Modelos de estadística clásica: muestreos aleatorios

Armando Ocampo

Librerías de trabajo

Para esta clase necesitamos cargar las siguientes librerias

```
library(dplyr)
library(ggplot2)
```

Dudas de la clase previa

Recodificar NA's

Los NA's son valores faltantes o no disponibles, los cuales suelen representar un problema durante la transformación y procesamiento de los datos. Por lo cual, previo a realizar cualquier análisis o función sobre el dataset es necesario conocer el comportamiento de nuestros datos. Para esto utilizaremos la función *summary()* de la paquetería *base* (Esta paquetería se encuentra instalada por defecto en el lenguaje de programación R, por lo cual no es necesario instalarla).

Generaremos un data frame con valores NA.

```
dummy_data <- data.frame(id=c(1,2,3,4,NA),
                         horas_estudio=c(2, 5, 4, 2, 1),
                         horas_recre=c(4, 2, 4, 3, 1),
                         edad = c(20, 15, 16, 22, 20))

summary(dummy_data)
```

```
##        id       horas_estudio  horas_recre       edad
##  Min.   :1.00   Min.   :1.0   Min.   :1.0   Min.   :15.0
##  1st Qu.:1.75   1st Qu.:2.0   1st Qu.:2.0   1st Qu.:16.0
##  Median :2.50   Median :2.0   Median :3.0   Median :20.0
##  Mean   :2.50   Mean   :2.8   Mean   :2.8   Mean   :18.6
##  3rd Qu.:3.25   3rd Qu.:4.0   3rd Qu.:4.0   3rd Qu.:20.0
##  Max.   :4.00   Max.   :5.0   Max.   :4.0   Max.   :22.0
##  NA's   :1
```

Además de algunas medidas de estadísitca descriptiva, esta función detalla si existen NA's en nuestro dataset. Otra manera de conocer si existen o no NA's es mediante la función *is.na()*. Esta generará como resultado TRUE, en el sitio donde encuentre un resultado faltante.

```
is.na(dummy_data)
```

```
##         id horas_estudio horas_recre  edad
## [1,] FALSE         FALSE       FALSE FALSE
## [2,] FALSE         FALSE       FALSE FALSE
## [3,] FALSE         FALSE       FALSE FALSE
## [4,] FALSE         FALSE       FALSE FALSE
## [5,]  TRUE         FALSE       FALSE FALSE
```

```
# podemos acompañarlo con la función sum(), para saber el total de NA's
sum(is.na(dummy_data))
```

```
## [1] 1
```

```
# También, es posible utilizar la función which() para determinar el sitio
# en la columna donde se encuentra el NA. En este caso, conocemos que el NA
# se encuentra en la columna id, ahora identificaremos su posición

which(is.na(dummy_data$id))
```

```
## [1] 5
```

```
# Es así, como determinamos qué el valor faltante se encuentra en el quinto
# lugar de la columna id
```

Otra manera es utilizar la función *sapply()* acompañado de la función *sum()* para determinar el total de NA's por columna

```
sapply(dummy_data, function(y) sum(length(which(is.na(y)))))
```

```
##            id horas_estudio   horas_recre          edad
##             1             0             0             0
```

Estas son algunas maneras de identificar los valores faltantes. A continuación, se mostrarán algunos métodos para su eliminación.

El primer método es la recodificación, en el cual se le agrega un valor concreto al NA. Esto, al colocar la posición en la que se encuentra y renombrarla. En este caso, al ver la numeración, suponemos que el id faltante es el numero 5.

```
dummy_data$id[is.na(dummy_data$id)] = 5
dummy_data
```

```
##   id horas_estudio horas_recre edad
## 1  1             2           4   20
## 2  2             5           2   15
## 3  3             4           4   16
## 4  4             2           3   22
## 5  5             1           1   20
```

Vamos a generar de nuevo el valor NA. Sin embargo, en el siguiente ejemplo utilizaremos la función *na.omit()* para quitar el valor faltante. Esta función se caracteriza por eliminar la fila donde se encuentra el valor NA.

```
dummy_data <- data.frame(id=c(1,2,3,4,NA),
                         horas_estudio=c(2, 5, 4, 2, 1),
                         horas_recre=c(4, 2, 4, 3, 1),
                         edad = c(20, 15, 16, 22, 20))

na.omit(dummy_data)
```

```
##   id horas_estudio horas_recre edad
## 1  1             2           4   20
## 2  2             5           2   15
## 3  3             4           4   16
## 4  4             2           3   22
```

Nota: *na.omit()* puede ser funcional si la recodificación no es posible. No obstante, hay que tener cuidado en no eliminar la mayor parte de la información. Esto lo veremos a continuación

Si tenemos varios NA a lo largo del dataset, *na.omit()* puede ser contraproducente. El siguiente ejemplo lo detalla.

```
dummy_data_2 <- data.frame(id=c(1,2,3,4,NA,6,7,8,9,10,11,12),
                           horas_estudio=c(NA,5,4,2,1,3,NA,2,4,3,5,NA),
                           horas_recre=c(9,NA,8,3,NA,8,5,6,8,NA,4,7),
                           edad = c(20,15,NA,22,NA,17,18,NA,18,20,18,15))

# contando NA's por columna
sapply(dummy_data_2, function(y) sum(length(which(is.na(y)))))
```

```
##            id horas_estudio   horas_recre          edad
##             1             3             3             3
```

Al aplicar la función *na.omit()* perdemos la mayor parte de la información

```
na.omit(dummy_data_2)
```

```
##    id horas_estudio horas_recre edad
## 4   4             2           3   22
## 6   6             3           8   17
## 9   9             4           8   18
## 11 11             5           4   18
```

Para este caso, podemos hacer dos maneras de recodificación. Colocar un 0 en todos los NA's, o definir un valor específico por columna.

```
dummy_data_2 <- data.frame(id=c(1,2,3,4,NA,6,7,8,9,10,11,12),
                           horas_estudio=c(NA,5,4,2,1,3,NA,2,4,3,5,NA),
                           horas_recre=c(9,NA,8,3,NA,8,5,6,8,NA,4,7),
                           edad = c(20,15,NA,22,NA,17,18,NA,18,20,18,15))

dummy_data_2[is.na(dummy_data_2)] = 0

dummy_data_2
```

```
##    id horas_estudio horas_recre edad
## 1   1             0           9   20
## 2   2             5           0   15
## 3   3             4           8    0
## 4   4             2           3   22
## 5   0             1           0    0
## 6   6             3           8   17
## 7   7             0           5   18
## 8   8             2           6    0
## 9   9             4           8   18
## 10 10             3           0   20
## 11 11             5           4   18
## 12 12             0           7   15
```

Para el segundo caso, colocaremos el valor de la media aritmética por columna para el proceso de recodificación.

```
dummy_data_2 <- data.frame(id=c(1,2,3,4,NA,6,7,8,9,10,11,12),
                           horas_estudio=c(NA,5,4,2,1,3,NA,2,4,3,5,NA),
                           horas_recre=c(9,NA,8,3,NA,8,5,6,8,NA,4,7),
                           edad = c(20,15,NA,22,NA,17,18,NA,18,20,18,15))

mean(dummy_data_2$horas_estudio, na.rm = TRUE)
```

```
## [1] 3.222222
```

```
dummy_data_2$horas_estudio[is.na(dummy_data_2$horas_estudio)]=2.4

mean(dummy_data_2$horas_recre, na.rm = TRUE)
```

```
## [1] 6.444444
```

```
dummy_data_2$horas_recre[is.na(dummy_data_2$horas_recre)] = 4.8

mean(dummy_data_2$edad, na.rm = TRUE)
```

```
## [1] 18.11111
```

```
dummy_data_2$edad[is.na(dummy_data_2$edad)] = 13

dummy_data_2
```

```
##    id horas_estudio horas_recre edad
## 1   1           2.4         9.0   20
## 2   2           5.0         4.8   15
## 3   3           4.0         8.0   13
## 4   4           2.0         3.0   22
## 5  NA           1.0         4.8   13
## 6   6           3.0         8.0   17
## 7   7           2.4         5.0   18
## 8   8           2.0         6.0   13
## 9   9           4.0         8.0   18
## 10 10           3.0         4.8   20
## 11 11           5.0         4.0   18
## 12 12           2.4         7.0   15
```

Nota: es posible combinar métodos de recodificación. Todo depende del objetivo de la limpieza del conjunto de datos

Identificando cuartiles

Los cuartiles dividen al conjunto de datos en 4 grupos, a partir de 3 puntos de corte. Primer cuartil (Q1, 25%), segundo cuartil (Q2, 50%), tercer cuartil (Q3, 75%). Derivado de esta información, podemos definir en que sitio se puede encontrar un dato nuevo. Los siguientes son solo ejemplo de cómo realizarlo, ya que pueden existir varias maneras de hacerlo.

Primero identificaremos los cuartiles de la longitud del sépalo del dataset iris

```
quantile(iris$Sepal.Length)
```

```
##   0%  25%  50%  75% 100%
##  4.3  5.1  5.8  6.4  7.9
```

La siguiente función genera un mensaje dependiendo del sitio en el cual se encuentra el nuevo valor

```
identificando_cuartiles <- function(x){
  if(x <= 5.1) print('debajo Q1')
  if(x > 5.1 & x <= 5.8) print('entre Q1 y Q2')
  if(x >5.8 & x <= 6.4) print('entre Q2 y Q3')
  if(x > 6.4) print('superior a Q3')
}

identificando_cuartiles(6.5)
```

```
## [1] "superior a Q3"
```

La función *case_when()* de la paquetería *dplyr* permite generar condiciones similares a las utilizadas en la función *if()*. Cuando se utiliza en conjunto a la función *mutate()*, los resultados se guardan en una columna nueva.

```
iris_df <- iris

quantile(iris_df$Sepal.Length)
```

```
##   0%  25%  50%  75% 100%
##  4.3  5.1  5.8  6.4  7.9
```

```
iris_df_cuartil <- iris_df %>%
  mutate(cuartil = case_when(Sepal.Length <= 5.1 ~ '< Q1',
                             Sepal.Length > 5.1 & Sepal.Length <= 5.8 ~ 'Q1 & Q2',
                             Sepal.Length > 5.8 & Sepal.Length <= 6.4 ~ 'Q1 & Q3',
                             Sepal.Length > 6.4 ~ '> Q3'))

head(iris_df_cuartil)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species cuartil
## 1          5.1         3.5          1.4         0.2  setosa    < Q1
## 2          4.9         3.0          1.4         0.2  setosa    < Q1
## 3          4.7         3.2          1.3         0.2  setosa    < Q1
## 4          4.6         3.1          1.5         0.2  setosa    < Q1
## 5          5.0         3.6          1.4         0.2  setosa    < Q1
## 6          5.4         3.9          1.7         0.4  setosa Q1 & Q2
```

De esta manera, se puede colocar una variable nueva, pegarla al dataset y conocer en que cuartil se encuentra.

```
iris_df <- iris
iris_nueva <- c(5.222,NA,NA,NA,NA)

new_iris <- rbind(iris_nueva, iris_df)

head(new_iris %>%
       mutate(cuartil = case_when(Sepal.Length <= 5.1 ~ '< Q1',
                                  Sepal.Length > 5.1 & Sepal.Length <= 5.8 ~ 'Q1 & Q2',
                                  Sepal.Length > 5.8 & Sepal.Length <= 6.4 ~ 'Q1 & Q3',
                                  Sepal.Length > 6.4 ~ '> Q3')),1)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species cuartil
## 1        5.222          NA           NA          NA    <NA> Q1 & Q2
```