella di estrarre/valutare un sottoinsieme di valori presenti nel vostro dataset:

```
1 # includo solo le righe in cui alla colonna1 i valori sono maggiori di 2
2 my_df[my_df$col1 > 2, ]
```

	col1	col2	col3
3	3	c	2.4530594
4	4	d	-0.9047478

```
1 my_df[my_df[1] > 2, ]
```

	col1	col2	col3
3	3	c	2.4530594
4	4	d	-0.9047478

Esempi

	col1	col2	col3
1	1	a	-0.7643168
2	2	b	-0.9250629
3	3	c	2.4530594
4	4	d	-0.9047478

- `my_df[my_df$col1 > 2 & my_df$col1 < 4,]`
 - “3” “c” “-1.155393”
- `my_df[my_df$col1== 2, "col2"]`
 - “b”
- `my_df[my_df$col1== 2, 2]`
 - “b”

Indicizzazione

Ci sono anche dei modi alternativi e più compatti di indicizzare. Ad esempio usando la funzione **subset()**:

```
1 subset(iris, subset = Species == "setosa" & Petal.Length > 1.7)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
25	4.8	3.4	1.9	0.2	setosa
45	5.1	3.8	1.9	0.4	setosa

equivalente a:

```
1 iris[iris$Species == "setosa" & iris$Petal.Length > 1.7,]
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
25	4.8	3.4	1.9	0.2	setosa
45	5.1	3.8	1.9	0.4	setosa

E' possibile anche selezionare colonne piuttosto che righe attraverso l'argomento ***select***:

```
1 head(subset(iris, select = c(Sepal.Length, Species)), n = 3)
```

	Sepal.Length	Species
1	5.1	setosa
2	4.9	setosa
3	4.7	setosa

Possiamo anche combinare le due cose:

```
1 head(subset(iris, Species == "setosa" & Sepal.Length > 4,  
2           c(Sepal.Length, Species)), n = 3)
```

	Sepal.Length	Species
1	5.1	setosa
2	4.9	setosa
3	4.7	setosa

equivalente a:

```
1 head(iris[iris$Species == "setosa" & iris$Sepal.Length > 4,  
2       c("Sepal.Length", "Species")], n = 3)
```

	Sepal.Length	Species
1	5.1	setosa
2	4.9	setosa
3	4.7	setosa

La funzione **subset()** essenzialmente prende delle espressioni ***Species == "setosa"*** che vengono eseguite all'interno dell'ambiente dataframe specificato come primo argomento.

Un'operazione simile viene svolta dalla funzione **with()**:

```
1 Species = "bo"  
2 with(iris, Sepal.Length[3] + Sepal.Width[3])
```

```
[1] 7.9
```

L'espressione `Sepal.Length + Sepal.Width` viene eseguita all'interno (**with()**) del dataframe `iris`.

Indicizzazione

La maggiorparte delle volte vi troverete ad accedere alle variabili tramite l'operatore `$`. Questo comando può essere utilizzato anche per creare una nuova variabile...

```
1 iris$somma = iris$Sepal.Length + iris$Sepal.Width
2 str(iris)
```

```
'data.frame':  150 obs. of  6 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species      : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1
1 1 ...
 $ somma        : num  8.6 7.9 7.9 7.7 8.6 9.3 8 8.4 7.3 8 ...
```

Si applicano gli stessi concetti che abbiamo visto per i vettori, potete quindi sia creare che modificare variabili.

```
1 my_df = data.frame(num = 1:4, let = letters[1:4])
2 my_df
```

	num	let
1	1	a
2	2	b
3	3	c
4	4	d

```
1 # Modifico la variabile num aggiungendo 1
2 my_df$num = my_df$num+1
3
4 # Creo una terza variabile composta dalla variabile num e let
5 my_df$both = paste(my_df$num,my_df$let, sep = "_")
6
7 str(my_df)
```

```
'data.frame':   4 obs. of  3 variables:
 $ num : num    2  3  4  5
 $ let  : chr    "a" "b" "c" "d"
 $ both: chr    "2_a" "3_b" "4_c" "5_d"
```

Combinare Dataframes

Essendo simili a delle matrici, i dataframe si possono combinare tra loro attraverso le funzioni **rbind()**:

```
1 # primo dataframe
2 str(my_df)
```

```
'data.frame':   4 obs. of  3 variables:
 $ num : num   2 3 4 5
 $ let : chr  "a" "b" "c" "d"
 $ both: chr  "2_a" "3_b" "4_c" "5_d"
```

```
1 # creo un secondo dataframe
2 my_df2 = data.frame(num = 4:7, lett = letters[1:4],
3                     both = paste(4:7, letters[1:4], sep = "_"))
4
5 str(my_df2)
```

```
'data.frame':   4 obs. of  3 variables:
 $ num : int   4 5 6 7
 $ lett: chr  "a" "b" "c" "d"
 $ both: chr  "4_a" "5_b" "6_c" "7_d"
```


Unisco i due dataframes

- I dataframes devono avere lo stesso numero di colonne
- I nomi delle colonne devono essere identici

```
1 my_df3 = rbind(my_df,my_df2)
```

```
Error in match.names(clabs, names(xi)): names do not match previous names
```

```
1 str(my_df)
```

```
'data.frame':  4 obs. of  3 variables:  
 $ num : num  2 3 4 5  
 $ let : chr  "a" "b" "c" "d"  
 $ both: chr  "2_a" "3_b" "4_c" "5_d"
```

```
1 str(my_df2)
```

```
'data.frame':  4 obs. of  3 variables:  
 $ num : int  4 5 6 7  
 $ lett: chr  "a" "b" "c" "d"  
 $ both: chr  "4_a" "5_b" "6_c" "7_d"
```

Sistemo i nomi

```
1 names(my_df2) = names(my_df)  
2 my_df3 = rbind(my_df,my_df2)  
3 str(my_df3)
```

```
'data.frame':  8 obs. of  3 variables:  
 $ num : num  2 3 4 5 4 5 6 7  
 $ let : chr  "a" "b" "c" "d" ...  
 $ both: chr  "2_a" "3_b" "4_c" "5_d" ...
```

Potrebbe anche capitarvi di dover raccogliere differenti tipi di dato dallo stesso partecipante, e successivamente combinare le informazioni raccolte...

Dataframe contenente i tempi di reazione:

```
1 df_rt = data.frame(subj = factor(rep(c("caio","tizio"),each = 400)),  
2                       cond = factor(rep(c("easy","hard"),  
3                                       each = 200, times = 2)),  
4                       rt = c(rlnorm(400, -1, .25), rlnorm(400, -.7, .3)))
```

Dataframe contenente l'età:

```
1 df_age = data.frame(subj = factor(c("caio","tizio")), age = c(20,3))
```

```
1 str(df_rt) # struttura dataframe tempi di reazione
```

```
'data.frame': 800 obs. of 3 variables:  
 $ subj: Factor w/ 2 levels "caio","tizio": 1 1 1 1 1 1 1 1 1 1 ...  
 $ cond: Factor w/ 2 levels "easy","hard": 1 1 1 1 1 1 1 1 1 1 ...  
 $ rt : num 0.246 0.332 0.371 0.346 0.383 ...
```

```
1 head(df_rt)
```

```
  subj cond      rt  
1 caio easy 0.2461750  
2 caio easy 0.3324239  
3 caio easy 0.3709496  
4 caio easy 0.3460079  
5 caio easy 0.3834675  
6 caio easy 0.4517216
```

```
1 str(df_age) # struttura dataframe età
```

```
'data.frame': 2 obs. of 2 variables:  
 $ subj: Factor w/ 2 levels "caio","tizio": 1 2  
 $ age : num 20 3
```

```
1 head(df_age)
```

```
  subj age  
1 caio 20  
2 tizio 3
```

In questo caso, è possibile utilizzare la funzione **merge()** e/o la funzione **_join()**:

```
1 dfAll1 = merge(df_rt, df_age, by="subj")
2 str(dfAll1)
```

```
'data.frame':  800 obs. of  4 variables:
 $ subj: Factor w/ 2 levels "caio","tizio": 1 1 1 1 1 1 1 1 1 1 ...
 $ cond: Factor w/ 2 levels "easy","hard": 1 1 1 1 1 1 1 1 1 1 ...
 $ rt  : num  0.246 0.332 0.371 0.346 0.383 ...
 $ age : num  20 20 20 20 20 20 20 20 20 20 ...
```

```
1 library(tidyverse)
2 dfAll2 = df_rt%>%
3   left_join(df_age, by = c("subj")) # esistono anche right_join; full_join
4 str(dfAll2)
```

```
'data.frame':  800 obs. of  4 variables:
 $ subj: Factor w/ 2 levels "caio","tizio": 1 1 1 1 1 1 1 1 1 1 ...
 $ cond: Factor w/ 2 levels "easy","hard": 1 1 1 1 1 1 1 1 1 1 ...
 $ rt  : num  0.246 0.332 0.371 0.346 0.383 ...
 $ age : num  20 20 20 20 20 20 20 20 20 20 ...
```

Tabelle di contingenza

E' possibile utilizzare la funzione **table()** sia per calcolare le frequenze che per computare delle tabelle di contingenze, per esempio per vedere quanti trial ho per ogni soggetto.

```
1 # tengo solo rt compresi tra .200 e 1.5 secondi
2 df_rt_clean = df_rt[df_rt$rt > .25 & df_rt$rt < 1.5 ,]
3
4 # quanti trial x ogni soggetto
5 table(df_rt_clean$subj)
```

```
caio tizio
375   393
```

```
1 # quanti trial x ogni soggetto x condizione
2 table(df_rt_clean$subj,df_rt_clean$cond)
```

```
      easy hard
caio   187   188
tizio  194   199
```

Esportazione e importazione dati

In R è possibile importare dati in molti formati differenti, più comunemente vi troverete ad importare dati `.csv` oppure `.xlsx`.

Qui per esempio, esporto i dataframe in due formati differenti...

```
1 library(readr) # carico il pacchetto readr
2 library(writexl) # carico il pacchetto writexl
3
4 write_csv(df_rt, file = "DATA/df_rt.csv", row.names = FALSE)
5 write_xlsx(df_age, path = "DATA/df_age.xlsx")
```

Importo

```
1 library(readxl) # carico il pacchetto readxl
2
3 df_rt_impo = read_csv("DATA/df_rt.csv") #utilizza il pacchetto readr
4
5 df_age_impo = read_xlsx("DATA/df_age.xlsx")
```


Controllare la struttura

Cosa notate di diverso tra i due dataframe?

```
1 str(df_rt) # Data frame creato
```

```
'data.frame':  800 obs. of  3 variables:
 $ subj: Factor w/ 2 levels "caio","tizio": 1 1 1 1 1 1 1 1 1 1 ...
 $ cond: Factor w/ 2 levels "easy","hard": 1 1 1 1 1 1 1 1 1 1 ...
 $ rt  : num  0.246 0.332 0.371 0.346 0.383 ...
```

```
1 str(df_rt_impo) # Data frame caricato
```

```
spc_tbl_ [800 × 3] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
 $ subj: chr [1:800] "caio" "caio" "caio" "caio" ...
 $ cond: chr [1:800] "easy" "easy" "easy" "easy" ...
 $ rt  : num [1:800] 0.246 0.332 0.371 0.346 0.383 ...
- attr(*, "spec")=
 .. cols(
 ..   subj = col_character(),
 ..   cond = col_character(),
 ..   rt = col_double()
 .. )
- attr(*, "problems")=<externalptr>
```

Il dataframe viene importato come tibble ed in questo caso le variabili **factor** sono state importate come **character**. Questo è modificabile o in importazione (**?read_csv**):

```
1 df_rt_impol = read_csv("DATA/df_rt.csv",
2                         col_types =
3                         list(col_double(),col_factor(),
4                             col_factor(),col_double()))
5 str(df_rt_impol)
```

```
spc_tbl_ [800 × 3] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
 $ subj: num [1:800] NA NA NA NA NA NA NA NA NA NA ...
 $ cond: Factor w/ 2 levels "easy","hard": 1 1 1 1 1 1 1 1 1 1 ...
 $ rt   : Factor w/ 800 levels "0.246175033345716",...: 1 2 3 4 5 6 7 8 9 10 ...
- attr(*, "spec")=
 .. cols(
 ..   subj = col_double(),
 ..   cond = col_factor(levels = NULL, ordered = FALSE, include_na = FALSE),
 ..   rt   = col_factor(levels = NULL, ordered = FALSE, include_na = FALSE)
 .. )
- attr(*, "problems")=<externalptr>
```

o a posteriori attraverso il comando **as.factor()**:

```
1 df_rt_impo$subj = as.factor(df_rt_impo$subj)
2 df_rt_impo$cond = as.factor(df_rt_impo$cond)
3 str(df_rt_impo)
```

```
spec_tbl_ [800 × 3] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
 $ subj: Factor w/ 2 levels "caio","tizio": 1 1 1 1 1 1 1 1 1 1 ...
 $ cond: Factor w/ 2 levels "easy","hard": 1 1 1 1 1 1 1 1 1 1 ...
 $ rt   : num [1:800] 0.246 0.332 0.371 0.346 0.383 ...
- attr(*, "spec")=
 .. cols(
 ..   subj = col_character(),
 ..   cond = col_character(),
 ..   rt = col_double()
 .. )
- attr(*, "problems")=<externalptr>
```

equivalente a:

```
1 df_rt_impo2 = read_csv("DATA/df_rt.csv") %>% #library(tidyverse)
2   mutate(subj = factor(subj),
3          cond = factor(cond))
4 str(df_rt_impo2)
```

```
tibble [800 × 3] (S3: tbl_df/tbl/data.frame)
 $ subj: Factor w/ 2 levels "caio","tizio": 1 1 1 1 1 1 1 1 1 1 ...
 $ cond: Factor w/ 2 levels "easy","hard": 1 1 1 1 1 1 1 1 1 1 ...
 $ rt   : num [1:800] 0.246 0.332 0.371 0.346 0.383 ...
```

(solo per farvi vedere come in *R*, si possa risolvere un problema in differenti modi)

Gli stessi concetti si applicano alla lettura di file.xlsx:

```
1 str(df_age_impo)
```

```
tibble [2 × 2] (S3: tbl_df/tbl/data.frame)
 $ subj: chr [1:2] "caio" "tizio"
 $ age : num [1:2] 20 3
```

```
1 df_age_impo$subj=as.factor(df_age_impo$subj)
2 str(df_age_impo)
```

```
tibble [2 × 2] (S3: tbl_df/tbl/data.frame)
 $ subj: Factor w/ 2 levels "caio","tizio": 1 2
 $ age : num [1:2] 20 3
```

```
1 df_age_impo$subj=factor(df_age_impo$subj)
2 str(df_age_impo)
```

```
tibble [2 × 2] (S3: tbl_df/tbl/data.frame)
 $ subj: Factor w/ 2 levels "caio","tizio": 1 2
 $ age : num [1:2] 20 3
```

Gli esempi che vi ho mostrato riguardano dati in formati **esterni** ad R, è possibile però anche salvare ed importare in formato **R**.

Il principale difetto è quello che sono appunto leggibili (principalmente) solo all'interno dell'ambiente R.

Se abbiamo la necessità di lavorare con software o linguaggi diversi è probabilmente meglio usare un formato più generico.

Se avanzerà del tempo andremo a vedere anche questo tipo di formato, il materiale è disponibile a questo [link](#).

Ora però facciamo un po' di pratica!

Aprirete e tenete aperto questo link:

<https://etherpad.wikimedia.org/p/arca-corsoR>

