Scrivere le proprie funzioni

Di solito, le funzioni personalizzate vengono scritte dall'utente al solo scopo di semplificare l'uso di funzioni impiegate in attività di routine, o per evitare di ripetere lo stesso comando più di due o tre volte all'interno di uno script. Molte delle funzioni del pacchetto LabRS sono nate in questo modo.

All'interno di un progetto di analisi dei dati, è meglio avere un file di script con le funzioni personalizzate, separato da quello con il codice dell'analisi. All'inizio di quest'ultimo, si inserirà il comando `source()`, ad esempio: `source(funzioni.R)`. In questa maniera, le funzioni verranno caricate esattamente come quando si carica un pacchetto.

Vedi anche Funzioni

La struttura delle funzioni

Prendiamo ad esempio due semplici funzioni, `nmiss()` e `nval()`. In casi come questi, la funzione personalizzata viene creata allo scopo di digitare più rapidamente comandi già di loro piuttosto semplici, come ad esempio quello per avere il numero dei casi mancanti (in questo caso una variabile):

```
sum(is.na(MYSLID$Lingua))
```

La funzione ha questa forma (è possibile visualizzare le funzioni caricate con il comando `View()`):

nmiss.R

```
nmiss <- function(x) {
  sum(is.na(x))
}
```

- In primo luogo, si sceglie il nome della funzione, che sarà il comando che permetterà di eseguirla: `nmiss <- function(x)`, quindi `nmiss(x);`
- Poi, fra le parentesi graffe, si scrive il comando, sostituendo il nome dell'oggetto (`MYSLID$Lingua`) con la lettera `x`.

`nval()`, che restituisce il numero dei casi mancanti, ha la stessa struttura:

nval.R

```
nval <- function(x) {
  sum(!is.na(x))
}
```

Gli argomenti

Vediamo come usare gli argomenti prendendo ad esempio la funzione `percent()`. In questo caso, il comando "normale" che ci permette di calcolare le percentuali per una tabella a doppia entrata, è:

```
round(prop.table(tab3, margin = 2) * 100, digits = 1)
```

Che produce le percentuali di colonna (`margin = 2`) con un decimale (`digits = 1`) di una tabella. La funzione è:

percent.R

```
percent <- function(x, digits = 8, margin = NULL) {
  (round (prop.table(x, margin) * 100, digits))
}
```

L'oggetto di cui calcolare le percentuali è `x`. Gli argomenti vengono dichiarati in `function(x, digits = 8, margin = NULL)`:

- non è necessario indicare i valori di default (`digits = 8, margin = NULL`);
- all'interno delle parentesi graffe, il comando viene scritto con gli argomenti al loro posto, ma senza i valori: se non verrà indicato nessun valore, la funzione userà quelli di default. In questo modo, la funzione può essere utilizzata per oggetti ad una, due o più dimensioni, e il numero di decimali può essere scelto dall'utente.

Argomenti non modificabili

Possiamo però anche impostare una funzione in modo che *i valori degli argomenti siano fissi*. In questo caso, non li dichiareremo. Consideriamo ad esempio la funzione `expdata()`, che serve a esportare un dataframe in un file csv con le impostazioni di lingua italiana:

expdata.R

```
expdata <- function(x, file, ...) {
  write.table(x, file, dec = ",", sep = ";", na = "", row.names =
FALSE,
    ...)
}
```

- i soli argomenti dichiarati sono x (l'oggetto da esportare) e `file` (il nome del file di esportazione);
- alcuni argomenti della funzione `write.table()` — quelli che servono ad esportare il file con il formato corretto — sono impostati *all'interno della funzione*: essi sono dunque fissi e non sono più modificabili dall'utente.

Ho scritto la funzione in questo modo, per non sbagliare, oltre che per non fare continuamente copia-

incolla dei comandi.

Argomenti ereditati

In ` function(x, file, \ldots)` troviamo dei “puntini di sospensione”, ripetuti in `write.table(x, file, dec = ",", sep = ";", na = "", row.names = FALSE, \ldots)`: in questo modo, la funzione eredita gli argomenti di `write.table()`, senza necessità di dichiararli.

Funzioni, Linguaggio