Informatica

Principi di programmazione ed analisi dati in linguaggio

Scienze Biologiche

Dr. Bruno Bellisario, PhD

Esercitazione

Applichiamo quanto più possibile abbiamo imparato finora utilizzando il database fish, un file Excel in cui sono tabulati una serie di dati relativi alle caratteristiche funzionali di 635 diverse specie di pesci. Il file Excel è suddiviso in tre fogli:

1. Alimentazione
2. Dimensioni
3. Comportamento

Scopo della esercitazione è mettere in pratica tutta una serie di comandi imparati finora per la gestione delle tabelle dati.

Esercitazione

Creazione della directory di lavoro

Per prima cosa creiamo una directory di lavoro:

1. Scegliamo un percorso (Desktop o Documenti) e creiamo una cartella chiamata Lezione8
3. All'interno della cartella appena creata inseriamo il file Excel fish.xlsx
5. Apriamo RStudio e creiamo un nuovo script File -> New File -> R script
7. Settiamo la working directory Session -> Set Working Directory -> Choose Directory
9. Salviamo lo script R appena creato File -> Save

Esercitazione

IMPORTANTE!

Ricordiamoci sempre quale è la struttura logica del flusso di lavoro in R

Tutto ciò che facciamo in R è un oggetto: funzioni, variabili, vettori, operazioni

Se non assegnamo un nome agli oggetti, questi non possono essere riutilizzati per successive operazioni

`NOME1 = FUNZIONE1(ARG)` -> Rendo permanente una FUNZIONE1(ARG) assegnandole un NOME1
`FUNZIONE1(ARG)` -> Eseguo una FUNZIONE1(ARG) ma non la memorizzo (rendere permanente nel workspace)

L'oggetto NOME1 può essere pertanto utilizzato come argomento (ARG) di altre funzioni

`NOME2 = FUNZIONE2(NOME1)`

Esercitazione

Installazione e caricamento delle librerie e del dataset

Installiamo e carichiamo le library:

1. Ai fini della esercitazione dobbiamo installare caricare le librerie **readxl, tidyr, dplyr.**

```
install.packages("readxl")
install.packages("tidyr")
install.packages("dplyr")
library(readxl)
library(tidyr)
library(dplyr)
```

2. Carichiamo il dataset

```
fish=read_xlsx("fish.xlsx")#Carica di default solo il primo foglio Excel
Alimentazione=read_xlsx("fish.xlsx",sheet="alimentazione")#Carica il primo foglio
Dimensioni=read_xlsx("fish.xlsx",sheet="dimensioni")#Carica il secondo foglio
Comportamento=read_xlsx("fish.xlsx",sheet="comportamento")#Carica il terzo foglio
```

Esercitazione

Esplorazione rapida del dataset

Una volta importato il dataset (fish.xlsx) e caricato i tre fogli che compongono il dataset possiamo procedere ad una veloce sessione esplorativa per capire se e come i nostri dati sono stati importati correttamente:

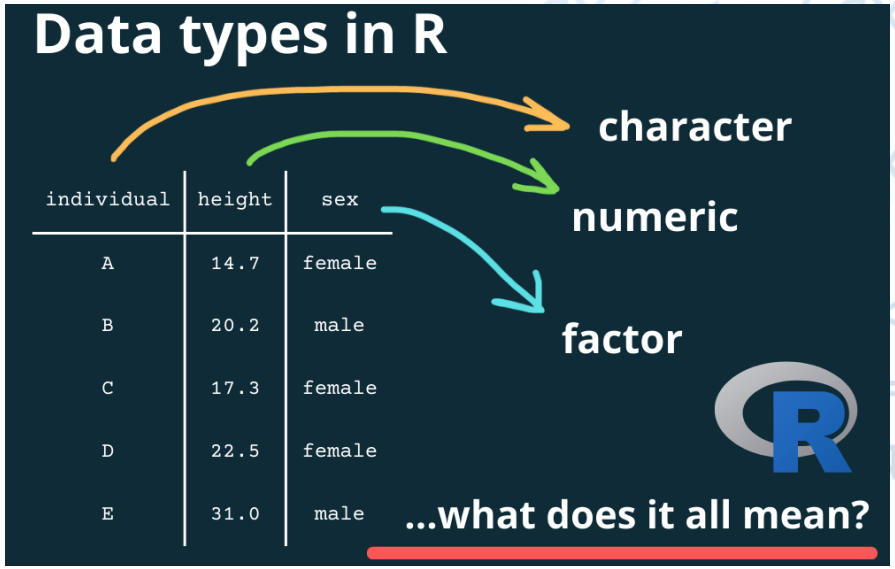
```
str(Alimentazione)
str(Dimensioni)
str(Comportamento)
```

La funzione str (**structure**) ci consente di esplorare le **classi di dati** che R interpreta quando carichiamo un dataset.

Esercitazione

Conversione delle classi di dati

L'importazione del dataset, soprattutto utilizzando file Excel, potrebbe portare a problemi dovuti alla diversa codifica dei dati.
Quello che in R viene riconosciuto come **chr** (character) è di per sé giusto. Tuttavia una classe dati **chr** non consente di operare operazioni di nessun tipo. Una più attenta lettura del dataset mostra come i dati **chr** sono in realtà una classe di dati particolari, **factor** (fattori).



Un **factor** in R è diverso da un **chr** perché un fattore rappresenta in realtà una classe di valori che racchiude in sé più character ripetuti, ovvero, dei **gruppi:**
Un esempio è il fattore sex, che contiene al suo interno diversi gruppi….male & female

Esercitazione

Conversione delle classi di dati

Per questo, dobbiamo operare delle "semplici" operazioni di conversione delle classi di dati:

Nella library **dplyr** esiste un comando molto utile in grado di identificare tutti i valori di una certa classe (es. **chr**) e convertirli in una classe più conveniente, usando la funzione coercitiva **as.***.

```
Alimentazione = Alimentazione %>% mutate_if(is.character, as.factor)
Dimensioni = Dimensioni %>% mutate_if(is.character, as.factor)
Comportamento = Comportamento %>% mutate_if(is.character, as.factor)
```

Con i comandi di cui sopra abbiamo creato un oggetto con lo stesso nome del dataframe originale (Alimentazione, Dimensioni e Comportamento) per evitare confusione con la creazione di n oggetti…

Esercitazione

Conversione delle classi di dati

Se andiamo nuovamente ad esplorare i vari dataframes, ci accorgeremo come anche i campi numerici sono diventati factor, perché nella importazione dei dati originali R aveva identificato questi come **chr** e non come numeri. Dobbiamo quindi convertire **questi** dati factor in numerici:

```
Alimentazione$Trophic_level = as.numeric(Alimentazione$Trophic_level)
Dimensioni$Common_length = as.numeric(Dimensioni$Common_length)
Dimensioni$Maximum_length = as.numeric(Dimensioni$Maximum_length)
```

Le funzioni di cui sopra prendono un campo preesistente (Alimentazione$Trophic_level) e lo convertono in una diversa classe di valori (in questo caso, **numeric**).

Riapplicando ora la funzione str ai dataframes dovremmo avere una più corretta struttura dei dati.

```
str(Alimentazione)
str(Dimensioni)
str(Comportamento)
```

Esercitazione

Effettuare la selezione per variabili, il subsetting

Molto spesso, diventa utile andare ad effettuare operazioni di selezione per variabili, definito subsetting.
Esistono diversi modi per effettuare tale operazione, sia con comandi di base, sia attraverso le funzioni presenti in dplyr e tidyr:

```
vettore_trophic_level = Alimentazione$Trophic_level#crea un vettore di dati
dframe_trophic_level = Alimentazione[,"Trophic_level"]#mantiene la struttura
dframe_trophic_level = Alimentazione[,numero_della_colonna_che_ci_interessa]#mantiene la struttura

#Usando dplyr le cose diventano molto più semplici
dframe_trophic_level = Alimentazione %>% select(Trophic_level)
```

Un aspetto interessante del subsetting con la funzione select è che possiamo estrarre tutte le variabili che vogliamo semplicemente utilizzandone il nome, ed il risultato rimarrà sempre un dataframe.

```
Alimentazione %>% select(Trophic_level, Family)#Estrae due tipi di variabili
```

Esercitazione

Effettuare la selezione per variabili, il subsetting

Così come è possibile selezionare le variabili di interesse, a volte può essere utile effettuare il subsetting sul numero di casi (osservazioni), ovvero le righe.

```
Alimentazione_prime_50_righe = Alimentazione[1:50,]#Estrae le righe da 1:50 mantenendo tutte le colonne (spazio vuoto dopo la virgola)

Alimentazione_prime_50_righe_e_variabile = Alimentazione[1:50,4]#Estrae le righe da 1:50 e la quarta colonna (4 dopo la virgola)
```

	[,1]	[,2]	[,3]
[1,]	10	3	6
[2,]	0	1	23
[3,]	8	4	55

→ `dataframe[righe , colonne]`

Esercitazione

Unione di dataframes

I dataset non sempre sono ordinati in maniera tale da avere tutti i dati a disposizione in un unico dataframe (o matrice). A volte possono presentarsi in diverse tabelle e, pertanto, potremmo avere la necessità di unirle.

In R esistono diversi modi per unire i dataframes, fermo restando la regola principale che, per essere uniti, debbano avere almeno un campo in comune. In generale, l'esplorazione delle colonne (variabili) comuni viene fatta visivamente. Esistono tuttavia una serie di comandi con i quali possiamo identificare quali e quante variabili una serie di tabelle hanno in comune.

Se abbiamo molte variabili e vogliamo ottimizzare i tempi, la library(janitor) è quello che ci serve.

```
install.packages("janitor")
library(janitor)
compare_df_cols(Alimentazione, Dimensioni, Comportamento, return="match", bind_method = "rbind")
```

La funzione compare_df_cols prende una serie di dataframes e restituisce le colonne in comune (return="match"), eliminando quelle non presenti in tutti i dataframes (bind_method = "rbind").

Esercitazione

Unione di dataframes

Una volta identificate le colonne in comune tra i dataframes non dovremmo fare altro che utilizzare le funzioni di unione (join o merge) utilizzando come campo di unione una colonna in comune.

Una delle funzioni base è merge():

```
Tab_unite = merge(Alimentazione, Dimensioni, by="Species")
```

La funzione *_join presente nel pacchetto **dplyr** consente di unire le tabelle in maniera più raffinata:

```
Alimentazione %>% inner_join(Dimensioni)#tiene solo gli elementi in comune
Alimentazione %>% left_join(Dimensioni)#tiene tutte le osservazioni a sx
Alimentazione %>% right_join(Dimensioni)#tiene tutte le osservazione a dx
Alimentazione %>% full_join(Dimensioni)#tiene tutte le osservazioni (in caso di osservazioni mancanti si creeranno NA)
```

Esercitazione

Unione di dataframes

Differentemente dalla funzione merge(), il *_join:

1. Riconosce automaticamente i campi in comune
2. Non replica i campi in comune

Le funzioni di merge, inoltre, può essere applicate soltanto tra due dataframes alla volta.

Come posso unire tre o più dataframes caratterizzati da campi in comune?
La più semplice: utilizzo una pipe (%>%) per concatenare gli output

```
Tab_unite = Alimentazione %>% inner_join(Dimensioni) %>% inner_join(Comportamento)
```

Esercitazione

Esportazione dei dataframes

Lavorando sulle tabelle si ottiene come risultato (molto spesso) una (o più) tabelle in cui sono presenti i valori delle analisi che abbiamo effettuato o un dataframe pulito (come nel nostro caso) che vogliamo esportare per conservarlo per future analisi.

Come detto, il formato più comune per salvare i dati è senza ombra di dubbio un formato di tipo testo .csv (o .txt), molto più "leggeri" di formati come l'Excel…

Per salvare in formato .csv possiamo usare la funzione presente in R write.csv():

```
write.csv(Tab_unite,"Tab_unite.csv")#salva in formato .csv (non specificare il separatore)
```

Se abbiamo la necessità di salvare in formato Excel allora abbiamo bisogno di una funzione presente nella library **writexl:**

```
install.packages("writexl")
library(writexl)
write_xlsx(Tab_unite,"Tab_unite.xlsx")
```

Esercitazione

Chiudere una sessione di lavoro

Una volta conclusa la sessione di lavoro può essere utile salvare non soltanto gli script in esecuzione, ma anche i risultati che abbiamo ottenuto.
Questo è particolarmente importante quando abbiamo a che fare con notevoli moli di dati che possono a volte portar via ore di tempo per fare delle semplici analisi di pulizia dei datasets.

Una opzione è quella di inserire una semplicissima riga di codice alla fine dello script:

```
save.image(file = “fish.RData”)
```

Il breve codice qui sopra salva tutto quello che abbiamo creato durante la sessione di lavoro (risultati compresi) in un file .RData, che può essere richiamato all’avvio della successiva sessione con:

```
load(“fish.RData”)
```

Esercitazione

Automatizzare la chiusura/avvio di una sessione di lavoro

Esiste tuttavia una maniera più rapida, efficace ed automatizzata di salvare e caricare un workspace.

L'unica "pecca" di tale metodologia è che sarà applicata sempre e comunque ad ogni sessione, salvando e caricando ogni volta tutti i workspace che abbiamo creato su diversi progetti: