Informatica

Principi di programmazione ed analisi dati in linguaggio

Scienze Biologiche

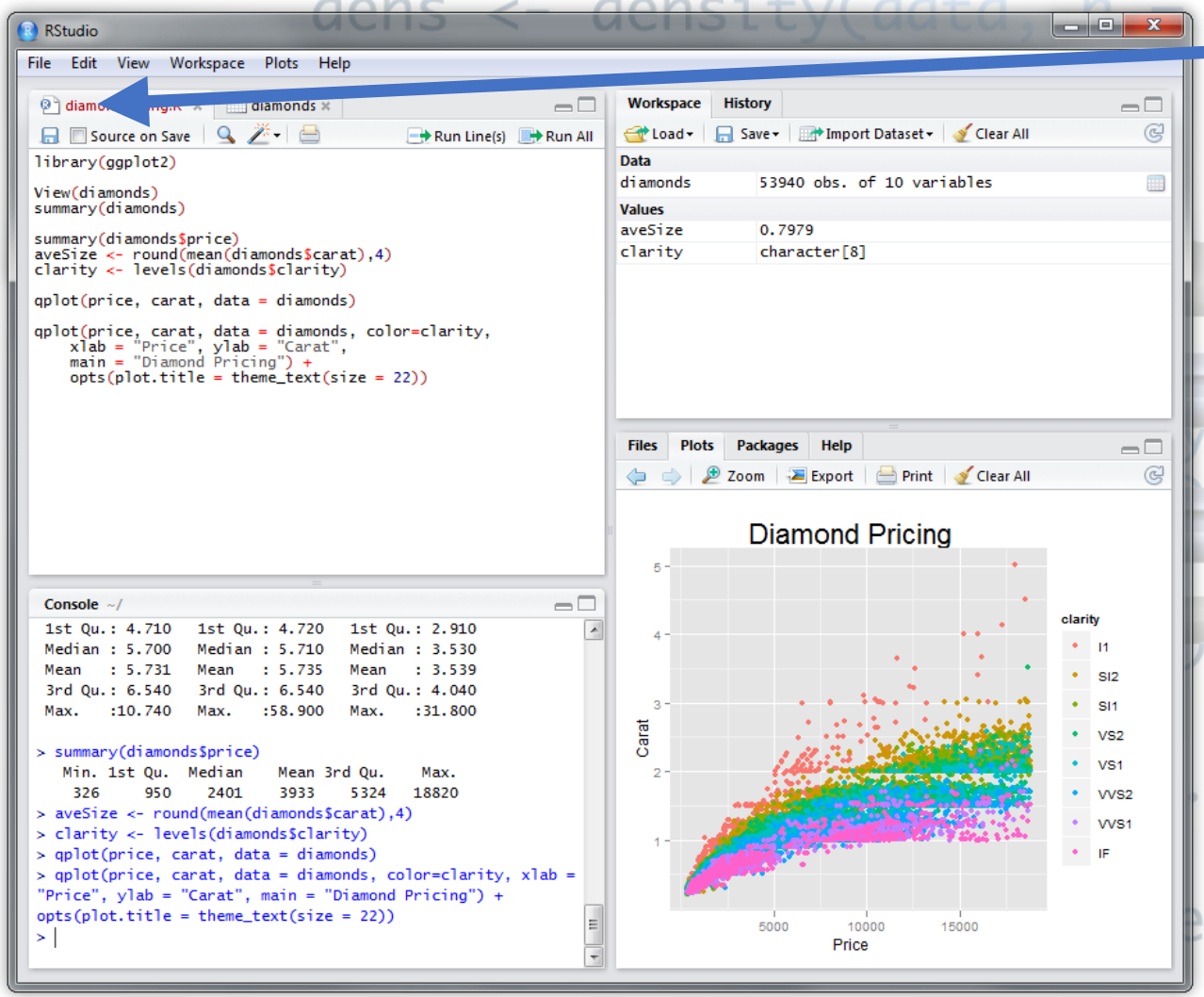
Dr. Bruno Bellisario, PhD

Prima di iniziare…

Creiamo una nuova directory di lavoro sul desktop e chiamiamola **LezioneR5**

Accertiamoci di caricare il workspace di RStudio puntando alla directory giusta.

Creiamo un nuovo file script .R da RStudio



TIPOLOGIE DI DATI: IL DATAFRAME

- data.frame (..., row.names = NULL, check.rows = FALSE, check.names = TRUE, stringsAsFactors = default.stringsAsFactors ())
- as.data.frame (x, row.names = NULL, optional = FALSE, ...) # funzione generica
- as.data.frame (x, ..., stringsAsFactors = default.stringsAsFactors ()) # S3 metodo per la classe 'character'
- as.data.frame (x, row.names = NULL, facoltativo = FALSE, ..., stringsAsFactors = default.stringsAsFactors ()) # S3 metodo per classe 'matrice'
- is.data.frame (x)

I frame di dati sono probabilmente la struttura dei dati che verrà utilizzata maggiormente nelle analisi. Un frame di dati è un tipo speciale di elenco che memorizza i vettori della stessa lunghezza di classi diverse. Si creano frame di dati utilizzando la funzione `data.frame` . L'esempio seguente mostra questo combinando un vettore numerico e un carattere in un frame di dati. Esso utilizza il `:` operatore, che crea un vettore contenente tutti i numeri interi da 1 a 3.

```
df1 <- data.frame(x = 1:3, y = c("a", "b", "c"))
df1
##   x y
## 1 1 a
## 2 2 b
## 3 3 c
class(df1)
## [1] "data.frame"
```

I dataframe non vengono stampati con le virgolette, quindi la classe delle colonne non è sempre evidente. Creiamo un nuovo dataframe df2….

```
df2 <- data.frame(x = c("1", "2", "3"), y = c("a", "b", "c"))
df2
##   x y
## 1 1 a
## 2 2 b
## 3 3 c
```

Senza ulteriori indagini, le colonne "x" in `df1` e `df2` non possono essere differenziate. La funzione `str` può essere utilizzata per descrivere oggetti con più dettagli rispetto alla classe.

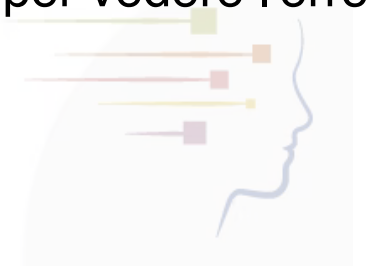
```
str(df1)
## 'data.frame':    3 obs. of  2 variables:
##  $ x: int  1 2 3
##  $ y: Factor w/ 3 levels "a","b","c": 1 2 3
str(df2)
## 'data.frame':    3 obs. of  2 variables:
##  $ x: Factor w/ 3 levels "1","2","3": 1 2 3
##  $ y: Factor w/ 3 levels "a","b","c": 1 2 3
```

Qui vedi che `df1` è un `data.frame` e ha 3 osservazioni di 2 variabili, "x" e "y". Quindi ti viene detto che "x" ha il numero intero di tipo di dati (non importante per questa classe, ma per i nostri scopi si comporta come un numerico) e "y" è un fattore con tre livelli (un'altra classe di dati che non stiamo discutendo).

Il comportamento predefinito può essere modificato con il parametro `stringsAsFactors` :

```
df3 <- data.frame(x = 1:3, y = c("a", "b", "c"), stringsAsFactors = FALSE)
str(df3)
## 'data.frame':    3 obs. of  2 variables:
##  $ x: int  1 2 3
##  $ y: chr  "a" "b" "c"
```

Come accennato in precedenza, ogni "colonna" di un frame di dati deve avere la stessa lunghezza. Provare a creare un data.frame da vettori con lunghezze diverse comporterà un errore. (Prova a eseguire `data.frame(x = 1:3, y = 1:4)` per vedere l'errore risultante.)



Crea un data.frame vuoto

Un data.frame è un tipo speciale di elenco: è *rettangolare* . Ogni elemento (colonna) dell'elenco ha la stessa lunghezza e ogni riga ha un "nome riga". Ogni colonna ha una sua classe, ma la classe di una colonna può essere diversa dalla classe di un'altra colonna (a differenza di una matrice, in cui tutti gli elementi devono avere la stessa classe).

In linea di principio, data.frame potrebbe non avere righe e nessuna colonna:

```
> structure(list(character()), class = "data.frame")
NULL
<0 rows> (or 0-length row.names)
```

Ma questo è insolito. È più comune per un data.frame avere molte colonne e molte righe. Ecco un data.frame con tre righe e due colonne (`a` è la classe numerica `b` è la classe carattere):

```
> structure(list(a = 1:3, b = letters[1:3]), class = "data.frame")
[1] a b
<0 rows> (or 0-length row.names)
```

Per stampare data.frame, è necessario fornire alcuni nomi di riga. Qui usiamo solo i numeri 1: 3:

```
> structure(list(a = 1:3, b = letters[1:3]), class = "data.frame", row.names = 1:3)
  a b
1 1 a
2 2 b
3 3 c
```

Ora diventa ovvio che abbiamo un data.frame con 3 righe e 2 colonne. Puoi verificarlo usando `nrow()` , `ncol()` e `dim()` :

```
> x <- structure(list(a = numeric(3), b = character(3)), class = "data.frame", row.names =
1:3)
> nrow(x)
[1] 3
> ncol(x)
[1] 2
> dim(x)
[1] 3 2
```

R fornisce altre due funzioni (oltre a `structure()`) che possono essere utilizzate per creare un data.frame. Il primo è chiamato, intuitivamente, `data.frame()` . Controlla che i nomi delle colonne che hai fornito siano validi, che gli elementi della lista siano tutti della stessa lunghezza e fornisca alcuni nomi di riga generati automaticamente. Ciò significa che l'output di `data.frame()` potrebbe essere sempre esattamente quello che ti aspetti:

```
> str(data.frame("a a a" = numeric(3), "b-b-b" = character(3)))
'data.frame':   3 obs. of  2 variables:
 $ a.a.a: num  0 0 0
 $ b.b.b: Factor w/ 1 level "": 1 1 1
```

L'altra funzione è chiamata `as.data.frame()` . Questo può essere usato per forzare un oggetto che non è un data.frame in essere un data.frame eseguendolo attraverso `data.frame()` . Ad esempio, considera una matrice:

```
> m <- matrix(letters[1:9], nrow = 3)
> m
     [,1] [,2] [,3]
[1,] "a"  "d"  "g"
[2,] "b"  "e"  "h"
[3,] "c"  "f"  "i"
```

```
> as.data.frame(m)
  V1 V2 V3
1  a  d  g
2  b  e  h
3  c  f  i
> str(as.data.frame(m))
'data.frame':   3 obs. of  3 variables:
 $ V1: Factor w/ 3 levels "a","b","c": 1 2 3
 $ V2: Factor w/ 3 levels "d","e","f": 1 2 3
 $ V3: Factor w/ 3 levels "g","h","i": 1 2 3
```

Subsetting di righe e colonne da un frame di dati

Questo argomento riguarda la sintassi più comune per accedere a righe e colonne specifiche di un frame di dati. Questi sono

- Come una `matrix` con `data[rows, columns]` parentesi quadre `data[rows, columns]`
  - Usando numeri di riga e colonna
  - Utilizzo dei nomi di colonna (e riga)
- Come una `list` :
  - Con i `data[columns]` parentesi quadre `data[columns]` per ottenere un frame di dati
  - Con i `data[[one_column]]` parentesi quadre `data[[one_column]]` per ottenere un vettore
- Con `$` per una singola colonna di `data$column_name`

Useremo il frame di dati `mtcars` per illustrare.

Usando il `mtcars` frame di dati `mtcars` , possiamo estrarre righe e colonne usando parentesi `[]` con una virgola inclusa. Gli indici prima della virgola sono righe:

```
# get the first row
mtcars[1, ]
# get the first five rows
mtcars[1:5, ]
```

Allo stesso modo, dopo la virgola sono colonne:

```
# get the first column
mtcars[, 1]
# get the first, third and fifth columns:
mtcars[, c(1, 3, 5)]
```

Come mostrato sopra, se le righe o le colonne sono vuote, tutto verrà selezionato. `mtcars[1, ]` indica la prima riga con *tutte* le colonne.

Con i nomi di colonna (e riga)

Finora, questo è identico a come si accede a righe e colonne di matrici. Con `data.frame` s, la maggior parte delle volte è preferibile utilizzare un nome di colonna per un indice di colonna. Questo viene fatto utilizzando un `character` con il nome della colonna anziché `numeric` con un numero di colonna:

```
# get the mpg column
mtcars[, "mpg"]
# get the mpg, cyl, and disp columns
```

```
mtcars[, c("mpg", "cyl", "disp")]
```

Anche se meno comuni, è possibile utilizzare anche i nomi delle righe:

```
mtcars["Mazda Rx4", ]
```

Righe e colonne insieme

Gli argomenti riga e colonna possono essere usati insieme:

```
# first four rows of the mpg column
mtcars[1:4, "mpg"]

# 2nd and 5th row of the mpg, cyl, and disp columns
mtcars[c(2, 5), c("mpg", "cyl", "disp")]
```

Un avvertimento sulle dimensioni:

Quando si utilizzano questi metodi, se si estrae più colonne, si otterrà un frame di dati
Tuttavia, se si estrae una *singola* colonna, si otterrà un vettore, non un frame di dati con le opzioni predefinite.

```
## multiple columns returns a data frame
class(mtcars[, c("mpg", "cyl")])
# [1] "data.frame"
## single column returns a vector
class(mtcars[, "mpg"])
# [1] "numeric"
```

Ci sono due modi per aggirare questo. Uno è quello di trattare il frame di dati come un elenco (vedi sotto), l'altro è quello di aggiungere un argomento `drop = FALSE` . Questo dice a R di non "eliminare le dimensioni inutilizzate":

```
class(mtcars[, "mpg", drop = FALSE])
# [1] "data.frame"
```

Si noti che le matrici funzionano allo stesso modo: per impostazione predefinita una singola colonna o riga sarà un vettore, ma se si specifica `drop = FALSE` è possibile mantenerlo come matrice a una o una riga.

Come una lista

I frame di dati sono essenzialmente `list` , cioè sono una lista di vettori di colonne (che devono avere tutti la stessa lunghezza). Le liste possono essere sottoinsieme usando parentesi singole `[` per un sottoelenco, o doppie parentesi `[[` per un singolo elemento.

Quando si utilizzano parentesi singole e nessuna virgola, si otterrà indietro la colonna perché i frame di dati sono elenchi di colonne.

```
mtcars["mpg"]
mtcars[c("mpg", "cyl", "disp")]
my_columns <- c("mpg", "cyl", "hp")
mtcars[my_columns]
```

Parentesi singole *come una lista* o parentesi *come una matrice*

La differenza tra `data[columns]` e `data[, columns]` è che quando si considera il `data.frame` come una `list` (nessuna virgola tra parentesi) l'oggetto restituito sarà *un* `data.frame` . Se si utilizza una virgola per trattare `data.frame` come una `matrix` selezione di una singola colonna restituirà un vettore ma selezionando più colonne verrà restituito un `data.frame` .

```
## When selecting a single column
## like a list will return a data frame
class(mtcars["mpg"])
# [1] "data.frame"
## like a matrix will return a vector
class(mtcars[, "mpg"])
# [1] "numeric"
```

Usare `$` per accedere alle colonne

Una singola colonna può essere estratta usando la scorciatoia magica `$` senza usare un nome di colonna quotato:

```
# get the column "mpg"
mtcars$mpg
```

Le colonne a cui si accede da `$` saranno sempre vettori, non frame di dati.

Svantaggi di `$` per l'accesso alle colonne

`$` Può essere una comoda scorciatoia, specialmente se si sta lavorando in un ambiente (come RStudio) che completerà automaticamente il nome della colonna in questo caso. **Tuttavia,** `$` ha anche degli svantaggi: utilizza *una valutazione non standard* per evitare la necessità di virgolette, il che significa che *non funzionerà* se il nome della colonna è memorizzato in una variabile.

```
my_column <- "mpg"
# the below will not work
mtcars$my_column
# but these will work
mtcars[, my_column]  # vector
mtcars[my_column]    # one-column data frame
mtcars[[my_column]]  # vector
```

A causa di questi timori, `$` viene utilizzato al meglio nelle sessioni R *interattive* quando i nomi delle colonne sono costanti. Per l'uso *programmatico* , ad esempio nella scrittura di una funzione generalizzabile che verrà utilizzata su set di dati diversi con nomi di colonne diversi, `$` dovrebbe essere evitato.

Si noti inoltre che il comportamento predefinito consiste nell'utilizzare la corrispondenza parziale solo quando si estrae da oggetti ricorsivi (eccetto ambienti) di `$`

```
# give you the values of "mpg" column
# as "mtcars" has only one column having name starting with "m"
mtcars$m
# will give you "NULL"
# as "mtcars" has more than one columns having name starting with "d"
mtcars$d
```

Indicizzazione avanzata: indici negativi e logici

Ogni volta che abbiamo la possibilità di utilizzare i numeri per un indice, possiamo anche usare numeri negativi per omettere determinati indici o un vettore booleano (logico) per indicare esattamente quali elementi conservare.

Gli indici negativi omettono elementi

```
mtcars[1, ]   # first row
mtcars[ -1, ] # everything but the first row
mtcars[-(1:10), ] # everything except the first 10 rows
```

I vettori logici indicano elementi specifici da mantenere

Possiamo usare una condizione come < per generare un vettore logico ed estrarre solo le righe che soddisfano la condizione:

```
# logical vector indicating TRUE when a row has mpg less than 15
# FALSE when a row has mpg >= 15
test <- mtcars$mpg < 15
```

```
# extract these rows from the data frame
mtcars[test, ]
```

Possiamo anche bypassare la fase di salvataggio della variabile intermedia

```
# extract all columns for rows where the value of cyl is 4.
mtcars[mtcars$cyl == 4, ]
# extract the cyl, mpg, and hp columns where the value of cyl is 4
mtcars[mtcars$cyl == 4, c("cyl", "mpg", "hp")]
```

Funzioni utili per manipolare data.frames

Alcune funzioni utili per manipolare `data.frames` sono `subset()` , `transform()` , `with()` e `within()` .

sottoinsieme

```
subset(mtcars, subset = cyl == 6, select = c("mpg", "hp"))
                mpg  hp
Mazda RX4      21.0 110
Mazda RX4 Wag  21.0 110
Hornet 4 Drive 21.4 110
Valiant        18.1 105
Merc 280       19.2 123
Merc 280C      17.8 123
Ferrari Dino   19.7 175
```

Nel codice sopra chiediamo solo le righe in cui `cyl == 6` e per le colonne `mpg` e `hp` . Puoi ottenere lo stesso risultato usando `[]` con il seguente codice:

```
mtcars[mtcars$cyl == 6, c("mpg", "hp")]
```

trasformare

La funzione `transform()` è una funzione utile per cambiare le colonne all'interno di un `data.frame` . Ad esempio, il codice seguente aggiunge un'altra colonna denominata `mpg2` con il risultato di `mpg^2` sul file `mtcars data.frame` :

```
mtcars <- transform(mtcars, mpg2 = mpg^2)
```

Introduzione rapida alle liste

In generale, la maggior parte degli oggetti con cui interagiresti come utente tenderebbe ad essere un vettore; ad esempio, il vettore numerico, il vettore logico. Questi oggetti possono contenere solo un singolo tipo di variabile (un vettore numerico può avere solo numeri al suo interno).

Una lista sarebbe in grado di memorizzare qualsiasi variabile di tipo in essa, rendendola all'oggetto generico in grado di memorizzare qualsiasi tipo di variabile di cui avremmo bisogno.

Esempio di inizializzazione di una lista

```
exampleList1 <- list('a', 'b')
exampleList2 <- list(1, 2)
exampleList3 <- list('a', 1, 2)
```

Per comprendere i dati che sono stati definiti nella lista, possiamo usare la funzione str.

```
str(exampleList1)
str(exampleList2)
str(exampleList3)
```

La suddivisione di elenchi distingue tra l'estrazione di una sezione dell'elenco, ovvero l'ottenimento di un elenco contenente un sottoinsieme degli elementi nell'elenco originale e l'estrazione di un singolo elemento. Usando `[` operatore comunemente usato per i vettori produce una nuova lista.

```
# Returns List
exampleList3[1]
exampleList3[1:2]
```

Per ottenere un singolo elemento usa `[[` invece.

```
# Returns Character
exampleList3[[1]]
```

È possibile accedere alle voci negli elenchi denominati in base al loro nome anziché al loro indice.

```
exampleList4[['char']]
```

In alternativa, l'operatore `$` può essere utilizzato per accedere agli elementi denominati.

```
exampleList4$num
```

Questo ha il vantaggio che è più veloce da digitare e può essere più facile da leggere, ma è importante essere consapevoli di una potenziale trappola. L'operatore `$` utilizza la corrispondenza parziale per identificare gli elementi della lista corrispondente e può produrre risultati imprevisti.

```
exampleList5 <- exampleList4[2:3]

exampleList4$num
# c(1, 2, 3)

exampleList5$num
# 0.5

exampleList5[['num']]
# NULL
```

Le liste possono essere particolarmente utili perché possono memorizzare oggetti di diverse lunghezze e di varie classi.

```
## Numeric vector
exampleVector1 <- c(12, 13, 14)
## Character vector
exampleVector2 <- c("a", "b", "c", "d", "e", "f")
## Matrix
exampleMatrix1 <- matrix(rnorm(4), ncol = 2, nrow = 2)
## List
exampleList3 <- list('a', 1, 2)

exampleList6 <- list(
    num = exampleVector1,
    char = exampleVector2,
    mat = exampleMatrix1,
    list = exampleList3
)
```