Tarea 1

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

Pablo Soriano González

13/9/2021

Ejercicio 0. Ejercicios de la práctica00:

Usando la función sample crea un vector dado_honesto con 100 números del 1 al 6. Haz una tabla de frecuencias absolutas (de dos maneras, con table y dplyr) y una tabla de frecuencias relativas:

Comenzamos cargando las librerías:

```
library(tidyverse)
library(dplyr)
```

Ahora continuamos creando el dado honesto y las tablas de frecuencia:

```
# Creamos el vector dado_honesto y lo visualizamos
dado_honesto = sample(1:6, 100, replace = TRUE)
dado_honesto
```

```
##   [1] 5 5 2 4 3 3 3 5 2 6 5 3 6 2 3 4 3 5 5 5 4 6 5 4 5 4 2 6 5 5 5 6 6 5 4 1 4
##  [38] 6 4 1 2 1 6 6 6 6 5 5 3 3 3 1 5 2 3 1 6 4 5 2 2 2 1 4 3 3 5 6 3 3 6 5 2 3
##  [75] 4 1 5 6 3 3 3 5 5 1 6 3 1 6 6 4 4 3 6 2 4 2 5 4 6 5
```

```
# Hacemos la tabla de frecuencias absolutas con table
table(dado_honesto)
```

```
## dado_honesto
##  1  2  3  4  5  6
##  9 12 20 15 24 20
```

```
# Creamos una tabla con un índice
datos <- data.frame(A = 1:100, B = dado_honesto)

# Hacemos la tabla usando dplyr
datos %>%
  count(B)
```

```
##   B  n
## 1 1  9
## 2 2 12
## 3 3 20
## 4 4 15
## 5 5 24
## 6 6 20
```

```
# Hacemos ahora la tabla con frecuencias relativas
signif(prop.table(table(datos$B)), 2)
```

```
##
##    1    2    3    4    5    6
## 0.09 0.12 0.20 0.15 0.24 0.20
```

```
# Y las frecuencias relativas con dplyr
datos %>%
  count(B) %>%
    mutate(B, relFreq = prop.table(n), n=NULL)
```

```
##   B relFreq
## 1 1    0.09
## 2 2    0.12
## 3 3    0.20
## 4 4    0.15
## 5 5    0.24
## 6 6    0.20
```

A continuación crea un nuevo vector dado_cargado de manera que la probabilidad de que el número elegido valga 6 sea el doble que la probabilidad de elegir cualquiera de los cinco números restantes. Lee la ayuda de sample si lo necesitas. De nuevo, haz tablas de frecuencias absolutas y relativas de este segundo vector:

Creamos primero el nuevo dado con mayor probabilidad en el 6:

```
# Creamos un vector donde el 6 aparece dos veces
carga = c(1:6, 6)
carga
```

```
## [1] 1 2 3 4 5 6 6
```

```
# Creamos el dado cargado con un sample del vector anterior
dado_cargado = sample(carga, 100, replace = TRUE)
dado_cargado
```

```
##   [1] 2 1 1 2 3 6 3 6 3 3 6 1 6 6 2 5 3 6 4 3 6 6 3 3 2 6 2 6 6 6 1 6 5 1 2 4 4
##  [38] 6 6 6 6 1 5 2 5 2 4 4 6 4 4 6 2 2 2 6 3 6 5 1 1 3 2 3 6 3 5 6 2 4 6 1 4 1
##  [75] 6 6 4 5 6 3 6 4 6 5 4 1 4 3 5 4 1 2 6 4 2 6 6 1 6 2
```

Y ahora realizamos las tablas como en el ejemplo del dado honesto:

```
# Hacemos la tabla de frecuencias absolutas con table
table(dado_cargado)
```

```
## dado_cargado
##  1  2  3  4  5  6
## 13 16 14 15  9 33
```

```
# Creamos una tabla con un índice
datos2 <-
  data.frame(A = 1:100, B = dado_cargado)

# Hacemos la tabla usando dplyr
datos2 %>%
  count(B)
```

```
##   B  n
## 1 1 13
## 2 2 16
```

```
## 3 3 14
## 4 4 15
## 5 5  9
## 6 6 33
```

```
# Hacemos ahora la tabla con frecuencias relativas
signif(prop.table(table(datos2$B)), 2)
```

```
##
##    1    2    3    4    5    6
## 0.13 0.16 0.14 0.15 0.09 0.33
```

```
# Y las frecuencias relativas con dplyr
datos2 %>%
  count(B) %>%
    mutate(B, relFreq = prop.table(n), n=NULL)
```

```
##   B relFreq
## 1 1    0.13
## 2 2    0.16
## 3 3    0.14
## 4 4    0.15
## 5 5    0.09
## 6 6    0.33
```

Utiliza las funciones rep y seq para crear tres vectores v1, v2 y v3 con estos elementos respectivamente

v1 = 4 4 4 4 3 3 3 3 2 2 2 2 1 1 1 1

v2 = 1 2 2 3 3 3 4 4 4 4 5 5 5 5 5

v3 = 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4

A continuación generamos los vectores citados:

```
v1 = rep(seq(4,1,-1), each = 4)
v1
```

```
##  [1] 4 4 4 4 3 3 3 3 2 2 2 2 1 1 1 1
```

```
v2 = rep(seq(1,5,1), seq(1,5,1))
v2
```

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

```
v3 = rep(seq(1,4,1), 4)
v3
```

```
##  [1] 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4
```

Utilizando la tabla mpg de la librería tidyverse crea una tabla mpg2 que contenga las filas en las que la variable class toma el valor pickup y las columnas de la tabla original cuyos nombres empiezan por c. No se trata de que las selecciones a mano, por sus nombres. Busca información sobre funciones auxiliares para select en la Sección 5.4 de R4DS.

A continuación, generamos la tabla y la visualizamos:

```
mpg2 <- (mpg %>%
           select(starts_with("c")) %>%
           filter(class == "pickup"))
mpg2
```

```
## # A tibble: 33 x 3
##      cyl   cty class
##    <int> <int> <chr>
##  1     6    15 pickup
##  2     6    14 pickup
##  3     6    13 pickup
##  4     6    14 pickup
##  5     8    14 pickup
##  6     8    14 pickup
##  7     8     9 pickup
##  8     8    11 pickup
##  9     8    11 pickup
## 10     8    12 pickup
## # ... with 23 more rows
```

Descarga el fichero census.dta. Averigua de qué tipo de fichero se trata y usa la herramienta Import DataSet del panel Environment de RStudio para leer con R los datos de ese fichero. Asegúrate de copiar en esta práctica los dos primeros comandos que llevan a cabo la importación (excluye el comando View) y que descubrirás al usar esa herramienta. Después completa los siguientes apartados con esos datos y usando dplyr y ggplot:

Copiamos los comandos de la importación del dataset census.dta:

```
library(haven)
census <- read_dta("data/census.dta")
```

¿Cuáles son las poblaciones totales de las regiones censales?

Creamos una tabla sumando las poblaciones de cada región censal:

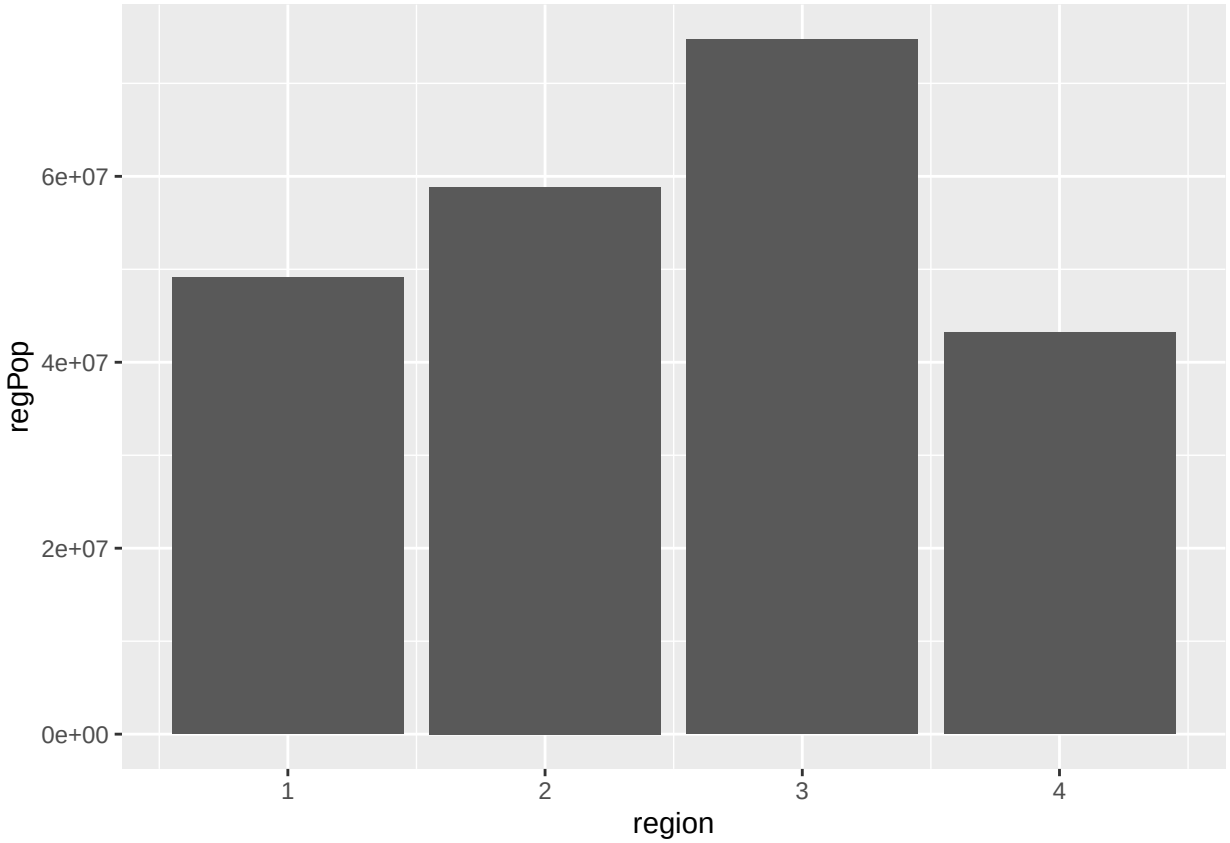
```
popReg <- (census %>%
         group_by(region) %>%
         summarise(regPop = sum(pop)))
popReg
```

```
## # A tibble: 4 x 2
##        region   regPop
##     <dbl+lbl>    <dbl>
## 1 1 [NE]      49135283
## 2 2 [N Cntrl] 58865670
## 3 3 [South]   74734029
## 4 4 [West]    43172490
```

Representa esas poblaciones totales en un diagrama de barras (una barra por región censal)

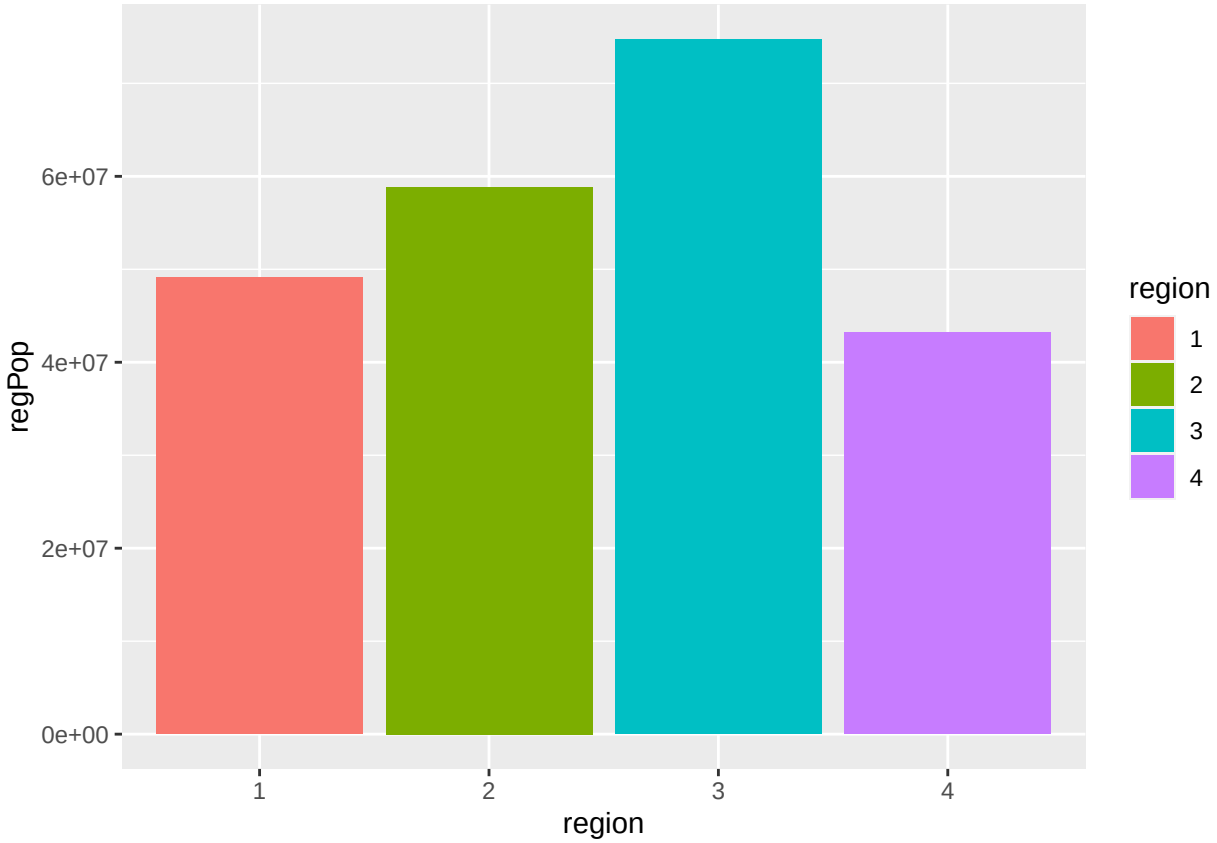
Lo haremos primero empleando el comando geom_col dentro de ggplot:

```
ggplot(data=popReg) +
  geom_col(aes(region, regPop))
```



También podemos hacerlo empleando el comando geom_bar dentro del ggplot, convirtiendo las regiones, que son números, a factores:

```
# Convertimos la columna numérica de región a una columna de factores
popReg$region <- as.factor(popReg$region)

# Y graficamos con geom_bar
ggplot(popReg) +
  geom_bar(aes(x=region, y=regPop, fill = region), stat = "identity")
```



Ordena los estados por población, de mayor a menor.

Lo hacemos empleando un arrange descendiente:

```
census %>%
  select(state, pop) %>%
  arrange(desc(pop))
```

```
## # A tibble: 50 x 2
##    state             pop
##    <chr>           <dbl>
##  1 California   23667902
##  2 New York     17558072
##  3 Texas        14229191
##  4 Pennsylvania 11863895
##  5 Illinois     11426518
##  6 Ohio         10797630
##  7 Florida       9746324
##  8 Michigan      9262078
##  9 New Jersey    7364823
## 10 N. Carolina   5881766
## # ... with 40 more rows
```

Crea una nueva variable que contenga la tasa de divorcios / matrimonios para cada estado.

Para ello creamos una nueva columna empleando el comando mutate:

```
census %>%
  mutate(divTasa = divorce / marriage) %>%
```

```
  select(state, marriage, divorce, divTasa) %>%
  arrange(desc(divTasa))
```

```
## # A tibble: 50 x 4
##    state       marriage divorce divTasa
##    <chr>          <dbl>   <dbl>   <dbl>
##  1 Oregon         23004   17762   0.772
##  2 Indiana        57853   40006   0.692
##  3 Florida       108344   71579   0.661
##  4 Arizona        30223   19908   0.659
##  5 Alaska          5361    3517   0.656
##  6 California    210864  133541   0.633
##  7 New Mexico     16641   10426   0.627
##  8 N. Carolina    46718   28050   0.600
##  9 Washington     47728   28642   0.600
## 10 Arkansas       26513   15882   0.599
## # ... with 40 more rows
```

Si nos preguntamos cuáles son los estados más envejecidos podemos responder de dos maneras. Mirando la edad mediana o mirando en qué estados la franja de mayor edad representa una proporción más alta de la población total. Haz una tabla en la que aparezcan los valores de estos dos criterios, ordenada según la edad mediana decreciente y muestra los 10 primeros estados de esa tabla.

La variable de edad mediana ya existe por lo que creamos una variable para la tasa de población más mayor por total de población:

```
census %>%
  mutate(oldTasa = pop65p / pop) %>%
  select(state, medage, oldTasa) %>%
  arrange(desc(medage)) %>%
  head(10)
```

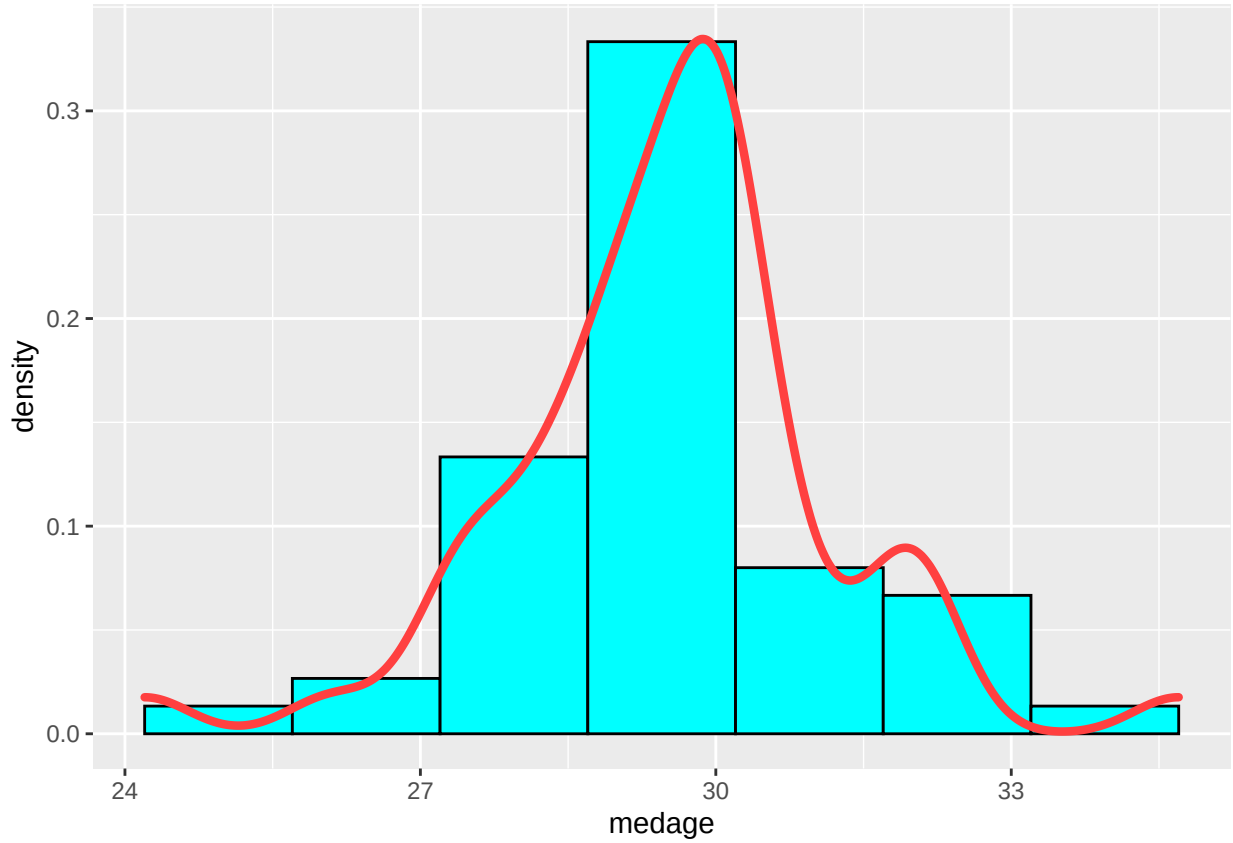
```
## # A tibble: 10 x 3
##    state         medage oldTasa
##    <chr>          <dbl>   <dbl>
##  1 Florida         34.7   0.173
##  2 New Jersey      32.2   0.117
##  3 Pennsylvania    32.1   0.129
##  4 Connecticut     32     0.117
##  5 New York        31.9   0.123
##  6 Rhode Island    31.8   0.134
##  7 Massachusetts   31.2   0.127
##  8 Missouri        30.9   0.132
##  9 Arkansas        30.6   0.137
## 10 Maine           30.4   0.125
```

Haz un histograma (con 10 intervalos) de los valores de la variable medage (edad mediana) y con la curva de densidad de la variable superpuesta.

Para ello primero definiremos un número de cortes que crearán los intervalos del histograma. Después graficaremos tanto el histograma como la curva de densidad superpuestas empleando los comandos geom_histogram y geom_density de ggplot:

```
# Definimos 8 cortes entre los valores mínimo y máximo que crearán 7 intervalos del histograma
cortes = seq(min(census$medage), max(census$medage), length.out = 8)
```

```r
# Y realizamos el histograma y la curva de densidad
ggplot(census, aes(x = medage)) +
  geom_histogram(aes(y=stat(density)),breaks = cortes, fill = "cyan", color="black") +
  geom_density(color="brown1", size=1.5)
```



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

Vamos a utilizar el conjunto de datos contenido en el fichero cholesterol.csv. Carga el conjunto de datos en un data.frame de R llamado chlstrl.

Cargamos el conjunto de datos:

```r
chlstrl <- read.csv(file = "data/cholesterol.csv", header = TRUE, sep = ",")
```

Empezaremos por información básica sobre el conjunto de datos. Cuántas observaciones contiene, cuáles son las variables y de qué tipos

Obtenemos esta información empleando el comando str:

```r
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 ...
```

Asegúrate de comprobar si hay datos ausentes y localízalos en la tabla.

Lo comprobaremos (y los localizaremos) usando los comandos is.na(), que busca las posiciones de datos NA, y which(), que nos dice cuales son las posiciones, encadenados. Las posiciones no se obtienen con 2 índices para cada una (fila y columna) si no que todas las posiciones de la matriz se enumeran de la primera a la última seguidas recorriendo las columnas en orden.

```
nulos <- which(is.na(chlstrl))
nulos
```

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

El análisis exploratorio (numérico y gráfico) debe cubrir todos los tipos de variable de la tabla. Es decir, que al menos debes estudiar una variable por cada tipo de variable presente en la tabla. El análisis debe contener, al menos:

- Para las variables cuantitativas (continuas o discretas). Resumen numérico básico. Gráficas (las adecuadas, a ser posible más de un tipo de gráfico).

Teniendo en cuenta el número de medidas y la extensión de los datos de los diferentes campos cuantitativos consideramos todas las variables cuantitativas como continuas. En este apartado analizamos la variable chol debido a que es la que presenta un comportamiento "más continuo" pues tiene un mayor rango de valores.

A continuación realizamos el resumen numérico básico de esta variable calculando distintos parámetros, para ello debemos exigir que se ignoren los datos ausentes (NA):

```
# Calculamos la media
mean(chlstrl$chol, na.rm = TRUE)
```

```
## [1] 207.8458
```

```
# Calculamos la mediana
median(chlstrl$chol, na.rm = TRUE)
```

```
## [1] 204
```

```
# Calculamos la desviación estándar
sd(chlstrl$chol, na.rm = TRUE)
```

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

```
# Calculamos el máximo
max(chlstrl$chol, na.rm = TRUE)
```

```
## [1] 443
```

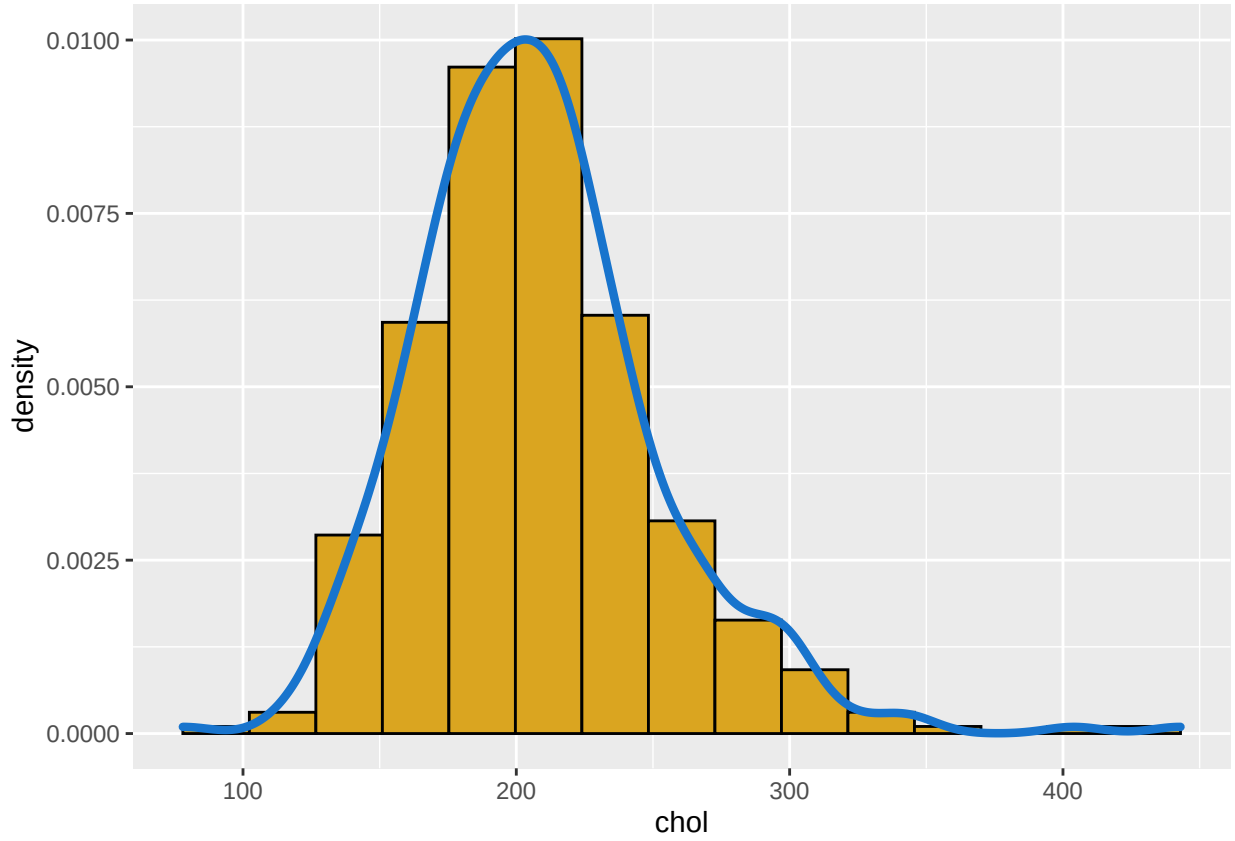
```
# Calculamos el mínimo
min(chlstrl$chol, na.rm = TRUE)
```

```
## [1] 78
```

Para las gráficas comenzamos con un histograma con la curva de densidad superpuesta. Lo realizamos de la misma forma que en el ejercicio de la práctica 0:

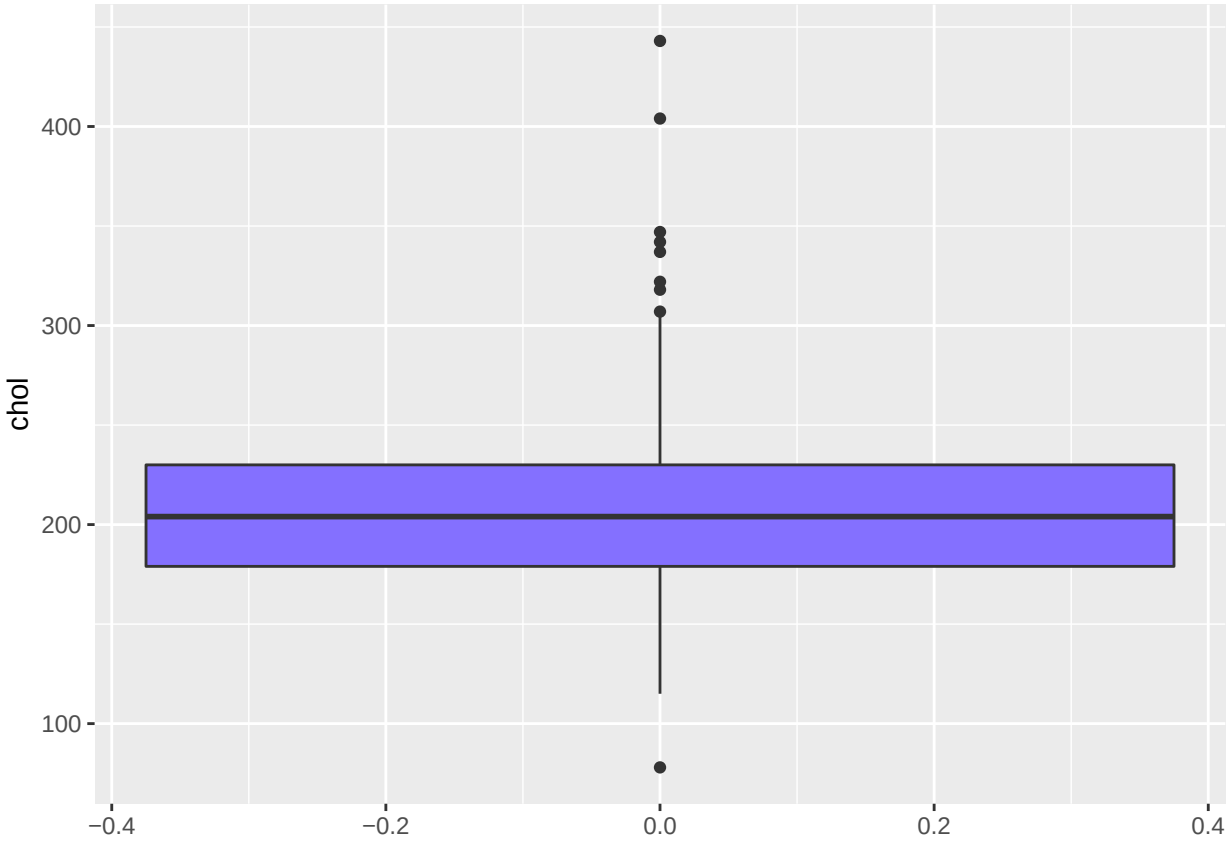
```
# Definimos los cortes
cortes = seq(min(chlstrl$chol, na.rm = TRUE), max(chlstrl$chol, na.rm = TRUE), length.out = 16)

# Y realizamos el histograma y la curva de densidad
ggplot(chlstrl, aes(x = chol)) +
```

```
  geom_histogram(aes(y=stat(density)),breaks = cortes, fill = "goldenrod", color="black")+
  geom_density(color="dodgerblue3", size=1.5)
```



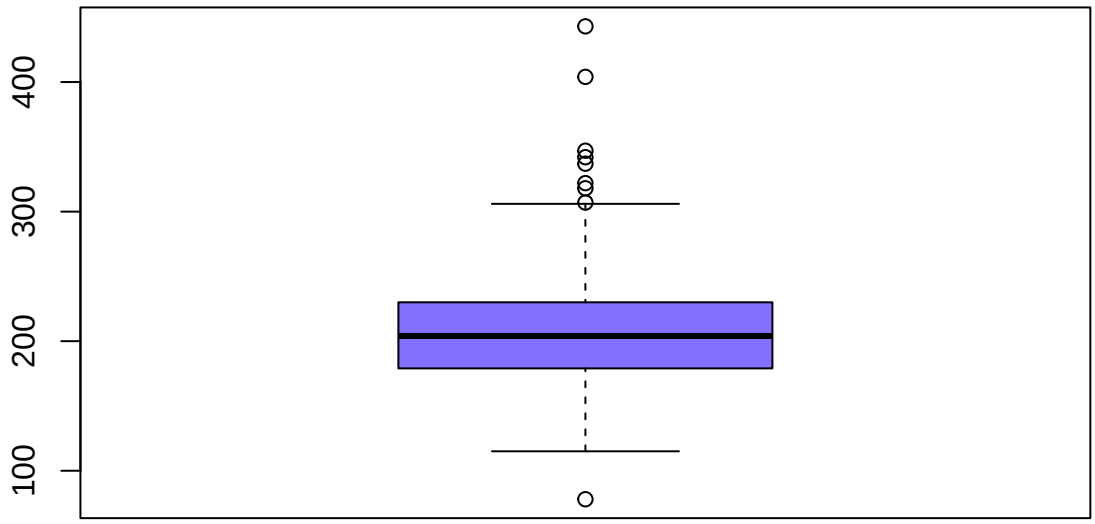
También realizaremos el boxplot. Usando el de ggplot obtenemos:

```
ggplot(chlstrl) +
  geom_boxplot(mapping = aes(y = chol), fill="lightslateblue")
```



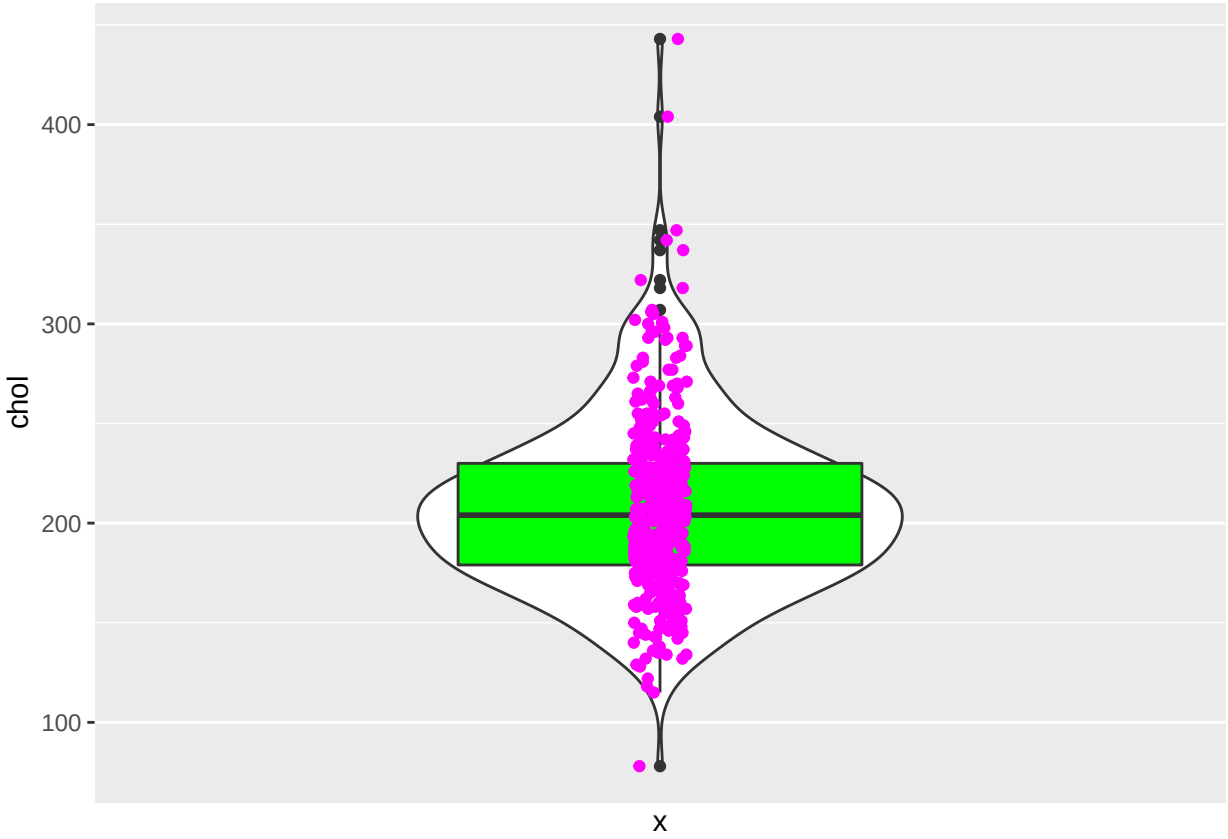
Usando el boxplot de R clásico tenemos:

```
bxp_cty = boxplot(chlstrl$chol, col="lightslateblue")
```



Por otro lado, también realizaremos un gráfico de violín superpuesto con el boxplot y con los datos de la variable (estos se encuentran "sacudidos" del eje vertical para poder observarlos mejor):

```
ggplot(chlstrl) +
  geom_violin(mapping = aes(x=0, y = chol)) +
  scale_x_discrete(breaks = c()) +
  geom_boxplot(mapping = aes(y = chol), fill="green1") +
  geom_jitter(aes(x=0, y = chol),
  position = position_jitter(w=0.05, h= 0), col="magenta")
```



- Variables categóricas (factores). Tablas de frecuencia (absolutas y relativas). Gráficas (diagrama de barras).

La unica variable categórica que encontramos en la tabla es el género.

El primer paso que realizamos con esta variable es convertir sus datos (cadenas de string) en factores:

```
chlstrl$gender = factor(chlstrl$gender)
```

Después obtenemos las tablas de frecuencias absolutas y relativas:

```
# Tabla de frecuencias absolutas
table(chlstrl$gender)
```

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

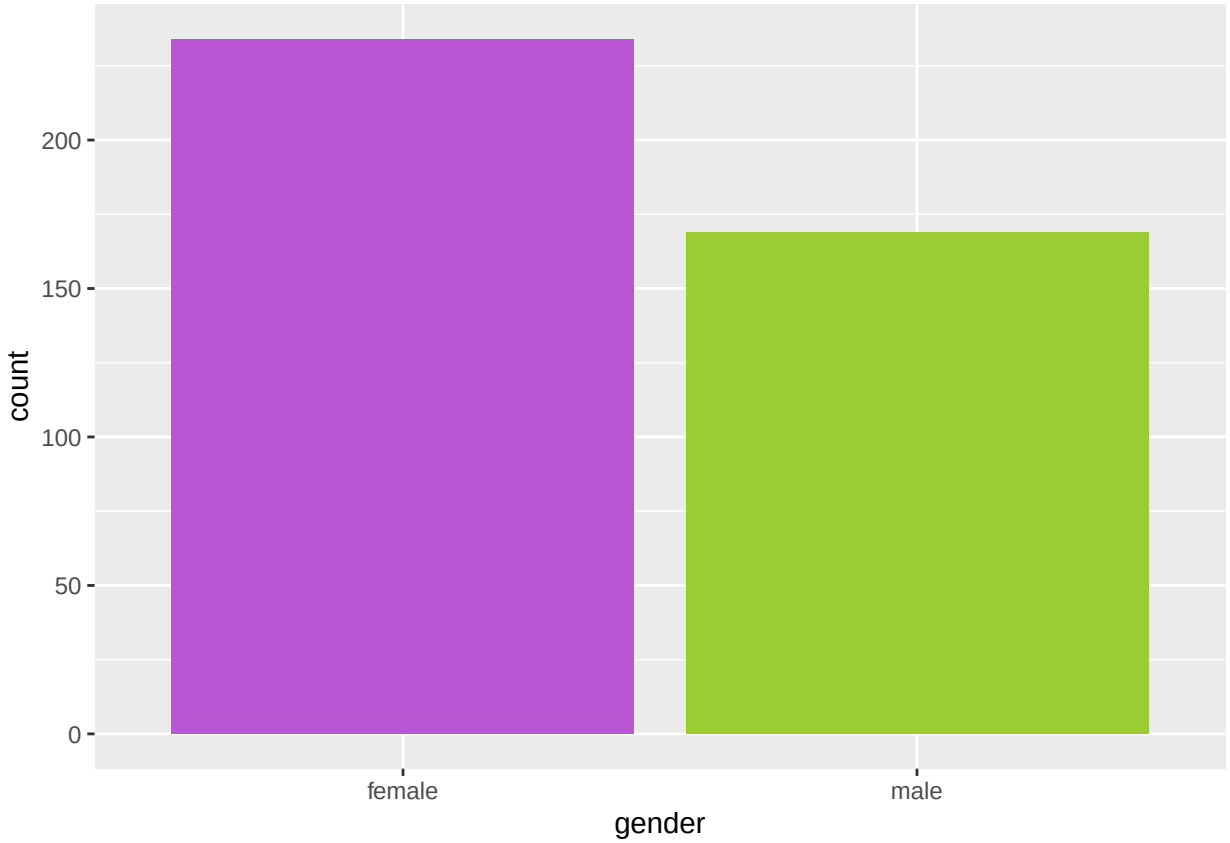
```
# Tabla de frecuencias relativas
prop.table(table(chlstrl$gender))
```

```
##
##    female      male
## 0.5806452 0.4193548
```

Ahora graficamos el diagrama de barras, para lo que debemos cargar al librería viridisLite:

```
# Cargamos la librería
library(viridisLite)
```

```r
# Realizamos el gráfico de barras
ggplot(chlstrl) +
  geom_bar(mapping = aes(x = gender), fill= c('mediumorchid','olivedrab3'))
```



Los valores de height y weight están en pulgadas (inches) y libras (pounds) respectivamente. Una libra son = 0.454kg y una pulgada son = 0.0254m, aproximadamente. Usa dplyr para convertir esas columnas a metros y kilogramos respectivamente. Las nuevas columnas deben llamarse igual que las originales.

Realizamos la conversión de las columnas de la tabla empleando mutate:

```r
chlstrl <- chlstrl %>%
             mutate("height" = height*0.0254, "weight" = weight*0.454 )
head(chlstrl, 10)
```

```
##    chol age gender height  weight waist hip
## 1   203  46 female 1.5748  54.934    29  38
## 2   165  29 female 1.6256  98.972    46  48
## 3   228  58 female 1.5494 116.224    49  57
## 4    78  67   male 1.7018  54.026    33  38
## 5   249  64   male 1.7272  83.082    44  41
## 6   248  34   male 1.8034  86.260    36  42
## 7   195  30   male 1.7526  86.714    46  49
## 8   227  37   male 1.4986  77.180    34  39
## 9   177  45   male 1.7526  75.364    34  40
## 10  263  55 female 1.6002  91.708    45  50
```

Ahora usa esos valores de height y weight para añadir una nueva columna llamada BMI,

definida mediante:

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

Generamos esta nueva columna volviendo a emplear mutate:

```
chlstrl <- chlstrl %>%
             mutate("BMI" = weight/(height)^2)
head(chlstrl, 10)
```

```
##    chol age gender height  weight waist hip      BMI
## 1   203  46 female 1.5748  54.934    29  38 22.15085
## 2   165  29 female 1.6256  98.972    46  48 37.45286
## 3   228  58 female 1.5494 116.224    49  57 48.41375
## 4    78  67   male 1.7018  54.026    33  38 18.65459
## 5   249  64   male 1.7272  83.082    44  41 27.84977
## 6   248  34   male 1.8034  86.260    36  42 26.52316
## 7   195  30   male 1.7526  86.714    46  49 28.23083
## 8   227  37   male 1.4986  77.180    34  39 34.36634
## 9   177  45   male 1.7526  75.364    34  40 24.53569
## 10  263  55 female 1.6002  91.708    45  50 35.81448
```

Crea una nueva columna llamada ageGroup dividiendo la edad en los siguientes tres niveles: (10,40], (40,70], (70,100]

Volvemos a usar mutate y el comando cut. Para definir los cortes que crean los intervalos usaremos seq(10,100,30):

```
chlstrl <- (chlstrl %>%
             mutate(ageGroup = cut(chlstrl$age, breaks = seq(10,100,30))))
head(chlstrl, 10)
```

```
##    chol age gender height  weight waist hip      BMI ageGroup
## 1   203  46 female 1.5748  54.934    29  38 22.15085  (40,70]
## 2   165  29 female 1.6256  98.972    46  48 37.45286  (10,40]
## 3   228  58 female 1.5494 116.224    49  57 48.41375  (40,70]
## 4    78  67   male 1.7018  54.026    33  38 18.65459  (40,70]
## 5   249  64   male 1.7272  83.082    44  41 27.84977  (40,70]
## 6   248  34   male 1.8034  86.260    36  42 26.52316  (10,40]
## 7   195  30   male 1.7526  86.714    46  49 28.23083  (10,40]
## 8   227  37   male 1.4986  77.180    34  39 34.36634  (10,40]
## 9   177  45   male 1.7526  75.364    34  40 24.53569  (40,70]
## 10  263  55 female 1.6002  91.708    45  50 35.81448  (40,70]
```

Usando dplyr calcula cuántas observaciones hay en cada nivel de ageGroup (indicación: usa group_by). Ahora, 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?

El número total de observaciones por cada grupo de edad es el siguiente:

```
chlstrl %>%
  group_by(ageGroup) %>%
  count()
```

```
## # A tibble: 3 x 2
## # Groups:   ageGroup [3]
##   ageGroup     n
```

```
##   <fct>    <int>
## 1 (10,40]    160
## 2 (40,70]    207
## 3 (70,100]    36
```

También podemos ver cuántas de estas observaciones corresponden a mujeres por grupo de edad:

```
chlstrl %>%
  group_by(ageGroup) %>%
  filter(gender=="female") %>%
  count()
```

```
## # A tibble: 3 x 2
## # Groups:   ageGroup [3]
##   ageGroup     n
##   <fct>    <int>
## 1 (10,40]     97
## 2 (40,70]    117
## 3 (70,100]    20
```

Para calcular las medias de colesterol y de BMI en las mujeres en cada uno de los distintos grupos de edad repetimos el proceso anterior añadiendo el comando summarise y haciendo las medias excluyendo los valores NA:

```
chlstrl %>%
  group_by(ageGroup) %>%
  filter(gender == "female") %>%
  summarise(media_col = mean(chol,na.rm=TRUE),media_bmi = mean(BMI,na.rm=TRUE))
```

```
## # A tibble: 3 x 3
##   ageGroup media_col media_bmi
##   <fct>        <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.

Crea una función de R llamada cambiosSigno que dado un vector x de números enteros no nulos, como [-12, -19, 9, -13, -14, -17, 8, -19, -14], calcule cuántos cambios de signo ha habido. Es decir, cuántas veces el signo de un elemento es distinto del signo del elemento previo. Por ejemplo, en el vector anterior hay 4 cambios de signo (en las posiciones 3, 4, 7 y 8).

Para crear esta función podríamos emplear bucles como en otros lenguajes. Sin embargo, mediante R podemos simplificar la tarea usando vectores y las operaciones entre ellos:

```
cambiosSigno = function(x){
  # Calculamos primero la longitud del vector x entrante
  long <- length(x)
  # En base a eso definimos dos nuevos vectores
  # El primero contiene todos los valores de x excepto el último
  inicio <- x[1:(long-1)]
  # El segundo contiene todos los valores de x excepto el primero
  final <- x[2:long]
  # Si multiplicamos ambos vectores tendremos un vector en el que cada posición representará un
  # posible cambio de signo. Valores positivos indican no cambio y valores negativos un cambio
  cambios <- inicio*final
```

```
  # El número total de cambios será la suma de los valores de este vector que sean menores que 0
  res <- sum(cambios < 0)
  return(res)
}
```

También se valorará que incluyas en el código como usar sample para generar vectores aleatorios de 20 enteros no nulos (el vector debe poder tomar valores positivos y negativos).

Los vectores que le pasaremos a la función deben ser aleatorios y de 20 enteros no nulos. Para ello empleamos el sample. Para evitar que el vector pueda contener aleatoriamente valores nulos no incluiremos al 0 en el espacio vectorial desde el cual el sample seleccionará los valores.

```
x = sample(c(-30:-1,1:30), 20, replace = TRUE)

# Observamos el vector
x
```

```
##  [1] -16  19 -28  27  -6  17 -27  10  19  -9  17  26  16  -1  16  15  -3  11  17
## [20]  27
```

A continuación probamos el funcionamiento de nuestra función y de la generación de un vector aleatorio:

```
#Llamamos a la función para que nos muestre el número de cambios
cambiosSigno(x)
```

```
## [1] 13
```

Modifica la función para que devuelva como resultado las posiciones donde hay cambios de signo. Llama cambiosSignoPos(x) a esa otra función.

Para esto simplemente cambiamos la suma de los cambios por un which que busque las posiciones en las que ocurren.

```
cambiosSignoPos = function(x){
  # Dejamos todo igual en la función
  long <- length(x)
  inicio <- x[1:(long-1)]
  final <- x[2:long]
  cambios <- inicio*final
  # Para obtener las posiciones donde ocurren los cambios las buscamos con which
  posiciones <- (which(cambios < 0) + 1)
  # Hemos añadido un +1 pues si no el which nos devolvería la posición anterior al cambio
  return(posiciones)
}
```

Si ponemos a prueba nuestras funciones comprobamos su funcionamiento:

```
# Volvemos a ver el vector x
x
```

```
##  [1] -16  19 -28  27  -6  17 -27  10  19  -9  17  26  16  -1  16  15  -3  11  17
## [20]  27
```

```
# Contamos los cambios totales
cambiosSigno(x)
```

```
## [1] 13
```

```
# Y localizamos las posiciones
cambiosSignoPos(x)
```

```
##  [1]  2  3  4  5  6  7  8 10 11 14 15 17 18
```

Ejercicio 3. R4DS.

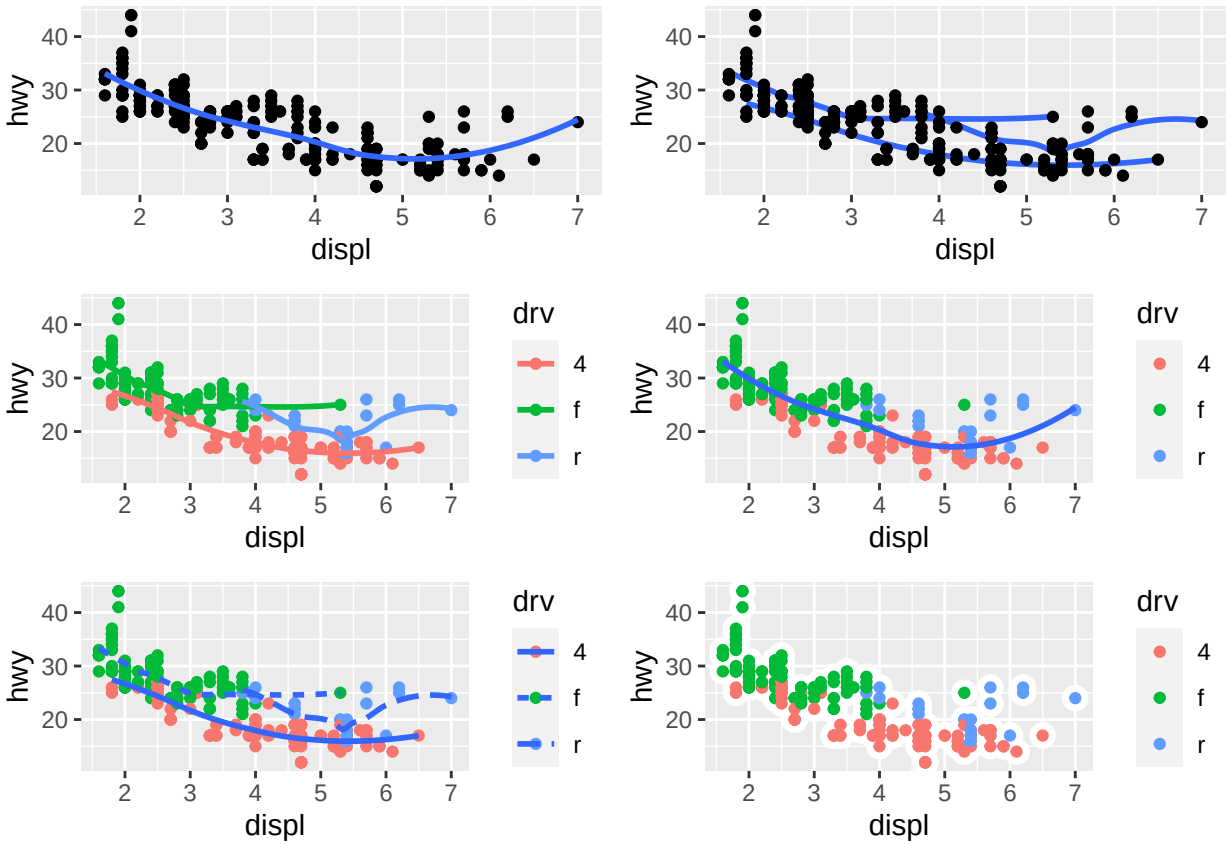
Haz el ejercicio 6 de la Sección 3.6.1 de R4DS

En este ejercicio se nos pide recrear un conjunto de gráficas mostradas en el libro que emplean los datos de la tabla mpg. A continuación generamos cada una de las tablas y posteriormente las graficaremos todas juntas:

```
# Primera gráfica
g1 <- ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  geom_smooth(se = FALSE)

# Segunda gráfica
g2 <- ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_smooth(mapping = aes(group = drv), se = FALSE) +
  geom_point()

# Tercera gráfica
g3 <- ggplot(mpg, aes(x = displ, y = hwy, colour = drv)) +
  geom_point() +
  geom_smooth(se = FALSE)

# Cuarta gráfica
g4 <- ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(aes(colour = drv)) +
  geom_smooth(se = FALSE)

# Quinta gráfica
g5 <- ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(aes(colour = drv)) +
  geom_smooth(aes(linetype = drv), se = FALSE)

# Sexta gráfica
g6 <- ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(size = 4, color = "white") +
  geom_point(aes(colour = drv))
```

Para graficar todas las figuras juntas primero debemos cargar la librería gridExtra y después usar grid.arrange:

```
# Cargamos la librería
library(gridExtra)

# Graficamos todas las figuras
grid.arrange(g1,g2,g3,g4,g5,g6, nrow = 3)
```



Haz el ejercicio 1 de la Sección 5.2.4 de R4DS.

En este ejercicio se nos pide que a partir de la base de datos de vuelos que partieron de Nueva York en 2013 (nycflights13) filtremos y seleccionemos los que cumplen ciertas condiciones.

```
# Comenzamos cargando la librería nycflights13
library(nycflights13)
```

Encontrar todos los vuelos que tuvieron un retraso de dos o más horas:

```
# Como los datos del tiempo de retraso están en minutos deberemos indicar 120 para las 2h
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>
```

Encontrar todos los vuelos que 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>
```

Encontrar todos los vuelos que fueron operados por United, American, o Delta:

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

```
## # 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>
```

Encontrar todos los vuelos que partieron en verano (julio, agosto y septiembre):

```
flights %>%
  filter(month %in% 7:9)
```

```
## # 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>
```

Encontrar todos los vuelos que llegaron más de dos horas tarde pero no salieron 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>
```

Encontrar todos los vuelos que se retrasaron al menos una hora pero recuperaron 30 minutos durante el vuelo:

```
flights %>%
  filter(dep_delay >= 60, dep_delay - arr_delay > 30)
```

```
## # A tibble: 1,844 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     2205           1720       285       46           2040
##  2  2013     1     1     2326           2130       116      131             18
##  3  2013     1     3     1503           1221       162     1803           1555
##  4  2013     1     3     1839           1700        99     2056           1950
##  5  2013     1     3     1850           1745        65     2148           2120
##  6  2013     1     3     1941           1759       102     2246           2139
##  7  2013     1     3     1950           1845        65     2228           2227
##  8  2013     1     3     2015           1915        60     2135           2111
##  9  2013     1     3     2257           2000       177       45           2224
## 10  2013     1     4     1917           1700       137     2135           1950
## # ... with 1,834 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>
```

Encontrar todos los vuelos que salieron entre medianoche y las 6am, ambas inclusive:

```r
# Debido a que la medianoche se expresa como 2400 no podemos exigir unicamente <= 600
flights %>%
  filter(dep_time <= 600 | dep_time == 2400)
```

```
## # A tibble: 9,373 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 9,363 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>
```