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

Tarea 1. Solución.

Departamento de Matemática Aplicada

Curso 2021-22. Última actualización: 2021-09-29

Contents

Instrucciones preliminares

- Empieza abriendo el proyecto de RStudio correspondiente a tu repositorio personal de la asignatura.
- En todas las tareas tendrás que repetir un proceso como el descrito en la sección *Repite los pasos Creando un fichero Rmarkdown para esta práctica* de la *Práctica00*. Puedes releer la sección *Practicando la entrega de las Tareas* de esa misma práctica para recordar el procedimiento de entrega.

Ejercicio 0

- Si no has hecho los *Ejercicios* de la *Práctica00* (págs. 12 y 13) hazlos ahora y añádelos a esta tarea. Si ya los has hecho y entregado a través de GitHub no hace falta que hagas nada.

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

- Vamos a utilizar el conjunto de datos contenido en el fichero (es un enlace):
  cholesterol.csv
  Los datos proceden de un estudio realizado en la *University of Virginia School of Medicine* que investiga

la prevalencia de la obesidad, la diabetes y otros factores de riesgo cardiovascular. Se puede encontrar más información sobre el fichero en este enlace:
https://biostat.app.vumc.org/wiki/pub/Main/DataSets/diabetes.html

- Carga el conjunto de datos en un data.frame de R llamado `chlstrl`.
Solución: Lo cargaremos con `read_csv` del tidyverse directamente desde la URL. El resultado es un `tibble` (que por tanto también es un `data.frame`).

```
library(tidyverse)

URL_chlstrl <-  "https://bit.ly/3EQsIMp"

chlstrl <-  read_csv(URL_chlstrl)
```

- Empezaremos por información básica sobre el conjunto de datos. Cuántas observaciones contiene, cuáles son las variables y de qué tipos,...
Solución: Puedes empezar usando `str` y `summary` de R base

```
str(chlstrl)
```

```
## spec_tbl_df [403 x 7] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ chol  : num [1:403] 203 165 228 78 249 248 195 227 177 263 ...
##  $ age   : num [1:403] 46 29 58 67 64 34 30 37 45 55 ...
##  $ gender: chr [1:403] "female" "female" "female" "male" ...
##  $ height: num [1:403] 62 64 61 67 68 71 69 59 69 63 ...
##  $ weight: num [1:403] 121 218 256 119 183 190 191 170 166 202 ...
##  $ waist : num [1:403] 29 46 49 33 44 36 46 34 34 45 ...
##  $ hip   : num [1:403] 38 48 57 38 41 42 49 39 40 50 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   chol = col_double(),
##   ..   age = col_double(),
##   ..   gender = col_character(),
##   ..   height = col_double(),
##   ..   weight = col_double(),
##   ..   waist = col_double(),
##   ..   hip = col_double()
##   .. )
```

```
summary(chlstrl)
```

```
##       chol            age           gender              height     
##  Min.   : 78.0   Min.   :19.00   Length:403         Min.   :52.00  
##  1st Qu.:179.0   1st Qu.:34.00   Class :character   1st Qu.:63.00  
##  Median :204.0   Median :45.00   Mode  :character   Median :66.00  
##  Mean   :207.8   Mean   :46.85                      Mean   :66.02  
##  3rd Qu.:230.0   3rd Qu.:60.00                      3rd Qu.:69.00  
##  Max.   :443.0   Max.   :92.00                      Max.   :76.00  
##  NA's   :1                                          NA's   :5      
##      weight          waist            hip       
##  Min.   : 99.0   Min.   :26.0   Min.   :30.00  
##  1st Qu.:151.0   1st Qu.:33.0   1st Qu.:39.00  
##  Median :172.5   Median :37.0   Median :42.00
```

```
##  Mean   :177.6   Mean   :37.9   Mean   :43.04
##  3rd Qu.:200.0   3rd Qu.:41.0   3rd Qu.:46.00
##  Max.   :325.0   Max.   :56.0   Max.   :64.00
##  NA's   :1       NA's   :2      NA's   :2
```

En particular presta atención a los avisos sobre valores ausentes de `summary`. Además, todas las variables salvo gender`parecen claramente variables cuantitativas continuas (toman suficientes valores distintos como para considerarlas así). Puedes ver el núemro de valores distintos de cada variable fácilmente así, con la función`mapde la librería`apurr‘ del *tidyverse* (ver capítulo 20 de R4DS):

```
chlstrl %>% map_dfr(.f = ~ length(unique(.x)))
```

```
## # A tibble: 1 x 7
##    chol   age gender height weight waist   hip
##   <int> <int>  <int>  <int>  <int> <int> <int>
## 1   155    68      2     23    141    31    33
```

En cuanto a `gender` es claramente un factor con dos niveles y lo mejor es tratarlo como tal:

```
chlstrl <- chlstrl %>%
  mutate(gender = factor(gender))
```

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

```
(whereNA = which(is.na(chlstrl), arr.ind = TRUE))
```

```
##       row col
##  [1,]  28   1
##  [2,]  64   4
##  [3,]  87   4
##  [4,] 196   4
##  [5,] 232   4
##  [6,] 318   4
##  [7,] 162   5
##  [8,] 337   6
##  [9,] 394   6
## [10,] 337   7
## [11,] 394   7
```

Vamos a aprovechar este momento para usar la función `map` de dplyr. Podéis leer más sobre la familia de funciones `map` en el Capítulo 21 de R4DS y os recomendamos que lo hagáis para introduciros en la librería *purr* del *tidyverse* y el mundo de la programación funcional con R.

```
chlstrl %>%
  map(~ which(is.na(.))) %>%
  keep(~ length(.) > 0)
```

```
## $chol
## [1] 28
##
```

```
## $height
## [1]  64  87 196 232 318
##
## $weight
## [1] 162
##
## $waist
## [1] 337 394
##
## $hip
## [1] 337 394
```

Una forma rápida de comprobar si hay datos ausentes en una tabla es ejecutar:

```
any(is.na(chlstrl))
```

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

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

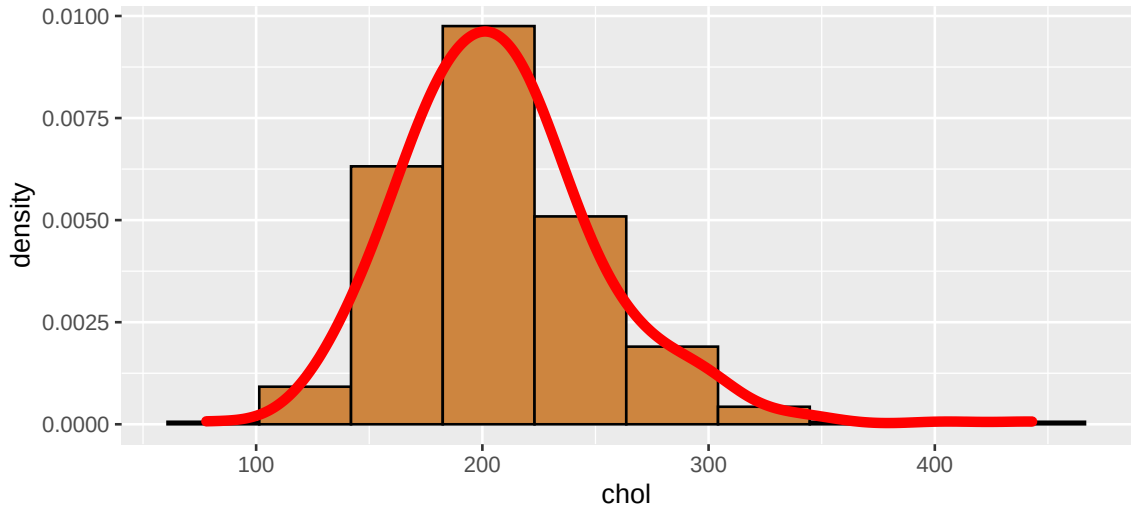
  – Variables categóricas (factores).
  Tablas de frecuencia (absolutas y relativas).
  Gráficas (diagrama de barras).
  Solución: Vamos a usar `chol` como ejemplo de variable cuantitativa. Empezamos calculando el resumen de esta función:

  ```
  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
  ```

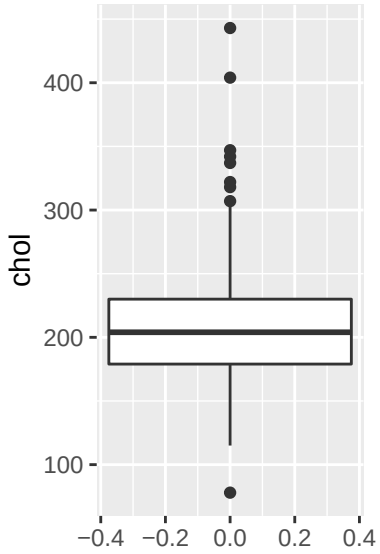
  A continuación la representaremos gráficamente mediante un histograma, con 10 intervalos y su curva de densidad. Vamos a eliminar el único valor ausente de esa variable antes de la gráfica:

  ```
  chlstrl %>%
    drop_na(chol) %>%
    ggplot(aes(x = chol)) +
    geom_histogram(aes(y = stat(density)), bins = 10, fill="tan3", color = "black") +
    geom_density(color = "red", size=2, adjust = 1.5)
  ```



Un bxplot de esta variable muestra la presencia de varios valores atípicos:

```
chlstrl %>%
  drop_na(chol) %>%
  ggplot(aes(y = chol)) +
  geom_boxplot()
```



Una forma sencilla de identificarlos es usando la función `boxplot` de R básico:

```
bxp_chol  <-  boxplot(chlstrl$chol, plot = FALSE)
(chol_outvalues <-  bxp_chol$out)
```

```
## [1]  78 443 318 347 342 404 307 337 322
```

```
(where_choloutliers <-  which(chlstrl$chol %in% chol_outvalues))
```

```
## [1]   4  63 134 148 213 295 360 378 381
```

- 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. Usa dplyr para convertir esas columnas a metros y kilogramos respectivamente. Las nuevas columnas deben llamarse igual que las originales.
  Solución:

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

Veamos el principio de la tabla:

```
chlstrl %>%
  slice_head(n = 10)
```

```
## # A tibble: 10 x 7
##     chol   age gender height weight waist   hip
##    <dbl> <dbl> <fct>   <dbl>  <dbl> <dbl> <dbl>
##  1   203    46 female   1.57   54.9    29    38
##  2   165    29 female   1.63   99.0    46    48
##  3   228    58 female   1.55  116.     49    57
##  4    78    67 male     1.70   54.0    33    38
##  5   249    64 male     1.73   83.1    44    41
##  6   248    34 male     1.80   86.3    36    42
##  7   195    30 male     1.75   86.7    46    49
##  8   227    37 male     1.50   77.2    34    39
##  9   177    45 male     1.75   75.4    34    40
## 10   263    55 female   1.60   91.7    45    50
```

- Ahora usa esos valores de `height` y `weight` para añadir una nueva columna llamada BMI, definida mediante:

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

(se divide por el cuadrado de la altura).
Solución:

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

Aunque las columnas creadas con `mutate` se añaden a la derecha de la tabla, vamos a usar `select` para mostrar el resultado colocando BMI en primera posición. Esta visualización no afecta a la tabla `chlstrl`.

```
chlstrl %>%
  select(BMI, everything())
```

```
## # A tibble: 403 x 8
##      BMI  chol   age gender height weight waist   hip
##    <dbl> <dbl> <dbl> <fct>   <dbl>  <dbl> <dbl> <dbl>
##  1  22.2   203    46 female   1.57   54.9    29    38
##  2  37.5   165    29 female   1.63   99.0    46    48
##  3  48.4   228    58 female   1.55  116.     49    57
##  4  18.7    78    67 male     1.70   54.0    33    38
##  5  27.8   249    64 male     1.73   83.1    44    41
##  6  26.5   248    34 male     1.80   86.3    36    42
##  7  28.2   195    30 male     1.75   86.7    46    49
##  8  34.4   227    37 male     1.50   77.2    34    39
##  9  24.5   177    45 male     1.75   75.4    34    40
## 10  35.8   263    55 female   1.60   91.7    45    50
## # ... with 393 more rows
```

- Crea una nueva columna llamada `ageGroup` dividiendo la edad en los siguientes tres niveles:

  `(10,40], (40,70], (70,100]`

  Solución:

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

  Veamos ahora los valores de `age` y `ageGroup` en una muestra aleatoria de filas de la tabla para comprobar que el resultado es el esperado.

```
chlstrl %>%
  select(age, ageGroup) %>%
  slice_sample(n = 15)
```

```
## # A tibble: 15 x 2
##      age ageGroup
##    <dbl> <fct>
##  1    52 (40,70]
##  2    33 (10,40]
##  3    20 (10,40]
##  4    37 (10,40]
##  5    91 (70,100]
##  6    68 (40,70]
##  7    19 (10,40]
##  8    41 (40,70]
##  9    68 (40,70]
## 10    65 (40,70]
## 11    37 (10,40]
## 12    55 (40,70]
## 13    38 (10,40]
## 14    28 (10,40]
## 15    52 (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?
  Solución:

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

```
## # A tibble: 3 x 2
##   ageGroup     n
##   <fct>    <int>
## 1 (10,40]    160
## 2 (40,70]    207
## 3 (70,100]    36
```

  Y si sólo usamos las observaciones correspondientes a mujeres:

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

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

Para la segunda parte:

```r
chlstrl %>%
  filter(gender == "female") %>%
  group_by(ageGroup) %>%
  drop_na(chol, BMI) %>%
  summarise(n = n(), media_chol = mean(chol), media_BMI = mean(BMI))
```

```
## # A tibble: 3 x 4
##   ageGroup     n media_chol media_BMI
##   <fct>    <int>      <dbl>     <dbl>
## 1 (10,40]     94       188.      30.5
## 2 (40,70]    115       221.      30.3
## 3 (70,100]    20       230.      29.4
```

Fíjate en el uso de `drop_na`. Si no lo empleamos las medias valdrían ambas `NA`. En particular ahora la tabla de frecuencia es distinta porque se han excluido las observaciones con datos ausentes de alguna de esas dos variables.

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).
  Solución: hay muchas posibles soluciones, desde el uso de bucles for para recorrer el vector buscando cambios de signo a otras que explotan el comportamiento vectorializado de muchas funciones de R. Aquí mostramos una posible solución que simplemente comprueba si el signo del producto de cada elemento (desde el segundo hasta el último) con el que lo precede es negativo.

  Para ponerla a prueba fabricaremos vectores de enteros no nulos:

- Modifica la función para que devuelva como resultado las posiciones donde hay cambios de signo. Llama `cambiosSignoPos(x)` a esa otra función. Por ejemplo, para el vector anterior el resultado de esta función sería [1] 3 4 7 8 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).
  Solución: una posible solución basada en la anterior.

```r
cambiosSignoPos = function(x){
  n = length(x)
  x2 = x[2:n]
  which(x2 * x[1:(n - 1)] < 0) + 1
}
```

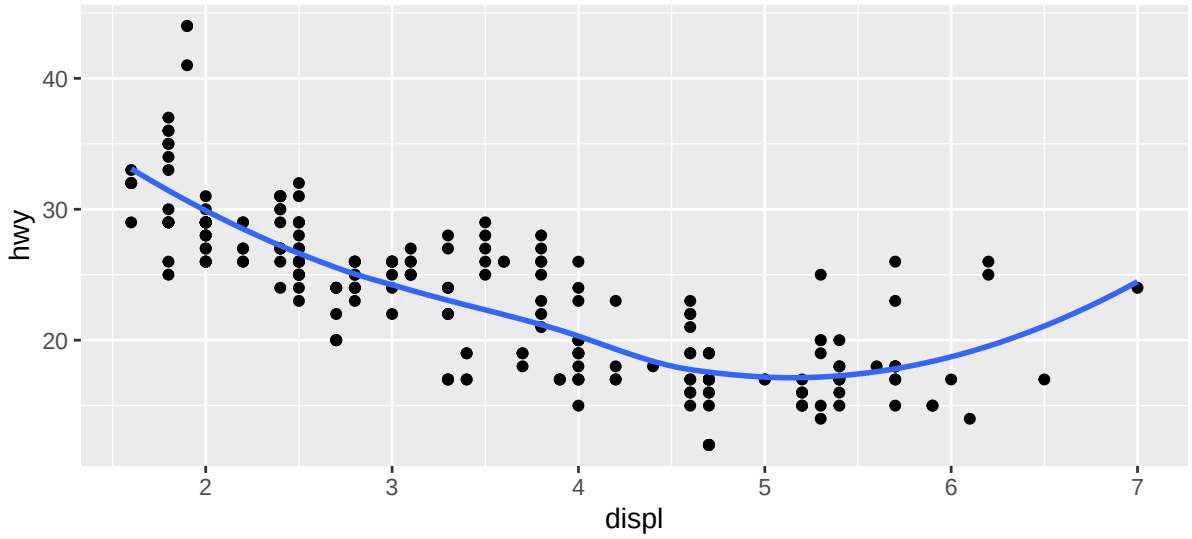
Ejercicio 3. R4DS.

Es recomendable que esta semana del curso hagas al menos una lectura somera de los Capítulos 1 a 5 de R for Data Science (R4DS), de H. Wickham, con énfasis especial en los Capítulos 3 y 5 (los capítulos 1, 2 y 4 son muy breves). Los siguientes apartados pretenden motivar esa lectura y por eso mismo pueden resultar un poco más laboriosos.
Solución: Existe una página web con soluciones a los ejercicios de R4DS recopiladas por Jeffrey B. Arnold en este enlace:
https://jrnold.github.io/r4ds-exercise-solutions.
Recomendamos usarla como complemento a vuestro trabajo con las advertencias habituales sobre cualquier colección de ejercicios resueltos.

- Haz el ejercicio 6 de la Sección 3.6.1 de R4DS.
  Solución: Debajo del enlace al gráfico original incluimos una posible solución para generarlo. En todos los casos hay otras opciones igualmente válidas.
  Graph 1:

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

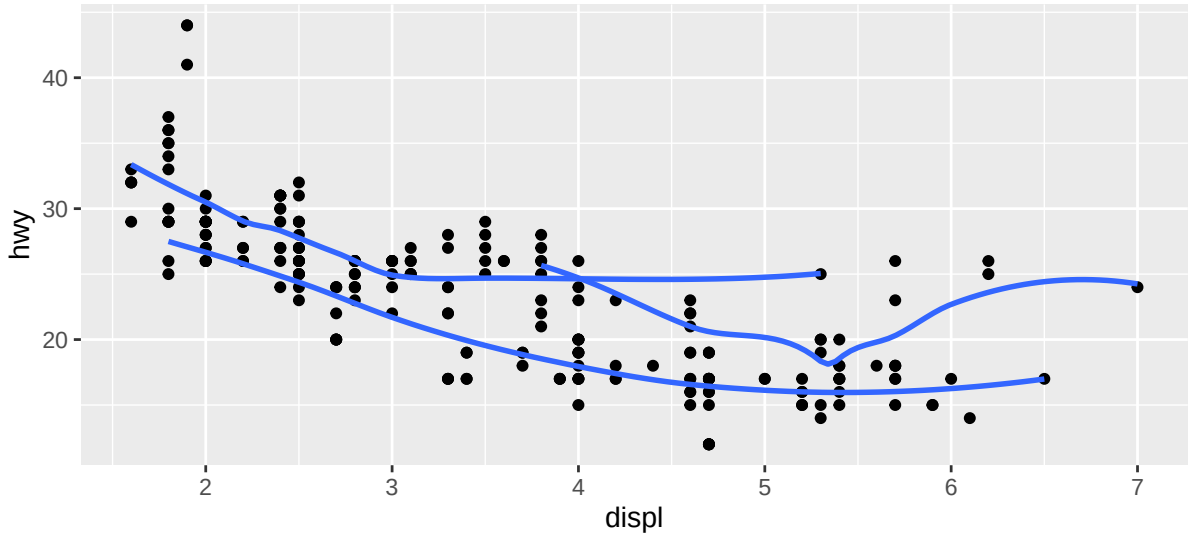
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Graph 2:

```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_point() +
  geom_smooth(aes(group = drv), show.legend = FALSE ,se = FALSE)
```

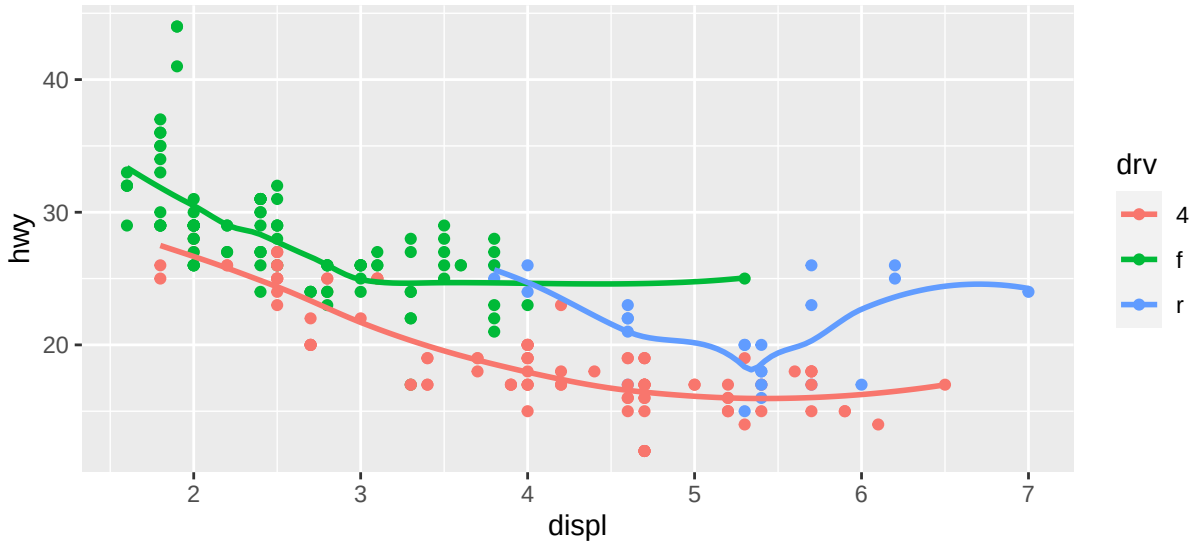
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Graph 3:

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

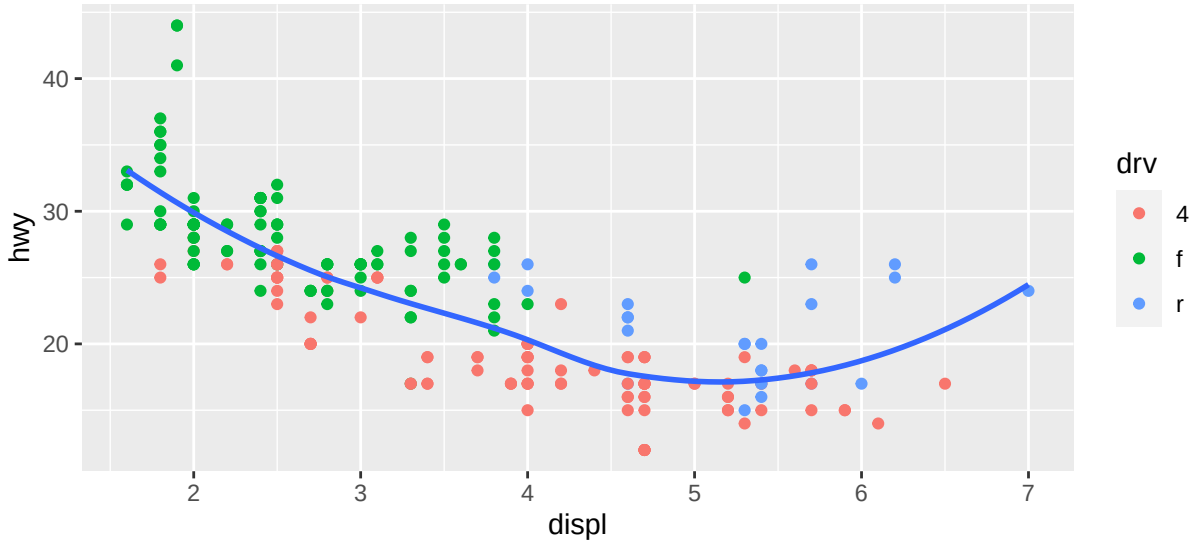
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Graph 4:

```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_point(aes(color = drv)) +
  geom_smooth(se = FALSE)
```

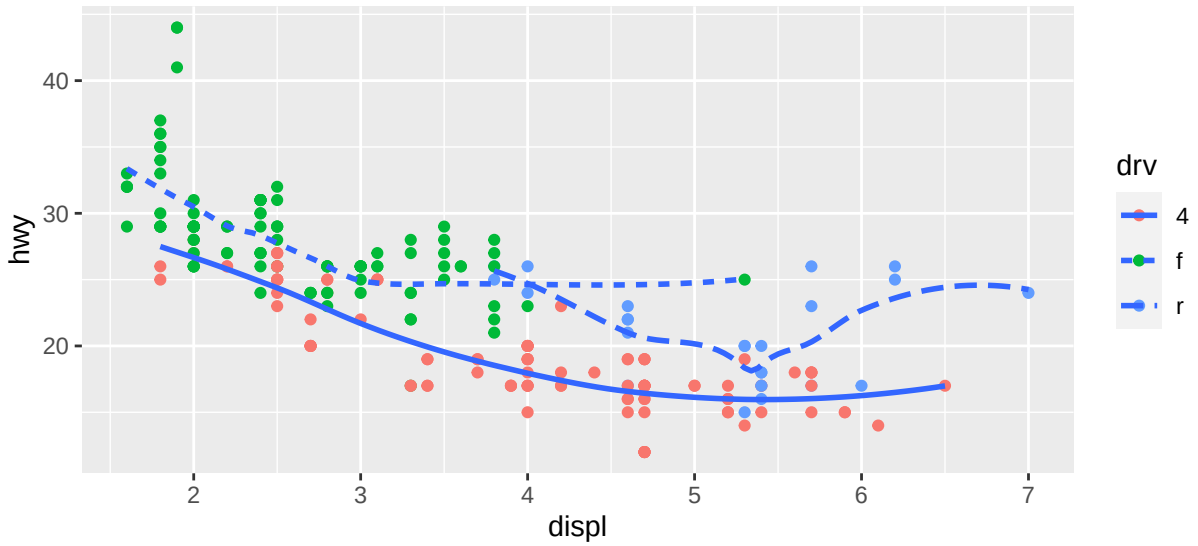
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Graph 5:

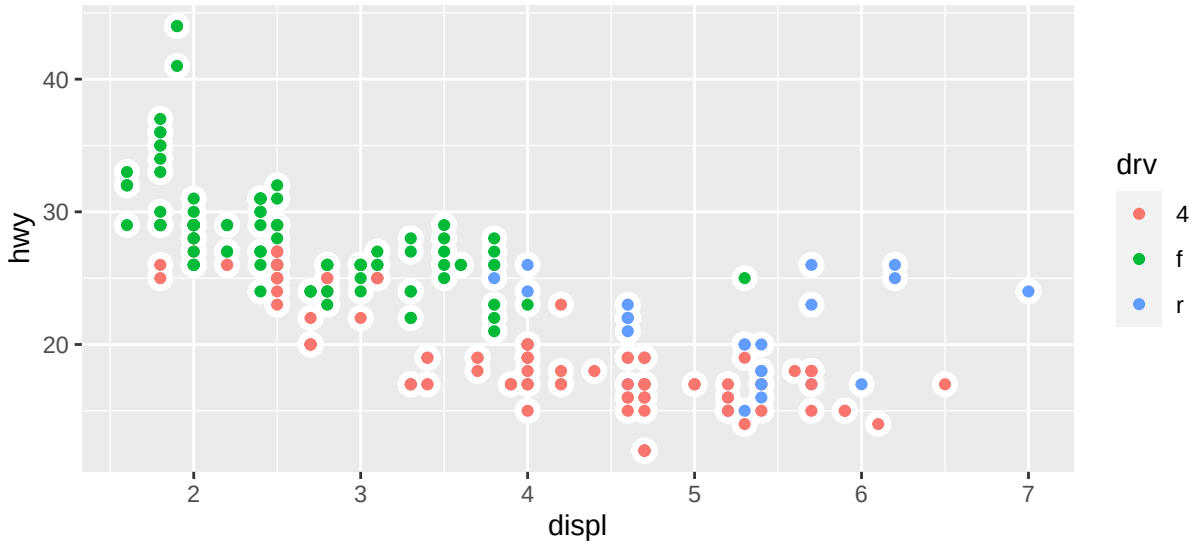
```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_point(aes(color = drv)) +
  geom_smooth(aes(linetype = drv), se = FALSE)
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Graph 6:

```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy, color = drv)) +
  geom_point(colour = "white", size = 4) +
  geom_point()
```



- Haz el ejercicio 1 de la Sección 5.2.4 de R4DS.
  Solución: De nuevo, las soluciones que se incluyen aquí son en la mayoría de los casos nada más que una de las muchas posibilidades. Recomendamos consultar el libro / web de Jeffrey B. Arnold al que nos hemos referido antes.

```
require(nycflights13)
```

```
## Loading required package: nycflights13
```

```
# 1. Had an arrival delay of two or more hours

summary(flights$arr_delay)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.     NA's
##  -86.000  -17.000   -5.000    6.895   14.000 1272.000     9430
```

```
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. Flew to Houston (IAH or HOU)

flights %>%
  filter(dest %in% c("IAH", "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. Were operated by United, American, or 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>
```

```r
# 4. Departed in summer (July, August, and September)

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>
```

```r
# 5. Arrived more than two hours late, but didn't leave late

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>
```

```r
# 6. Were delayed by at least an hour, but made up over 30 minutes in flight

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

```
## # 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. Departed between midnight and 6am (inclusive)

flights %>%
  filter(dep_time < 600 |   dep_time == 2400)
```

```
## # A tibble: 8,759 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 8,749 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>
```

```
flights %>%
  mutate(condi = between(hour, 0, 6)) %>%
  select(hour, condi) %>%
  group_by(hour) %>%
  count(condi)
```

```
## # A tibble: 20 x 3
## # Groups:   hour [20]
##     hour condi     n
##    <dbl> <lgl> <int>
##  1     1 TRUE      1
##  2     5 TRUE   1953
##  3     6 TRUE  25951
##  4     7 FALSE 22821
##  5     8 FALSE 27242
##  6     9 FALSE 20312
##  7    10 FALSE 16708
```

```
##  8    11 FALSE 16033
##  9    12 FALSE 18181
## 10    13 FALSE 19956
## 11    14 FALSE 21706
## 12    15 FALSE 23888
## 13    16 FALSE 23002
## 14    17 FALSE 24426
## 15    18 FALSE 21783
## 16    19 FALSE 21441
## 17    20 FALSE 16739
## 18    21 FALSE 10933
## 19    22 FALSE  2639
## 20    23 FALSE  1061
```