Tarea_1

Nicolás Núñez de Cela Román

12/9/2021

Sección previa

Cargamos las librerías que son necesarias para la realización de la práctica:

```
library(tidyverse)
library(viridisLite)
library(gridExtra)
library(nycflights13)
library(haven)
```

Práctica 0

Ejercicio 0

Comenzamos incluyendo la realización de los ejercicios de la *Práctica00*.

Ejercicio 0.1

Creamos un vector dado_honesto con 100 elementos del 1 al 6. Con este, realizamos una tabla de frecuencias con el comando table y con la librería dplyr (metemos el vector en un data frame y utilizamos las *pipes* y la función count()).

```
dado_honesto = sample(1:6, 100, replace = TRUE)

table(dado_honesto)
```

```
## dado_honesto
##  1  2  3  4  5  6
## 17 19 19 16 11 18
```

```
dado_honesto_2 <- data.frame(dado_honesto)
dado_honesto_2 %>%
  count(dado_honesto)
```

```
##   dado_honesto  n
## 1            1 17
## 2            2 19
```

```
## 3            3 19
## 4            4 16
## 5            5 11
## 6            6 18
```

A continuación, realizamos una tabla de frecuencias relativas con la función prop.table(), otra vez de dos formas diferentes.

```
prop.table(table(dado_honesto))
```

```
## dado_honesto
##    1    2    3    4    5    6
## 0.17 0.19 0.19 0.16 0.11 0.18
```

```
dado_honesto_2 %>%
  count(dado_honesto) %>%
  mutate(dado_honesto, relFreq = prop.table(n), n = NULL)
```

```
##   dado_honesto relFreq
## 1            1    0.17
## 2            2    0.19
## 3            3    0.19
## 4            4    0.16
## 5            5    0.11
## 6            6    0.18
```

Ejercicio 0.2

Creamos ahora un nuevo vector, dado_cargado, en el que la probabilidad de obtener un 6 es el doble que la de obtener el resto de números. Volvemos a crear, con este vector, las tablas de frecuencia absolutas y relativas, utilizando la función table y dplyr.

```
truco <- c(1/7,1/7,1/7,1/7,1/7,2/7)

dado_cargado = sample(1:6, 100, replace = TRUE, prob = truco)

table(dado_cargado)
```

```
## dado_cargado
##  1  2  3  4  5  6
## 13 15 15 13 12 32
```

```
dado_cargado_2 <- data.frame(dado_cargado)
dado_cargado_2 %>%
  count(dado_cargado)
```

```
##   dado_cargado  n
## 1            1 13
## 2            2 15
## 3            3 15
## 4            4 13
## 5            5 12
## 6            6 32
```

```r
prop.table(table(dado_cargado))
```

```
## dado_cargado
##    1    2    3    4    5    6
## 0.13 0.15 0.15 0.13 0.12 0.32
```

```r
dado_cargado_2 %>%
  count(dado_cargado) %>%
  mutate(dado_cargado, relFreq = prop.table(n), n = NULL)
```

```
##   dado_cargado relFreq
## 1            1    0.13
## 2            2    0.15
## 3            3    0.15
## 4            4    0.13
## 5            5    0.12
## 6            6    0.32
```

Vemos que, efectivamente, el número de veces que sale el número 6 es aproximadamente el doble de las veces que salen los demás números.

Ejercicio 0.3

Creamos los vectores v1, v2 y v3 con las funciones rep() y seq().

```r
(v1 <- rev(rep(seq(from = 1, to = 4, by = 1), each = 4)))
```

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

```r
(v2 <- rep(seq(from = 1, to = 5, by = 1), times = seq(from = 1, to = 5, by = 1)))
```

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

```r
(v3 <- rep(seq(from = 1, to = 4, by = 1), times = 4))
```

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

Ejercicio 0.4

Creamos la tabla mpg2 a partir de la tabla mpg. Para ello, seleccionamos las columnas cuyo nombre empieza por “c” con la función select(starts_with()) y las filas que contengan el valor “pickup” en la columna class con la función filter().

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

```
## # 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
```

Ejercicio 0.5

Descargamos el fichero census.dta, que se trata de un archivo de tipo Stata.

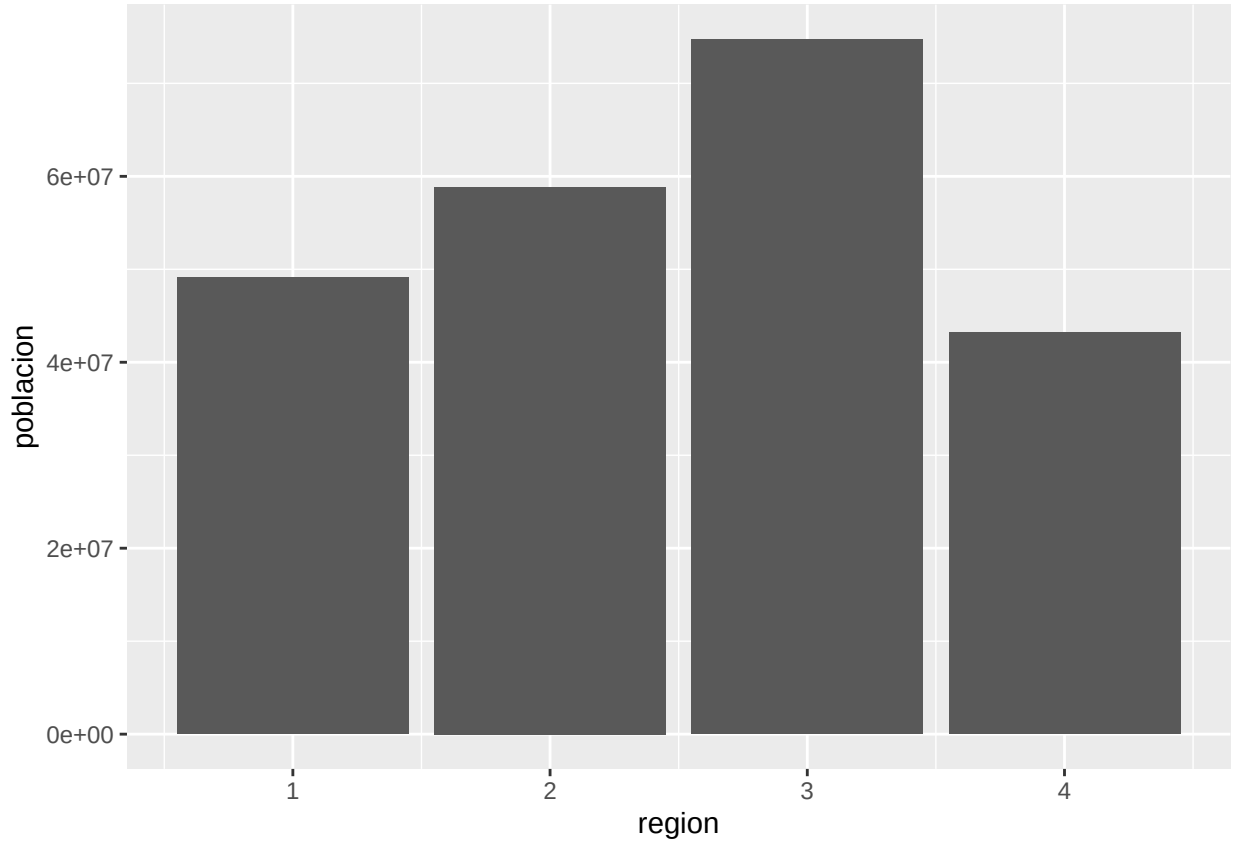
```
census <- read_dta("http://www.stata-press.com/data/r8/census.dta")
```

Calculamos las poblaciones totales de las regiones censales y las representamos en un diagrama de barras:

```
(prueba<-census %>%
  group_by(region) %>%
  summarise(poblacion = sum(pop)))
```

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

```
ggplot(data = prueba) +
  geom_col(mapping = aes(x = region, y = poblacion))
```



Ordenamos los estados por población, de mayor a menor:

```
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
```

Creamos una nueva variable, div_marr, que contiene la tasa de divorcios y matrimonios por estado:

```
(div_marr <- census %>%
  mutate(tasa_div_marr = divorce/marriage) %>%
  select(state,tasa_div_marr))
```

```
## # A tibble: 50 x 2
##    state       tasa_div_marr
##    <chr>               <dbl>
##  1 Alabama             0.546
##  2 Alaska              0.656
##  3 Arizona             0.659
##  4 Arkansas            0.599
##  5 California          0.633
##  6 Colorado            0.532
##  7 Connecticut         0.518
##  8 Delaware            0.521
##  9 Florida             0.661
## 10 Georgia             0.492
## # ... with 40 more rows
```

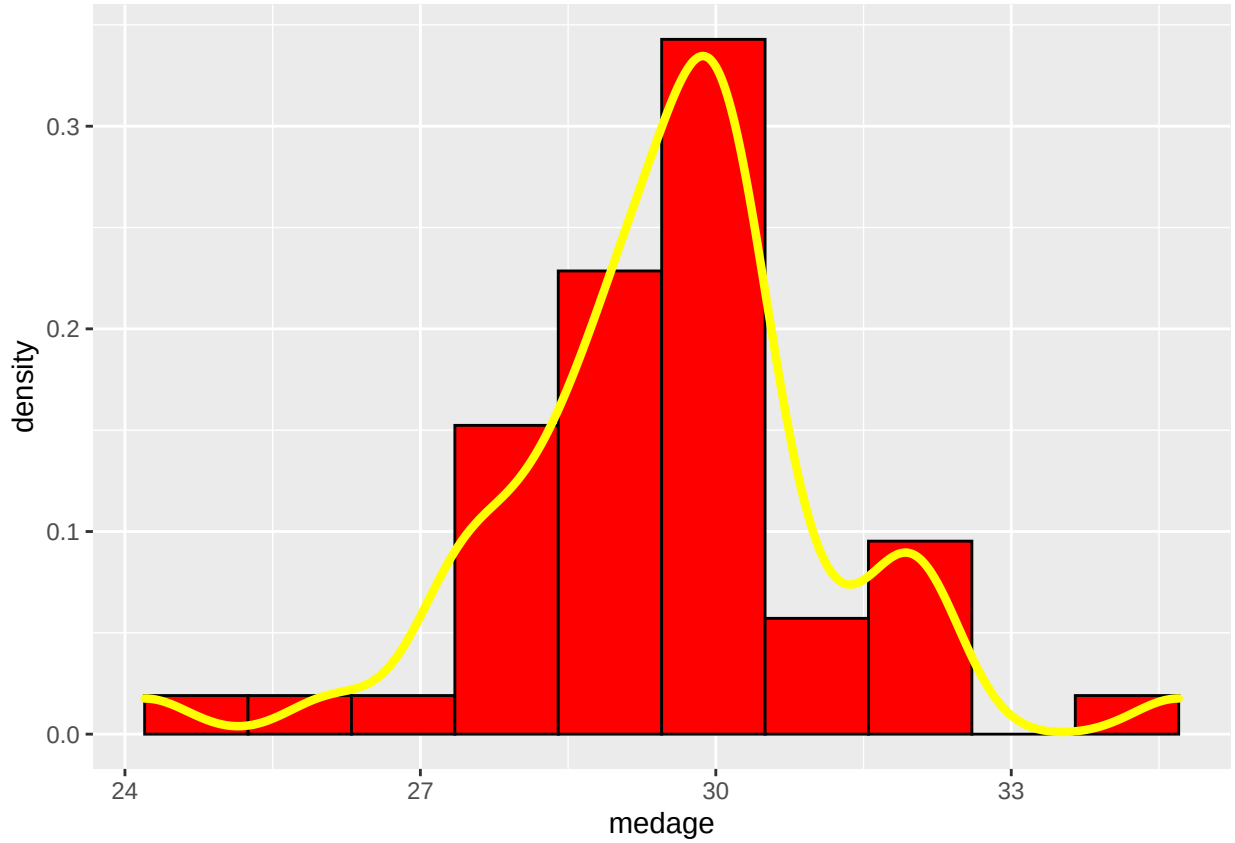
Para ver cuáles son los 10 estados más envejecidos, calculamos la edad mediana y el porcentaje de la población mayor de 65 años. En la siguiente tabla se muestran los 10 estados con mayor edad mediana.

```
(pop_mayor <- census %>%
  summarise(state, edad_mediana = medage, porc_mayor = pop65p/pop) %>%
  arrange(desc(edad_mediana)) %>%
  head(10))
```

```
## # A tibble: 10 x 3
##    state         edad_mediana porc_mayor
##    <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
```

Por último, representamos un histograma de la variable medage junto con la curva de densidad.

```
cortes = seq(min(census$medage),max(census$medage),length.out=11)
ggplot(data = census,aes(x=medage)) +
  geom_histogram(aes(y=stat(density)), breaks = cortes, fill = "red", color = "black")+
  geom_density(color = "yellow",size =1.5)
```



Práctica 1

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

Antes de nada, guardamos en la variable chlstrl el fichero cholesterol.csv.

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

La información básica del fichero nos la proporciona el comando str(). Vemos que es un data.frame con 403 observaciones y 7 variables, cuyos nombres también se muestran, junto con el tipo de dato que guardan.

```
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 ...
```

Podemos ver los datos de tipo NA que hay en la tabla utilizando la función is.na(). Si sobre esta aplicamos which(), nos devuelve las posiciones donde hay datos de tipo NA. Los números que devuelve cuentan en vertical (de columna en columna), comenzando en la primera y acabando en la última. Sin embargo, también podemos ver los valores NA por columna si aplicamos la misma función a alguna de estas columnas, tal y como se muestra a continuación. Así vemos, por posición en cada una de las columnas, dónde hay datos de tipo NA.

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

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

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

```
## [1] 28
```

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

```
## integer(0)
```

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

```
## integer(0)
```

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

```
## [1]  64  87 196 232 318
```

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

```
## [1] 162
```

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

```
## [1] 337 394
```

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

```
## [1] 337 394
```

Vemos que no hay datos de tipo NA ni en la columna age ni en la columna gender.

Ahora analizamos una variable numérica y continua, en este caso la variable chol. Para ello, primero mostramos toda su información estadística con summary(), es decir, el mínimo y el máximo, la media y la mediana, el primer y tercer cuartiles y el número de NA que tiene. Por otro lado, calculamos su desvición típica.

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

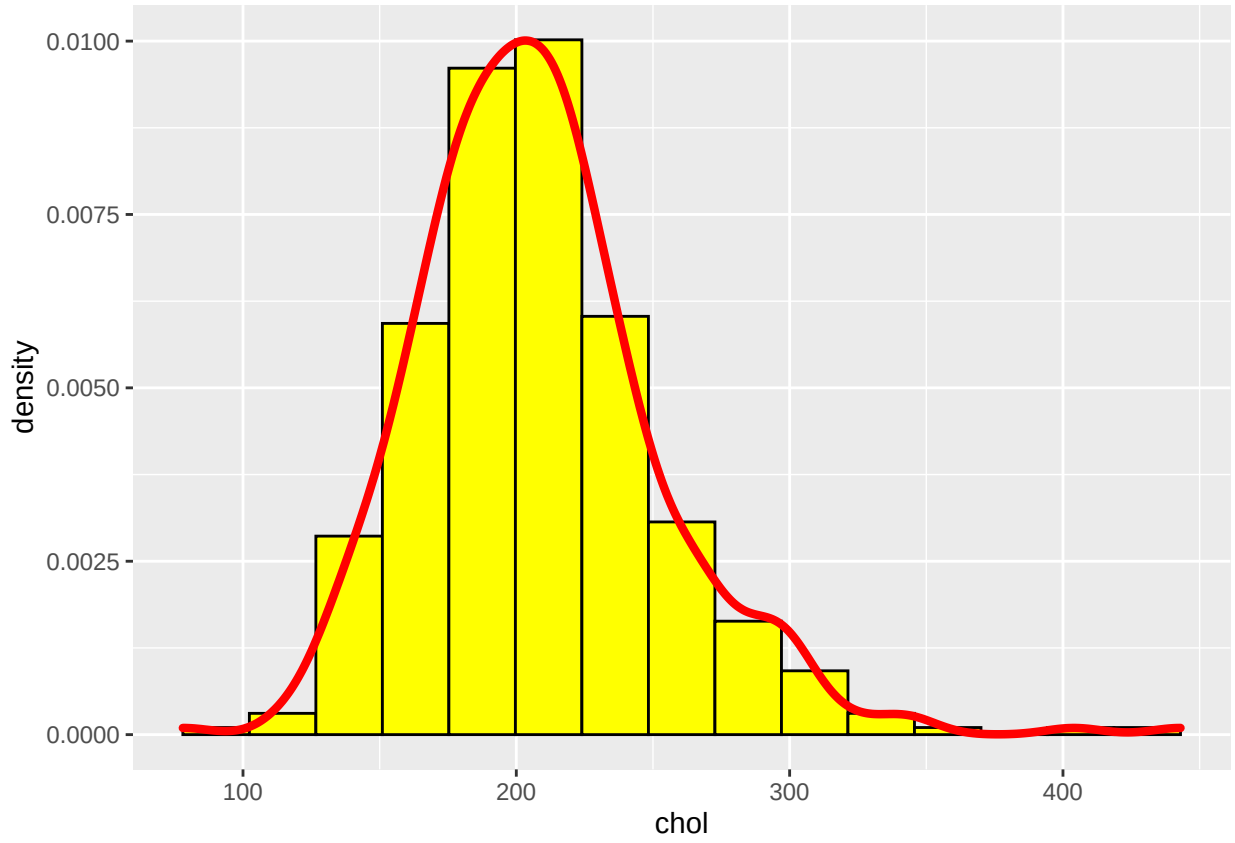
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##    78.0   179.0   204.0   207.8   230.0   443.0       1
```

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

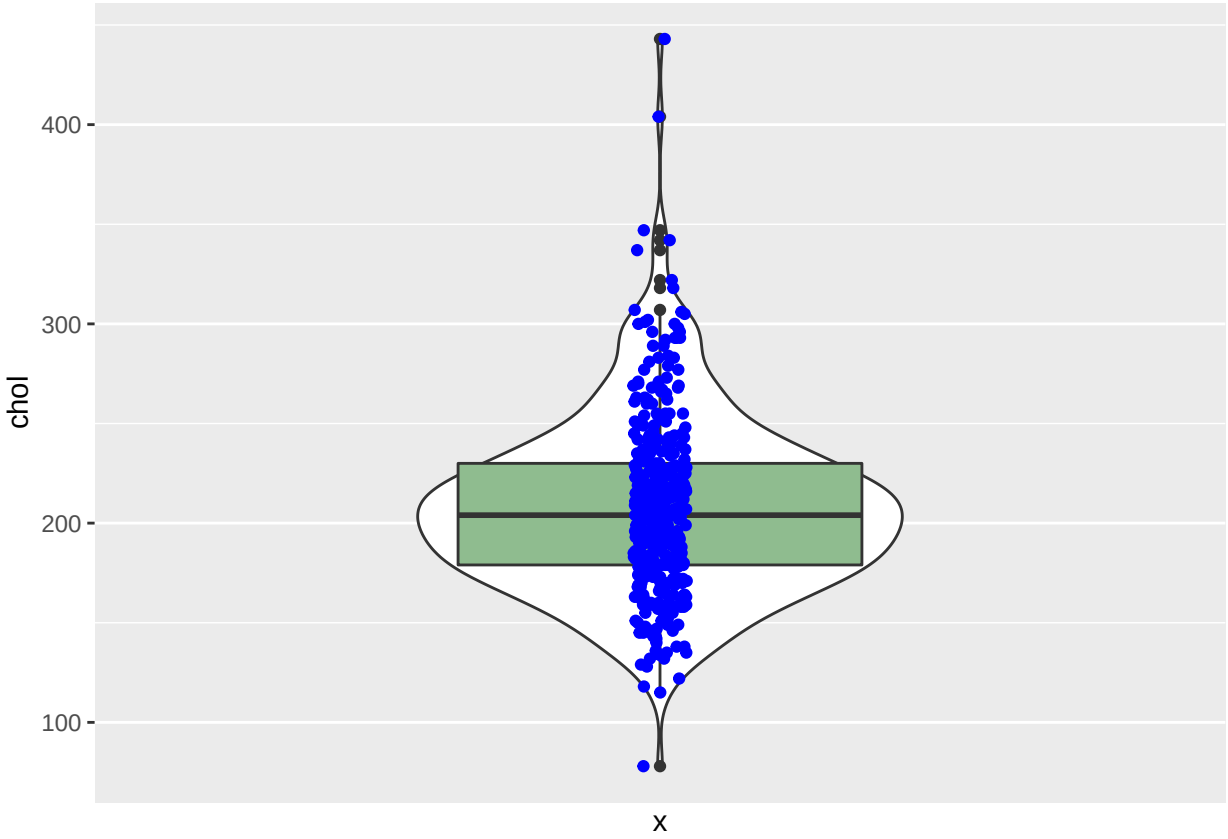
```
## [1] 44.44556
```

Para terminar de comprender la variable, a continuación se representan una serie de gráficas. En la primera de ellas, vemos un histograma junto con la curva de densidad correspondiente. En la segunda, un boxplot junto con las curvas de densidad de los puntos.

```
cortes = seq(min(chlstrl$chol, na.rm = TRUE),max(chlstrl$chol, na.rm = TRUE),length.out = 16)
ggplot(data = chlstrl, mapping = aes(chol)) +
  geom_histogram(aes(y = stat(density)), breaks = cortes, fill = "yellow", color = "black") +
  geom_density(color = "red", size = 1.5)
```




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



Analizamos ahora una varible de tipo factor. En este caso es la variable gender que puede tomar los valores male/female. El primer paso es convertirla a tipo factor. Después, calculamos las tablas de frecuencias absolutas (que nos da el número de hombres y de mujeres en el estudio) y la de frecuencias relativas (que nos da sus correspondientes proporciones). Por último, representamos en un gráfico de barras los dos factores, de forma que podemos observar esta proporción de manera más visual.

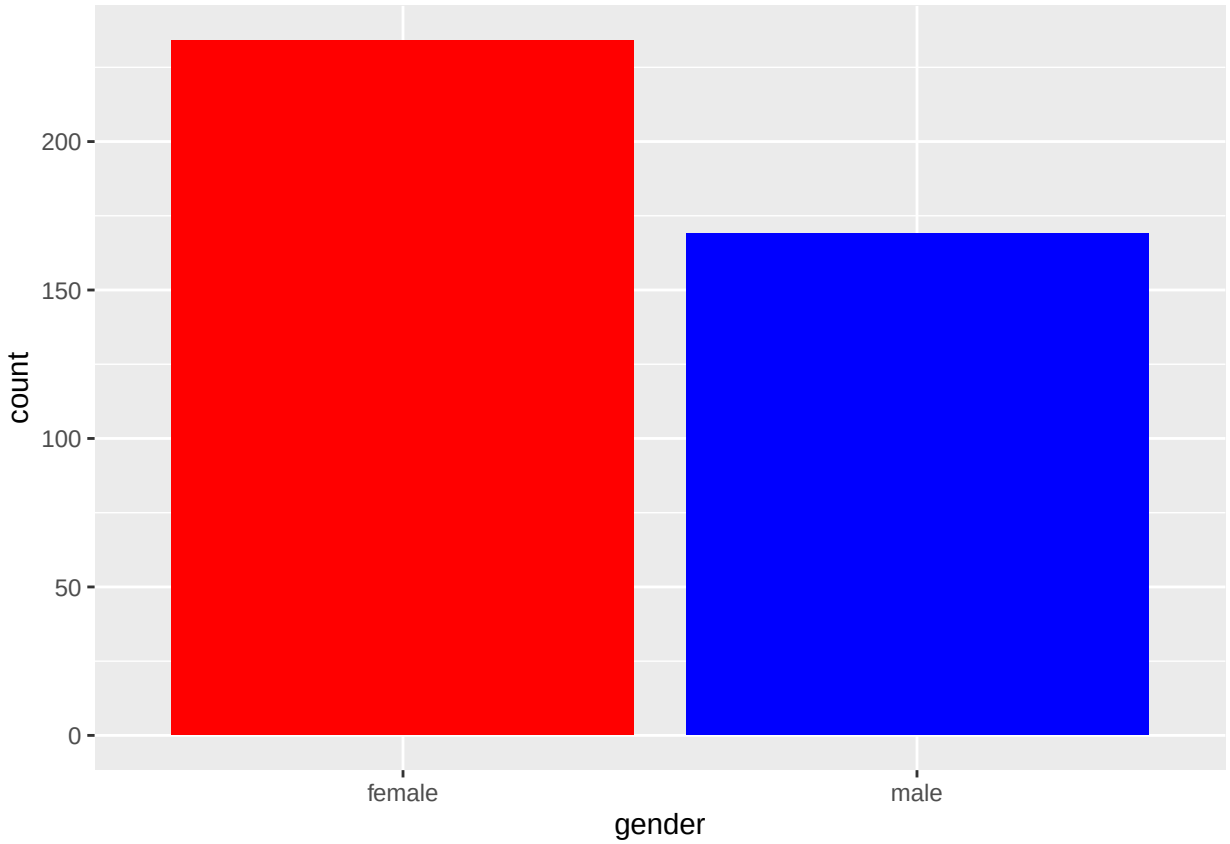
```
chlstrl$gender = factor(chlstrl$gender)

chlstrl %>%
  count(gender)
```

```
##   gender   n
## 1 female 234
## 2   male 169
```

```
chlstrl %>%
  count(gender) %>%
  mutate(gender,prop.table(n),n= NULL)
```

```
##   gender prop.table(n)
## 1 female     0.5806452
## 2   male     0.4193548
```

```
ggplot(chlstrl) +
  geom_bar(mapping = aes(x = gender), fill=c("red","blue"))
```



Transformamos ahora las medidas de altura y peso (height y weight) al sistema internacional.

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

Añadimos la columna BMI:

```
chlstrl_si <- chlstrl_si %>%
  mutate("BMI" = weight/(height)^2)
```

Creamos ahora una nueva columna en la que se divide la edad en tres intervalos (se pueden ver en dicha tabla). Con esto último, se puede hacer un conteo del número de personas en cada uno de los intervalos, así como una media del BMI y del cholesterol de las mujeres en cada uno de los grupos de edad. Todo ello se muestra a continuación.

```
edades <- cut(chlstrl$age,breaks = seq(10,100,30))

chlstrl_age <- chlstrl_si %>%
  mutate("ageGroup" = cut(age,breaks = seq(10,100,30)))

chlstrl_age %>%
  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
```

```
chlstrl_age %>%
  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.

Creamos la función cambiosSigno(v) que nos devuelve el número de cambios de signo que hay en un vector que le pasamos como argumento.

```
cambiosSigno = function(vector_num, cambios = 0){
  long = length(vector_num)
  for (k in 2:long){
    if (vector_num[k]*vector_num[k-1] < 0){
        cambios = cambios + 1
      }
    }
    return(cambios)
}
```

La podemos modificar para que nos devuelva la posición donde se producen esos cambios de signo y la llamamos cambiosSignoPos(v).

```
cambiosSignoPos = function(vector_num, cambios = c()){
  long = length(vector_num)
  for (k in 2:long){
    if (vector_num[k]*vector_num[k-1] < 0){
        cambios = append(cambios,k)
      }
    }
    return(cambios)
}
```

Incluimos el código que genera vectores aleatorios sin tener en cuenta el 0 y se lo pasamos a las funciones para que nos devuelvan los cambios de signo y dónde se han producido estos.

```
(vector = sample(c(-10:-1,1:10),20,replace = TRUE))
```

```
##  [1]  2 -5  8 -9 -9  4  8 -3 -9 -1 -4 10  8 -9  2 10  6  3 -8  5
```

```
cambiosSigno(vector)
```

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

```
cambiosSignoPos(vector)
```

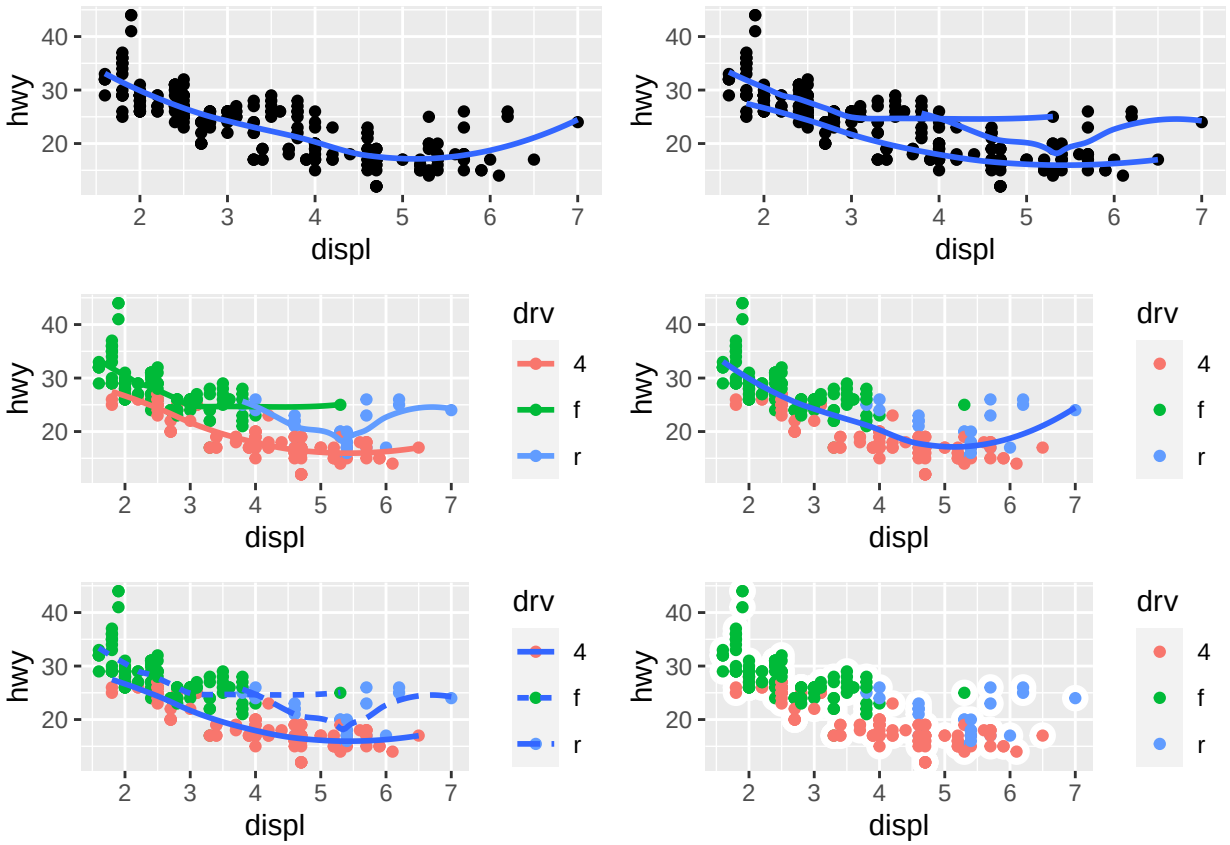
```
##  [1]  2  3  4  6  8 12 14 15 19 20
```

Ejercicio 3. R4DS.

Ejercicio 6 de la sección 3.6.1 de R4DS

En este ejercicio se pretende replicar el código que genera la figura del enunciado. Este código se presenta a continuación. Hemos creado 6 figuras y luego hemos hecho un grid con ellas, todo utilizando los datos de la tabla mpg.

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

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

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

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

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

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

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



Ejercicio 1 de la sección 5.2.4 de r4DS

Este ejercicio demanda hacer una serie de filtros con el comando filter(). Estos son:

Vuelos que:

1. Tengan un retraso de llegada de más de dos horas:

```
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:

```
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. Su operadora fue “United”, “American” o “Delta”:

```
flights %>%
  filter(carrier %in% c("UA","AA","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>
```

4. Volaron en verano:

```
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>
```

5. No salieron tarde pero llegaron más de dos horas 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 por lo menos una hora, pero recuperaron media hora en 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>
```

7. Salieron entre medianoche y las 6 AM:

```
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>
```