3.3_apply

`*apply` family

Immaginate di avere una lista di vettori, e di voler applicare la stessa funzione/i ad ogni elemento della lista:

- applico manualmente la funzione selezionando gli elementi
- ciclo `for` che itera sugli elementi della lista e applica la funzione/i

  ...

```r
my_list=list(
    vec1=rnorm(100),
    vec2=runif(100),
    vec3=rnorm(100),
    vec4=rnorm(100)
)
```

*apply family

Applichiamo media, mediana e std

```r
# inizializzo i vettori
means=vector(mode = "numeric",
             length = length(my_list))
medians=vector(mode = "numeric",
               length = length(my_list))
stds=vector(mode = "numeric",
            length = length(my_list))

# Loop
for(i in 1:length(my_list)){

  means[i] <- mean(my_list[[i]])
  medians[i] <- median(my_list[[i]])
  stds[i] <- sd(my_list[[i]])
}
```

Risultato

```r
means
```

```
[1] -0.06244243  0.52132264
0.05718438 -0.02769332
```

```r
medians
```

```
[1] -0.04877713  0.52255677
0.09084072  0.07004321
```

```r
stds
```

```
[1] 0.9455751 0.2614234
1.1453929 1.0212599
```

`*apply` family

Funziona tutto! ma:

- il for è molto laborioso da scrivere gli indici sia per la lista che per il vettore che stiamo popolando
- dobbiamo pre-allocare delle variabili (per il motivo della velocità che dicevo)
- 8 righe di codice (per questo esempio semplice)

*apply family

In R è presente una famiglia di funzioni apply come **lapply**, **sapply**, etc. che permettono di ottenere lo stesso risultato in modo più conciso, rapido e semplice:

```
means=sapply(my_list, mean)
medians=sapply(my_list, median)
stds=sapply(my_list, sd)

means
```

```
       vec1         vec2         vec3         vec4 
-0.06244243  0.52132264  0.05718438 -0.02769332 
```

```
medians
```

```
       vec1         vec2         vec3         vec4 
-0.04877713  0.52255677  0.09084072  0.07004321 
```

```
stds
```

```
     vec1       vec2       vec3       vec4 
0.9455751 0.2614234 1.1453929 1.0212599 
```

`*apply` family

apply(< lista > , < funzione >)

- Cosa può essere la **lista**?
  - lista
  - dataframe
  - vettore
- Cosa può essere la **funzione**?
  - base o importata da un pacchetto
  - custom
  - anonima

*apply family

Prima di analizzare l'*apply family, credo sia utile un ulteriore parallelismo con il ciclo for che abbiamo visto. **apply** non è altro che un ciclo **for**, leggermente semplificato.

Ciclo for

```
1  vec = 1:5
2  for(i in vec){
3    print(i)}
```

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

sapply

```
1  vec = 1:5
2  res = sapply(vec, print)
```

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

*apply family - funzione custom

Possiamo utlizzare anche funzioni create da noi:

```r
center_var = function(x) x - mean(x)

my_list = list(
    vec1 = runif(10),
    vec2 = runif(10),
    vec3 = runif(10)
)

lapply(my_list, center_var)
```

```
$vec1
 [1] -0.0869647245 -0.2413591818  0.3104610218 -0.1848986781 -0.0006296035
 [6]  0.4223377566 -0.0869682684  0.2025990224 -0.0119473644 -0.3226299802

$vec2
 [1] -0.37761481  0.44598750  0.22337399 -0.34515876 -0.02681381  0.07327309
 [7]  0.04524927 -0.33909030  0.52355547 -0.22276163

$vec3
 [1]  0.27791786 -0.25086747 -0.24096777 -0.24830888  0.03075864 -0.25943905
 [7]  0.07306628  0.15212384  0.24749477  0.21822178
```

`*apply` family - implicito vs. esplicito

Quindi come il ciclo `for` scritto come **i** in **vec** assegna al valore **i** un **elemento** per volta dell'oggetto **vec**, internamente le funzioni `*apply` prendono il **primo elemento** dell'oggetto in **input** (lista) e **applicano** direttamente la funzione che abbiamo scelto.

`sapply` implicito

```
1 vec = 1:5
2 res = sapply(vec, print)
```

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

`sapply` esplicito

```
1 vec = 1:5
2 res = sapply(vec, function(i) print(i))
```

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

*apply family - funzione anonima

Una funzione anonima è una funzione non salvata in un oggetto ma scritta per essere **eseguita direttamente**, all'interno di altre funzioni che lo permettono:

```
1  lapply(my_list, function(x) x - mean(x))
```

```
$vec1
 [1] -0.0869647245 -0.2413591818  0.3104610218 -0.1848986781 -0.0006296035
 [6]  0.4223377566 -0.0869682684  0.2025990224 -0.0119473644 -0.3226299802

$vec2
 [1] -0.37761481  0.44598750  0.22337399 -0.34515876 -0.02681381  0.07327309
 [7]  0.04524927 -0.33909030  0.52355547 -0.22276163

$vec3
 [1]  0.27791786 -0.25086747 -0.24096777 -0.24830888  0.03075864 -0.25943905
 [7]  0.07306628  0.15212384  0.24749477  0.21822178
```

x è solo un **placeholder** (analogo di i) e può essere qualsiasi lettera o nome!

Tutte le tipologie di `*apply`

- `lapply()`: la funzione di base
- `sapply()`: simplified-apply
- `tapply()`: poco utilizzata, utile con i fattori
- `apply()`: utile per i dataframe/matrici
- `mapply()`: versione multivariata, utilizza più liste contemporaneamente
- `vapply()`: utilizzata dentro le funzioni e pacchetti

lapply

lapply sta per list-apply e restituisce sempre una lista, applicando la funzione ad ogni elemento della lista in input:

```
1  res=lapply(my_list, mean)
2  res
```

```
$vec1
[1] 0.355799

$vec2
[1] 0.4039008

$vec3
[1] 0.4435136
```

```
1  class(res)
```

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

sapply

sapply sta per simplified-apply e (cerca) di restituire una versione più semplice di una lista, applicando la funzione ad ogni elemento della lista in input:

```
1 res=sapply(my_list, mean)
2 res
```

```
     vec1      vec2      vec3 
0.3557990 0.4039008 0.4435136 
```

```
1 class(res)
```

```
[1] "numeric"
```

apply

apply funziona in modo specifico per dataframe o matrici, applicando una funzione alle righe o alle colonne:

```
1 my_df =data.frame(x1 = runif(5,1,10), x2 = runif(5,3,4))
2 my_df
```

```
        x1       x2
1 8.056291 3.009874
2 6.782558 3.776416
3 3.007746 3.832786
4 8.077636 3.666191
5 9.784749 3.227990
```

apply

Applico a tutte le righe (1) la funzione mean:

```
1 apply(my_df, MARGIN = 1, FUN = mean)
```

```
[1] 5.533083 5.279487 3.420266 5.871914 6.506370
```

Applico a tutte le colonne (2) la funzione mean:

```
1 apply(my_df, MARGIN = 2, FUN = mean)
```

```
      x1       x2 
7.141796 3.502651 
```

tapply

tapply permette di applicare una funzione ad un vettore, dividendo questo vettore in base ad una variabile categoriale:

```r
vec=rnorm(75)

index=rep(c("a", "b", "c"), each = 25)

tapply(vec, INDEX = index, mean)
```

```
         a          b          c 
0.07363169 0.17311330 0.02408607 
```

Qui dove *INDEX* è un vettore stringa o un fattore.

tapply

In questo caso calcoliamo la media per ogni categoria d'età:

```
1 my_df = readr::read_csv("data/mydf_2.csv")
2 head(my_df)
```

```
# A tibble: 6 × 4
     id   age age_cat        age_z
  <dbl> <dbl> <chr>          <dbl>
1     1    44 adulto        1.15
2     2    18 adolescente  -1.19
3     3    32 adulto        0.0721
4     4    25 giovane      -0.559
5     5    45 adulto        1.24
6     6    33 adulto        0.162
```

```
1 tapply(my_df$age, my_df$age_cat, mean)
```

```
adolescente      adulto     giovane
   16.50000    40.73333    25.11111
```

vapply

vapply è una versione più solida delle precedenti dal punto di vista di programmazione. In pratica permette (e richiede) di specificare in anticipo la tipologia di dato che ci aspettiamo come risultato:

```
1 vapply(my_list, FUN = mean, FUN.VALUE = numeric(length = 1))
```

```
     vec1      vec2      vec3
0.3557990 0.4039008 0.4435136
```

FUN.VALUE = numeric(length = 1): indica che ogni risultato è un singolo valore numerico.

mapply

mapply permette di gestire **più liste contemporaneamente** per scenari più complessi. Ad esempio vogliamo usare la funzione **rnorm()** e generare 4 con diverse medie e deviazioni stardard in combinazione:

```
1 medie=list(10, 20, 30, 40)
2 stds=list(1, 2, 3, 4)
3 mapply(function(x,y) rnorm(n = 5, mean = x, sd = y), medie, stds, SIMPLIFY
```

```
[[1]]
[1] 10.23285 11.63268 10.06450 11.19754 11.41108

[[2]]
[1] 22.70248 18.05308 20.48019 22.12845 22.07240

[[3]]
[1] 33.17335 32.46833 25.14690 25.13352 33.61143

[[4]]
[1] 37.29529 41.58475 42.75991 47.85355 41.24995
```

IMPORTANTE, tutte le liste incluse devono avere la stessa dimensione!

mapply

mapply(function(x,y) rnorm(n = 4, mean = x, sd = y), medie, stds, SIMPLIFY = FALSE)

- `function(...)`: è una funzione anonima come abbiamo visto prima che può avere n elementi
- `rnorm(n = 10, mean = x, sd = y)`: è l'effettiva funzione anonima dove abbiamo i placeholders x and y
- `medie, stds`: sono in **ordine** le liste corrispondenti ai placeholders indicati, quindi x = medie e y = stds
- `SIMPLIFY = FALSE`: semplicemente dice di restituire una lista e non cercare (come sapply) di semplificare il risultato

`mapply` come `for`

Lo stesso risultato (in modo più verboso) si ottiene con un `for` usando più volte l'iteratore i:

```
1 medie=list(10, 20, 30)
2 stds=list(1,2,3)
3 res=vector(mode = "list", length = length(medie)) # lista vuota
4
5 for(i in 1:length(medie)){
6   res[[i]] = rnorm(6, mean = medie[[i]], sd = stds[[i]])
7 }
8 res
```

```
[[1]]
[1] 10.371779 10.216703  8.472698 11.663369  8.420294 10.753599

[[2]]
[1] 17.86660 20.17946 17.63329 20.26231 16.86248 20.23591

[[3]]
[1] 31.45917 29.89232 31.99641 24.74793 23.14769 28.89921
```

`replicate`

Questa funzione permette di ripetere un operazione n volte, senza però utilizzare un iteratore o un placeholder.

`replicate(n, expr)`

- ***`n`*** è il numero di ripetizioni
- ***`expr`*** è la porzione di codice da ripetere

replicate

Campioniamo 1000 volte da una normale e facciamo la media
AKA distribuzione campionaria della media

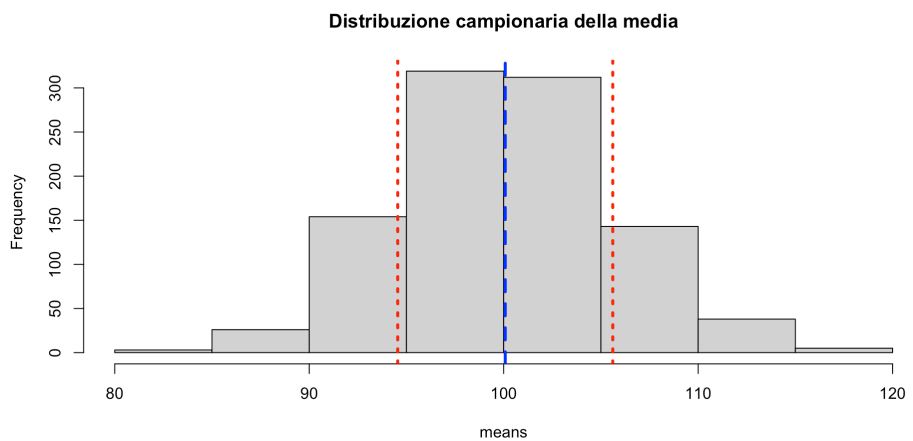
```r
nrep=1000

nsample=30

media=100

sd=30

means=
  replicate(
    n = nrep,
    expr = {mean(
      rnorm(nsample, media, sd))}
    )
```



Ora facciamo un po’ di pratica!

Aprite e tenete aperto questo link:

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