

Big Analytics VI: Programación en **R**

Harold A. Hernández-Roig

4-5 Febrero 2022

Contents

1	Introducción	5
1.1	Referencias	5
2	Visualización	7
2.1	Paquetes	7
2.2	Datos	8
2.3	Visualización con R base	9
2.4	Visualización con <code>ggplot2</code>	10
2.5	Resumen	43
3	Transformaciones	45
3.1	Datos	45
3.2	El paquete <code>dplyr</code>	47
4	Tidy	73
4.1	Datos	73
4.2	Pivotar	74
4.3	Separar y unir	79
4.4	Lidiar con los datos faltantes	82
4.5	Case study	85
5	Relational Data	89
5.1	Datos	89
5.2	Keys	89
5.3	Mutating Joins	90
5.4	Filtering Joins	92

Chapter 1

Introducción

Estos son los ejercicios del curso + soluciones :)

Recuerda que tienes disponibles las diapositivas en: <https://hhroig.github.io/BA-VI-Xaringan-Slides/>

1.1 Referencias

Chang, Winston. 2012. R Graphics Cookbook: Practical Recipes for Visualizing Data. O'Reilly Media. (Versión parcial online libre: <http://www.cookbook-r.com/Graphs/>).

Wickham, H. 2015. Advanced R. Chapman & Hall. (Versión online libre: <http://adv-r.had.co.nz/>)

Wickham, Hadley, and Garrett Grolemund. 2017. R for Data Science: Import, Tidy, Transform, Visualize, and Model Data. 1st ed. O'Reilly Media. (Versión online libre: [r4ds](https://r4ds.org/)).

Holtz, Yan. 2018. The R Graph Gallery. Recurso online: <https://www.r-graph-gallery.com/index.html>

Chapter 2

Visualización

Recuerda, trabajaremos en un *script* de **R**, no en la *Consola*. Además lo haremos de forma segura y organizada creando un *RStudio Project*:

- Ir a *File > New Project...*
- Podemos crear un nuevo directorio donde guardar nuestros scripts, figuras, datos, etc.;
- Por ejemplo, en el Escritorio creamos el proyecto “*intro_R*”;
- Siempre que trabajemos en este proyecto, “*intro_R*” será nuestro *Working Directory*
- Ahora, creamos un nuevo script “*plots_mpg.R*” y a programar!

2.1 Paquetes

Necesitamos cargar el paquete `tidyverse`:

```
library(tidyverse)
```

```
## -- Attaching packages ----- tidyverse 1.3.1 --

## v ggplot2 3.3.5      v purrr  0.3.4
## v tibble  3.1.6      v dplyr  1.0.7
## v tidyr   1.1.4      v stringr 1.4.0
## v readr   2.1.1      v forcats 0.5.1

## -- Conflicts ----- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Notamos que este comando carga a su vez una serie de paquetes, no solo uno. Los *conflictos* son importantes a tener en cuenta porque indican que dos paquetes diferentes comparten el mismo nombre para una función. Por ejemplo, la función

`select` está repetida tanto en el paquete `dplyr` como en el paquete `MASS`. Si cargamos ambos paquetes en nuestro script, entonces para evitar conflictos debemos especificar `dplyr::select(...)` o `MASS::select(...)`.

2.2 Datos

Vamos a trabajar con los **data frames** `mpg`:

```
mpg
```

```
## # A tibble: 234 x 11
##   manufacturer model      displ  year   cyl trans drv      cty   hwy fl      class
##   <chr>          <chr>    <dbl> <int> <int> <chr> <chr> <int> <int> <chr> <chr>
## 1 audi          a4         1.8  1999     4 auto~ f      18    29 p      comp~
## 2 audi          a4         1.8  1999     4 manu~ f      21    29 p      comp~
## 3 audi          a4         2    2008     4 manu~ f      20    31 p      comp~
## 4 audi          a4         2    2008     4 auto~ f      21    30 p      comp~
## 5 audi          a4         2.8  1999     6 auto~ f      16    26 p      comp~
## 6 audi          a4         2.8  1999     6 manu~ f      18    26 p      comp~
## 7 audi          a4         3.1  2008     6 auto~ f      18    27 p      comp~
## 8 audi          a4 quattro 1.8  1999     4 manu~ 4      18    26 p      comp~
## 9 audi          a4 quattro 1.8  1999     4 auto~ 4      16    25 p      comp~
## 10 audi          a4 quattro 2    2008     4 manu~ 4      20    28 p      comp~
## # ... with 224 more rows
```

y `diamonds` de `ggplot2`:

```
head(diamonds, n = 10)
```

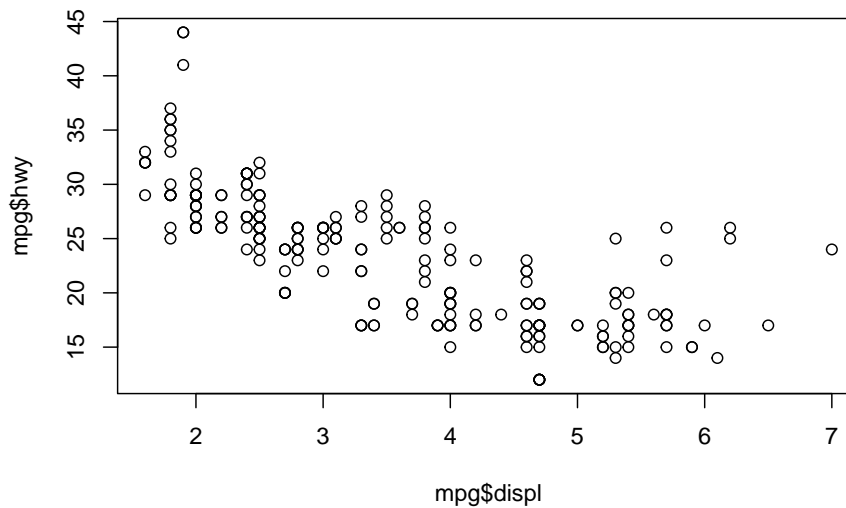
```
## # A tibble: 10 x 10
##   carat cut      color clarity depth table price      x      y      z
##   <dbl> <ord>    <ord> <ord>    <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1 0.23 Ideal    E     SI2     61.5    55    326  3.95  3.98  2.43
## 2 0.21 Premium E     SI1     59.8    61    326  3.89  3.84  2.31
## 3 0.23 Good    E     VS1     56.9    65    327  4.05  4.07  2.31
## 4 0.29 Premium I     VS2     62.4    58    334  4.2   4.23  2.63
## 5 0.31 Good    J     SI2     63.3    58    335  4.34  4.35  2.75
## 6 0.24 Very Good J     VVS2     62.8    57    336  3.94  3.96  2.48
## 7 0.24 Very Good I     VVS1     62.3    57    336  3.95  3.98  2.47
## 8 0.26 Very Good H     SI1     61.9    55    337  4.07  4.11  2.53
## 9 0.22 Fair    E     VS2     65.1    61    337  3.87  3.78  2.49
## 10 0.23 Very Good H     VS1     59.4    61    338  4     4.05  2.39
```

Un *data frame* es una colección rectangular de datos donde las variables están organizadas por columnas y las observaciones por filas. Si ejecutamos `?mpg` (o `?diamonds`) el panel de Ayuda brinda una descripción de los datos.

2.3 Visualización con R base

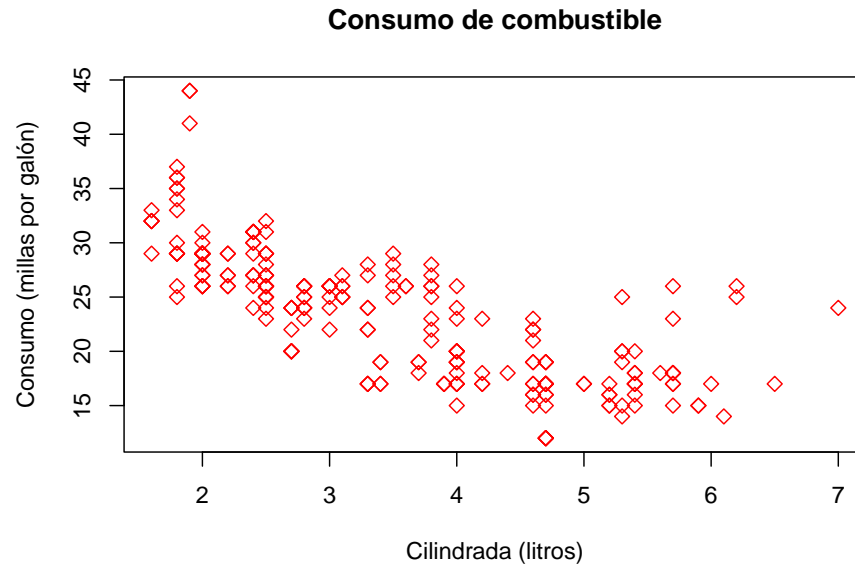
Nos vamos a concentrar en las variables `displ` y `hwy`:

```
plot(mpg$displ, mpg$hwy)
```



Esto es un *diagrama de dispersión*. Si hacemos `?plot` vemos las características que podemos variar. Por ejemplo:

```
plot(mpg$displ, mpg$hwy,  
     main = "Consumo de combustible",  
     xlab = "Cilindrada (litros)",  
     ylab = "Consumo (millas por galón)",  
     pch = 5,  
     col = "red")
```



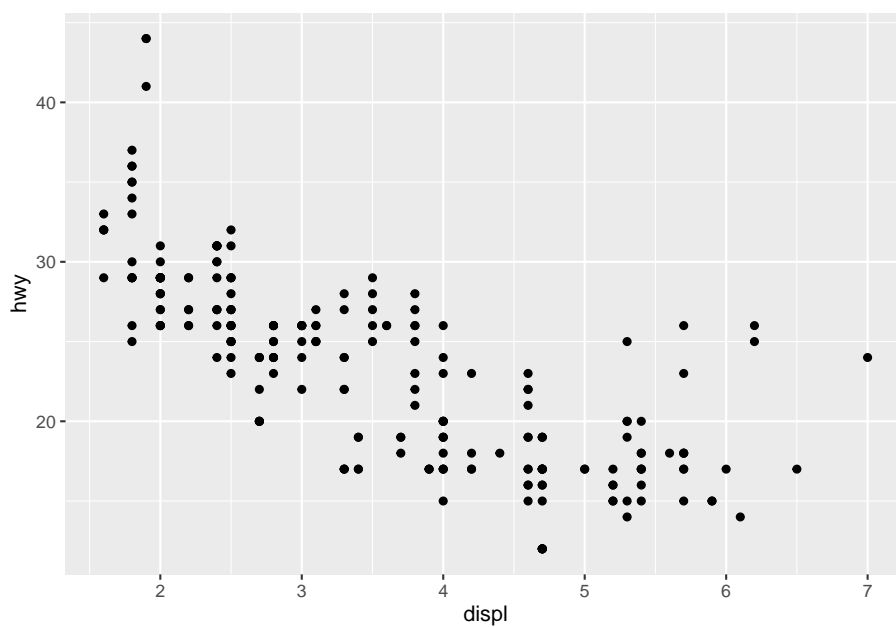
2.4 Visualización con ggplot2

El modelo básico para crear un *ggplot* tiene la forma:

```
ggplot(data = <DATA>) +  
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

Así que para emular el gráfico previo hacemos:

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

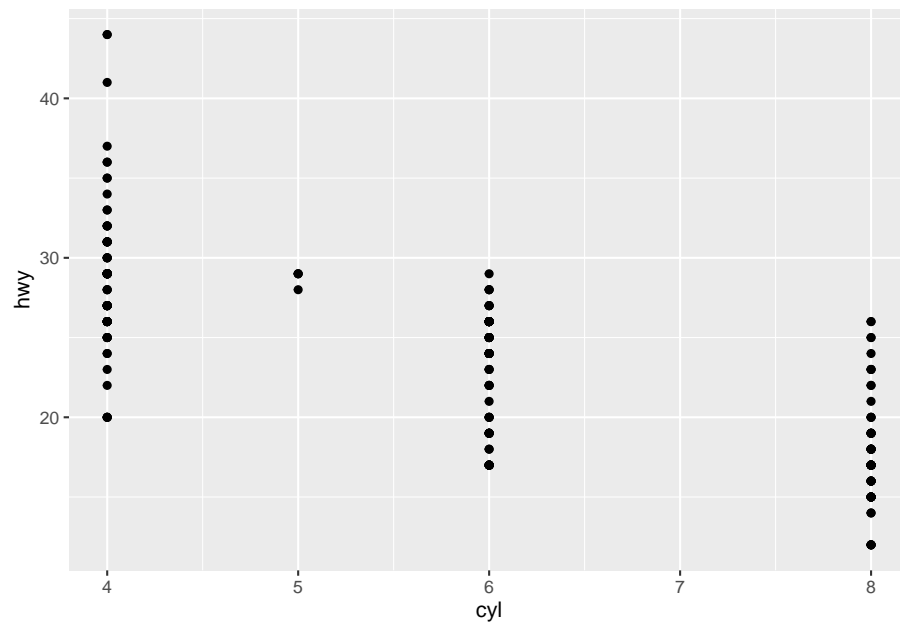


2.4.1 Ejercicios

1. Hacer el diagrama de dispersión de `hwy` vs. `cyl` ¿qué crees del gráfico obtenido?

R/

```
ggplot(mpg, aes(x = cyl, y = hwy)) +  
  geom_point()
```

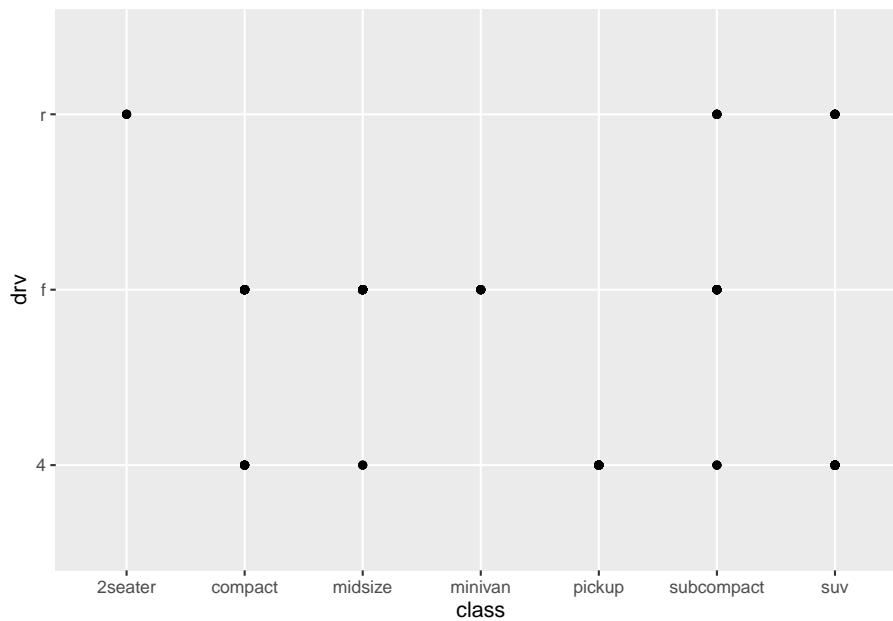


2. ¿Qué pasa si hacemos el diagrama de `class` vs. `drv`? ¿por qué crees que hay menos puntos?

R/

Ambas son categóricas, por tanto, no es un buen plot.

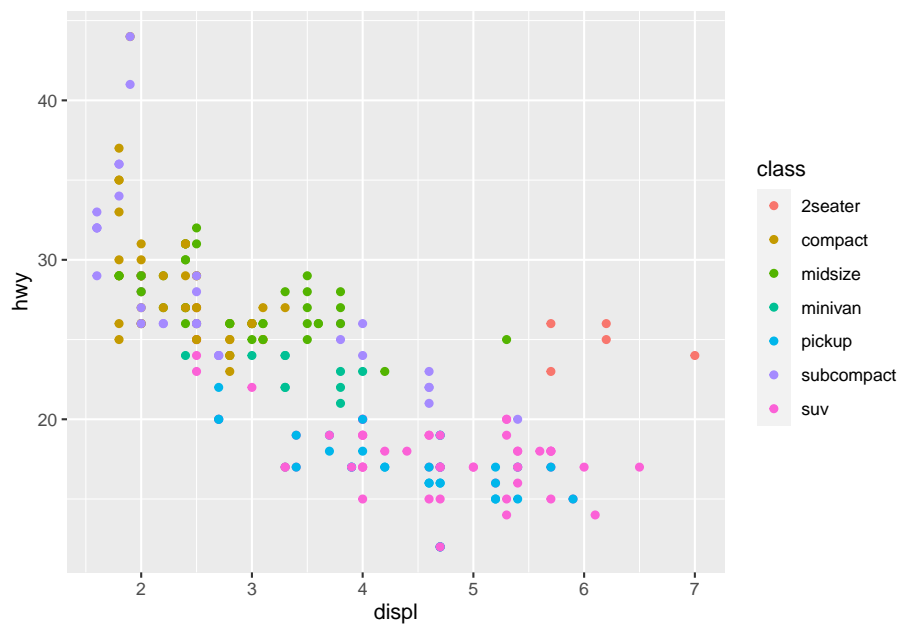
```
ggplot(mpg, aes(x = class, y = drv)) +  
  geom_point()
```



2.4.2 Cambiando la *estética*

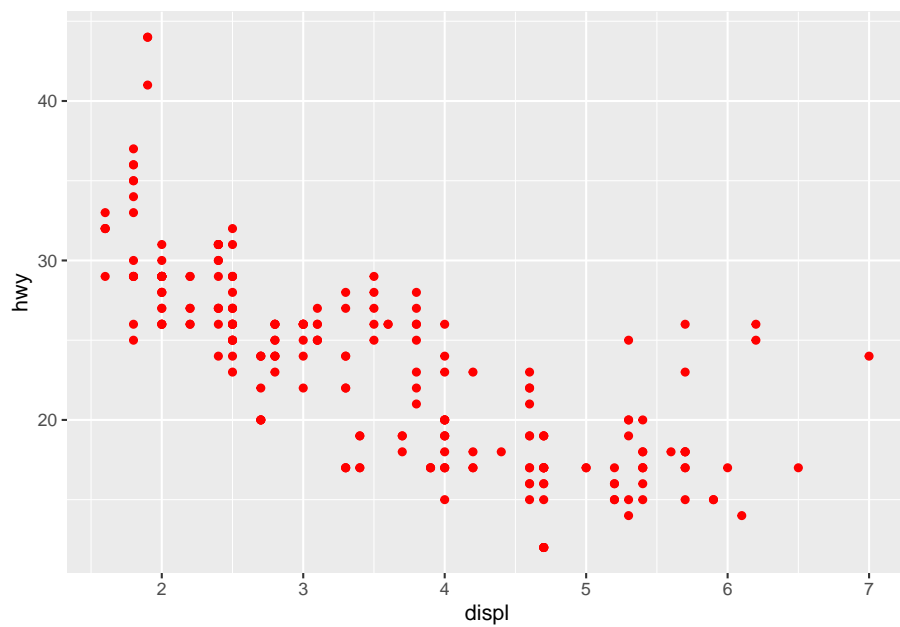
Habréis notado la instrucción `aes(x = ..., y = ...)`. Si vamos a la ayuda (presionando F1 una vez que el cursor está sobre la función deseada) notaremos que corresponde al **aesthetic mapping** de `ggplot`. Además de definir qué va en el *eje x* y qué va en el *eje y*, podemos incluir más información de los datos en nuestro plot, por ejemplo, definiendo un color, forma o tamaño diferente en función del tipo de vehículo (variable `class`). Veamos un ejemplo, asignando un color diferente para cada tipo de vehículo:

```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy, color = class))
```



Notar que para fijar las características de forma manual debemos escribimos la instrucción fuera de `aes()`:

```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy), color = "red")
```



2.4.2.1 Ejercicios

3. ¿Qué pasa si en lugar de `color`, usamos `alpha`, `shape` o `size`?

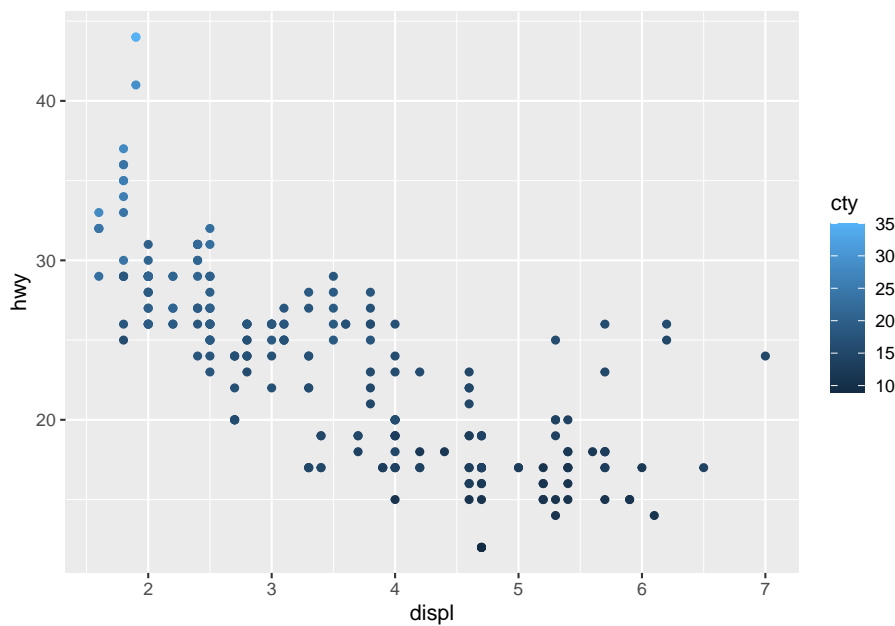
R/

Explora con algún ejemplo en el que uses `color` y cambia a `alpha`, `shape` o `size`...

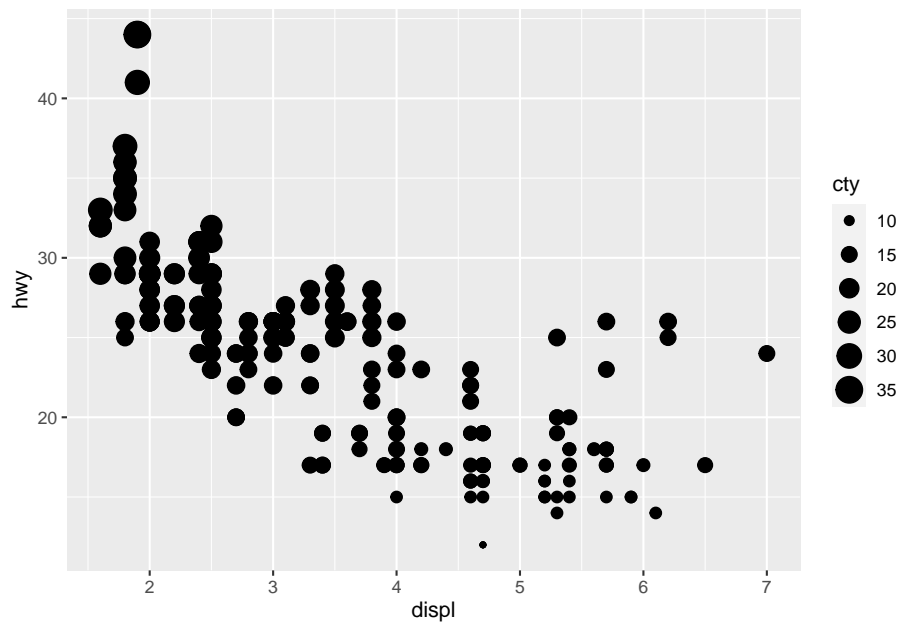
4. ¿Qué pasa al asignar una variable continua (e.g. `cty`) a `color`, `size` o `shape`? Hint: para el caso de `shape` visita <https://ggplot2.tidyverse.org/articles/ggplot2-specs.html#point-1>.

R/

```
ggplot(mpg, aes(x = displ, y = hwy, colour = cty)) +  
  geom_point()
```



```
ggplot(mpg, aes(x = displ, y = hwy, size = cty)) +  
  geom_point()
```



`shape` no funcionará porque no podemos pasarle un argumento continuo. Intenta con este ejemplo:

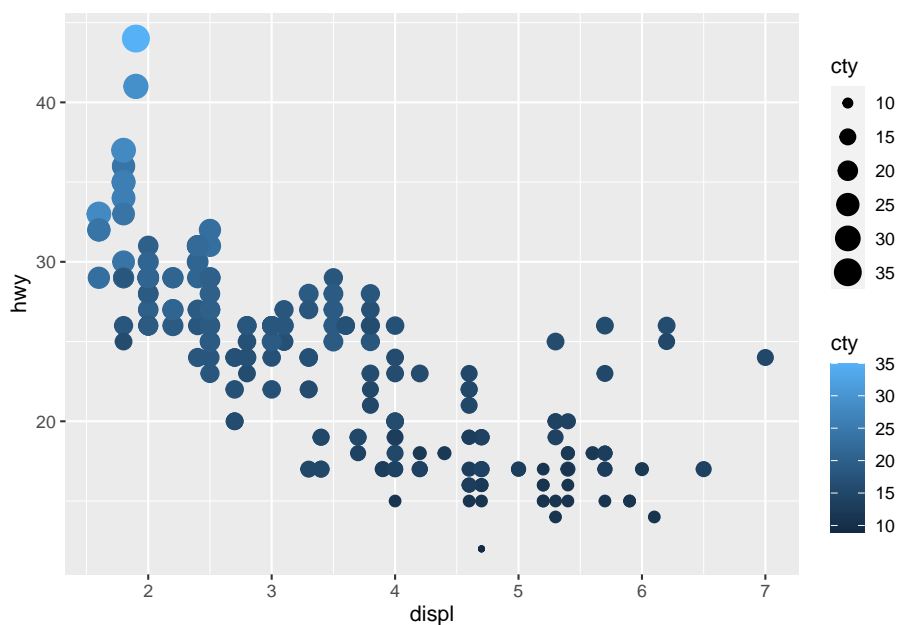
```
ggplot(mpg, aes(x = displ, y = hwy, shape = cty)) +  
  geom_point()
```

5. ¿Qué pasa si asignamos la misma variable continua (e.g. `cty`) a `color` y `size` a la vez?

R/

Simplemente estaremos construyendo un plot con información redundante:

```
ggplot(mpg, aes(x = displ, y = hwy, colour = cty, size = cty)) +  
  geom_point()
```

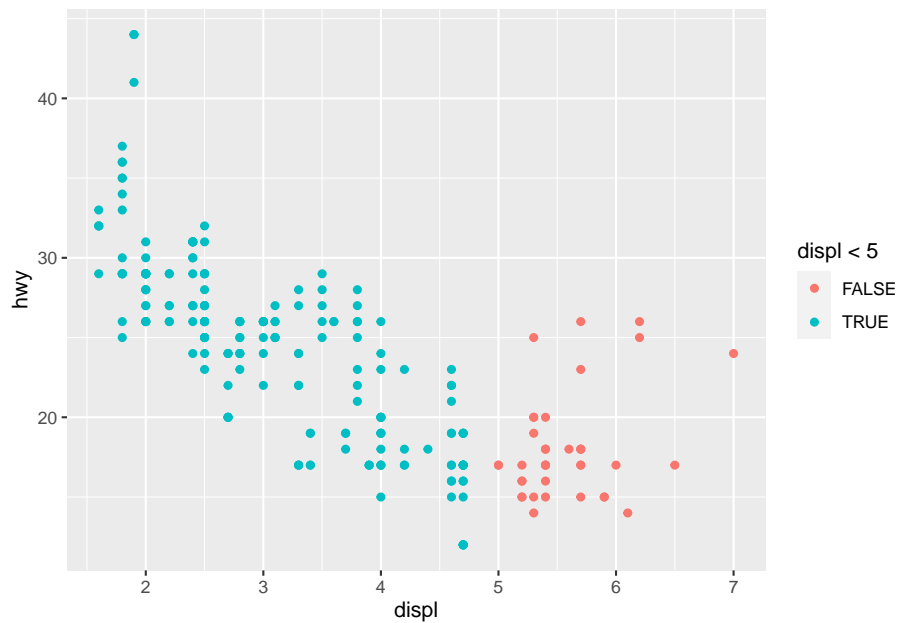
6. Visita la ayuda `?geom_point` (también https://ggplot2.tidyverse.org/reference/geom_point.html) y explora los diferentes *aesthetic* que puedes especificar.

7. Agrega al *aesthetic* de tu plot la expresión `colour = displ < 5`. Esto ya no es una variable si no una expresión que devuelve un booleano. ¿Puedes explicar el plot resultante?

R/

Es equivalente a agregar una nueva etiqueta que diferencia las observaciones que cumplen `displ < 5` y `displ >= 5`

```
ggplot(mpg, aes(x = displ, y = hwy, colour = displ < 5)) +
  geom_point()
```

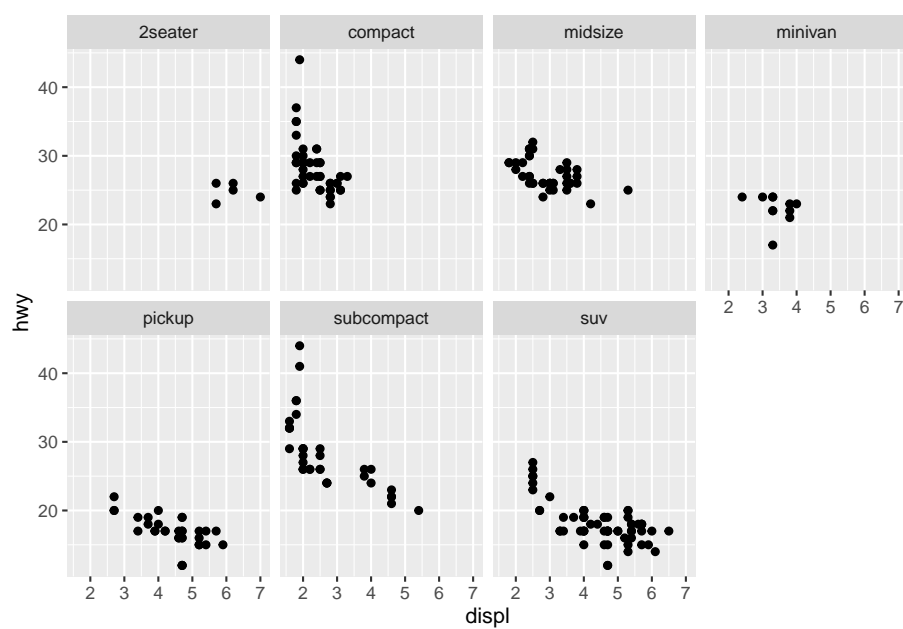


2.4.3 Las *facet*as

Habréis notado que en la sección anterior estábamos representando *3 dimensiones* (3D) en el plano (que tiene solo 2D). Con las *facet*as (**facets**) particionamos un gráfico de acuerdo a cierta (o ciertas) variables.

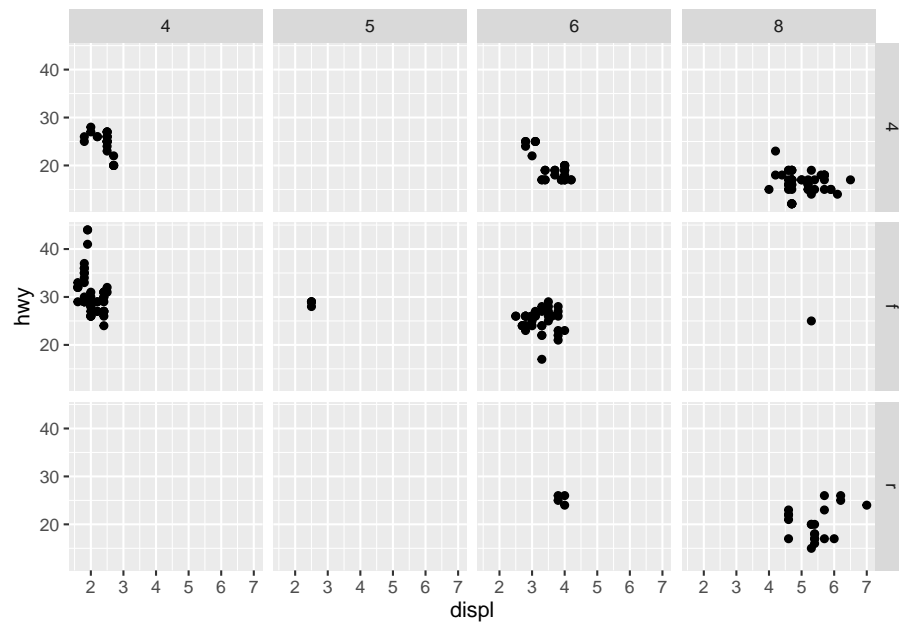
Para crear *facet*as de acuerdo a una única variable usamos `facet_wrap()`. El primer argumento será una “fórmula” de **R**. Las fórmulas son una estructura del lenguaje, formadas con el símbolo `~` y que permite relacionar variables o transformaciones de variables (i.e. sumas, logaritmos o la identidad). En este caso, debemos tener cuidado de pasar a `facet_wrap()` una variable discreta:

```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy)) +  
  facet_wrap(~ class, nrow = 2)
```



Si queremos particionar nuestro gráfico de acuerdo a una combinación de variables usamos `facet_grid`. Por ejemplo:

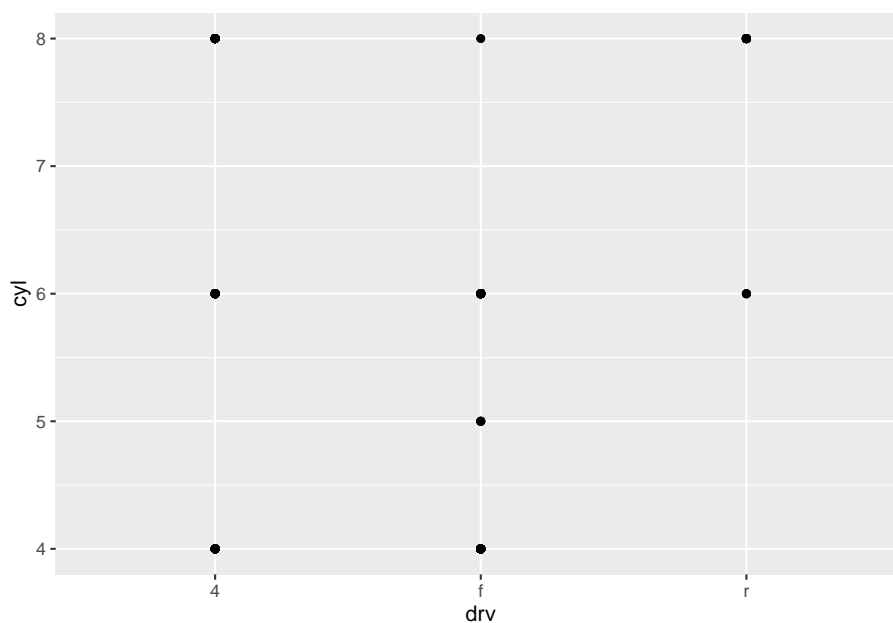
```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy)) +  
  facet_grid(drv ~ cyl)
```



2.4.3.1 Ejercicios

8. ¿Qué hemos hecho en el gráfico de arriba? ¿Por qué hay facetas vacías?
Hint: intenta relacionar tus impresiones con el siguiente gráfico:

```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = drv, y = cyl))
```



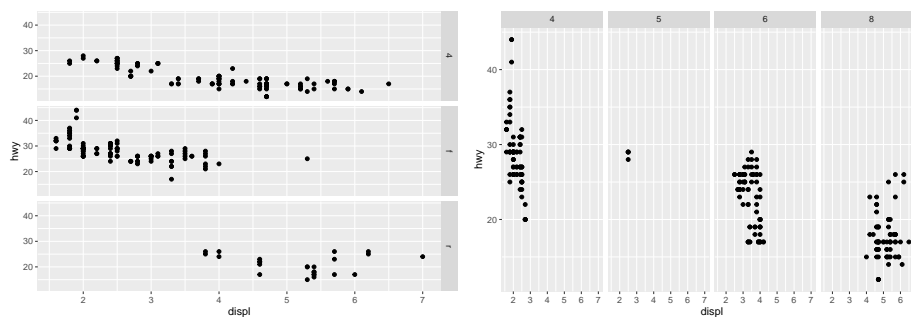
R/

No hay observaciones con todas las combinaciones de posibles niveles de ambas variables categóricas.

9. Explica el uso del punto `.` en los siguientes plots:

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  facet_grid(drv ~ .)
```

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  facet_grid(. ~ cyl)
```

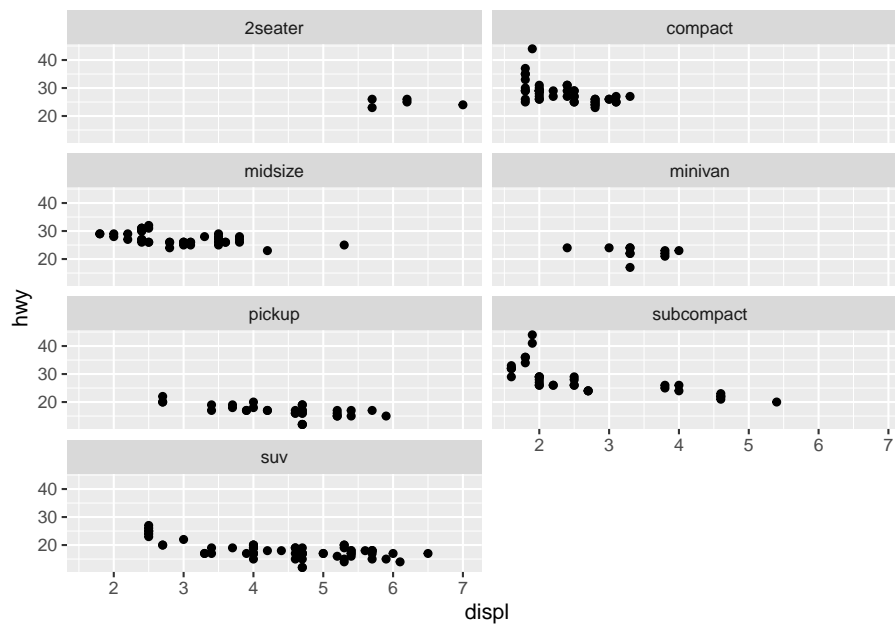


R/

El primero arregla los plots por filas y el segundo por columnas.

10. ¿Para qué sirven los argumentos `nrow` y `ncol`? ¿En qué tipo de facetas se pueden usar? Explora la ayuda `facet_wrap` y `facet_grid` o el manual en <https://ggplot2.tidyverse.org/reference/index.html>.

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  facet_wrap(~class, nrow = 4)
```

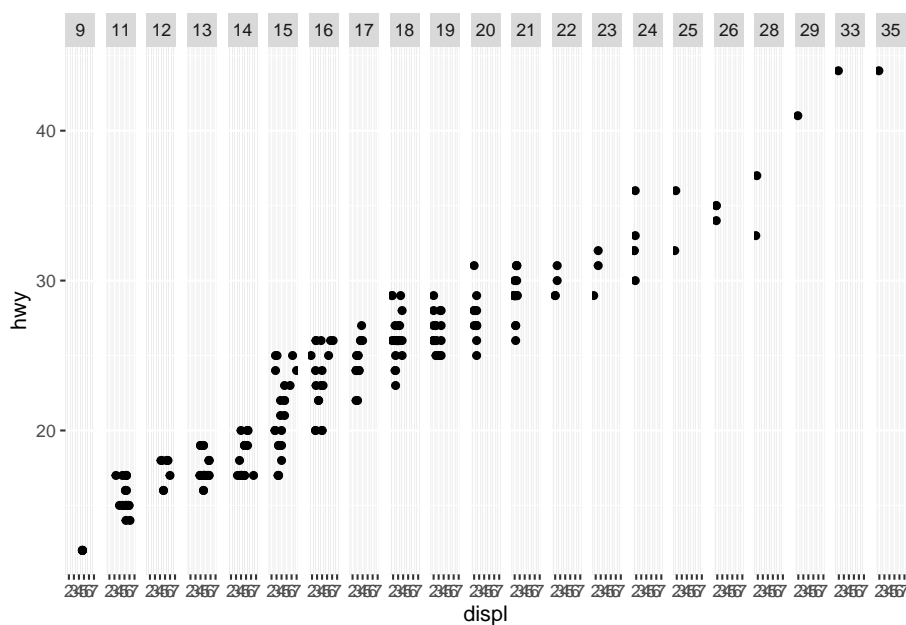


R/

Ajusta el número de filas en las que arreglamos los plots.

11. ¿Qué pasa si usamos una variable continua para hacer facetas? Intenta hacerlo con `cty`.

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  facet_grid(. ~ cty)
```



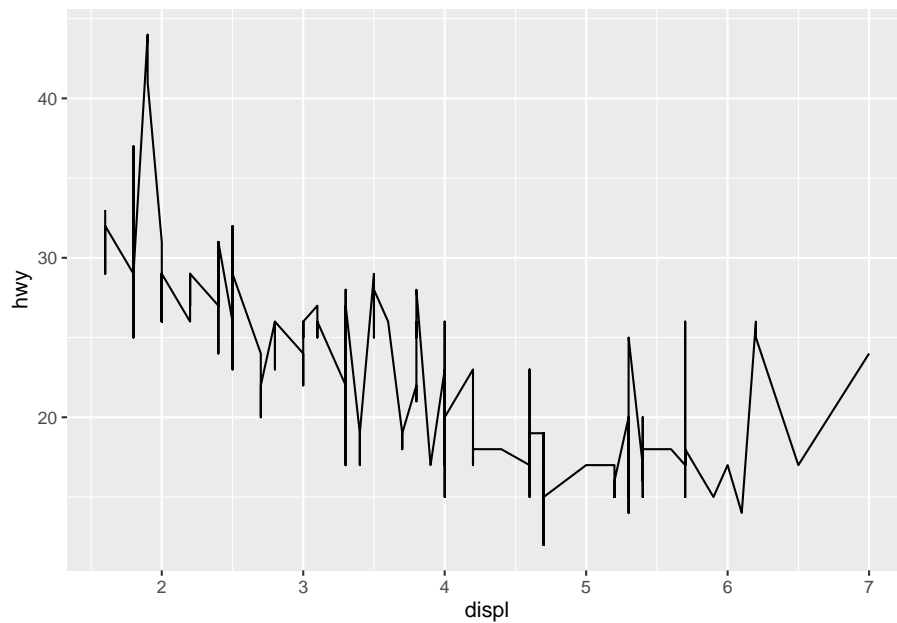
R/

No es muy útil tener tantos paneles. En este caso, los plots son medianamente interpretables porque la variable `cty` tiene pocos valores únicos.

2.4.4 Objetos geométricos geoms

Hasta ahora solo hemos hecho diagramas de dispersión usando `geom_point`. En `ggplot` es muy sencillo cambiar el tipo de gráfico cambiando a otro **geom** (*objeto geométrico*). Aún así, los argumentos de cada **geom** pueden variar un poco (parecido a lo que pasa entre `facet_wrap` y `facet_grid`). Por ejemplo, si en lugar de un diagrama de dispersión quisiéramos un gráfico de líneas:

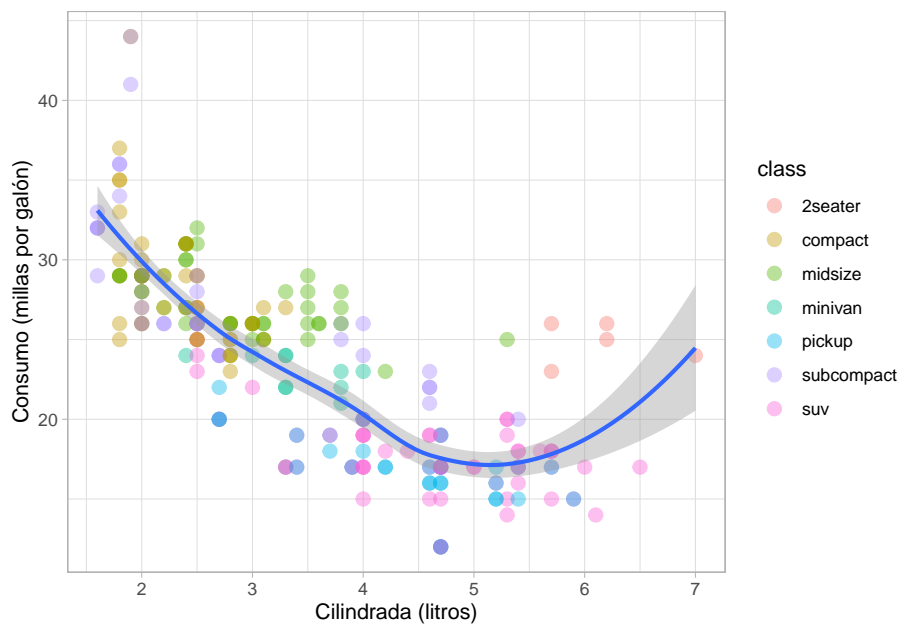
```
ggplot(data = mpg) +
  geom_line(mapping = aes(x = displ, y = hwy))
```



Este gráfico no es muy útil (además de ser estéticamente horrible). Sin embargo tanto este como el diagrama de dispersión parecen indicar que a mayor cilindrada (`displ`) mayor consumo (menor cantidad de millas autopista por galón `hwy`), excepto para algunos vehículos de gran cilindrada (los puntos más a la derecha). Sin dudas, debe haber una “curva suave” que pueda describir esta relación entre `hwy` y `displ`... así que es un buen momento para echarle un ojo a “la chuleta” ([Cheatsheet](#)) del paquete `ggplot`. Te adelanto que la curva se puede estimar con `geom_smooth`:

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, colour = class), alpha = 0.4, size = 3) +
  geom_smooth(mapping = aes(x = displ, y = hwy)) +
  xlab('Cilindrada (litros)') +
  ylab('Consumo (millas por galón)') +
  theme_light()
```

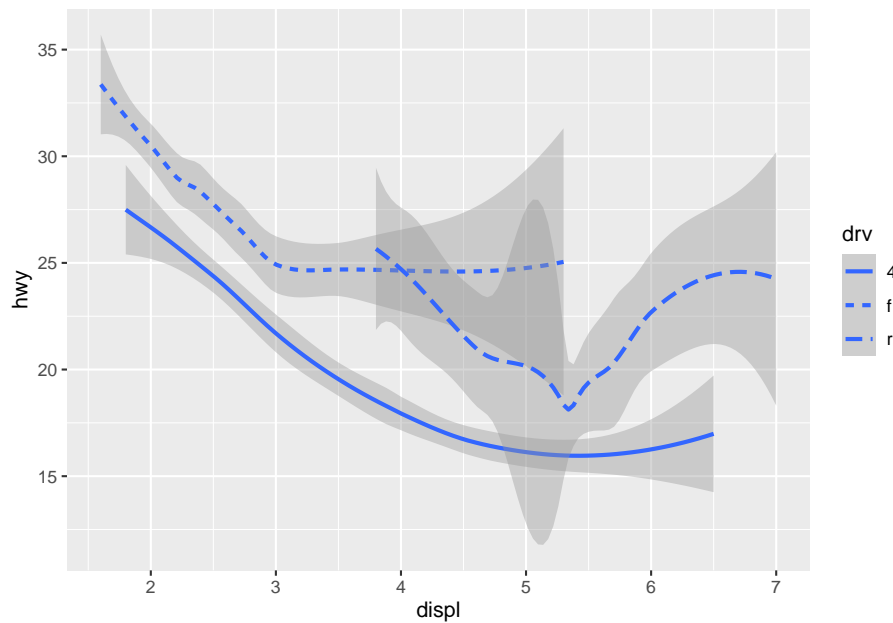
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

En este caso ya hemos añadido dos diferentes **geoms** a un mismo plot, además hemos modificado los nombres de los ejes, hemos modificado un poco la estética de los puntos y hemos usado un “tema” (**theme**) con fondo blanco. Aún así, los tipos de vehículos son muchos y es complicado establecer una relación entre el tipo de vehículo y la monotonía de la curva suave. Vamos a ver qué pasa si hacemos el “suavizado” según el tipo de tracción (**drv**):

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

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

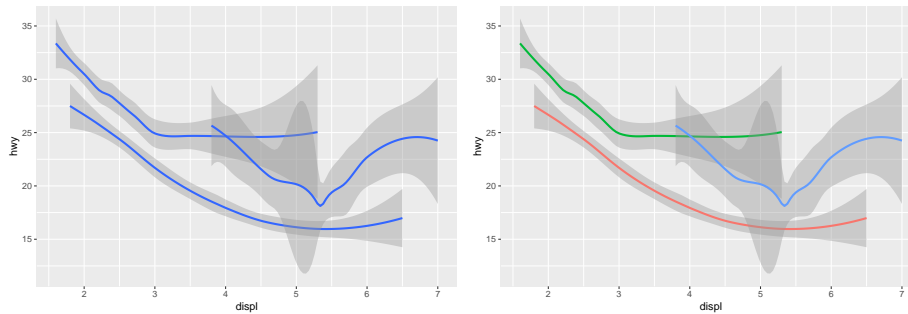


Fíjate que ahora estamos describiendo la relación entre cilindrada y consumo 3 curvas suaves que corresponden al tipo de tracción (*r*: rear/trasera, *f*: front/delantero y *4*: ambos ejes delantero y trasero). Hemos usado `linetype` para diferenciar la estética de las 3 clases que describe `drv`... si tienes dudas consulta los argumentos estéticos de `geom_smooth` https://ggplot2.tidyverse.org/reference/geom_smooth.html#aesthetics. Si usamos, por ejemplo, `group` o `color`:

```
# suavizar de acuerdo a los niveles de 'drv'
# agrupa, pero no diferencia con colores o tipos de línea
ggplot(data = mpg) +
  geom_smooth(mapping = aes(x = displ, y = hwy, group = drv))
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
# suavizar de acuerdo a los niveles de 'drv'
# agrupa y diferencia con colores
ggplot(data = mpg) +
  geom_smooth(
    mapping = aes(x = displ, y = hwy, color = drv),
    show.legend = FALSE
  )
```

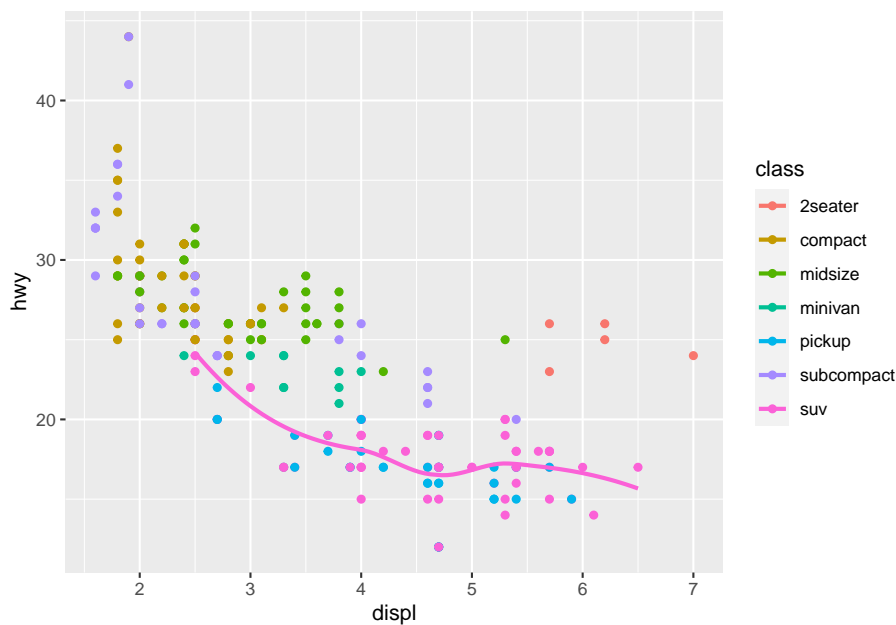
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Podemos además ir un poco más lejos e intentar hacer el suavizado (estimar la curva suave) para un tipo de vehículo determinado (de acuerdo a los niveles de `class`). Por ejemplo, en el caso de vehículos `suv`:

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

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



2.4.4.1 Ejercicios

12. ¿Cuál será la diferencia entre estos dos gráficos?

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

R/

Ninguna diferencia, aunque podemos hacerlo más legible:

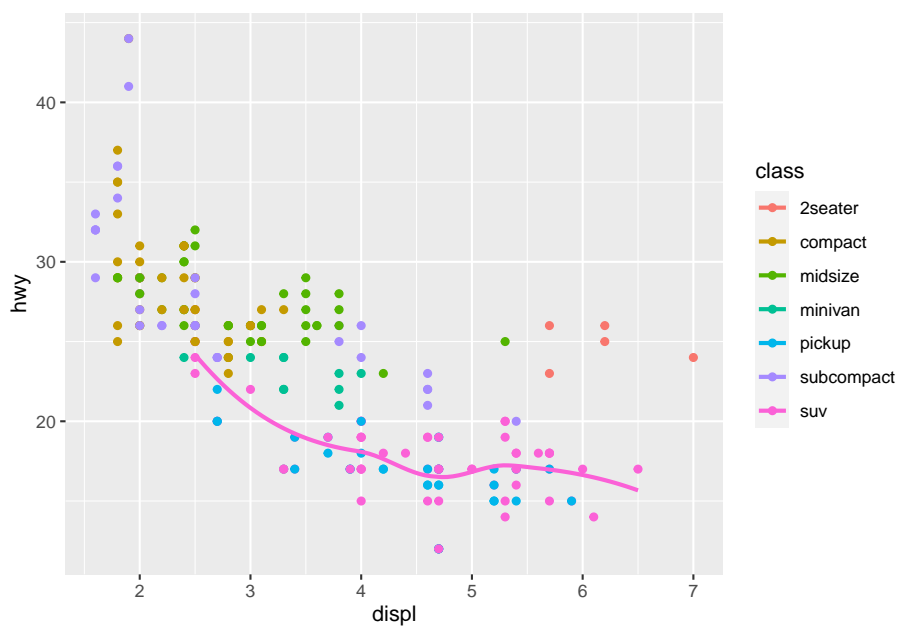
```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +  
  geom_point() +  
  geom_smooth()
```

De acuerdo a tus impresiones, reescribe el código que hace el suavizado solo para los vehículos `suv`. El objetivo es lograr un código legible y sin argumentos innecesarios. ¿Qué produce la instrucción `se = FALSE`?

R/

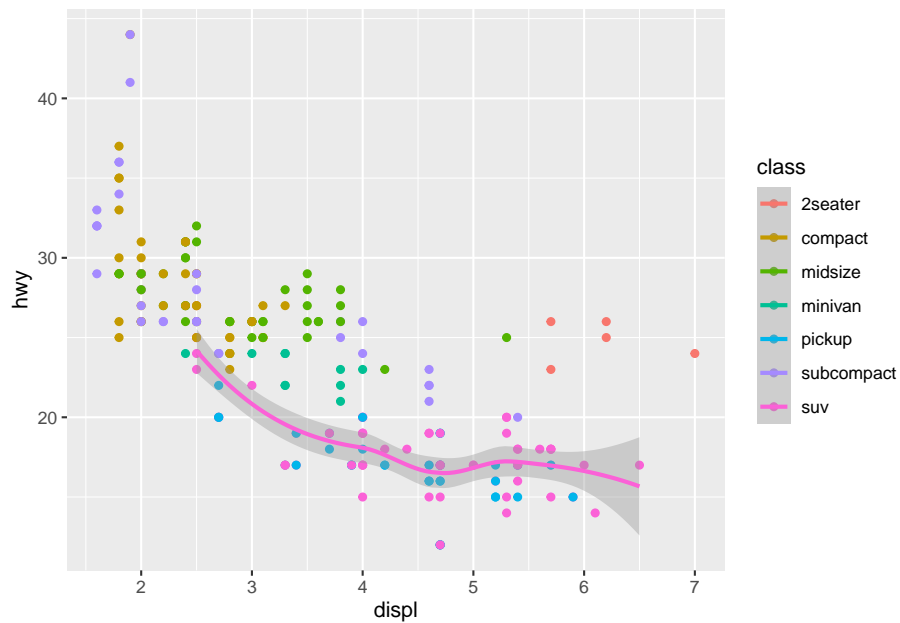
```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy, color = class)) +  
  geom_point() +  
  geom_smooth(data = filter(mpg, class == "suv"),  
              se = FALSE)
```

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



```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy, color = class)) +
  geom_point() +
  geom_smooth(data = filter(mpg, class == "suv"),
              se = TRUE)
```

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



13. Reproducir los siguientes gráficos:

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

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

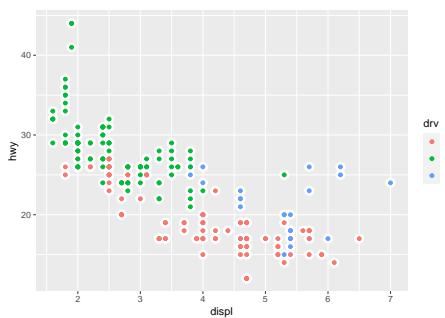
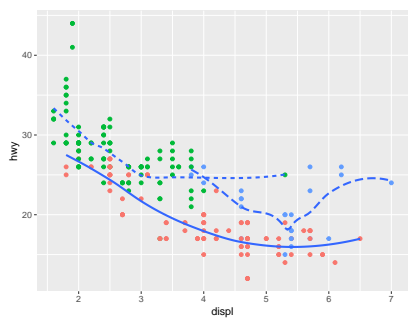
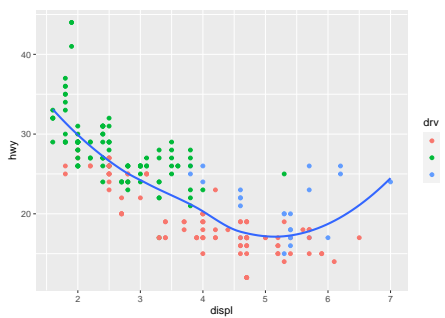
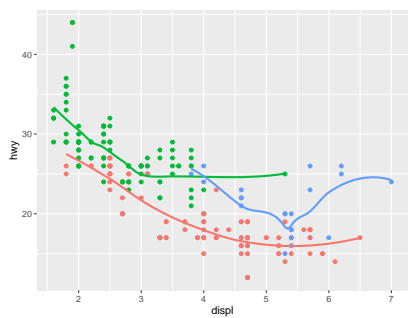
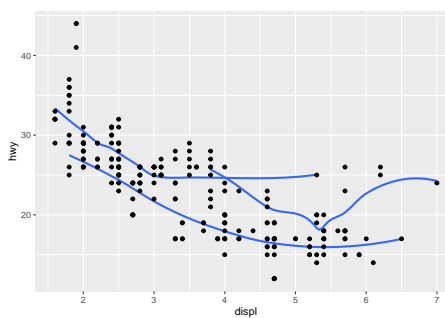
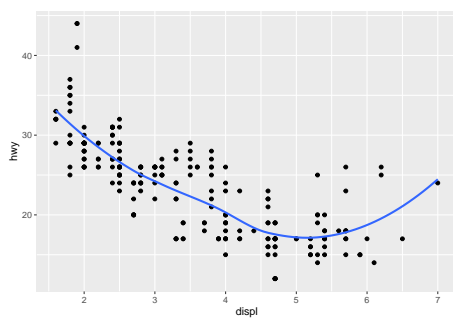
ggplot(mpg, aes(x = displ, y = hwy, colour = drv)) +
  geom_point() +
  geom_smooth(se = FALSE)

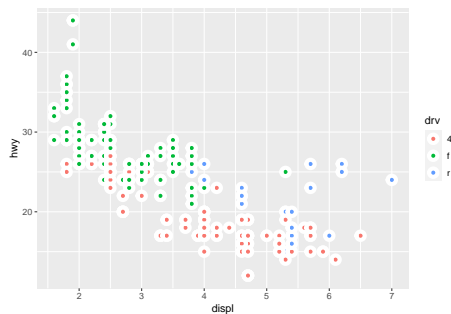
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(aes(colour = drv)) +
  geom_smooth(se = FALSE)

ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(aes(colour = drv)) +
  geom_smooth(aes(linetype = drv), se = FALSE)

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

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(aes(fill = drv),
            shape = 21,
            colour = "white",
            size = 2,
            stroke = 3)
```





R/

Códigos incluidos con los gráficos.

2.4.5 Transformaciones estadísticas

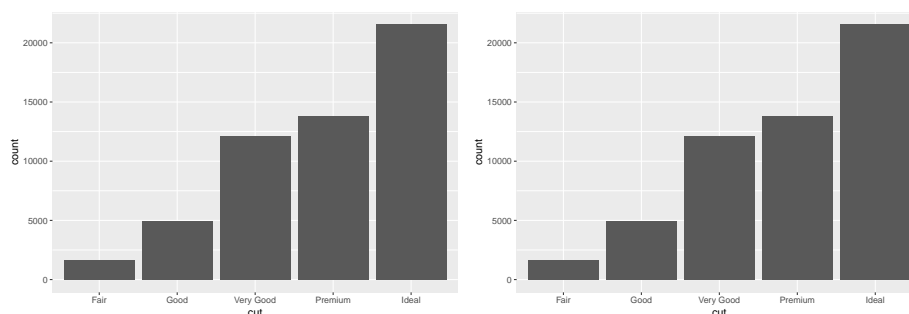
En la sección anterior `ggplot` hizo algunas transformaciones por nosotros. Está claro que la “curva suave” con la que hemos trabajado no forma parte de `mpg`, sino que es una estimación a partir de una regresión lineal, local (loess) o un spline. De hecho, muchos de los **geoms** de `ggplot` hacen transformaciones estadísticas por nosotros:

- los gráficos de barras, histogramas y polígonos de frecuencia construyen intervalos (*bins*) y cuentan el número de observaciones que “caen” dentro de estos;
- los *smoothers* (¿suavizadores? :) como ya hemos visto;
- los diagramas de cajas (*boxplots*) calculan estadísticos importantes para entender la distribución de cierta variable continua (mediana, media, cuartiles, *outliers*).

Vamos a hacer algunos diagramas de barras con los datos `diamonds`:

```
# geom_bar tiene a stat_count como el "stat" por defecto:
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut))

# stat_count tiene a geom_bar como el "geom" por defecto:
ggplot(data = diamonds) +
  stat_count(mapping = aes(x = cut))
```

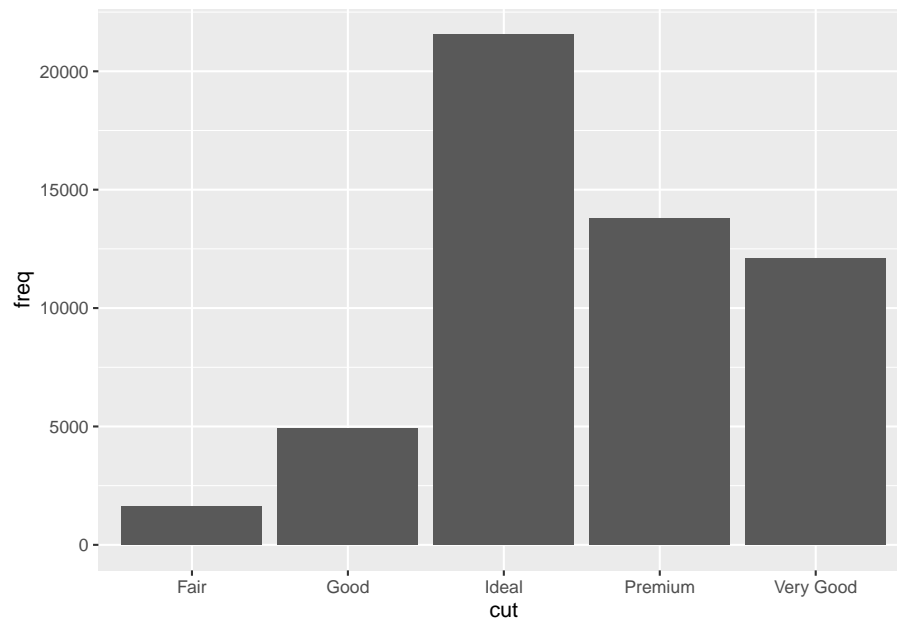



También lo podemos hacer “a mano” si contamos los elementos de cada clase y cambiamos el **stat** por defecto de **geom_bar**:

```
summary(diamonds$cut)
```

```
##      Fair      Good Very Good   Premium    Ideal
##      1610     4906    12082    13791    21551
mi_df <- data.frame( cut = c("Fair", "Good", "Very Good", "Premium", "Ideal"),
                     freq = c(1610, 4906, 12082, 13791, 21551) )
print(mi_df)
```

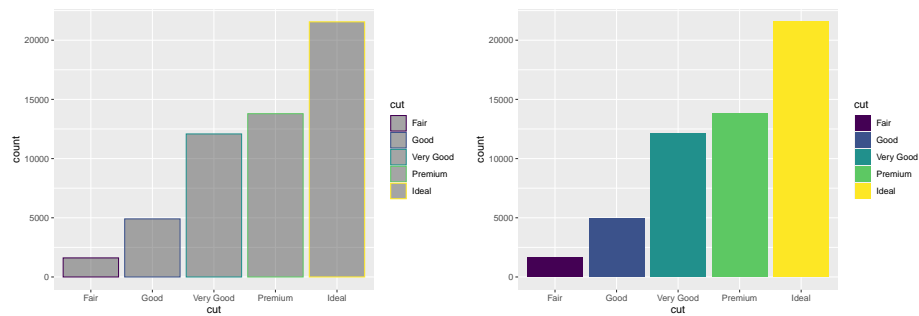
```
##      cut  freq
## 1  Fair  1610
## 2  Good  4906
## 3 Very Good 12082
## 4  Premium 13791
## 5  Ideal 21551
ggplot(data = mi_df) +
  geom_bar(mapping = aes(x = cut, y = freq), stat = "identity")
```



Notarás que son los mismos diagramas de barras, salvo por el orden de los cortes (ahora están ordenados alfabéticamente). Esto se podría arreglar convirtiendo `mi_df$cut` a clase `factor` y reordenando los niveles... pero ya lo veremos luego :)

Podemos también añadir color, modificar la transparencia de las barras, etc.:

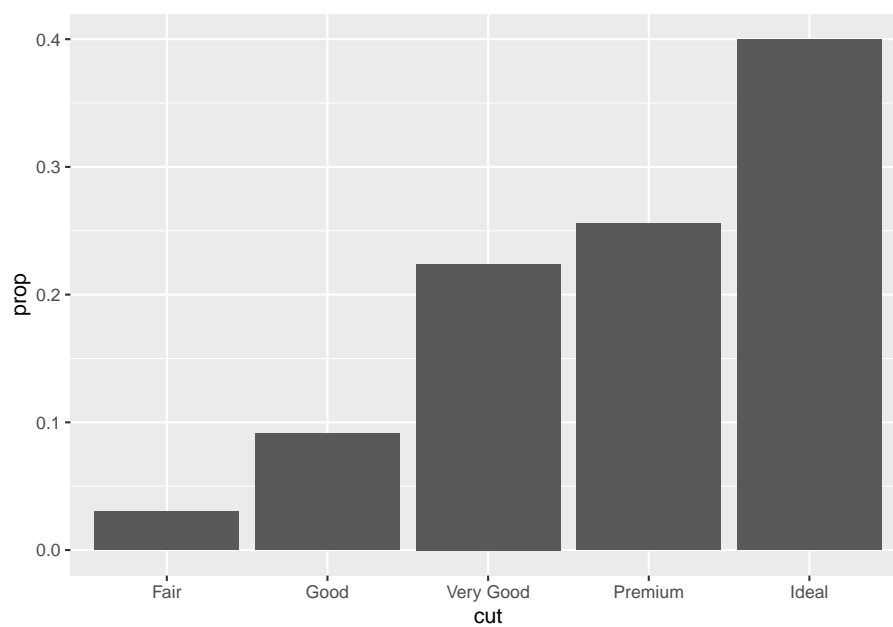
```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, colour = cut), alpha = 0.5)
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = cut))
```



2.4.5.1 Ejercicios

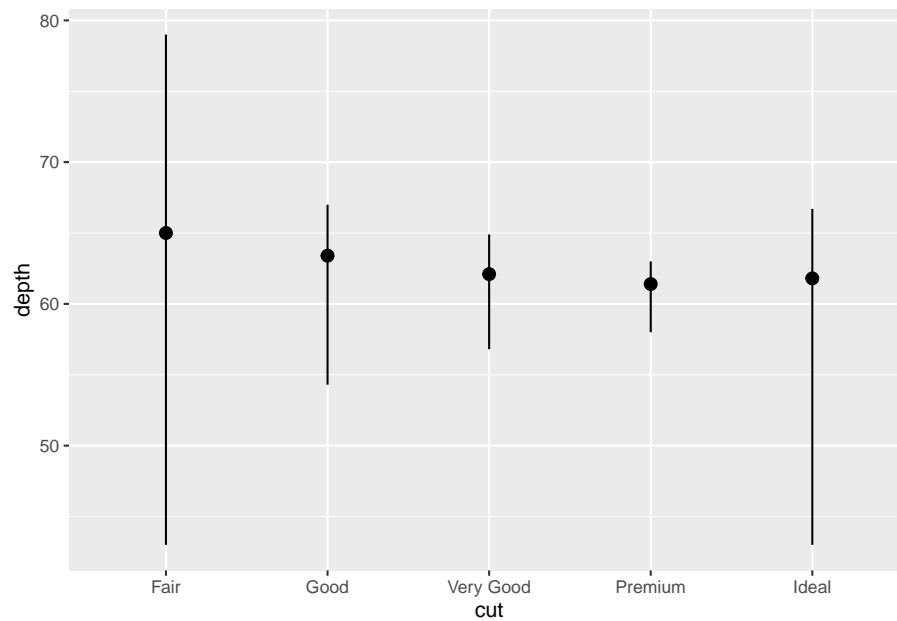
14. ¿Qué hace el siguiente código?

```
ggplot(data = diamonds) +  
  geom_bar(mapping = aes(x = cut, y = stat(prop), group = 1))
```



15. Interpreta los resultados de ejecutar:

```
ggplot(data = diamonds) +  
  stat_summary(  
    mapping = aes(x = cut, y = depth),  
    fun.min = min,  
    fun.max = max,  
    fun = median  
  )
```



R/

Te muestran la profundidad mínima, máxima y la mediana. No es tan útil como un diagrama de cajas (boxplot).

16. ¿Cuál es la diferencia entre `geom_bar` y `geom_col`? ¿Qué datos necesitaríamos introducir en cada función para obtener el mismo diagrama de barras en cada caso?

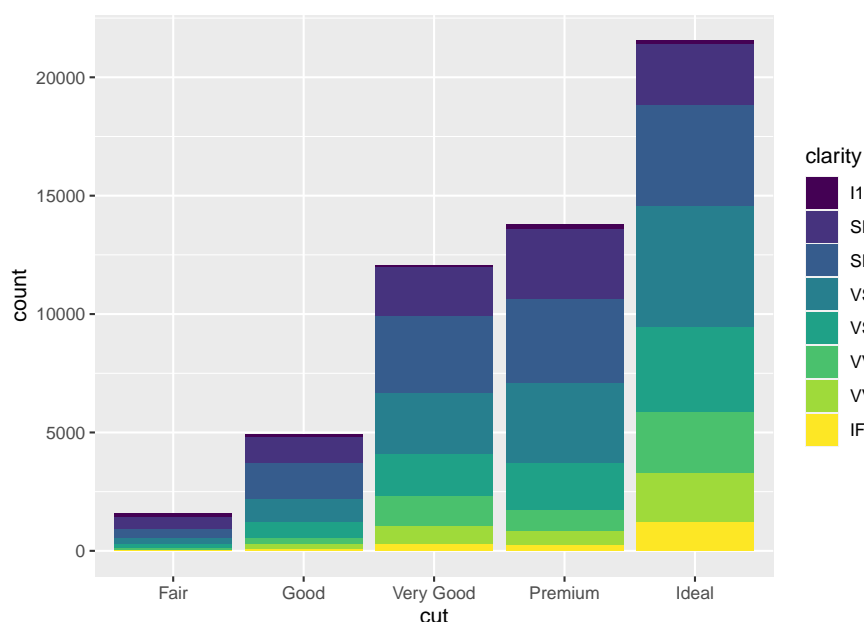
R/

- `geom_col()` tiene un “stat” diferente a `geom_bar()`
- El stat por defecto de `geom_col()` es `stat_identity()`
- El stat por defecto de `geom_bar()` es `stat_count()`

2.4.6 Ajuste de posición y sistemas de coordenadas

Los diagramas de barras también permiten añadir una tercera variable (además de la frecuencia en el *eje y* y la clase correspondiente en el *eje x*), como ya hemos hecho con los diagramas de dispersión. Por ejemplo si utilizamos la variable `clarity` para “rellenar” las barras:

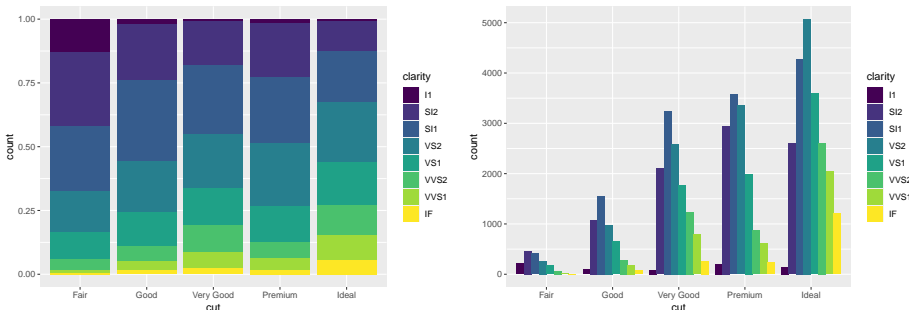
```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = clarity))
```



Si variamos el parámetro de posición (**position adjustment**) podemos hacer más fácil la comparación de acuerdo a la claridad de los diamantes (variable clarity):

```
# todas las barras iguales para comparar proporciones de claridad
# de acuerdo al tipo de corte:
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = clarity), position = "fill")

# frecuencias por tipo de corte y claridad:
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = clarity), position = "dodge")
```



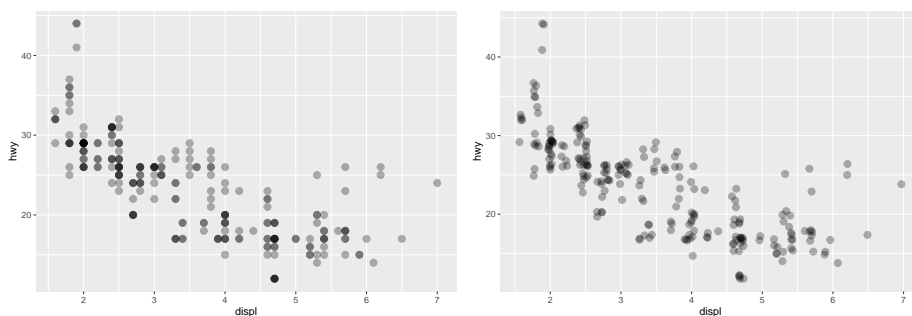
Hay otro tipo de ajuste (`position = "jitter"`) que no tiene utilidad para los diagramas de barra, pero sí para los diagramas de dispersión y de cajas

(*boxplots*). Por ejemplo, en el caso de los datos `mpg` es muy difícil notar que muchos de los puntos del diagrama `hwy` vs. `displ` están superpuestos. Con `jitter` podemos añadir un poco de “ruido” a las observaciones para que así los puntos del diagrama aparezcan más dispersos y así tener una idea más acertada del tamaño muestral:

```
p <- ggplot(data = mpg, mapping = aes(x = displ, y = hwy))

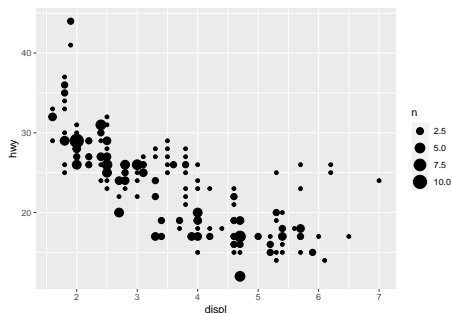
# con algo transparencia los superpuestos producen un color oscuro:
p + geom_point(alpha = 0.3, size = 3)

# dispersamos con "jitter":
p + geom_point(alpha = 0.3, size = 3, position = "jitter")
```



Otra opción es usar `geom_count` para “contar” los puntos solapados:

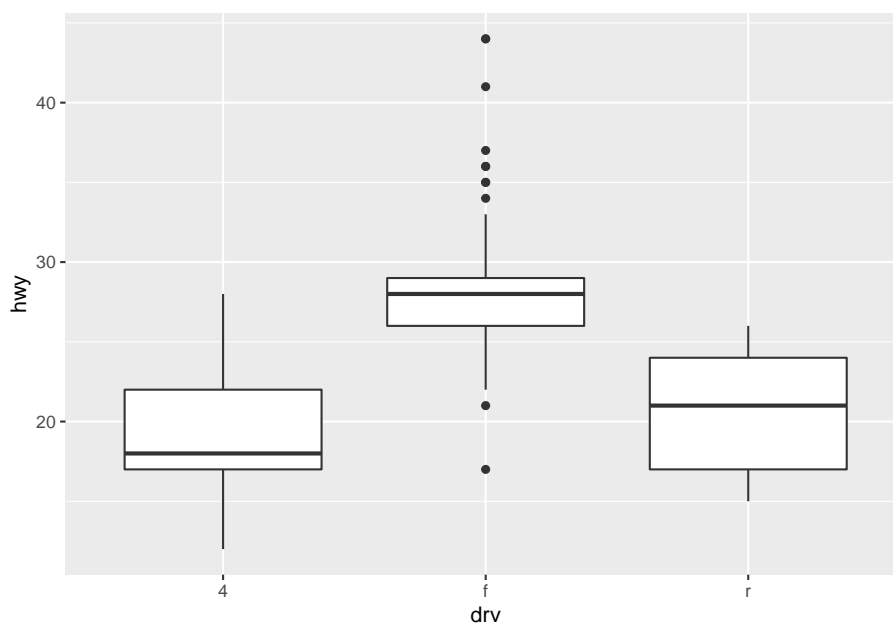
```
p + geom_point() +
  geom_count()
```



2.4.6.1 Ejercicios

17. Los diagramas de cajas se logran con `geom_boxplot`. Este tipo de gráficos permiten comparar las distribuciones de una variable continua para diferentes grupos o clases. Por ejemplo, para los datos `mpg` podemos comparar la distribución del consumo de acuerdo al tipo de tracción:

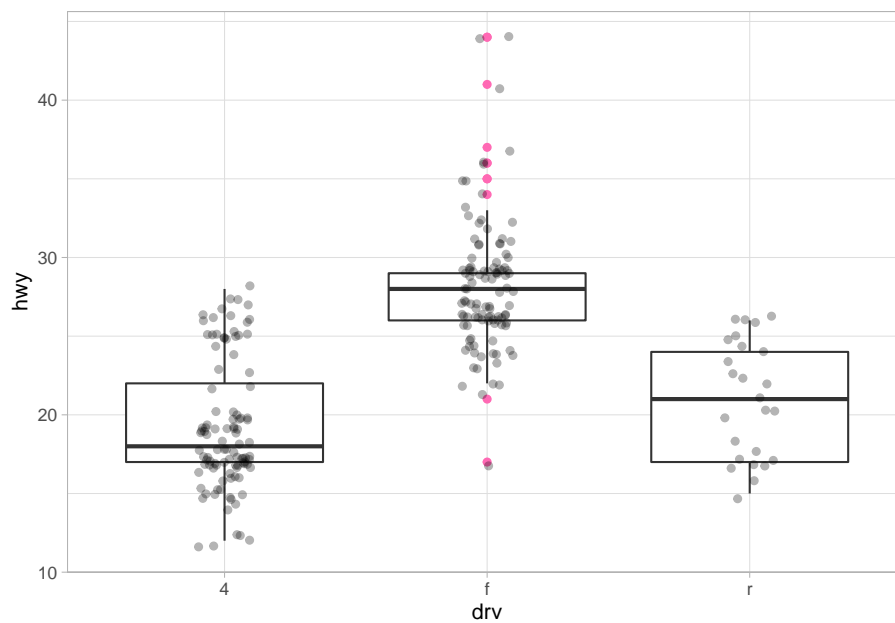
```
p_box <- ggplot(data = mpg, aes(x = drv, y = hwy)) +  
  geom_boxplot()  
p_box
```



Consulta la ayuda de `geom_jitter` e incluye las observaciones como puntos superpuestos al diagrama de cajas. Deberías obtener algo como esto:

R/

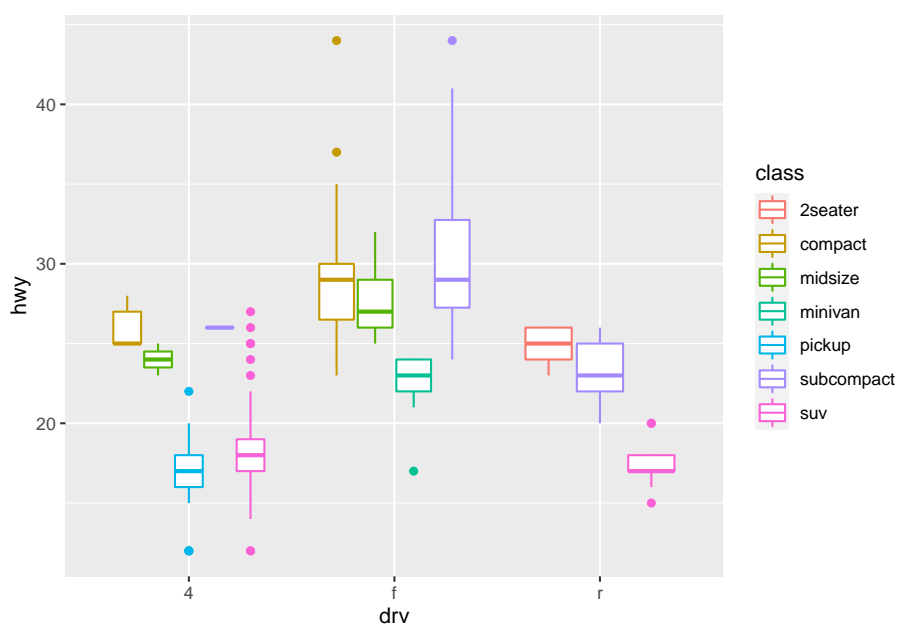
```
ggplot(data = mpg, aes(x = drv, y = hwy)) +  
  geom_boxplot(outlier.colour = "hotpink") +  
  geom_jitter(alpha = 0.3, width = 0.1) +  
  theme_light()
```



18. Modifica `p_box` para que represente también la información relativa al tipo de vehículo (variable `class`). ¿Puedes identificar el *ajuste de posición* por defecto de `geom_boxplot`?

R/

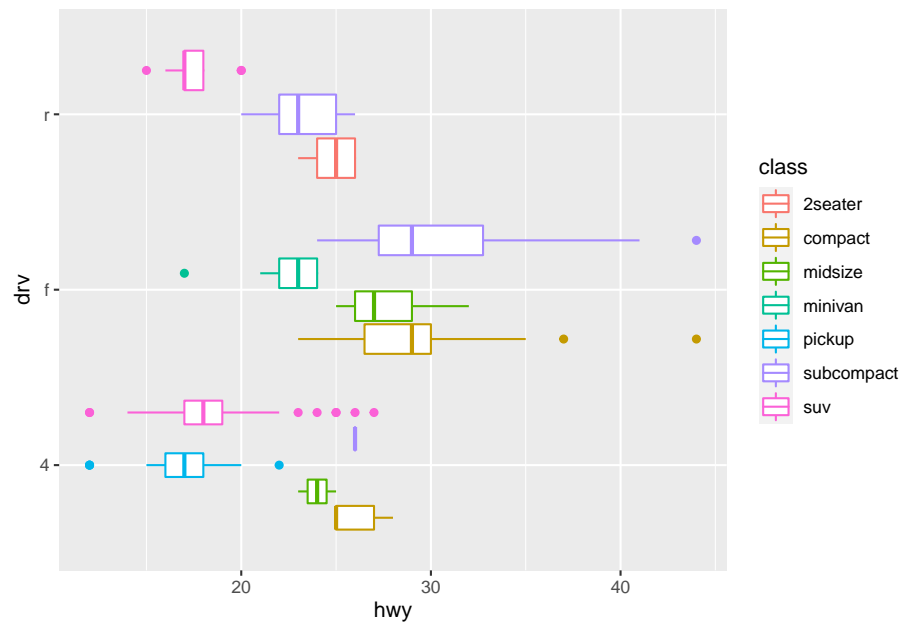
```
p_box2 <- ggplot(data = mpg, aes(x = drv, y = hwy, color = class)) +
  geom_boxplot()
p_box2
```

Es “dodge2”, que es un shortcut para “position_dodge2”. Lo que hace es mover las cajas en la horizontal (sin afectar la vertical), para evitar el solapamiento de cajas.

19. Cambia la orientación de los diagramas de cajas de verticales a horizontales.
Hint: consulta la documentación de `coord_flip`.

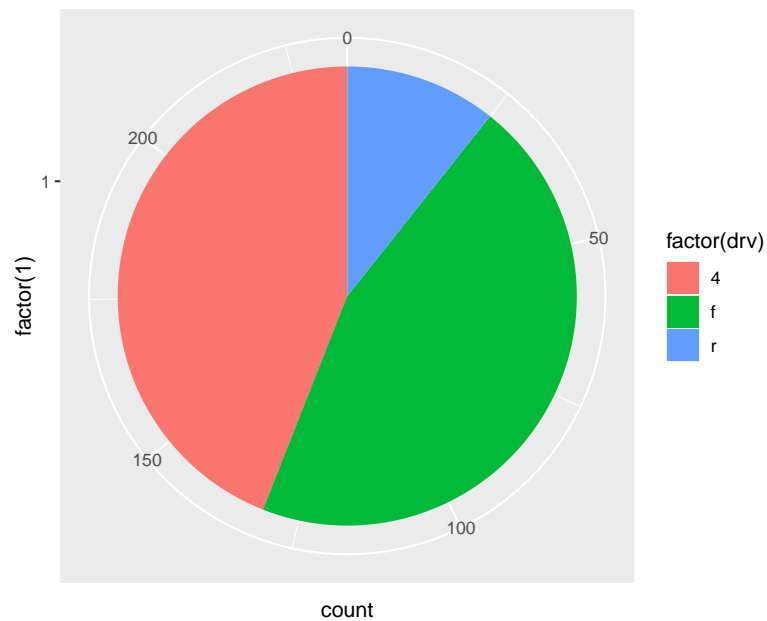
```
R/
p_box2 <- ggplot(data = mpg, aes(x = drv, y = hwy, color = class)) +
  geom_boxplot() + coord_flip()
p_box2
```



20. Consulta la documentación de `coord_polar` y construye un diagrama circular (“de pastel”) de la variable tipo de tracción (`drv`).

R/

```
pie <- ggplot(mpg, aes(x = factor(1), fill = factor(drv))) +  
  geom_bar(width = 1)  
pie + coord_polar(theta = "y")
```



2.5 Resumen

En las secciones anteriores has asimilado la “gramática estratificada de los gráficos” (*The layered grammar of graphics*) de `ggplot`. Aunque no lo parezca ahora, ya eres capaz de construir cualquier tipo de gráfico en 2D. Resumiendo, dispones de un modelo con 7 parámetros a definir (no necesitas definirlos todos) y tantas capas de **geoms** como necesites:

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION> (
    mapping = aes(<MAPPINGS>),
    stat = <STAT>,
    position = <POSITION>
  ) +
  <COORDINATE_FUNCTION> +
  <FACET_FUNCTION>
```

Finalmente, podemos añadir otros 2 parámetros a este modelo que te permitirán modificar otros elementos necesarios a la hora de “comunicar” con tus gráficos (título, leyenda, etiquetado de los ejes, escala de los ejes, etc.):

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION> (
    mapping = aes(<MAPPINGS>),
    stat = <STAT>,
```

```
    position = <POSITION>
  ) +
  <COORDINATE_FUNCTION> +
  <FACET_FUNCTION> +
  <SCALE_FUNCTION> +
  <THEME_FUNCTION>
```

2.5.0.1 Ejercicios

21. Cambia la escala y tema de algunos de los gráficos que has desarrollado.
Hint: en el *Cheatsheet: Data Visualization with ggplot2* tienes un resumen muy completo de las herramientas que necesitas.

Chapter 3

Transformaciones

3.1 Datos

Vamos a trabajar con el **data frame** `nycflights13::flights`. Una vez más ten en cuenta los “conflictos” y asegúrate de usar la función correcta (`paquete_correcto::fun_repetida(...)`).

```
library(nycflights13)
library(tidyverse)
```

```
flights
```

```
## # A tibble: 336,776 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 336,766 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 <dtm>
```

3.1.0.1 Ejercicios

1. ¿Puedes identificar los tipos de variables?

R/

Al imprimir un tibble en la consola, se muestra el tipo de variable debajo de cada columna: <int>, <dbl>, <chr>, etc.

2. ¿Qué información puedes extraer de los datos con la función `summary()`?

R/

```
summary(flights)
```

##	year	month	day	dep_time	sched_dep_time
##	Min. :2013	Min. : 1.000	Min. : 1.00	Min. : 1	Min. : 106
##	1st Qu.:2013	1st Qu.: 4.000	1st Qu.: 8.00	1st Qu.: 907	1st Qu.: 906
##	Median :2013	Median : 7.000	Median :16.00	Median :1401	Median :1359
##	Mean :2013	Mean : 6.549	Mean :15.71	Mean :1349	Mean :1344
##	3rd Qu.:2013	3rd Qu.:10.000	3rd Qu.:23.00	3rd Qu.:1744	3rd Qu.:1729
##	Max. :2013	Max. :12.000	Max. :31.00	Max. :2400	Max. :2359
##				NA's :8255	
##	dep_delay	arr_time	sched_arr_time	arr_delay	
##	Min. : -43.00	Min. : 1	Min. : 1	Min. : -86.000	
##	1st Qu.: -5.00	1st Qu.:1104	1st Qu.:1124	1st Qu.: -17.000	
##	Median : -2.00	Median :1535	Median :1556	Median : -5.000	
##	Mean : 12.64	Mean :1502	Mean :1536	Mean : 6.895	
##	3rd Qu.: 11.00	3rd Qu.:1940	3rd Qu.:1945	3rd Qu.: 14.000	
##	Max. :1301.00	Max. :2400	Max. :2359	Max. :1272.000	
##	NA's :8255	NA's :8713		NA's :9430	
##	carrier	flight	tailnum	origin	
##	Length:336776	Min. : 1	Length:336776	Length:336776	
##	Class :character	1st Qu.: 553	Class :character	Class :character	
##	Mode :character	Median :1496	Mode :character	Mode :character	
##		Mean :1972			
##		3rd Qu.:3465			
##		Max. :8500			
##	dest	air_time	distance	hour	
##	Length:336776	Min. : 20.0	Min. : 17	Min. : 1.00	
##	Class :character	1st Qu.: 82.0	1st Qu.: 502	1st Qu.: 9.00	
##	Mode :character	Median :129.0	Median : 872	Median :13.00	
##		Mean :150.7	Mean :1040	Mean :13.18	
##		3rd Qu.:192.0	3rd Qu.:1389	3rd Qu.:17.00	
##		Max. :695.0	Max. :4983	Max. :23.00	
##		NA's :9430			
##	minute	time_hour			
##	Min. : 0.00	Min. :2013-01-01 05:00:00			

```
## 1st Qu.: 8.00    1st Qu.:2013-04-04 13:00:00
## Median :29.00   Median :2013-07-03 10:00:00
## Mean   :26.23   Mean   :2013-07-03 05:22:54
## 3rd Qu.:44.00   3rd Qu.:2013-10-01 07:00:00
## Max.    :59.00   Max.    :2013-12-31 23:00:00
##
```

3.2 El paquete dplyr

El objetivo ahora es asimilar las transformaciones de datos que ofrece `dplyr`:

- Filtrar observaciones (filas) con `filter()`,
- Reordenar observaciones (filas) con `arrange()`,
- Seleccionar variables (columnas) con `select()`,
- Crear nuevas variables (columnas) aplicando transformaciones (funciones) a las ya existentes con `mutate()`,
- Resumir la información de muchos valores con `summarise()`,
- ... puede ser usado con `group_by()` que agrupa las observaciones de acuerdo a cierta variable categórica.

3.2.1 Filtrar filas

Con `filter()` podemos filtrar/extraer las observaciones de acuerdo a características de una o varias variables, usando los operadores de comparación lógicos. Por ejemplo, para filtrar todos los vuelos ocurridos en los 1eros de Enero:

```
filter(flights, month == 1, day == 1)
```

```
## # A tibble: 842 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 832 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 <dtm>
```

Todos los vuelos de Enero a Febrero:

```
# nivel: "beginner"
flights_1_2 <- filter(flights, month == 1 | month == 2)

# nivel: "beginner" adelantado
flights_1_2 <- filter(flights, month %in% c(1, 2))

# nivel: "tidyverser" :)
flights_1_2 <- flights %>%
  filter(month %in% c(1, 2))
```

Vuelos que no se han retrasado más de 2hrs (tanto salida como llegada):

```
not_delayed <- filter(flights, arr_delay <= 120, dep_delay <= 120)
```

Algo interesante de `filter()` es que deja fuera directamente los NAs.

3.2.1.1 Ejercicios:

3. Encontrar los vuelos (asignar a una nueva variable que nombres apropiadamente):
 - a. Se atrasaron más de 2hrs en llegar
 - b. Volaron a Houston (IAH or HOU)
 - c. Fueron operados por “United”, “American” o “Deta”
 - d. Salieron en el verano (Julio, Agosto y Septiembre)
 - e. Llegaron más de 2hrs tarde, pero no salieron tarde
 - f. Se retrasaron al menos 1hr, pero compensaron 30min en vuelo
 - g. Salieron entre medianoche y 6am (inclusive)

R/

- a. Se atrasaron más de 2hrs en llegar

```
filter(flights, 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 <dtm>
```

b. Volaron a Houston (IAH or HOU)

```
filter(flights, 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 <dtm>
```

c. Fueron operados por “United”, “American” o “Delta”

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

```
## # 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 <dtm>
```

d. Salieron en el verano (Julio, Agosto y Septiembre)

```
filter(flights, month >= 7, 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 <dtm>
```

e. Llegaron más de 2hrs tarde, pero no salieron tarde

```
filter(flights, 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 <dtm>
```

f. Se retrasaron al menos 1hr, pero compesaron 30min en vuelo

```
filter(flights, 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 <dtm>
```

g. Salieron entre medianoche y 6am (inclusive). No

Opción 1:

```
filter(flights, 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 <dtm>
```

Opción 2:

```
filter(flights, dep_time %% 2400 <= 600)
```

```
## # 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 <dtm>
```

4. Busca la ayuda de `between()` e intenta simplificar un poco tus respuestas al ejercicio anterior.

R/

La expresión `between(x, left, right)` es equivalente a `x >= left & x <= right`. En el ejercicio “d. Salieron en el verano (Julio, Agosto y Septiembre)”, podemos hacer:

```
filter(flights, between(month, 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 <dtm>
```

5. ¿Cuántos vuelos no tienen información sobre `dep_time`? ¿Alguna otra variable tiene datos perdidos? ¿Qué crees que representan en cada caso?

R/

Parecen ser vuelos cancelados, también `arr_time` presenta NAs en estos casos:

```
filter(flights, is.na(dep_time))
```

```
## # A tibble: 8,255 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      NA           1630          NA     NA         1815
## 2 2013     1     1      NA           1935          NA     NA         2240
## 3 2013     1     1      NA           1500          NA     NA         1825
## 4 2013     1     1      NA           600           NA     NA          901
## 5 2013     1     2      NA           1540          NA     NA         1747
```

```
## 6 2013      1      2      NA      1620      NA      NA      1746
## 7 2013      1      2      NA      1355      NA      NA      1459
## 8 2013      1      2      NA      1420      NA      NA      1644
## 9 2013      1      2      NA      1321      NA      NA      1536
## 10 2013     1      2      NA      1545      NA      NA      1910
## # ... with 8,245 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 <dtm>
```

6. ¿Qué crees de los siguientes resultados?

```
NA^0 # R/ todo x^0 = 1
```

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

```
NA | TRUE # R/
```

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

```
FALSE & NA
```

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

```
NA * 0
```

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

R/

- NA^0 es 0 porque $x^0 = 1$ para todo x .
- $NA | TRUE$ es `TRUE` porque si el valor faltante fuera `TRUE`: entonces `TRUE | TRUE == TRUE`. Si el faltante es `FALSE`, entonces `FALSE | TRUE == TRUE`.
- `FALSE & NA` es `FALSE`. Si el `NA` es `TRUE`: `TRUE & FALSE == FALSE`. Si el `NA` fuera `FALSE`: `FALSE & FALSE == FALSE`.
- $NA * 0$ es 0, porque tiene en cuenta las indeterminaciones $\pm\infty \times 0$, en cuyo caso el resultado sería `NaN`. Para cualquier $x \neq \pm\infty$ tendríamos $x * 0 == 0$.

3.2.2 Rerodenar filas

Con `arrange()` podemos ordenar las observaciones (filas) de nuestros data frame, de acuerdo a una o más variables (columnas). En general, la ordenación se hará de acuerdo a la primera variable y el resto se usará en caso de “empate”. Por defecto, la ordenación es ascendente y los NA se colocan al final:

```
fl_asc <- arrange(flights, year, month, day, dep_time)
head(fl_asc, 7)
```

```
## # A tibble: 7 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
## # ... with 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 <dtm>

tail(fl_asc, 7)
```

```
## # A tibble: 7 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     12    31      NA          1430          NA        NA          1750
## 2 2013     12    31      NA          855          NA        NA          1142
## 3 2013     12    31      NA          705          NA        NA          931
## 4 2013     12    31      NA          825          NA        NA          1029
## 5 2013     12    31      NA          1615         NA        NA          1800
## 6 2013     12    31      NA          600          NA        NA          735
## 7 2013     12    31      NA          830          NA        NA          1154
## # ... with 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 <dtm>
```

Orden descendente, de acuerdo a dep_time:

```
fl_dsc <- arrange(flights, desc(dep_time))
head(fl_dsc, 7)
```

```
## # A tibble: 7 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     10    30     2400          2359          1        327          337
## 2 2013     11    27     2400          2359          1        515          445
## 3 2013     12     5     2400          2359          1        427          440
## 4 2013     12     9     2400          2359          1        432          440
## 5 2013     12     9     2400          2250         70         59          2356
## 6 2013     12    13     2400          2359          1        432          440
## 7 2013     12    19     2400          2359          1        434          440
## # ... with 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 <dtm>
```

3.2.2.1 Ejercicios

7. Si por defecto `arrange()` coloca los NA al final, ¿hay alguna forma de colocarlos al inicio? Hint: usa `is.na()`.

R/

```
arrange(flights, desc(is.na(dep_time)), dep_time)
```

```
## # A tibble: 336,776 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     NA           1630         NA       NA           1815
## 2  2013     1     1     NA           1935         NA       NA           2240
## 3  2013     1     1     NA           1500         NA       NA           1825
## 4  2013     1     1     NA            600         NA       NA            901
## 5  2013     1     2     NA           1540         NA       NA           1747
## 6  2013     1     2     NA           1620         NA       NA           1746
## 7  2013     1     2     NA           1355         NA       NA           1459
## 8  2013     1     2     NA           1420         NA       NA           1644
## 9  2013     1     2     NA           1321         NA       NA           1536
## 10 2013     1     2     NA           1545         NA       NA           1910
## # ... with 336,766 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 <dtm>
```

8. Ordena los vuelos para encontrar los que más se retrasaron.

R/

```
arrange(flights, desc(dep_delay))
```

```
## # A tibble: 336,776 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     9     641           900       1301    1242           1530
## 2  2013     6    15    1432          1935       1137    1607           2120
## 3  2013     1    10    1121          1635       1126    1239           1810
## 4  2013     9    20    1139          1845       1014    1457           2210
## 5  2013     7    22     845          1600       1005    1044           1815
## 6  2013     4    10    1100          1900        960    1342           2211
## 7  2013     3    17    2321           810        911     135           1020
## 8  2013     6    27     959          1900        899    1236           2226
## 9  2013     7    22    2257           759        898     121           1026
## 10 2013    12     5     756          1700        896    1058           2020
## # ... with 336,766 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 <dtm>
```

9. Encuentra los que despegaron antes.

R/

```
arrange(flights, dep_delay)
```

```
## # A tibble: 336,776 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    12     7    2040           2123         -43     40           2352
## 2  2013     2     3    2022           2055         -33    2240           2338
## 3  2013    11    10    1408           1440         -32    1549           1559
## 4  2013     1    11    1900           1930         -30    2233           2243
## 5  2013     1    29    1703           1730         -27    1947           1957
## 6  2013     8     9     729            755         -26    1002            955
## 7  2013    10    23    1907           1932         -25    2143           2143
## 8  2013     3    30    2030           2055         -25    2213           2250
## 9  2013     3     2    1431           1455         -24    1601           1631
## 10 2013     5     5     934            958         -24    1225           1309
## # ... with 336,766 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 <dtm>
```

10. Ordena los vuelos de forma tal que permita encontrar los de mayor velocidad.

R/

```
# Si lo entendemos por tiempo de vuelo:
```

```
head(arrange(flights, air_time))
```

```
## # A tibble: 6 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    16    1355           1315          40    1442           1411
## 2  2013     4    13     537            527          10     622            628
## 3  2013    12     6     922            851          31    1021            954
## 4  2013     2     3    2153           2129          24    2247           2224
## 5  2013     2     5    1303           1315         -12    1342           1411
## 6  2013     2    12    2123           2130          -7    2211           2225
## # ... with 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 <dtm>
```

```
# Si lo entendemos por velocidad media:
```

```
head(arrange(flights, desc(distance / air_time)))
```

```
## # A tibble: 6 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      5      25      1709          1700        9      1923          1937
## 2  2013      7       2      1558          1513       45      1745          1719
## 3  2013      5      13      2040          2025       15      2225          2226
## 4  2013      3      23      1914          1910        4      2045          2043
## 5  2013      1      12      1559          1600       -1      1849          1917
## 6  2013     11      17       650           655       -5     1059          1150
## # ... with 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 <dtm>
```

11. ¿Cuáles son los vuelos que mayor (menor) distancia recorrieron?

R/

```
# Mayor distancia:
arrange(flights, desc(distance))
```

```
## # A tibble: 336,776 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     857           900         -3    1516           1530
## 2  2013     1     2     909           900          9    1525           1530
## 3  2013     1     3     914           900         14    1504           1530
## 4  2013     1     4     900           900          0    1516           1530
## 5  2013     1     5     858           900         -2    1519           1530
## 6  2013     1     6    1019           900         79    1558           1530
## 7  2013     1     7    1042           900        102    1620           1530
## 8  2013     1     8     901           900          1    1504           1530
## 9  2013     1     9     641           900       1301    1242           1530
## 10 2013     1    10     859           900         -1    1449           1530
## # ... with 336,766 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 <dtm>
```

```
# Menor distancia:
arrange(flights, distance)
```

```
## # A tibble: 336,776 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    27      NA           106         NA      NA           245
## 2  2013     1     3    2127          2129         -2    2222           2224
## 3  2013     1     4    1240          1200         40    1333           1306
## 4  2013     1     4    1829          1615        134    1937           1721
## 5  2013     1     4    2128          2129         -1    2218           2224
## 6  2013     1     5    1155          1200         -5    1241           1306
## 7  2013     1     6    2125          2129         -4    2224           2224
```

```
## 8 2013      1      7      2124      2129      -5      2212      2224
## 9 2013      1      8      2127      2130      -3      2304      2225
## 10 2013     1      9      2126      2129      -3      2217      2224
## # ... with 336,766 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 <dtm>
```

3.2.3 Seleccionar variables

Con `select()` podemos justamente seleccionar variables (columnas) de interés.

```
# seleccionamos año, mes y día
flights %>%
  select(year, month, day) %>%
  head(5)
```

```
## # A tibble: 5 x 3
##   year month   day
##   <int> <int> <int>
## 1  2013     1     1
## 2  2013     1     1
## 3  2013     1     1
## 4  2013     1     1
## 5  2013     1     1
```

```
# seleccionamos todas las columnas desde año (year) hasta día (day),
# ambas inclusive
flights %>%
  select(year:day) %>%
  head(5)
```

```
## # A tibble: 5 x 3
##   year month   day
##   <int> <int> <int>
## 1  2013     1     1
## 2  2013     1     1
## 3  2013     1     1
## 4  2013     1     1
## 5  2013     1     1
```

```
# seleccionamos todas las columnas excepto las que van desde año (year)
# hasta día (day), ambas inclusive
flights %>%
  select(-(year:day)) %>%
  head(5)
```

```
## # A tibble: 5 x 16
##   dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier
```

```
##      <int>          <int>      <dbl>    <int>          <int>      <dbl> <chr>
## 1      517          515         2      830          819        11 UA
## 2      533          529         4      850          830        20 UA
## 3      542          540         2      923          850        33 AA
## 4      544          545        -1     1004         1022       -18 B6
## 5      554          600        -6      812          837       -25 DL
## # ... with 9 more variables: flight <int>, tailnum <chr>, origin <chr>,
## #   dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
## #   time_hour <dtm>
```

También dispondremos de las “funciones de ayuda a la selección”:

- `starts_with("abc")`: columnas que empiezan en “abc”.
- `ends_with("xyz")`: columnas que terminan en “xyz”.
- `contains("ijk")`: columnas que contienen la expresión “ijk”.
- `matches("[pt]xyz")`: selecciona variables que coinciden con una expresión regular.
- `num_range("x", 1:3)`: equivalente a `select(x, 1:3)`.
- `everything()`: selecciona todas las variables. Útil si deseamos poner algunas columnas de interés al inicio, porque `select()` no incluye columnas repetidas:

```
flights %>%
  select(time_hour, air_time, everything()) %>%
  head(5)
```

```
## # A tibble: 5 x 19
##   time_hour          air_time year month   day dep_time sched_dep_time
##   <dtm>          <dbl> <int> <int> <int>   <int>         <int>
## 1 2013-01-01 05:00:00    227  2013     1     1     517           515
## 2 2013-01-01 05:00:00    227  2013     1     1     533           529
## 3 2013-01-01 05:00:00    160  2013     1     1     542           540
## 4 2013-01-01 05:00:00    183  2013     1     1     544           545
## 5 2013-01-01 06:00:00    116  2013     1     1     554           600
## # ... with 12 more variables: dep_delay <dbl>, arr_time <int>,
## #   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, distance <dbl>, hour <dbl>,
## #   minute <dbl>
```

3.2.3.1 Ejercicios

12. ¿Cuál será la forma más corta de seleccionar: `dep_time`, `dep_delay`, `arr_time`, `arr_delay`?

R/

```
# Opción elegante:
select(flights, starts_with("dep_"), starts_with("arr_"))
```

```
## # A tibble: 336,776 x 4
##   dep_time dep_delay arr_time arr_delay
##   <int>     <dbl>   <int>     <dbl>
## 1     517         2     830         11
## 2     533         4     850         20
## 3     542         2     923         33
## 4     544        -1    1004        -18
## 5     554        -6     812        -25
## 6     554        -4     740         12
## 7     555        -5     913         19
## 8     557        -3     709        -14
## 9     557        -3     838         -8
## 10    558        -2     753          8
## # ... with 336,766 more rows
```

```
# Menos elegante, pero funcional y corta:
select(flights, 4, 6, 7, 9)
```

```
## # A tibble: 336,776 x 4
##   dep_time dep_delay arr_time arr_delay
##   <int>     <dbl>   <int>     <dbl>
## 1     517         2     830         11
## 2     533         4     850         20
## 3     542         2     923         33
## 4     544        -1    1004        -18
## 5     554        -6     812        -25
## 6     554        -4     740         12
## 7     555        -5     913         19
## 8     557        -3     709        -14
## 9     557        -3     838         -8
## 10    558        -2     753          8
## # ... with 336,766 more rows
```

13. Queremos seleccionar las variables indicadas en el vector `vars`. Hint: usar `any_of`.

```
vars <- c("year", "month", "day", "dep_delay", "arr_delay")
```

R/

```
select(flights, any_of(vars))
```

```
## # A tibble: 336,776 x 5
##   year month day dep_delay arr_delay
##   <int> <int> <int>     <dbl>     <dbl>
## 1  2013     1   1         2         11
## 2  2013     1   1         4         20
## 3  2013     1   1         2         33
```

```
## 4 2013      1      1      -1      -18
## 5 2013      1      1      -6      -25
## 6 2013      1      1      -4       12
## 7 2013      1      1      -5       19
## 8 2013      1      1      -3      -14
## 9 2013      1      1      -3       -8
## 10 2013     1      1      -2        8
## # ... with 336,766 more rows
```

14. ¿Qué pasa con el siguiente código? ¿Debería seleccionar todas esas variables?

```
select(flights, contains("TiMe"))
```

```
## # A tibble: 336,776 x 6
##   dep_time sched_dep_time arr_time sched_arr_time air_time time_hour
##   <int>         <int>      <int>         <int>      <dbl> <dtm>
## 1     517           515        830           819      227 2013-01-01 05:00:00
## 2     533           529        850           830      227 2013-01-01 05:00:00
## 3     542           540        923           850      160 2013-01-01 05:00:00
## 4     544           545       1004          1022      183 2013-01-01 05:00:00
## 5     554           600        812           837      116 2013-01-01 06:00:00
## 6     554           558        740           728      150 2013-01-01 05:00:00
## 7     555           600        913           854      158 2013-01-01 06:00:00
## 8     557           600        709           723       53 2013-01-01 06:00:00
## 9     557           600        838           846      140 2013-01-01 06:00:00
## 10    558           600        753           745      138 2013-01-01 06:00:00
## # ... with 336,766 more rows
```

R/

Para cambiar el resultado, podemos hacer que coincidan también las letras en mayúscula y minúscula:

```
select(flights, contains("TiMe", ignore.case = FALSE))
```

```
## # A tibble: 336,776 x 0
```

3.2.4 Crear nuevas variables

Con `mutate()` podemos añadir nuevas columnas a nuestro data frame. Estas columnas se crean al aplicar las funciones que conocemos (operaciones aritméticas, *lags*, acumulados, etc.) a las columnas ya existentes.

```
flights %>%
  mutate(gain = dep_delay - arr_delay,
         speed = distance / air_time * 60,
         hours = air_time / 60,
         gain_per_hour = gain / hours) %>% # ¡usamos las columnas nuevas!
```

```
select(gain, speed, hours, gain_per_hour, everything()) %>%
head(5)
```

```
## # A tibble: 5 x 23
##   gain speed hours gain_per_hour year month   day dep_time sched_dep_time
##   <dbl> <dbl> <dbl>         <dbl> <int> <int> <int>   <int>         <int>
## 1    -9  370.  3.78         -2.38  2013     1     1     517             515
## 2   -16  374.  3.78         -4.23  2013     1     1     533             529
## 3   -31  408.  2.67        -11.6  2013     1     1     542             540
## 4    17  517.  3.05          5.57  2013     1     1     544             545
## 5    19  394.  1.93          9.83  2013     1     1     554             600
## # ... with 14 more variables: dep_delay <dbl>, arr_time <int>,
## #   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dtm>
```

Si solamente nos interesan las nuevas columnas que hemos creado, usamos `transmute()`:

```
flights %>%
  transmute(gain = dep_delay - arr_delay,
            speed = distance / air_time * 60,
            hours = air_time / 60,
            gain_per_hour = gain / hours) %>% # ¡usamos las columnas nuevas!
head(5)
```

```
## # A tibble: 5 x 4
##   gain speed hours gain_per_hour
##   <dbl> <dbl> <dbl>         <dbl>
## 1    -9  370.  3.78         -2.38
## 2   -16  374.  3.78         -4.23
## 3   -31  408.  2.67        -11.6
## 4    17  517.  3.05          5.57
## 5    19  394.  1.93          9.83
```

3.2.4.1 Ejercicios

15. Convertir `dep_time` y `sched_dep_time` a minutos transcurridos desde la medianoche. Notar que son variables importantes pero con un formato difícil de trabajar (es complicado hacer operaciones aritméticas con ellos). Sigue las siguientes directrices:
 - Una observación de `dep_time` sería por ejemplo 2021 que indica las 20:21hrs (8:21pm).
 - Para obtener las horas transcurridas desde la medianoche hasta las 20:21hrs tendremos que usar la división entera `2021 %/% 100 == 20`. Luego es fácil obtener los minutos multiplicando por 60.

- Los 21 minutos restantes podemos obtenerlos con el resto de la división `2021 %% 100 == 21`... no olvides sumar ambas cantidades
- Finalmente, tendrás que lidiar con la medianoche, representada con 2400. Primero, comprueba a cuántos minutos corresponde según nuestras operaciones. Luego, considera calcular el resto de la división por esta cantidad de minutos (siempre que `x <= y` y ambos sean positivos, tendremos `x %% y == 0`).

R/

```
flights_times <- mutate(flights,
  dep_time_mins = (dep_time %% 100 * 60 + dep_time %% 100) %% 1440,
  sched_dep_time_mins = (sched_dep_time %% 100 * 60 +
    sched_dep_time %% 100) %% 1440
)
select(
  flights_times, dep_time, dep_time_mins, sched_dep_time,
  sched_dep_time_mins
)
```

```
## # A tibble: 336,776 x 4
##   dep_time dep_time_mins sched_dep_time sched_dep_time_mins
##   <int>      <dbl>         <int>         <dbl>
## 1     517         317           515           315
## 2     533         333           529           329
## 3     542         342           540           340
## 4     544         344           545           345
## 5     554         354           600           360
## 6     554         354           558           358
## 7     555         355           600           360
## 8     557         357           600           360
## 9     557         357           600           360
## 10    558         358           600           360
## # ... with 336,766 more rows
```

16. Compara `air_time` con `arr_time - dep_time`. ¿Es necesaria hacer la transformación del ejercicio anterior? ¿Puedes encontrar en cuántos casos `air_time != arr_time - dep_time`? ¿Por qué pasa esto, no deberíamos obtener que el tiempo de vuelo es la diferencia entre la llegada y la salida?

R/

Calculamos primero tiempo de vuelo:

```
flights_airtime <-
  mutate(flights,
    dep_time = (dep_time %% 100 * 60 + dep_time %% 100) %% 1440,
    arr_time = (arr_time %% 100 * 60 + arr_time %% 100) %% 1440,
    air_time_diff = air_time - arr_time + dep_time
```

```
)
```

Pero notamos que `air_time != arr_time - dep_time` lo cual no es intuitivo:

```
nrow(filter(flights_airtime, air_time_diff != 0))
```

```
## [1] 327150
```

La relación que sí se cumple es `air_time <= arr_time - dep_time`. El problema es que `air_time` solo cuenta el tiempo desde que despegue hasta que aterriza, sin contar el desplazamiento hasta la pista o el tiempo en alcanzar la velocidad apropiada antes de despegar.

17. ¿Qué relación crees que habrá entre `dep_time`, `sched_dep_time` y `dep_delay`? Encuentra el número de observaciones en las que no se cumple tu hipótesis.

R/

Esperaríamos que `dep_time - sched_dep_time == dep_delay`, pero esto no siempre es así:

```
flights_deptime <-
  mutate(flights,
    dep_time_min = (dep_time %% 100 * 60 + dep_time %/% 100) %/% 1440,
    sched_dep_time_min = (sched_dep_time %% 100 * 60 +
      sched_dep_time %/% 100) %/% 1440,
    dep_delay_diff = dep_delay - dep_time_min + sched_dep_time_min
  )
```

```
filter(flights_deptime, dep_delay_diff != 0)
```

```
## # A tibble: 1,236 x 22
```

```
##   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     848           1835        853     1001           1950
## 2  2013     1     2      42           2359         43      518           442
## 3  2013     1     2     126           2250        156      233           2359
## 4  2013     1     3      32           2359         33      504           442
## 5  2013     1     3      50           2145        185      203           2311
## 6  2013     1     3     235           2359        156      700           437
## 7  2013     1     4      25           2359         26      505           442
## 8  2013     1     4     106           2245        141      201           2356
## 9  2013     1     5      14           2359         15      503           445
## 10 2013     1     5      37           2230        127      341           131
## # ... with 1,226 more rows, and 14 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 <dtm>,
## #   dep_time_min <dbl>, sched_dep_time_min <dbl>, dep_delay_diff <dbl>
```


En este caso, las discrepancias podrían estar debidas a algún error al recoger los datos.

18. Encuentra los 10 vuelos que más se retrasaron.

R/

```
flights_delayed10 <- flights %>%
  top_n(10, dep_delay) %>%
  arrange(desc(dep_delay)) %>%
  select(month, day, carrier, flight, dep_delay) %>%
  print()
```

```
## # A tibble: 10 x 5
##   month   day carrier flight dep_delay
##   <int> <int> <chr>    <int>    <dbl>
## 1     1     9  HA         51      1301
## 2     6    15  MQ        3535      1137
## 3     1    10  MQ        3695      1126
## 4     9    20  AA         177      1014
## 5     7    22  MQ        3075      1005
## 6     4    10  DL        2391       960
## 7     3    17  DL        2119       911
## 8     6    27  DL        2007       899
## 9     7    22  DL        2047       898
## 10    12     5  AA         172       896
```

3.2.5 Resumir variables

Con `summarise()` logramos “resumir” la información de determinadas variables, de acuerdo a cierta función que fijemos (media, mediana, IQR, etc.). Debes tener en cuenta que esto “colapsa” el data frame inicial.

```
summarise(flights, delay = mean(dep_delay, na.rm = TRUE))
```

```
## # A tibble: 1 x 1
##   delay
##   <dbl>
## 1  12.6
```

```
summarise(flights, delay = mean(dep_delay))
```

```
## # A tibble: 1 x 1
##   delay
##   <dbl>
## 1    NA
```

Ahora, lo verdaderamente interesante de esta función es usarla para “observaciones agrupadas” con `group_by()`. Por ejemplo, queremos saber la media de

los retrasos por mes y año:

```
mean_m_y <- flights %>%
  group_by(year, month) %>%
  summarise(delay = mean(dep_delay, na.rm = TRUE))
```

```
## `summarise()` has grouped output by 'year'. You can override using the `.groups` arg
mean_m_y
```

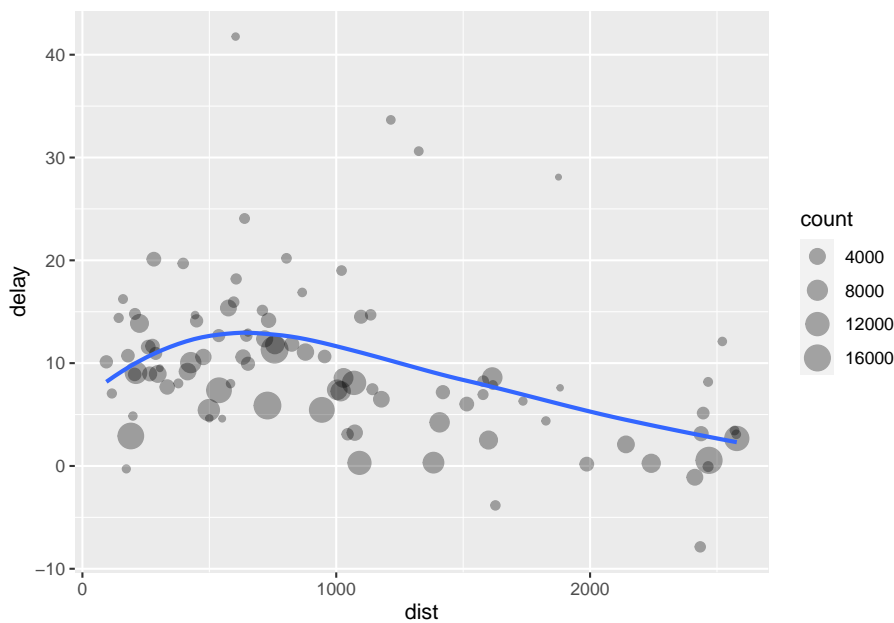
```
## # A tibble: 12 x 3
## # Groups:   year [1]
##   year month delay
##   <int> <int> <dbl>
## 1  2013     1  10.0
## 2  2013     2  10.8
## 3  2013     3  13.2
## 4  2013     4  13.9
## 5  2013     5  13.0
## 6  2013     6  20.8
## 7  2013     7  21.7
## 8  2013     8  12.6
## 9  2013     9   6.72
## 10 2013    10   6.24
## 11 2013    11   5.44
## 12 2013    12  16.6
```

Cambiando la variable de agrupamiento (debe ser categórica) podemos obtener la media (o cualquier otro estadístico que deseemos) para cada categoría. Veamos un ejemplo en combinación con `ggplot`:

```
delays <- flights %>%
  group_by(dest) %>%
  summarise(
    count = n(),
    dist = mean(distance, na.rm = TRUE),
    delay = mean(arr_delay, na.rm = TRUE)
  ) %>%
  filter(count > 20, dest != "HNL")

ggplot(data = delays, mapping = aes(x = dist, y = delay)) +
  geom_point(aes(size = count), alpha = 1/3) +
  geom_smooth(se = FALSE)
```

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



Estamos agrupando por destino (`dest`) y luego contamos la cantidad de vuelos que van a cada destino (`count`), la distancia media (`dist`) entre los aeropuertos de origen y el destino, y el retraso medio en minutos de la llegada (`delay`). Habrás notado que filtramos los destinos con pocas visitas (pueden ser outliers) y Honolulu (está muy lejos de casi cualquier aeropuerto), para eliminar un poco de “ruido” en nuestro plot (intenta omitir el filtrado y notarás que es más difícil la interpretación). De este gráfico entendemos que mientras más cercano el destino, mayor probabilidad de retraso. Sin embargo, los vuelos a destinos lejanos parecen presentar menos retrasos (tal vez en el aire puedan compensar el retraso).

Otros ejemplos usando varias variables de agrupamiento:

```
daily <- group_by(flights, year, month, day)
(per_day <- summarise(daily, flights = n()))
```

`summarise()` has grouped output by 'year', 'month'. You can override using the `.groups` argu

```
## # A tibble: 365 x 4
## # Groups:   year, month [12]
##   year month   day flights
##   <int> <int> <int>   <int>
## 1  2013     1     1     842
## 2  2013     1     2     943
## 3  2013     1     3     914
## 4  2013     1     4     915
## 5  2013     1     5     720
## 6  2013     1     6     832
```

```
## 7 2013      1      7      933
## 8 2013      1      8      899
## 9 2013      1      9      902
## 10 2013     1     10      932
## # ... with 355 more rows
```

```
(per_month <- summarise(per_day, flights = sum(flights)))
```

`summarise()` has grouped output by 'year'. You can override using the `.groups` argument

```
## # A tibble: 12 x 3
## # Groups:   year [1]
##   year month flights
##   <int> <int>   <int>
## 1  2013     1   27004
## 2  2013     2   24951
## 3  2013     3   28834
## 4  2013     4   28330
## 5  2013     5   28796
## 6  2013     6   28243
## 7  2013     7   29425
## 8  2013     8   29327
## 9  2013     9   27574
## 10 2013    10   28889
## 11 2013    11   27268
## 12 2013    12   28135
```

```
(per_year <- summarise(per_month, flights = sum(flights)))
```

```
## # A tibble: 1 x 2
##   year flights
##   <int>   <int>
## 1  2013  336776
```

También, si deseas deshacer la agrupación, por ejemplo, si quieres contar el total de vuelos sin agrupar:

```
daily %>%
  ungroup() %>%           # deshacemoos la agrupación por fecha
  summarise(flights = n()) # toooooooooodos los vuelos :)
```

```
## # A tibble: 1 x 1
##   flights
##   <int>
## 1  336776
```

3.2.5.1 Ejercicios

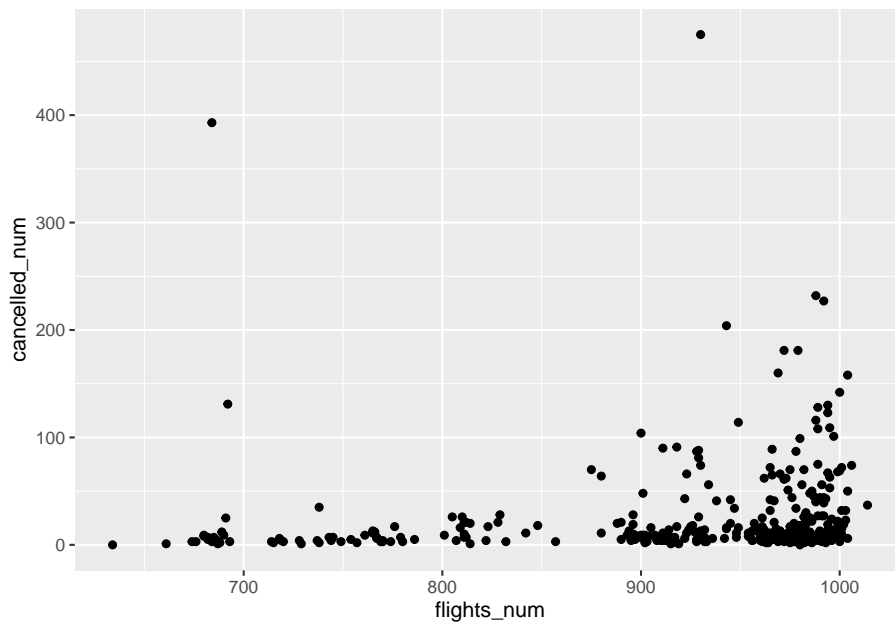
19. Mira el número de vuelos cancelados por día e intenta encontrar algún patrón. ¿Está relacionada la proporción de vuelos cancelados con el retraso medio? Hint:

- Crear una nueva variable/columna que indique si un vuelo se ha cancelado o no (definiremos `cancelado = (is.na(arr_delay) | is.na(dep_delay))`),
- No olvides agrupar (año, mes, día) y luego cuenta el número total de vuelos y el número de cancelados,
- Haz un plot de cancelados vs. número de vuelos e intenta describir posibles patrones,
- Para responder la pregunta tendrás que crear una variable `prop_cancelados` (media) y la media de `dep_delay` o `arr_delay`,
- Realiza los diagramas de dispersión correspondientes e intenta describir posibles patrones.

R/

```
cancelled_per_day <-  
  flights %>%  
  mutate(cancelled = (is.na(arr_delay) | is.na(dep_delay))) %>%  
  group_by(year, month, day) %>%  
  summarise(  
    cancelled_num = sum(cancelled),  
    flights_num = n(),  
  )
```

```
## `summarise()` has grouped output by 'year', 'month'. You can override using the `.groups` argu  
ggplot(cancelled_per_day) +  
  geom_point(aes(x = flights_num, y = cancelled_num))
```

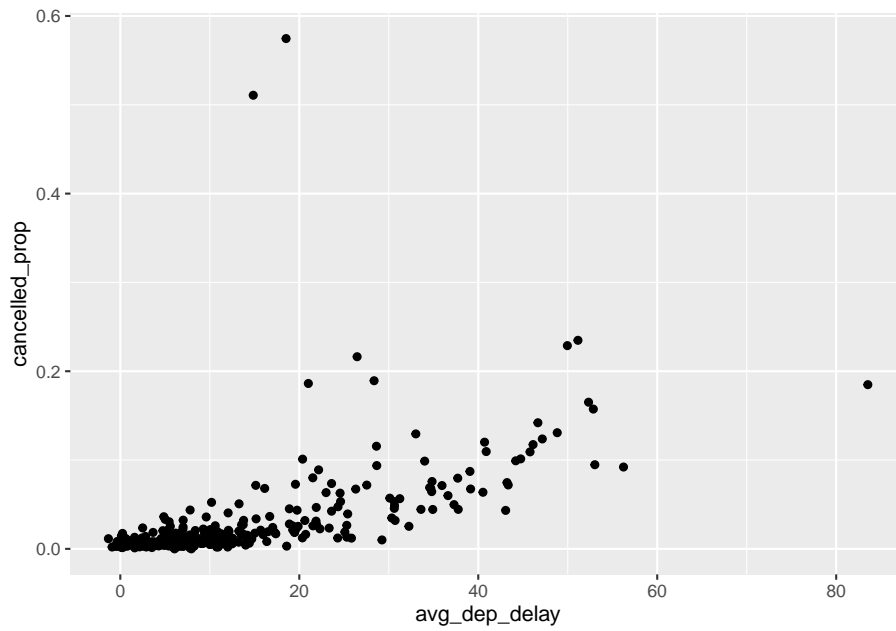


Respuesta a la pregunta:

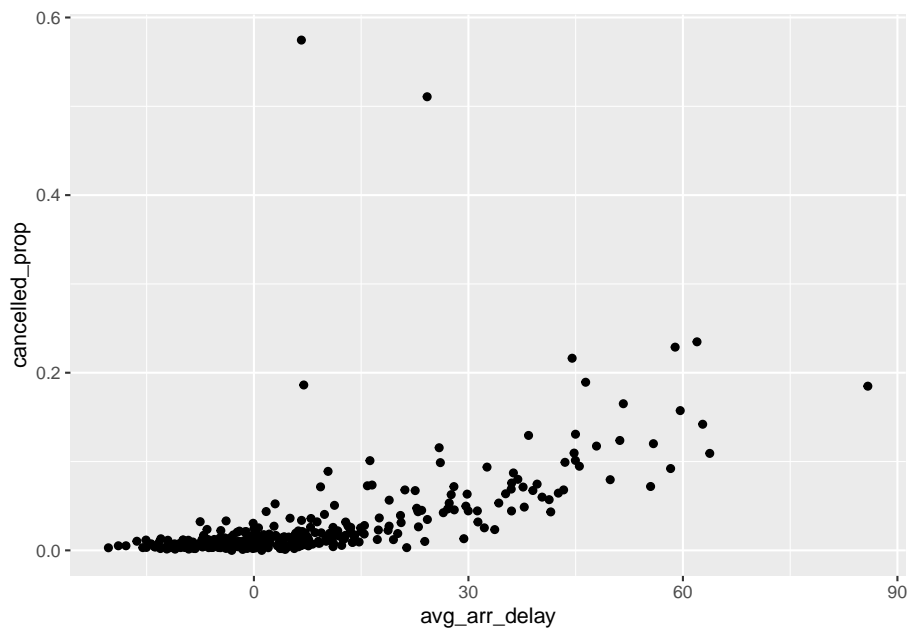
```
cancelled_and_delays <-
  flights %>%
  mutate(cancelled = (is.na(arr_delay) | is.na(dep_delay))) %>%
  group_by(year, month, day) %>%
  summarise(
    cancelled_prop = mean(cancelled),
    avg_dep_delay = mean(dep_delay, na.rm = TRUE),
    avg_arr_delay = mean(arr_delay, na.rm = TRUE)
  ) %>%
  ungroup()
```

```
## `summarise()` has grouped output by 'year', 'month'. You can override using the `.groups` argument.

ggplot(cancelled_and_delays) +
  geom_point(aes(x = avg_dep_delay, y = cancelled_prop))
```



```
ggplot(cancelled_and_delays) +  
  geom_point(aes(x = avg_arr_delay, y = cancelled_prop))
```



20. ¿A qué hora del día (hour) deberías viajar si quieres evitar retrasos tanto como sea posible? Hints:

- Agrupar por la variable que consideres oportuna,
- Resume el tiempo que ha demorado el vuelo (¿es más importante `arr_delay` o `dep_delay`?),
- Reordena adecuadamente.

R/

```
flights %>%  
  group_by(hour) %>%  
  summarise(arr_delay = mean(arr_delay, na.rm = TRUE)) %>%  
  arrange(arr_delay)
```

```
## # A tibble: 20 x 2  
##       hour arr_delay  
##   <dbl>   <dbl>  
## 1     7    -5.30  
## 2     5    -4.80  
## 3     6    -3.38  
## 4     9    -1.45  
## 5     8    -1.11  
## 6    10     0.954  
## 7    11     1.48  
## 8    12     3.49  
## 9    13     6.54  
## 10   14     9.20  
## 11   23    11.8  
## 12   15    12.3  
## 13   16    12.6  
## 14   18    14.8  
## 15   22    16.0  
## 16   17    16.0  
## 17   19    16.7  
## 18   20    16.7  
## 19   21    18.4  
## 20    1    NaN
```


Chapter 4

Tidy

4.1 Datos

Vamos a trabajar con unos datasets sencillos que recopilan la misma información sobre 4 variables: país (*country*), año (*year*), población (*population*) y casos (*cases*) de Tuberculosis (TB). ¿Puedes identificar cuál de ellos está en forma *tidy*?

```
library(tidyverse)
table1
```

```
## # A tibble: 6 x 4
##   country      year cases population
##   <chr>      <int> <int>      <int>
## 1 Afghanistan 1999     745   19987071
## 2 Afghanistan 2000    2666   20595360
## 3 Brazil      1999   37737   172006362
## 4 Brazil      2000   80488   174504898
## 5 China       1999  212258  1272915272
## 6 China       2000  213766  1280428583
```

```
table2
```

```
## # A tibble: 12 x 4
##   country      year type      count
##   <chr>      <int> <chr>      <int>
## 1 Afghanistan 1999 cases         745
## 2 Afghanistan 1999 population 19987071
## 3 Afghanistan 2000 cases         2666
## 4 Afghanistan 2000 population 20595360
## 5 Brazil      1999 cases         37737
## 6 Brazil      1999 population 172006362
```

```
## 7 Brazil      2000 cases      80488
## 8 Brazil      2000 population 174504898
## 9 China       1999 cases      212258
## 10 China      1999 population 1272915272
## 11 China      2000 cases      213766
## 12 China      2000 population 1280428583
```

```
table3
```

```
## # A tibble: 6 x 3
##   country      year rate
## * <chr>      <int> <chr>
## 1 Afghanistan 1999 745/19987071
## 2 Afghanistan 2000 2666/20595360
## 3 Brazil      1999 37737/172006362
## 4 Brazil      2000 80488/174504898
## 5 China       1999 212258/1272915272
## 6 China       2000 213766/1280428583
```

```
table4a
```

```
## # A tibble: 3 x 3
##   country      `1999` `2000`
## * <chr>      <int> <int>
## 1 Afghanistan    745    2666
## 2 Brazil        37737  80488
## 3 China         212258 213766
```

```
table4b
```

```
## # A tibble: 3 x 3
##   country      `1999`      `2000`
## * <chr>      <int>      <int>
## 1 Afghanistan 19987071 20595360
## 2 Brazil      172006362 174504898
## 3 China       1272915272 1280428583
```

4.2 Pivotar

Generalmente, para ordenar tus datos (*tidying*) tendrás que seguir 2 pasos básicos:

1. Identificar qué es variable (lo que irá en las columnas) y qué es observación (lo que irá en las filas);
2. Resolver una de estas situaciones:
 - Las variables podrían estar distribuidas en varias columnas
 - Las observaciones podrían estar distribuidas en varias filas

- Ambas a la vez :(

Esto lo resolveremos con las funciones `pivot_longer()` y `pivot_wider()`.

4.2.1 *Pivot longer*

Cuando nuestro dataset tiene por columnas los valores de una variable, usamos `pivot_longer()`. La `table4a` es un caso claro de esta situación: tenemos dos columnas con nombre 1990 y 2000, que corresponden a valores de la variable `year`. El proceso para hacerlos *tidy* pasa por arreglar estas columnas creando dos nuevas variables: `year` y `cases`:

```
table4a %>%
  pivot_longer(c(`1999`, `2000`), names_to = "year", values_to = "cases")

## # A tibble: 6 x 3
##   country    year  cases
##   <chr>      <chr> <int>
## 1 Afghanistan 1999     745
## 2 Afghanistan 2000    2666
## 3 Brazil      1999   37737
## 4 Brazil      2000   80488
## 5 China       1999  212258
## 6 China       2000  213766
```

De forma similar, podemos arreglar `table4b`:

```
table4b %>%
  pivot_longer(c(`1999`, `2000`), names_to = "year", values_to = "population")

## # A tibble: 6 x 3
##   country    year population
##   <chr>      <chr>      <int>
## 1 Afghanistan 1999    19987071
## 2 Afghanistan 2000    20595360
## 3 Brazil      1999    172006362
## 4 Brazil      2000    174504898
## 5 China       1999    1272915272
## 6 China       2000    1280428583
```

Finalmente, si queremos unir ambos resultados, podemos usar `left_join`, que ya estudiaremos con los *Datos relacionales*:

```
tidy4a <- table4a %>%
  pivot_longer(c(`1999`, `2000`), names_to = "year", values_to = "cases")
tidy4b <- table4b %>%
  pivot_longer(c(`1999`, `2000`), names_to = "year", values_to = "population")
left_join(tidy4a, tidy4b)
```

```
## Joining, by = c("country", "year")

## # A tibble: 6 x 4
##   country    year  cases population
##   <chr>      <chr> <int>      <int>
## 1 Afghanistan 1999     745    19987071
## 2 Afghanistan 2000    2666    20595360
## 3 Brazil      1999   37737    172006362
## 4 Brazil      2000   80488    174504898
## 5 China       1999  212258    1272915272
## 6 China       2000  213766    1280428583
```

4.2.2 *Pivot wider*

Lo opuesto a *alargar* un dataset es hacerlo *más ancho*. Por tanto, es de entender que con `pivot_wider()` crearemos más columnas. Si prestamos atención a la `table2` notaremos que cada observación a sido expandida en dos filas que recogen los casos y la población. Esto lo solucionamos creando dos nuevas variables (columnas) para los casos y la población:

```
table2 %>%
  pivot_wider(names_from = type, values_from = count)
```

```
## # A tibble: 6 x 4
##   country    year  cases population
##   <chr>      <int> <int>      <int>
## 1 Afghanistan 1999     745    19987071
## 2 Afghanistan 2000    2666    20595360
## 3 Brazil      1999   37737    172006362
## 4 Brazil      2000   80488    174504898
## 5 China       1999  212258    1272915272
## 6 China       2000  213766    1280428583
```

4.2.3 Ejercicios

1. Aunque opuestas, no son perfectamente simétricas. ¿Puedes deducir por qué?

```
stocks <- tibble(
  year = c(2015, 2015, 2016, 2016),
  half = c( 1,   2,   1,   2),
  return = c(1.88, 0.59, 0.92, 0.17)
)
```

```
stocks
```

```
## # A tibble: 4 x 3
##   year half return
```

```
##      <dbl> <dbl> <dbl>
## 1  2015      1  1.88
## 2  2015      2  0.59
## 3  2016      1  0.92
## 4  2016      2  0.17

stocks %>%
  pivot_wider(names_from = year, values_from = return) %>%
  pivot_longer(`2015`:`2016`, names_to = "year", values_to = "return")
```

```
## # A tibble: 4 x 3
##   half year return
##   <dbl> <chr> <dbl>
## 1     1 2015  1.88
## 2     1 2016  0.92
## 3     2 2015  0.59
## 4     2 2016  0.17
```

R/

Al usar *wider* pasamos los años a nombres de columnas (de enteros a caracteres). Luego con *longer* leemos los años como caracteres.

2. Intenta arreglarlo usando el argumento `names_transform = list(year = as.numeric)`.

R/

```
stocks %>%
  pivot_wider(names_from = year, values_from = return) %>%
  pivot_longer(`2015`:`2016`, names_to = "year", values_to = "return",
              names_transform = list(year = as.numeric )) %>%
  select(year, half, return) %>%
  arrange(year)
```

```
## # A tibble: 4 x 3
##   year half return
##   <dbl> <dbl> <dbl>
## 1  2015      1  1.88
## 2  2015      2  0.59
## 3  2016      1  0.92
## 4  2016      2  0.17
```

3. ¿Por qué esto no funciona?

```
table4a %>%
  pivot_longer(c(1999, 2000), names_to = "year", values_to = "cases")
```

R/

Las columnas (variables) son caracteres, por tanto debemos usar:

```
table4a %>%
  pivot_longer(c(`1999`, `2000`), names_to = "year", values_to = "cases")
```

4. ¿Qué pasa si ampliamos esta tabla?

```
people <- tribble(
  ~name,          ~names, ~values,
  #-----/-----/-----
  "Phillip Woods", "age",    45,
  "Phillip Woods", "height", 186,
  "Phillip Woods", "age",    50,
  "Jessica Cordero", "age",   37,
  "Jessica Cordero", "height", 156
)
```

R/

```
pivot_wider(people, names_from="name", values_from = "values")
```

```
## Warning: Values are not uniquely identified; output will contain list-cols.
## * Use `values_fn = list` to suppress this warning.
## * Use `values_fn = length` to identify where the duplicates arise
## * Use `values_fn = {summary_fun}` to summarise duplicates

## # A tibble: 2 x 3
##   names `Phillip Woods` `Jessica Cordero`
##   <chr> <list>          <list>
## 1 age   <dbl [2]>          <dbl [1]>
## 2 height <dbl [1]>          <dbl [1]>
```

La opción de arriba no nos vale porque `name` no identifica de forma única a cada fila. Si pasamos los nombres a las columnas:

```
people %>%
  group_by(name, names) %>%
  mutate(obs = row_number()) %>%
  pivot_wider(names_from = "name", values_from = "values")
```

```
## # A tibble: 3 x 4
## # Groups:   names [2]
##   names  obs `Phillip Woods` `Jessica Cordero`
##   <chr> <int>          <dbl>          <dbl>
## 1 age     1           45             37
## 2 height  1          186             156
## 3 age     2           50              NA
```

Observamos que algunos outliers implícitos son ahora explícitos.

4.3 Separar y unir

La `table3` tiene una columna `rate` con los casos y la población. Evidentemente, esta proporción no es realmente útil porque no está calculada. Con `separate()` podemos “partirla” en dos nuevas columnas con la información que deseamos:

```
table3 %>%
  separate(rate, into = c("cases", "population"))
```

```
## # A tibble: 6 x 4
##   country      year cases population
##   <chr>      <int> <chr>    <chr>
## 1 Afghanistan 1999   745    19987071
## 2 Afghanistan 2000  2666    20595360
## 3 Brazil      1999 37737    172006362
## 4 Brazil      2000 80488    174504898
## 5 China       1999 212258   1272915272
## 6 China       2000 213766   1280428583
```

Automáticamente, la función separa los datos cuando encuentra algún carácter no alfanumérico. Esto se puede personalizar:

```
table3 %>%
  separate(rate, into = c("cases", "population"), sep = "/")
```

```
## # A tibble: 6 x 4
##   country      year cases population
##   <chr>      <int> <chr>    <chr>
## 1 Afghanistan 1999   745    19987071
## 2 Afghanistan 2000  2666    20595360
## 3 Brazil      1999 37737    172006362
## 4 Brazil      2000 80488    174504898
## 5 China       1999 212258   1272915272
## 6 China       2000 213766   1280428583
```

Habrás notado que al separar convierte las nuevas variables a tipo `character`. Para lidiar con esto, podemos decirle a `separate` que encuentre el tipo de datos correspondiente a cada caso:

```
table3 %>%
  separate(rate, into = c("cases", "population"), convert = TRUE)
```

```
## # A tibble: 6 x 4
##   country      year cases population
##   <chr>      <int> <int>      <int>
## 1 Afghanistan 1999     745    19987071
## 2 Afghanistan 2000    2666    20595360
## 3 Brazil      1999   37737    172006362
## 4 Brazil      2000   80488    174504898
```

```
## 5 China      1999 212258 1272915272
## 6 China      2000 213766 1280428583
```

También podemos separar enteros si proporcionamos el número de dígitos a separar:

```
table3 %>%
  separate(year, into = c("first_3", "last_digit"), sep = -1) %>%
  separate(rate, into = c("cases", "population"), convert = TRUE)
```

```
## # A tibble: 6 x 5
##   country    first_3 last_digit cases population
##   <chr>      <chr>   <chr>    <int>      <int>
## 1 Afghanistan 199     9         745    19987071
## 2 Afghanistan 200     0         2666   20595360
## 3 Brazil      199     9        37737   172006362
## 4 Brazil      200     0        80488   174504898
## 5 China       199     9       212258 1272915272
## 6 China       200     0       213766 1280428583
```

```
table3 %>%
  separate(year, into = c("century", "year"), sep = 2) %>%
  separate(rate, into = c("cases", "population"), convert = TRUE)
```

```
## # A tibble: 6 x 5
##   country    century year    cases population
##   <chr>      <chr>   <chr>    <int>      <int>
## 1 Afghanistan 19     99         745    19987071
## 2 Afghanistan 20     00         2666   20595360
## 3 Brazil      19     99        37737   172006362
## 4 Brazil      20     00        80488   174504898
## 5 China       19     99       212258 1272915272
## 6 China       20     00       213766 1280428583
```

Con `unite()` hacemos justamente lo contrario, especificando el separador (por defecto será `_`) que en este caso será un espacio en blanco:

```
table5 %>%
  unite(new, century, year, sep = " ")
```

```
## # A tibble: 6 x 3
##   country    new    rate
##   <chr>      <chr> <chr>
## 1 Afghanistan 1999 745/19987071
## 2 Afghanistan 2000 2666/20595360
## 3 Brazil      1999 37737/172006362
## 4 Brazil      2000 80488/174504898
## 5 China       1999 212258/1272915272
## 6 China       2000 213766/1280428583
```


4.3.1 Ejercicios

5. Experimenta con los argumentos `extra` y `fill` de `separate()`, usando estos datos:

```
tibble(x = c("a,b,c", "d,e,f,g", "h,i,j")) %>%
  separate(x, c("one", "two", "three"))
```

```
## Warning: Expected 3 pieces. Additional pieces discarded in 1 rows [2].
```

```
## # A tibble: 3 x 3
##   one   two  three
##   <chr> <chr> <chr>
## 1 a     b     c
## 2 d     e     f
## 3 h     i     j
```

```
tibble(x = c("a,b,c", "d,e", "f,g,i")) %>%
  separate(x, c("one", "two", "three"))
```

```
## Warning: Expected 3 pieces. Missing pieces filled with `NA` in 1 rows [2].
```

```
## # A tibble: 3 x 3
##   one   two  three
##   <chr> <chr> <chr>
## 1 a     b     c
## 2 d     e    <NA>
## 3 f     g     i
```

R/

El argumento `extra` le dice a `separate()` qué hacer si hay muchos elementos. Por defecto, los valores extra son descartados con un *Warning*, como habrás notado en los códigos arriba. Por ejemplo, si usamos `extra = "merge"` podemos retener los elementos adicionales que no puedan ser separados en solo 3 columnas:

```
tibble(x = c("a,b,c", "d,e,f,g", "h,i,j")) %>%
  separate(x, c("one", "two", "three"), extra = "merge")
```

```
## # A tibble: 3 x 3
##   one   two  three
##   <chr> <chr> <chr>
## 1 a     b     c
## 2 d     e   f,g
## 3 h     i     j
```

Por otro lado, `fill` rellena los espacios en blanco de acuerdo al criterio que se fije. Por defecto, rellena con `NA` si no hay suficientes elementos como para separar en la cantidad de columnas que hemos pedido. Podemos variar este comportamiento con los argumentos `fill = "left"` (rellenar a la izquierda con `NAs`) o `fill = "right"` (rellenar a la derecha con `NAs`), sin emitir *Warnings*:

```
tibble(x = c("a,b,c", "d,e", "f,g,i")) %>%
  separate(x, c("one", "two", "three"), fill = "left")
```

```
## # A tibble: 3 x 3
##   one   two  three
##   <chr> <chr> <chr>
## 1 a     b     c
## 2 <NA> d     e
## 3 f     g     i
```

```
tibble(x = c("a,b,c", "d,e", "f,g,i")) %>%
  separate(x, c("one", "two", "three"), fill = "right")
```

```
## # A tibble: 3 x 3
##   one   two  three
##   <chr> <chr> <chr>
## 1 a     b     c
## 2 d     e    <NA>
## 3 f     g     i
```

4.4 Lidar con los datos faltantes

Habrás notado que al cambiar la forma en que presentamos los datos, pueden aparecer valores perdidos (NAs). Estos perdidos pueden ser de dos formas:

1. Explícitos, cuando vemos un NA en los datos.
2. Implícitos, cuando no están presentes en los datos.

¿Podrías identificarlos aquí? **R/** Explícito es el valor de **return** en el cuatrimestre 4 del 2015. Implícito el caso del 1er cuatrimestre del 2016.

```
stocks <- tibble(
  year   = c(2015, 2015, 2015, 2015, 2016, 2016, 2016),
  qtr    = c( 1,    2,    3,    4,    2,    3,    4),
  return = c(1.88, 0.59, 0.35, NA, 0.92, 0.17, 2.66)
)
```

Observa cómo los implícitos pasan a ser explícitos:

```
stocks %>%
  pivot_wider(names_from = year, values_from = return)
```

```
## # A tibble: 4 x 3
##   qtr `2015` `2016`
##   <dbl> <dbl> <dbl>
## 1     1  1.88  NA
## 2     2  0.59  0.92
## 3     3  0.35  0.17
```

```
## 4      4  NA      2.66
```

Si hacemos la operación inversa con `pivot_longer()`, tal vez no deseamos que esos perdidos aparezcan de forma explícita:

```
stocks %>%
  pivot_wider(names_from = year, values_from = return) %>%
  pivot_longer(
    cols = c(`2015`, `2016`),
    names_to = "year",
    values_to = "return",
    values_drop_na = TRUE
  )
```

```
## # A tibble: 6 x 3
##   qtr year  return
##   <dbl> <chr>  <dbl>
## 1     1  2015    1.88
## 2     2  2015    0.59
## 3     2  2016    0.92
## 4     3  2015    0.35
## 5     3  2016    0.17
## 6     4  2016    2.66
```

Por otro lado, si queremos que los perdidos implícitos aparezcan de forma explícita (sí, ¡vaya lío!):

```
stocks %>%
  complete(year, qtr)
```

```
## # A tibble: 8 x 3
##   year  qtr return
##   <dbl> <dbl>  <dbl>
## 1  2015     1   1.88
## 2  2015     2   0.59
## 3  2015     3   0.35
## 4  2015     4    NA
## 5  2016     1    NA
## 6  2016     2   0.92
## 7  2016     3   0.17
## 8  2016     4   2.66
```

4.4.1 Ejercicios

6. Otra función interesante es `fill`. ¿Puedes entender cómo funciona a partir de este ejemplo?

```
treatment <- tribble(
  ~ person,      ~ treatment, ~response,
```

```

"Derrick Whitmore", 1,      7,
NA,                2,      10,
NA,                3,      9,
"Katherine Burke", 1,      4
)

treatment

```

```

## # A tibble: 4 x 3
##   person      treatment response
##   <chr>          <dbl>    <dbl>
## 1 Derrick Whitmore      1         7
## 2 <NA>                 2        10
## 3 <NA>                 3         9
## 4 Katherine Burke      1         4

treatment %>%
  fill(person)

```

```

## # A tibble: 4 x 3
##   person      treatment response
##   <chr>          <dbl>    <dbl>
## 1 Derrick Whitmore      1         7
## 2 Derrick Whitmore      2        10
## 3 Derrick Whitmore      3         9
## 4 Katherine Burke      1         4

```

R/ Rellena los faltantes con el valor no faltante anterior.

7. ¿Para qué sirve el argumento `direction` de `fill()`?

R/

Te permite escoger con cuál valor rellenar. Por ejemplo, tomando el siguiente valor:

```

treatment %>%
  fill(person, .direction = "up")

```

```

## # A tibble: 4 x 3
##   person      treatment response
##   <chr>          <dbl>    <dbl>
## 1 Derrick Whitmore      1         7
## 2 Katherine Burke      2        10
## 3 Katherine Burke      3         9
## 4 Katherine Burke      1         4

```

4.5 Case study

Vamos con unos datos reales. En este caso, usaremos el dataset `who` de `dplyr`, con información sobre el número de casos de TB en el 2014, proporcionados por la Organización Mundial de la Salud (OMS, o WHO en inglés).

```
data("who")
```

El primer paso es crear una nueva columna auxiliar para agrupar las categorías `new_sp_m014` a `new_rel_f65`, que no parecen ser variables:

```
who1 <- who %>%
  pivot_longer(
    cols = new_sp_m014:newrel_f65,
    names_to = "key",
    values_to = "cases",
    values_drop_na = TRUE
  )
who1
```

```
## # A tibble: 76,046 x 6
##   country    iso2 iso3   year key          cases
##   <chr>      <chr> <chr> <int> <chr>      <int>
## 1 Afghanistan AF    AFG   1997 new_sp_m014      0
## 2 Afghanistan AF    AFG   1997 new_sp_m1524     10
## 3 Afghanistan AF    AFG   1997 new_sp_m2534      6
## 4 Afghanistan AF    AFG   1997 new_sp_m3544      3
## 5 Afghanistan AF    AFG   1997 new_sp_m4554      5
## 6 Afghanistan AF    AFG   1997 new_sp_m5564      2
## 7 Afghanistan AF    AFG   1997 new_sp_m65       0
## 8 Afghanistan AF    AFG   1997 new_sp_f014      5
## 9 Afghanistan AF    AFG   1997 new_sp_f1524     38
## 10 Afghanistan AF    AFG   1997 new_sp_f2534     36
## # ... with 76,036 more rows
```

Antes de separar la columna `key`, de acuerdo a la información consultada en la ayuda `?who`, tenemos que lidiar con unos *tipos* muy difíciles de observar: hay cierta inconsistencia entre `new_rel` y `newrel`. Para resolver esto, solo tenemos que emplear una de las funciones de `stringr`... **Arréglalo y guarda los datos en un nuevo tibble `who2`.**

```
who2 <- who1 %>%
  mutate(key = stringr::str_replace(key, "newrel", "new_rel"))
who2
```

```
## # A tibble: 76,046 x 6
##   country    iso2 iso3   year key          cases
##   <chr>      <chr> <chr> <int> <chr>      <int>
## 1 Afghanistan AF    AFG   1997 new_sp_m014      0
```

```
## 2 Afghanistan AF AFG 1997 new_sp_m1524 10
## 3 Afghanistan AF AFG 1997 new_sp_m2534 6
## 4 Afghanistan AF AFG 1997 new_sp_m3544 3
## 5 Afghanistan AF AFG 1997 new_sp_m4554 5
## 6 Afghanistan AF AFG 1997 new_sp_m5564 2
## 7 Afghanistan AF AFG 1997 new_sp_m65 0
## 8 Afghanistan AF AFG 1997 new_sp_f014 5
## 9 Afghanistan AF AFG 1997 new_sp_f1524 38
## 10 Afghanistan AF AFG 1997 new_sp_f2534 36
## # ... with 76,036 more rows
```

Ahora vamos a hacer dos pases de `separate()` . Primero, separamos todo lo que esté unido por `_`:

```
who3 <- who2 %>%
  separate(key, c("new", "type", "sexage"), sep = "_")
who3
```

```
## # A tibble: 76,046 x 8
##   country    iso2 iso3  year new  type sexage cases
##   <chr>      <chr> <chr> <int> <chr> <chr> <chr> <int>
## 1 Afghanistan AF AFG 1997 new  sp  m014 0
## 2 Afghanistan AF AFG 1997 new  sp  m1524 10
## 3 Afghanistan AF AFG 1997 new  sp  m2534 6
## 4 Afghanistan AF AFG 1997 new  sp  m3544 3
## 5 Afghanistan AF AFG 1997 new  sp  m4554 5
## 6 Afghanistan AF AFG 1997 new  sp  m5564 2
## 7 Afghanistan AF AFG 1997 new  sp  m65 0
## 8 Afghanistan AF AFG 1997 new  sp  f014 5
## 9 Afghanistan AF AFG 1997 new  sp  f1524 38
## 10 Afghanistan AF AFG 1997 new  sp  f2534 36
## # ... with 76,036 more rows
```

Antes del segundo pase, elimina lo que no te interesa: `new`, `iso2` e `iso3`. Cuando lo hayas hecho, guarda los nuevos datos en `who4`, y hacemos al segundo `separate()` para obtener el sexo y rangos de edades por separado:

```
who4 <- who3 %>%
  select(-new, -iso2, -iso3)

who5 <- who4 %>%
  separate(sexage, c("sex", "age"), sep = 1)
who5
```

```
## # A tibble: 76,046 x 6
##   country    year type sex  age  cases
##   <chr>      <int> <chr> <chr> <chr> <int>
## 1 Afghanistan 1997 sp  m   014 0
## 2 Afghanistan 1997 sp  m  1524 10
```

```
## 3 Afghanistan 1997 sp m 2534 6
## 4 Afghanistan 1997 sp m 3544 3
## 5 Afghanistan 1997 sp m 4554 5
## 6 Afghanistan 1997 sp m 5564 2
## 7 Afghanistan 1997 sp m 65 0
## 8 Afghanistan 1997 sp f 014 5
## 9 Afghanistan 1997 sp f 1524 38
## 10 Afghanistan 1997 sp f 2534 36
## # ... with 76,036 more rows
```

4.5.1 Ejercicios

8. Escribe todas las transformaciones con un único *pipe*.

R/

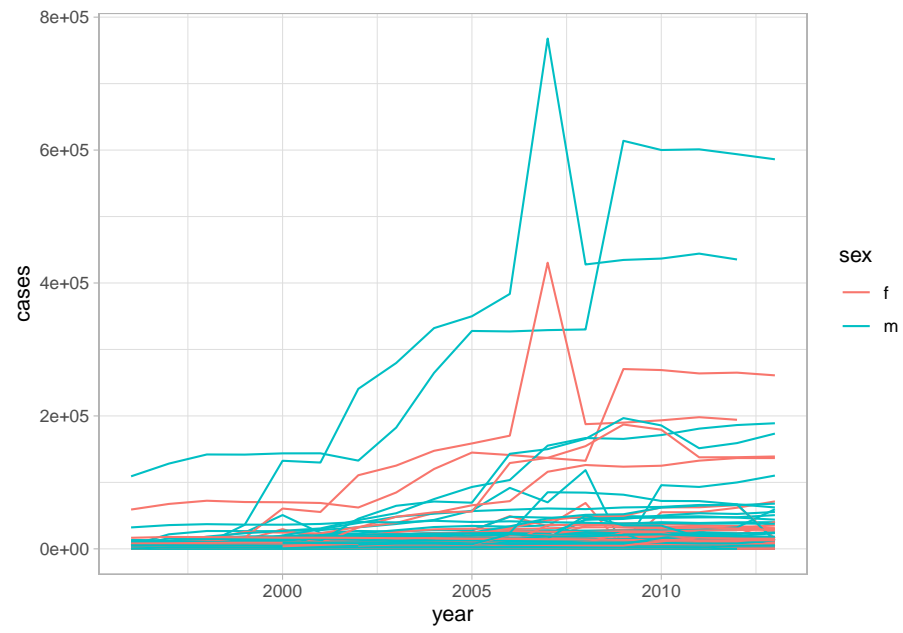
```
who_piped <- who %>%
  pivot_longer(
    cols = new_sp_m014:newrel_f65,
    names_to = "key",
    values_to = "cases",
    values_drop_na = TRUE
  ) %>%
  mutate(
    key = stringr::str_replace(key, "newrel", "new_rel")
  ) %>%
  separate(key, c("new", "var", "sexage")) %>%
  select(-new, -iso2, -iso3) %>%
  separate(sexage, c("sex", "age"), sep = 1)
```

9. Para cada país, año y sexo calcula el número total de casos de TB. Haz un plot de los resultados, de la forma que consideres más informativa.

R/

```
who5 %>%
  group_by(country, year, sex) %>%
  filter(year > 1995) %>%
  summarise(cases = sum(cases)) %>%
  unite(country_sex, country, sex, remove = FALSE) %>%
  ggplot(aes(x = year, y = cases, group = country_sex, colour = sex)) +
  geom_line() +
  theme_light()
```

`summarise()` has grouped output by 'country', 'year'. You can override using the `.groups` argument



Chapter 5

Relational Data

5.1 Datos

Vamos a trabajar con los datos de `nycflights13`: `airlines`, `airports`, `planes`, `weather`

```
library(tidyverse)
library(nycflights13)
data("airlines", "airports", "planes", "weather")
```

Las relaciones entre ellos se resumen en:

- `flights` con `planes` a través de la variable `tailnum`.
- `flights` con `airlines` a través de la variable `carrier`.
- `flights` con `airports` a través de las variables `origin` y `dest`.
- `flights` con `weather` a través de las variables `origin` (lugar) y `year`, `month`, `day`, `hour` (fecha + hora).

5.2 Keys

La variable `tailnum` es un identificador único de cada avión para los datos `planes`:

```
planes %>%
  count(tailnum) %>%
  filter(n > 1)
```

```
## # A tibble: 0 x 2
## # ... with 2 variables: tailnum <chr>, n <int>
```

Algunas tablas no tienen un *key* primario. ¿Qué crees de estos casos? ¿Tienen sentido? ¿Se te ocurre alguna otra combinación de variables que pueda identificar

de forma única a cada observación?

```
flights %>%
  count(year, month, day, flight) %>%
  filter(n > 1)

flights %>%
  count(year, month, day, tailnum) %>%
  filter(n > 1)
```

Recuerda que podemos añadir una *surrogate key*:

```
flights %>%
  arrange(year, month, day, sched_dep_time, carrier, flight) %>%
  mutate(flight_id = row_number()) %>%
  glimpse()
```

5.3 Mutating Joins

Primero vamos a reducir un poco la cantidad de columnas de `flights` para notar las columnas añadidas:

```
flights2 <- flights %>%
  select(year:day, hour, origin, dest, tailnum, carrier)
```

Veamos cómo definir la *key*/clave de referencia:

- Por defecto: `by = NULL` usa las variables comunes a ambas tablas:

```
flights2 %>%
  left_join(weather)
```

- Podemos introducir un vector de caracteres `by = x`, donde `x` es alguna de las columnas en común. A continuación, lo hacemos para `by = tailnum`. ¿Qué son `year.x` y `year.y`?

```
flights2 %>%
  left_join(planes, by = "tailnum")
```

- Podemos introducir un vector de caracteres con nombre: `by = c("a" = "b")`. Esto empareja las variables `a` (de la tabla `x`) y `b` (de la tabla `y`). Por ejemplo, para combinar `flights` y `airports` necesitamos combinar el destino (`dest`) u origen (`origin`) en `flights` con el código de cada aeropuerto (`faa`) en `airports`:

```
flights2 %>%
  left_join(airports, c("dest" = "faa"))

flights2 %>%
```

```
left_join(airports, c("origin" = "faa"))
```

5.3.1 Ejercicios:

1. Añadir latitud y longitud (lat y lon) del origen y destino a la tabla flights.

R/

```
airport_locations <- airports %>%
  select(faa, lat, lon)

flights %>%
  select(year:day, hour, origin, dest) %>%
  left_join(
    airport_locations,
    by = c("origin" = "faa")
  ) %>%
  left_join(
    airport_locations,
    by = c("dest" = "faa")
  )
```

```
## # A tibble: 336,776 x 10
##   year month   day hour origin dest lat.x lon.x lat.y lon.y
##   <int> <int> <int> <dbl> <chr> <chr> <dbl> <dbl> <dbl> <dbl>
## 1  2013     1     1     5 EWR   IAH  40.7 -74.2  30.0 -95.3
## 2  2013     1     1     5 LGA   IAH  40.8 -73.9  30.0 -95.3
## 3  2013     1     1     5 JFK   MIA  40.6 -73.8  25.8 -80.3
## 4  2013     1     1     5 JFK   BQN  40.6 -73.8   NA    NA
## 5  2013     1     1     6 LGA   ATL  40.8 -73.9  33.6 -84.4
## 6  2013     1     1     5 EWR   ORD  40.7 -74.2  42.0 -87.9
## 7  2013     1     1     6 EWR   FLL  40.7 -74.2  26.1 -80.2
## 8  2013     1     1     6 LGA   IAD  40.8 -73.9  38.9 -77.5
## 9  2013     1     1     6 JFK   MCO  40.6 -73.8  28.4 -81.3
## 10 2013     1     1     6 LGA   ORD  40.8 -73.9  42.0 -87.9
## # ... with 336,766 more rows
```

```
airport_locations <- airports %>%
  select(faa, lat, lon)

flights %>%
  select(year:day, hour, origin, dest) %>%
  left_join(
    airport_locations,
    by = c("origin" = "faa")
  ) %>%
```

```

left_join(
  airport_locations,
  by = c("dest" = "faa"),
  suffix = c("_origin", "_dest")
  # las variables lat y lon existentes en el tibble toman el sufijo _origin
  # las nuevas lat y lon tendrán el sufijo _dest
)

```

```

## # A tibble: 336,776 x 10
##   year month   day hour origin dest lat_origin lon_origin lat_dest lon_dest
##   <int> <int> <int> <dbl> <chr> <chr>   <dbl>   <dbl>   <dbl>   <dbl>
## 1  2013     1     1     5 EWR   IAH     40.7    -74.2    30.0   -95.3
## 2  2013     1     1     5 LGA   IAH     40.8    -73.9    30.0   -95.3
## 3  2013     1     1     5 JFK   MIA     40.6    -73.8    25.8   -80.3
## 4  2013     1     1     5 JFK   BQN     40.6    -73.8     NA     NA
## 5  2013     1     1     6 LGA   ATL     40.8    -73.9    33.6   -84.4
## 6  2013     1     1     5 EWR   ORD     40.7    -74.2    42.0   -87.9
## 7  2013     1     1     6 EWR   FLL     40.7    -74.2    26.1   -80.2
## 8  2013     1     1     6 LGA   IAD     40.8    -73.9    38.9   -77.5
## 9  2013     1     1     6 JFK   MCO     40.6    -73.8    28.4   -81.3
## 10 2013     1     1     6 LGA   ORD     40.8    -73.9    42.0   -87.9
## # ... with 336,766 more rows

```

5.4 Filtering Joins

Los semi-joins son útiles cuando hacemos un resumen de los datos y luego queremos emparejar estos resultados con las observaciones originales. Por ejemplo, si calculamos los 10 destinos más populares:

```

top_dest <- flights %>%
  count(dest, sort = TRUE) %>%
  head(10)
top_dest

```

```

## # A tibble: 10 x 2
##   dest      n
##   <chr> <int>
## 1 ORD   17283
## 2 ATL   17215
## 3 LAX   16174
## 4 BOS   15508
## 5 MCO   14082
## 6 CLT   14064
## 7 SFO   13331
## 8 FLL   12055
## 9 MIA   11728

```

```
## 10 DCA      9705
```

... y luego queremos encontrar todos los vuelos (en `flights`) que tuvieron este destino:

```
flights %>%
  semi_join(top_dest)
```

Los anti-joins son útiles para diagnosticar las discrepancias en las uniones. Por ejemplo, en `planes` hay aviones que no aparecen en `flights`:

```
flights %>%
  anti_join(planes, by = "tailnum") %>%
  count(tailnum, sort = TRUE)
```

5.4.1 Ejercicios

2. Encuentra otra forma de obtener el mismo resultado que:

```
flights %>%
  semi_join(top_dest)
```

sin usar `semi_join()`. Hint: Filtrar los destinos de `flights` de acuerdo a los 10 más populares.

R/

```
flights %>%
  filter(dest %in% top_dest$dest)
```

```
## # A tibble: 141,145 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     542           540           2     923           850
## 2  2013     1     1     554           600          -6     812           837
## 3  2013     1     1     554           558          -4     740           728
## 4  2013     1     1     555           600          -5     913           854
## 5  2013     1     1     557           600          -3     838           846
## 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           559           0     702           706
## 10 2013     1     1     600           600           0     851           858
## # ... with 141,135 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 <dtm>
```

3. Filtra `flights` para que solo recoja los datos de aquellos aviones que han volado al menos 100 veces.

R/

```

# usando semi-join
planes_gte100 <- flights %>%
  filter(!is.na(tailnum)) %>%
  group_by(tailnum) %>%
  count() %>%
  filter(n >= 100)

flights %>%
  semi_join(planes_gte100, by = "tailnum")

## # A tibble: 228,390 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     544             545          -1    1004            1022
## 4  2013     1     1     554             558          -4     740             728
## 5  2013     1     1     555             600          -5     913             854
## 6  2013     1     1     557             600          -3     709             723
## 7  2013     1     1     557             600          -3     838             846
## 8  2013     1     1     558             600          -2     849             851
## 9  2013     1     1     558             600          -2     853             856
## 10 2013     1     1     558             600          -2     923             937
## # ... with 228,380 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 <dtm>

# sin semi-join
flights %>%
  filter(!is.na(tailnum)) %>%
  group_by(tailnum) %>%
  mutate(n = n()) %>%
  filter(n >= 100)

## # A tibble: 228,390 x 20
## # Groups:   tailnum [1,217]
##   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     544             545          -1    1004            1022
## 4  2013     1     1     554             558          -4     740             728
## 5  2013     1     1     555             600          -5     913             854
## 6  2013     1     1     557             600          -3     709             723
## 7  2013     1     1     557             600          -3     838             846

```

```
## 8 2013      1      1      558          600      -2      849          851
## 9 2013      1      1      558          600      -2      853          856
## 10 2013     1      1      558          600      -2      923          937
## # ... with 228,380 more rows, and 12 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 <dtm>,
## #   n <int>
```