Tarea 1. FMAD 2021-2022

ICAI. Máster en Big Data. Fundamentos Matemáticos del Análisis de Datos (FMAD).

Rodríguez González, Álvaro

Curso 2021-22. Última actualización: 2021-09-17

Contents

Ejercicio 1. Análisis exploratorio de un conjunto de datos y operaciones con dplyr.

Lo primero que haremos será cargar el fichero de datos choresterol.csv como chlstrl:

```
chlstrl = read.csv("./data/cholesterol.csv",header = TRUE, sep= ",")
knitr::kable(head(chlstrl,10))
```

chol	age	gender	height	weight	waist	hip
203	46	female	62	121	29	38
165	29	female	64	218	46	48
228	58	female	61	256	49	57
78	67	male	67	119	33	38
249	64	male	68	183	44	41
248	34	male	71	190	36	42
195	30	male	69	191	46	49
227	37	male	59	170	34	39
177	45	male	69	166	34	40
263	55	female	63	202	45	50

Antes de comenzar a realizar un análisis más técnico vamos a comentar como es la estructura del data frame sobre el que vamos a trabajar:

- El data frame está formado por 7 columnas (es decir, las variables con las que vamos a trabajar) y de 403 filas (las observaciones que tenemos, nuestros datos a analizar).
- Cabe destacar que dentro del data frame encontramos datos ausentes como son el chol de la fila 28, el height de la fila 64, 87, 196, 232 y 318, el weight de la fila 162, el weist de la fila 337 y 394 y la heist de la fila 337 y 394.

Esto también lo podemos hacer con R a través de los siguientes bloques de código:

```
str(chlstrl)
```

```
## 'data.frame':    403 obs. of  7 variables:
##  $ chol  : int  203 165 228 78 249 248 195 227 177 263 ...
##  $ age   : int  46 29 58 67 64 34 30 37 45 55 ...
##  $ gender: chr  "female" "female" "female" "male" ...
##  $ height: int  62 64 61 67 68 71 69 59 69 63 ...
##  $ weight: int  121 218 256 119 183 190 191 170 166 202 ...
##  $ waist : int  29 46 49 33 44 36 46 34 34 45 ...
##  $ hip   : int  38 48 57 38 41 42 49 39 40 50 ...
```

```
sum(is.na(chlstrl))
```

```
## [1] 11
```

```
which(is.na(chlstrl))
```

```
##  [1]   28 1273 1296 1405 1441 1527 1774 2352 2409 2755 2812
```

Una vez revisada la estructura del data frame podemos comenzar con el análisis exploratorio, para el cual trabajaremos en función del tipo de variable, distinguiendo entre variable cuantitativa y cualitativa. Comenzaremos con la única variable cualitativa que encontramos en el data frame, el género.

Para comenzar el análisis exploratorio de la variable género calcularemos la frecuencia absoluta y la frecuencia relativa:

```
table(chlstrl$gender)
```

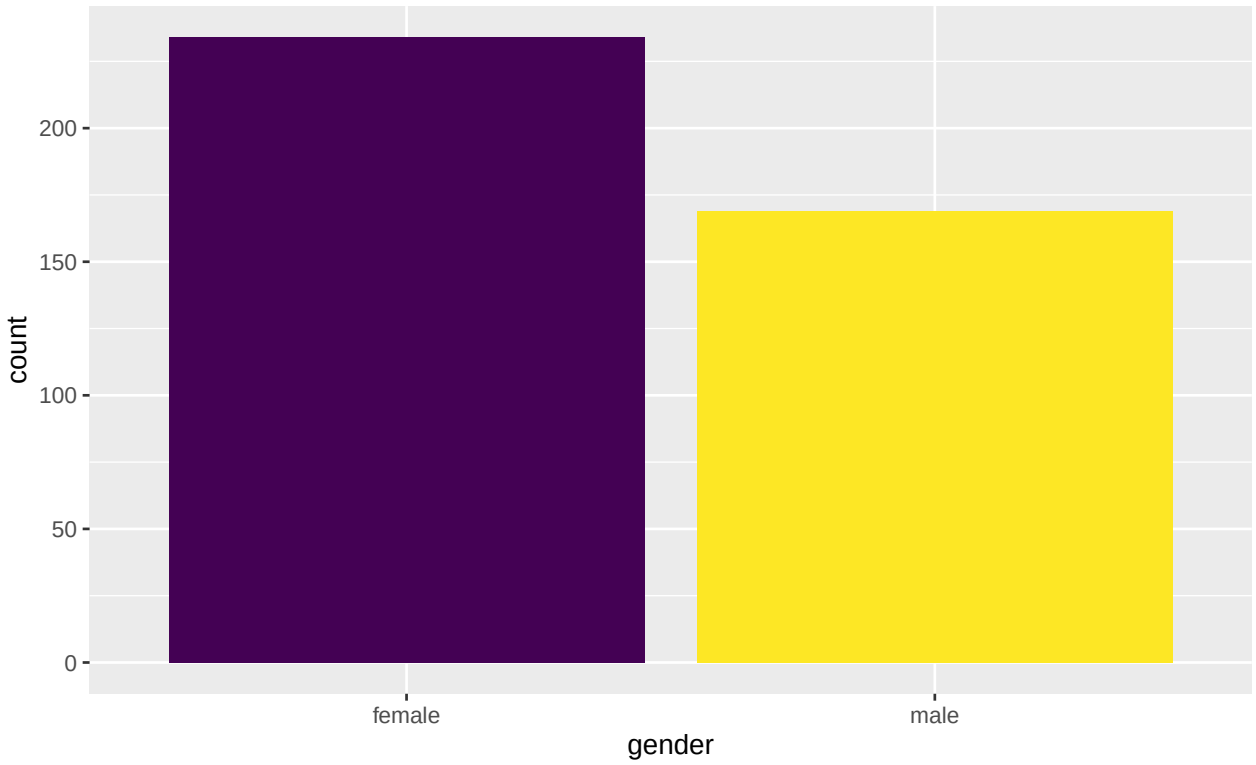
```
##
## female   male
##    234    169
```

```
signif(prop.table(table(chlstrl$gender)),3)
```

```
##
## female   male
##  0.581  0.419
```

Por último haremos un diagrama de barras donde veremos la distribución de hombres y mujeres en los encuestados:

```
ggplot(chlstrl) + geom_bar(mapping = aes(x = gender), fill= viridis(2))
```



Una vez hecho el análisis exploratorio del caso cualitativo realizaremos el análisis exploratorio de una variable cuantitativa, que en nuestro caso serán tanto la edad (que no contiene valores nulos) como el colesterol (que sí contiene valores nulos).

Lo primero que haremos será calcular un resumen numérico básico (donde se calculará el valor mínimo, máximo, el primer y tercer cuartil, la media y la mediana así como la desviación típica y la varianza), primero lo haremos de la edad y posteriormente del colesterol.

```
summary(chlstrl$age)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   19.00   34.00   45.00   46.85   60.00   92.00
```

```
sd(chlstrl$age)
```

```
## [1] 16.31233
```

```
var(chlstrl$age)
```

```
## [1] 266.0922
```

```
summary(chlstrl$chol)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##    78.0   179.0   204.0   207.8   230.0   443.0       1
```

```
sd(chlstrl$chol, na.rm = TRUE)
```

```
## [1] 44.44556
```

```
var(chlstrl$chol, na.rm = TRUE)
```

```
## [1] 1975.408
```

Otra forma de hacerlo, aunque bastante más laboriosa, si nuestra variable tiene valores NA es la siguiente:

```
chlstrl_na <- chlstrl[!is.na(chlstrl$chol),]
chol_sin_na <- chlstrl_na$chol
sd(chol_sin_na)
```

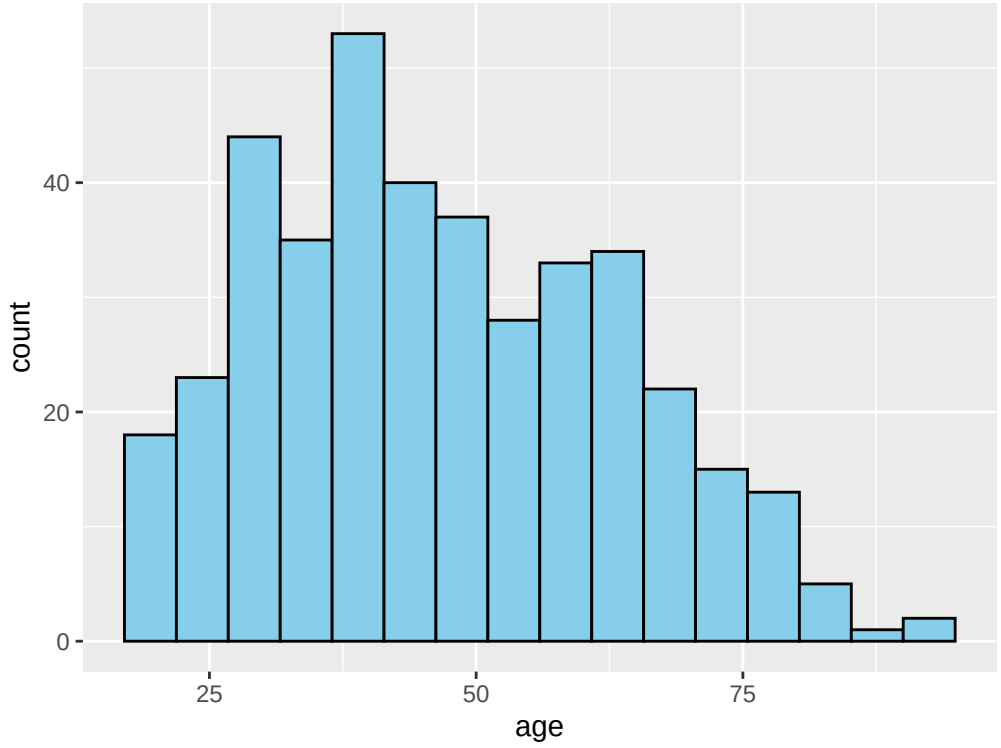
```
## [1] 44.44556
```

```
var(chol_sin_na)
```

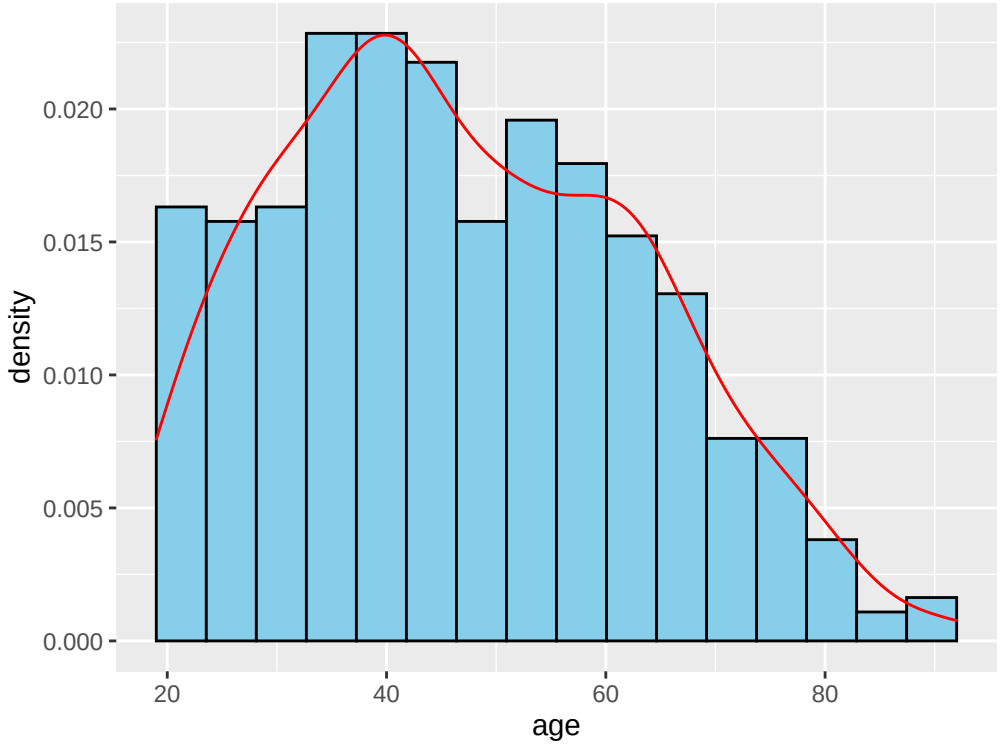
```
## [1] 1975.408
```

Una vez hecho el resumen numérico de ambas variables pasaremos al resumen gráfico, el cual se adaptará al tipo de variable y la diferente interpretación de los datos.

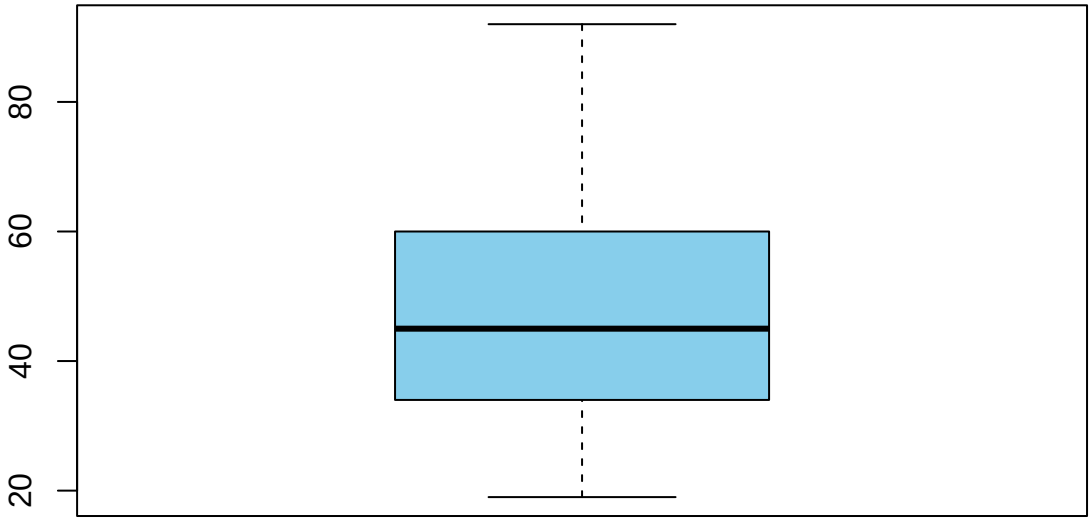
```
ggplot(chlstrl) + geom_histogram(mapping = aes(x = age), bins = 16,
                fill = "skyblue", color="black" )
```




```
ggplot(chlstrl, aes(x =age)) +
  geom_histogram(aes(y=stat(density)), breaks = seq(min(chlstrl$age),
      max(chlstrl$age), length.out = 17) , fill = "skyblue", color="black") +
      geom_density(color ="red")
```



```
boxplot(chlstrl$age,col="skyblue", mapping = aes(y = age))
```

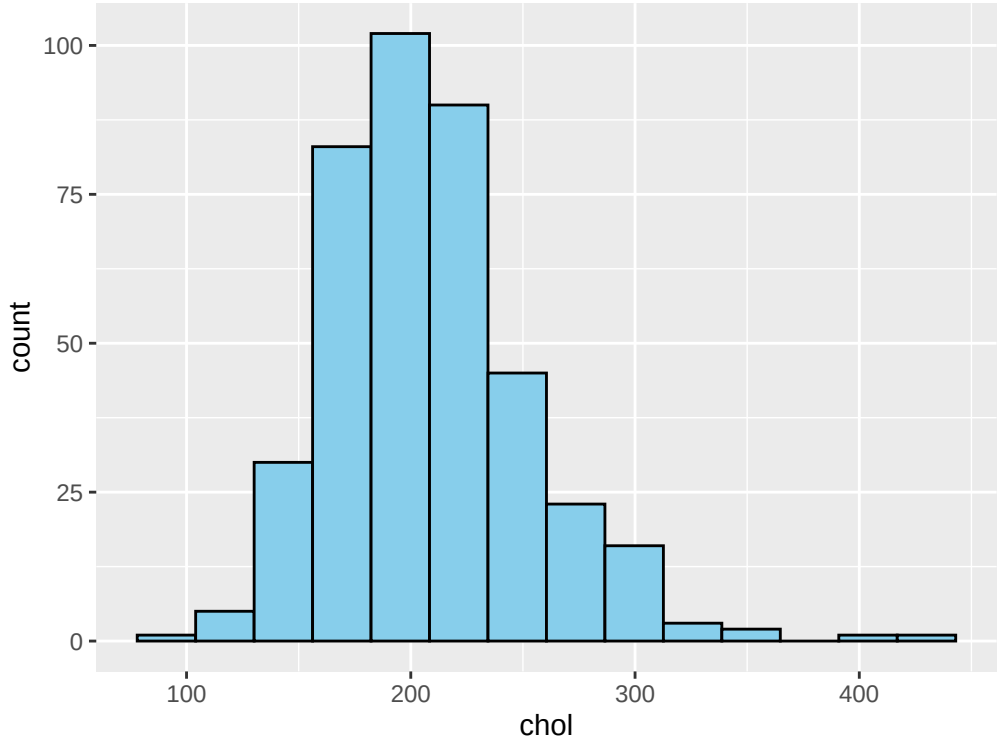


Para trabajar con los datos del colesterol será importante eliminar aquellas filas donde los valores son nulos, ya que si no los quitamos nos puede dar error.

```
datos_sinna_chol <- chlstrl[!is.na(chlstrl$chol),]
```

Haremos varios tipos de gráficas, donde encontraremos gráficos de barras, boxplot e histogramas.

```
ggplot(datos_sinna_chol) + geom_histogram(mapping = aes(x = chol),
            breaks = seq(min(datos_sinna_chol$chol), max(datos_sinna_chol$chol),
                         length.out=15 ), fill = "skyblue", color="black" )
```

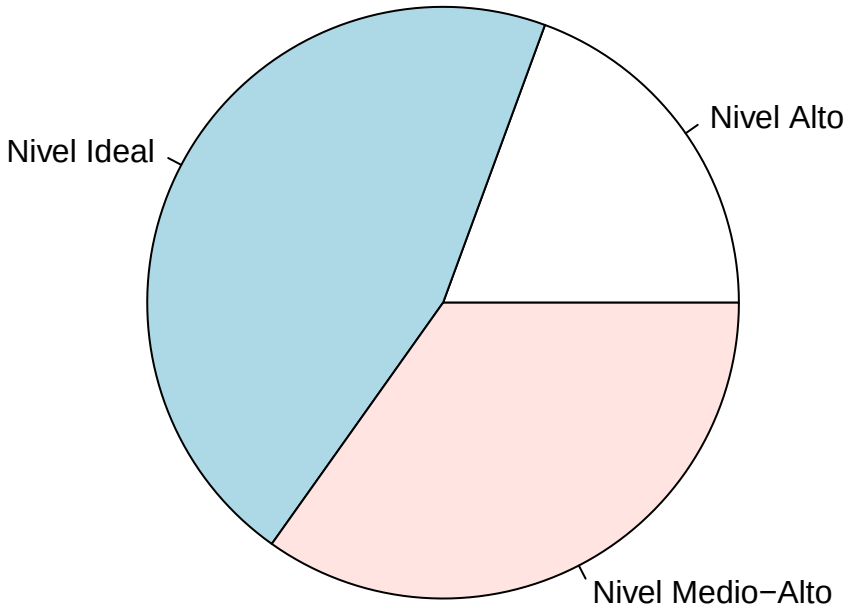


Cuando se habla de colesterol puede ser interesante ver que cantidad de personas tienen colesterol a niveles saludables, a niveles medio-altos y niveles altos. Pero como nuestra variable de colesterol viene dada de forma cuantitativa y a nosotros nos gustaría obtener tres tramos distintos vamos a crear una función que nos distinga el nivel de colesterol en función de los datos. Esta función también cambiará el tipo de variable, ya que pasaremos a una variable de tipo cualitativa que tendrá 3 posibles valores.

```
divi_chol = function(x){
  distr_col = c()
  for (i in 1:length(x)){
    if (x[i] < 200){
      distr_col = c(distr_col,"Nivel_ideal")
    } else {
      if (x[i]>= 200 & x[i]<240){
        distr_col = c(distr_col,"Nivel_medio_alto")
      }
      else {
        distr_col = c(distr_col,"Nivel_alto")
      }
    }
  }
  distr_col
}
```

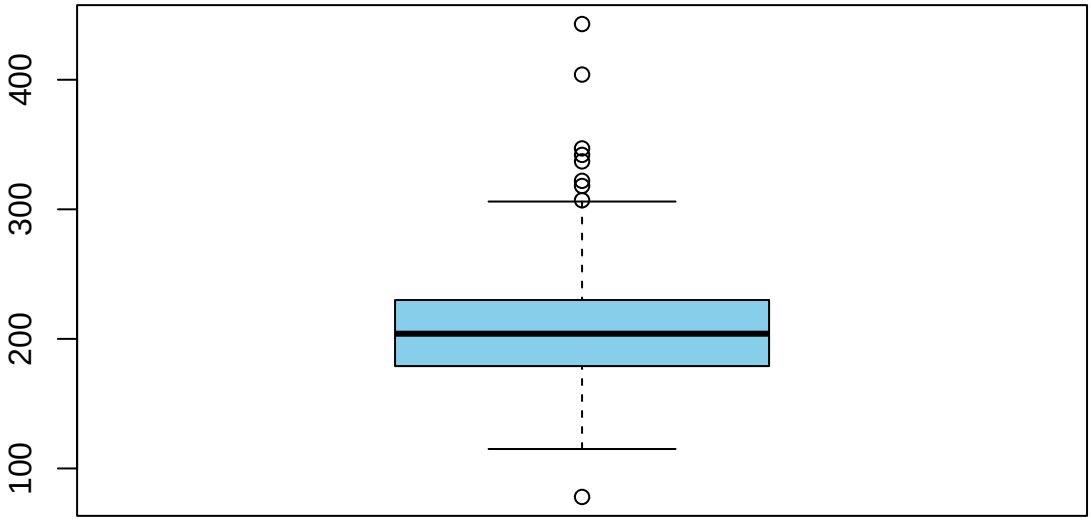
Una vez creada la función, vamos a aplicarla sobre la columna del colesterol. Tras ello, haremos un gráfico circular o de tarta donde podremos ver como ser distribuyen los diferentes niveles de colesterol.

```
distrib_por_niv <- data.frame(divi_chol(datos_sinna_chol$chol))
pie(table(distrib_por_niv), labels = c("Nivel Alto", "Nivel Ideal", "Nivel Medio-Alto"))
```



Por último hacemos el boxplot del colesterol:

```
boxplot(datos_sinna_chol$chol,col="skyblue", mapping = aes(y = chol))
```



Los valores de height y weight están en pulgadas y libras respectivamente, por ello vamos a pasarlos al unidades del sistema internacional como son el metro y el kilo. Además vamos a definir un nuevo data frame donde ambos valores esten en unidades del sistema internacional.

```
datos_si <- chlstrl %>%
  mutate(height = height * 0.0254, weight = weight * 0.454)
knitr::kable(head(datos_si,10))
```

chol	age	gender	height	weight	waist	hip
203	46	female	1.5748	54.934	29	38
165	29	female	1.6256	98.972	46	48
228	58	female	1.5494	116.224	49	57
78	67	male	1.7018	54.026	33	38
249	64	male	1.7272	83.082	44	41
248	34	male	1.8034	86.260	36	42
195	30	male	1.7526	86.714	46	49
227	37	male	1.4986	77.180	34	39
177	45	male	1.7526	75.364	34	40
263	55	female	1.6002	91.708	45	50

Ahora vamos a añadir una nueva columna que llamaremos BMI que vendrá dada por la siguiente ecuación:

$$BMI = \frac{weight}{height^2}$$

Pero antes tenemos que eliminar todas aquellas filas que tienen espacios en blanco ya sea en weight o en height.

```
datos_si <- datos_si %>%
  mutate(datos_si, BMI = weight/height^2)
knitr::kable(head(datos_si,10))
```

chol	age	gender	height	weight	waist	hip	BMI
203	46	female	1.5748	54.934	29	38	22.15085
165	29	female	1.6256	98.972	46	48	37.45286
228	58	female	1.5494	116.224	49	57	48.41375
78	67	male	1.7018	54.026	33	38	18.65459
249	64	male	1.7272	83.082	44	41	27.84977
248	34	male	1.8034	86.260	36	42	26.52316
195	30	male	1.7526	86.714	46	49	28.23083
227	37	male	1.4986	77.180	34	39	34.36634
177	45	male	1.7526	75.364	34	40	24.53569
263	55	female	1.6002	91.708	45	50	35.81448

La siguiente parte de este ejercicio consistirá en crear una nueva columna llamada ageGroup dividiendo la edad en tres niveles.

Para ello, primero tendremos que crear una función que nos diga en que rango de edad se encuentra:

```
divid_age = function(x){
  distr_col = c()
  for (i in 1:length(x)){
    if (x[i] <= 40){
      distr_col = c(distr_col,"(10,40]")
    } else {
      if (x[i]>40 & x[i]<=70){
        distr_col = c(distr_col,"(40,70]")
      }
      else {
        distr_col = c(distr_col,"(70,100]")
      }
    }
  }
  distr_col
}
```

```
datos_si <- datos_si %>%
  mutate(datos_si, ageGroup = divid_age(age))
knitr::kable(head(datos_si,10))
```

chol	age	gender	height	weight	waist	hip	BMI	ageGroup
203	46	female	1.5748	54.934	29	38	22.15085	(40,70]
165	29	female	1.6256	98.972	46	48	37.45286	(10,40]
228	58	female	1.5494	116.224	49	57	48.41375	(40,70]
78	67	male	1.7018	54.026	33	38	18.65459	(40,70]
249	64	male	1.7272	83.082	44	41	27.84977	(40,70]
248	34	male	1.8034	86.260	36	42	26.52316	(10,40]
195	30	male	1.7526	86.714	46	49	28.23083	(10,40]
227	37	male	1.4986	77.180	34	39	34.36634	(10,40]
177	45	male	1.7526	75.364	34	40	24.53569	(40,70]
263	55	female	1.6002	91.708	45	50	35.81448	(40,70]

Ahora usando dplyr vamos calcular cuántas observaciones hay en cada nivel de ageGroup.

```
datos_si %>%
  count(ageGroup)
```

```
##   ageGroup   n
## 1  (10,40] 160
## 2  (40,70] 207
## 3 (70,100]  36
```

Y por último usando aquellas observaciones que corresponden a mujeres, ¿cuál es la media del nivel de colesterol y de BMI en cada uno de esos grupos de edad?

Primero filtraremos para coger todos los datos de las mujeres y luego usaremos summarise para sacar la media de ambos:

```
datos_si %>%
  group_by(ageGroup) %>%
  filter(gender == "female") %>%
  summarise(media_chol = mean(chol, na.rm = TRUE), media_BMI = mean(BMI, na.rm = TRUE))
```

```
## # A tibble: 3 x 3
##   ageGroup media_chol media_BMI
##   <chr>         <dbl>     <dbl>
## 1 (10,40]        189.      30.5
## 2 (40,70]        221.      30.3
## 3 (70,100]       230.      29.4
```

Ejercicio 2: Funciones de R.

Enunciado: Crea una función de R llamada `cambiosSigno` que dado un vector `x` de números enteros no nulos calcule cuántos cambios de signo ha habido. Es decir, cuántas veces el signo de un elemento es distinto del signo del elemento previo.

Solución:

Comenzamos creando la función `cambiosSigno` y comprobando que funciona correctamente:

```
cambioSigno = function(vect_num){
  cambios = 0
  for (i in 1:(length(vect_num)-1)){
    if (vect_num[i]*vect_num[i+1] < 0){
      cambios = cambios + 1
    }
  }
  cambios
}
```

```
(vect = sample(c(-20:-1,1:20),20, replace = TRUE))
```

```
##  [1] -17  -4 -10  15 -15  -7 -14  -4  -9  15  -3   1 -11  -3   3   6 -20  20   1
## [20] -12
```

```
cambioSigno(vect)
```

```
## [1] 10
```

Ahora vamos a modificar la función anterior para que en vez de devolvernos cuantos cambios se han producido nos devuelva un vector formado por las posiciones donde se han producido dichos cambios de signo:

```
cambiosSignoPos = function(vect_num){
  posiciones = c()
  for (i in 1:(length(vect_num)-1)){
    if (vect_num[i]*vect_num[i+1] < 0){
      posiciones = c(posiciones,i+1)
    }
  }
  posiciones
}
```

```
(vect = sample(c(-20:-1,1:20),20, replace = TRUE))
```

```
##  [1] -20  14 -19 -11   1  -9  10  18  -3  -8   1 -13  12   9 -11 -12  20  -7   7
## [20]  -8
```

```
cambiosSignoPos(vect)
```

```
##  [1]  2  3  5  6  7  9 11 12 13 15 17 18 19 20
```

Ejercicio 3. Ejercicios libro R4DS.

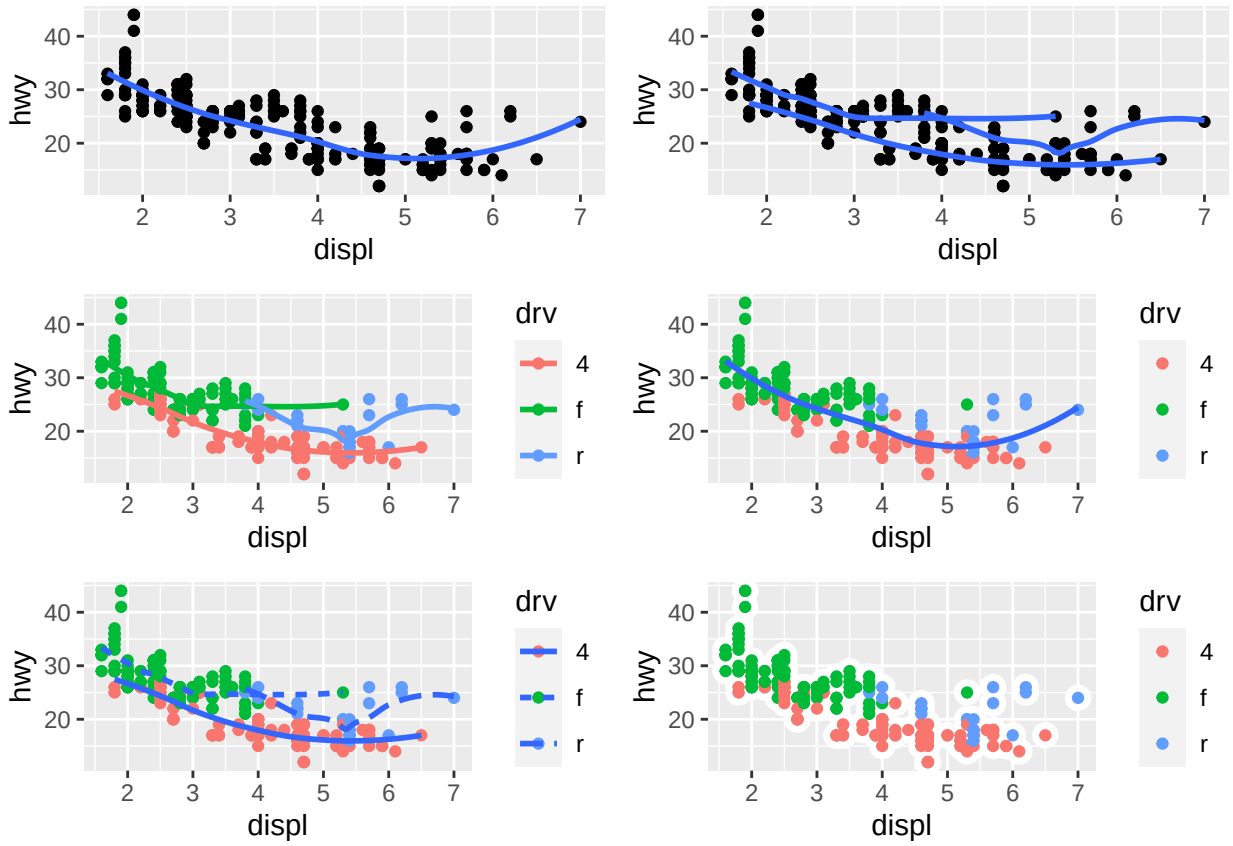
Ejercicio 3.1

Enunciado: Del libro de R4DS:

- Haz el ejercicio 6 de la Sección 3.6.1.

Solución:

```
p1 <- ggplot(mpg, mapping = aes(x = displ, y=hwy)) +
  geom_point(color = "black") +
  geom_smooth(se = FALSE)

p2 <- ggplot(mpg, mapping = aes(x = displ, y=hwy , group= drv)) +
  geom_point(color = "black") +
  geom_smooth(se = FALSE)

p3 <- ggplot(mpg, mapping = aes(x = displ, y=hwy, color = drv)) +
  geom_point() +
  geom_smooth(se = FALSE)

p4 <- ggplot(mpg, mapping = aes(x = displ, y=hwy)) +
  geom_point(mapping = aes(color = drv)) +
  geom_smooth(se = FALSE)

p5 <- ggplot(mpg, mapping = aes(x = displ, y=hwy)) +
  geom_point(mapping = aes(color = drv)) +
  geom_smooth(mapping = aes(linetype = drv),se = FALSE)

p6 <- ggplot(mpg, mapping = aes(x = displ, y=hwy)) +
  geom_point(size = 4, color = "white") +
  geom_point(aes(color = drv))

grid.arrange(p1,p2,p3,p4,p5,p6, nrow = 3)
```



Ejercicio 3.2

Enunciado y soluciones: Encuentra todos los vuelos que:

1. Tuvieron un retraso de dos horas o más.

```
flights %>%
  filter(arr_delay >= 120)
```

```
## # A tibble: 10,200 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      811            630       101     1047            830
##  2  2013     1     1      848           1835       853     1001           1950
##  3  2013     1     1      957            733       144     1056            853
##  4  2013     1     1     1114            900       134     1447           1222
##  5  2013     1     1     1505           1310       115     1638           1431
##  6  2013     1     1     1525           1340       105     1831           1626
##  7  2013     1     1     1549           1445        64     1912           1656
##  8  2013     1     1     1558           1359       119     1718           1515
##  9  2013     1     1     1732           1630        62     2028           1825
## 10  2013     1     1     1803           1620       103     2008           1750
## # ... with 10,190 more rows, and 11 more variables: arr_delay <dbl>,
```

```
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

2. Volaron a Houston (IAH or HOU).

```
flights %>%
  filter(dest == "IAH" | dest == "HOU")
```

```
## # A tibble: 9,313 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      517            515         2      830            819
##  2  2013     1     1      533            529         4      850            830
##  3  2013     1     1      623            627        -4      933            932
##  4  2013     1     1      728            732        -4     1041           1038
##  5  2013     1     1      739            739         0     1104           1038
##  6  2013     1     1      908            908         0     1228           1219
##  7  2013     1     1     1028           1026         2     1350           1339
##  8  2013     1     1     1044           1045        -1     1352           1351
##  9  2013     1     1     1114            900       134     1447           1222
## 10  2013     1     1     1205           1200         5     1503           1505
## # ... with 9,303 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

3. Fueron operados por United, American, or Delta.

Para conocer cuales son las abreviaturas tenemos que recurrir a una tabla distinta, la de airlines. Una vez vemos cuales son las abreviaturas hacemos el filtrado.

```
flights %>%
  filter(carrier == "DL" | carrier == "UA" | carrier == "AA")
```

```
## # A tibble: 139,504 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      517            515         2      830            819
##  2  2013     1     1      533            529         4      850            830
##  3  2013     1     1      542            540         2      923            850
##  4  2013     1     1      554            600        -6      812            837
##  5  2013     1     1      554            558        -4      740            728
##  6  2013     1     1      558            600        -2      753            745
```

```
##  7  2013     1     1      558            600        -2      924            917
##  8  2013     1     1      558            600        -2      923            937
##  9  2013     1     1      559            600        -1      941            910
## 10  2013     1     1      559            600        -1      854            902
## # ... with 139,494 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

4. Salió en verano (julio, agosto y septiembre).

```
flights %>%
  filter(month == 7 | month == 8 | month == 9)
```

```
## # A tibble: 86,326 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     7     1        1           2029       212      236           2359
##  2  2013     7     1        2           2359         3      344            344
##  3  2013     7     1       29           2245       104      151              1
##  4  2013     7     1       43           2130       193      322             14
##  5  2013     7     1       44           2150       174      300            100
##  6  2013     7     1       46           2051       235      304           2358
##  7  2013     7     1       48           2001       287      308           2305
##  8  2013     7     1       58           2155       183      335             43
##  9  2013     7     1      100           2146       194      327             30
## 10  2013     7     1      100           2245       135      337            135
## # ... with 86,316 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

5. Llegó con retraso de más de 2 horas, pero no salió tarde.

```
flights %>%
  filter(arr_delay > 120 & dep_delay <= 0)
```

```
## # A tibble: 29 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1    27     1419           1420        -1     1754           1550
##  2  2013    10     7     1350           1350         0     1736           1526
##  3  2013    10     7     1357           1359        -2     1858           1654
##  4  2013    10    16      657            700        -3     1258           1056
```

```
##  5  2013    11     1      658            700        -2     1329           1015
##  6  2013     3    18     1844           1847        -3       39           2219
##  7  2013     4    17     1635           1640        -5     2049           1845
##  8  2013     4    18      558            600        -2     1149            850
##  9  2013     4    18      655            700        -5     1213            950
## 10  2013     5    22     1827           1830        -3     2217           2010
## # ... with 19 more rows, and 11 more variables: arr_delay <dbl>, carrier <chr>,
## #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
## #   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

6. Se retrasaron al menos una hora, pero recuperaron más de 30 minutos en vuelo.

```
flights %>%
  filter(dep_delay >= 60 & arr_delay < 30)
```

```
## # A tibble: 206 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     3     1850           1745        65     2148           2120
##  2  2013     1     3     1950           1845        65     2228           2227
##  3  2013     1     3     2015           1915        60     2135           2111
##  4  2013     1     6     1019            900        79     1558           1530
##  5  2013     1     7     1543           1430        73     1758           1735
##  6  2013     1    11     1020            920        60     1311           1245
##  7  2013     1    12     1706           1600        66     1949           1927
##  8  2013     1    12     1953           1845        68     2154           2137
##  9  2013     1    19     1456           1355        61     1636           1615
## 10  2013     1    21     1531           1430        61     1843           1815
## # ... with 196 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

7. Partió entre la medianoche y las 6 a.m. (inclusive).

```
flights %>%
  filter(dep_time >= 000 & dep_time <= 600)
```

```
## # A tibble: 9,344 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      517            515         2      830            819
##  2  2013     1     1      533            529         4      850            830
```

```
##  3  2013     1     1      542            540         2      923            850
##  4  2013     1     1      544            545        -1     1004           1022
##  5  2013     1     1      554            600        -6      812            837
##  6  2013     1     1      554            558        -4      740            728
##  7  2013     1     1      555            600        -5      913            854
##  8  2013     1     1      557            600        -3      709            723
##  9  2013     1     1      557            600        -3      838            846
## 10  2013     1     1      558            600        -2      753            745
## # ... with 9,334 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```