Introducción a la modelización con R

Primeros pasos en la regresión, la clasificación y en modelos no supervisados

Leonardo Hansa

Julio 2022

Contents

Chapter 1

Bienvenida

Nos adentramos en una breve e informal introducción a la modelización con R.
Por favor, instalar estos paquetes, si no los tiene aún:

```
needed_packages <- c("tidyverse",
                     "randomForests",
                     "ggRandomForests",
                     "e1071",
                     "tree",
                     "rpart",
                     "rattle",
                     "rpart.plot",
                     "tree",
                     "ROCR",
                     "pROC",
                     "caret",
                     "factoextra",
                     "broom",
                     "car",
                     "lmtest",
                     "corrplot",
                     "MASS")

not_installed <- needed_packages[!needed_packages %in% row.names(installed.packages())]

install.packages(not_installed)
```

Chapter 2

Introducción

Un modelo es una aproximación a la realidad. Se puede ver con una persona cuyas fotos ponen en catalógos de tiendas de moda, de manera que sirvan para hacerse una idea de cómo nos quedará la próxima prenda que nos compremos. Hay gente más agraciada, a la que esos modelos les sirvan como referencia, y gente a la que no.

En nuestro ámbito, un modelo es una expresión matemática que relaciona información de la vida real.

$$y = \alpha + \beta \cdot x + \varepsilon$$

suele usarse para representar una relación entre dos variables, x e y. Por ejemplo, y podrían ser las ventas de un producto y x, la inversión publicitaria que se ha hecho sobre ese producto. Estaríamos diciendo con nuestro modelo que las ventas dependen de la inversión publicitaria que hagamos, aparte de un cierto valor indendiente de ello (α, que podría ser, por ejemplo, la media de las ventas, de forma que la intervención de la publicidad hace variar las ventas sobre su media). También estaría ε, un error, porque un modelo nunca se ajusta perfectamente.

En estas páginas aprenderemos a ajustar modelos con R.

Todas estas notas se han escrito con el paquete **bookdown** (Xie, 2022) y **knitr** (Xie, 2015).

Chapter 3

Regresión

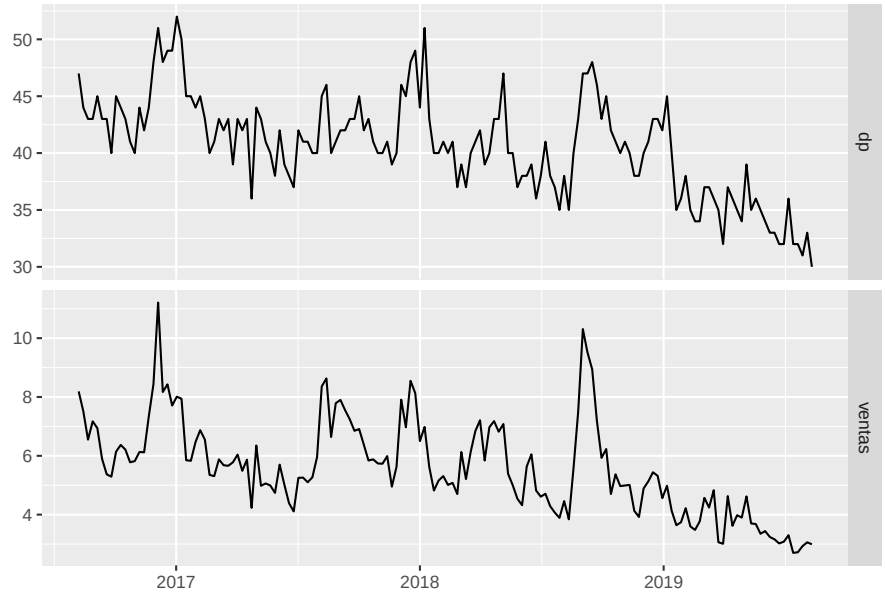
3.1 Relación entre variables

En una empresa de gran consumo quieren estudiar los factores que mueven las ventas de un determinado producto.

Aunque son muchas las variables que pueden influir, vamos a realizar un primer análisis con un único factor: la distribución del producto. Luego añadiremos más.

¿Métodos posibles? Todos. Se nos desbordaría la cabeza si nos pusiéramos a buscar todas las posibilidades en internet. Así que vamos a centrarnos. En primer lugar, algo intuitivo: un gráfico.

```
library(ggplot2)
library(dplyr)
library(tidyr)

df_consumo %>%
  select(fecha, ventas, dp) %>%
  pivot_longer(-fecha) %>%
  ggplot() +
  geom_line(aes(x = fecha, y = value)) +
  facet_grid(name ~ ., scales = "free_y") +
  labs(x = "", y = "")
```



A simple vista, algo de relación parece que tienen: por lo menos, las dos bajan. Podemos recurrir al coeficiente de correlación de Pearson, que algo numérico puede resultar más fiable que echar un ojo, sin más.

Este número va a cuantificar cuánto acompaña una variable a la otra en las subidas y las bajadas. Si en general cuando una sube (o baja) la otra también lo hace, entonces el valor será cercano 1. Si cuando una sube (o baja) la otra hace lo contrario, entonces el valor será cercano a -1. Si los comportamientos tienen poco que ver, entonces cercano a 0.

```
cor(df_consumo$ventas, df_consumo$dp)
```

```
## [1] 0.8792131
```

Es un valor bastante alto. Sugiere que hay algo de relación.

Cuidado. Que la correlación sea alta no signfica que uua variable influya en la otra. Significa que los datos muestran que, como ya hemos dicho, cuando una sube (o baja) la otra hace lo mismo. Pero puede ser casualidad. La relación real entre ambas variables se tiene que extraer, finalmente, del sentido común (acompañado del conocimiento que se tenga en el campo de estudio, claro: no vale inventarse todo).

3.2 Regresión lineal

Como ya hemos visto que las variables van de la mano, vamos a cuantificar su relación con una regresión lineal. Dada una variable y (ventas de un producto) y otra x (distribución del producto), una regresión lineal nos permite construir una expresión de este tipo:

$$y = \beta_0 + \beta_1 x + \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, \sigma^2),$$

donde β_0 y β_1 son números que cuantifican la relación de x sobre y. Cuando calculemos $\beta_0 + \beta_1 x$ tendremos una aproximación a y, pero habrá alguna diferencia. Ese error en la estimación es lo que se representa por *varepsilon* y tiene que cumplir algunos requisitos que veremos más adelante (que se resumen en la expresión de la derecha).

```
fit_lm <- lm(ventas ~ dp, data = df_consumo)
summary(fit_lm)
```

```
##
## Call:
## lm(formula = ventas ~ dp, data = df_consumo)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.9933 -0.4537 -0.1023  0.3600  2.7006
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -7.3406     0.5644  -13.01   <2e-16 ***
## dp            0.3181     0.0138   23.05   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7626 on 156 degrees of freedom
## Multiple R-squared:  0.773,  Adjusted R-squared:  0.7716
## F-statistic: 531.3 on 1 and 156 DF,  p-value: < 2.2e-16
```

En esta salida observamos la estimación de los coeficientes β_0 (-7.3406) y β_1 (0.3181). Además realiza algunos test de hipótesis sobre los coeficientes y nos devuelve los p-valores de estos tests, así como el *R2*, un indicador de la calidad del ajuste.

Vamos a introducir más información y luego entramos en el detalle de la interpretación.

3.2.1 Regresión lineal múltiple

Si incluimos varias variables explicativas, la expresión pasa a ser algo así:

$$y = \beta_0 + \sum_{i=1}^{p} \beta_i x_i + \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, \sigma^2),$$

donde p es el número de variables explicativas (o predictores) incluidas en el modelo.

Ajustar un modelo es un proceso interactivo que lleva un rato. No se trata de meter toda la información disponible y ver qué sale. Presentamos aquí un modelo con parte de información que no pretende ser el modelo definitivo, sino que sirva de guía para lo que queremos exponer a continuación: a saber, sintaxis e interpretación de resultados.

```
fit_lm <- lm(ventas ~
     + 1 # Cuidado
   + p2ola1_ad40
   + p3_ola1_ad40
   + p4_ola1_ad40
   + p5_ola1_ad40
   + p1_ola2_ad40
   + p1_ola3_ad40
   + dp
   + competencia1
   + competencia2
   , data = df_consumo)

summary(fit_lm)
```

```
##
## Call:
## lm(formula = ventas ~ +1 + p2ola1_ad40 + p3_ola1_ad40 + p4_ola1_ad40 +
##     p5_ola1_ad40 + p1_ola2_ad40 + p1_ola3_ad40 + dp + competencia1 +
##     competencia2, data = df_consumo)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.14202 -0.34507 -0.09212  0.25147  1.87867
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -5.2851497  0.4408759 -11.988  < 2e-16 ***
## p2ola1_ad40   0.0078508  0.0013079   6.003 1.43e-08 ***
```

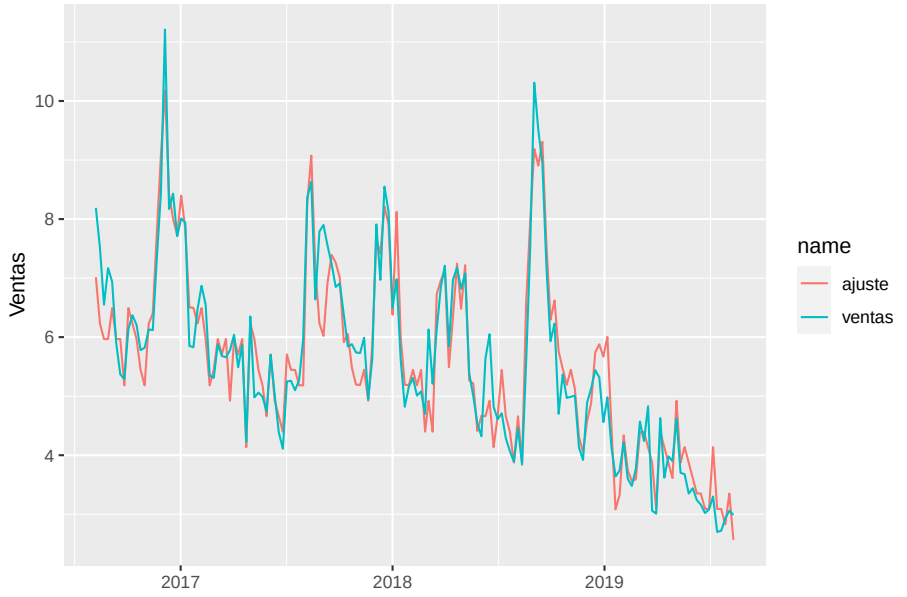
```
## p3_ola1_ad40  0.0039914  0.0005200   7.676 2.05e-12 ***
## p4_ola1_ad40  0.0050948  0.0008755   5.819 3.53e-08 ***
## p5_ola1_ad40  0.0057182  0.0008666   6.598 6.94e-10 ***
## p1_ola2_ad40  0.0045156  0.0011075   4.077 7.41e-05 ***
## p1_ola3_ad40  0.0029815  0.0009479   3.145  0.00201 **
## dp            0.2617109  0.0110016  23.789  < 2e-16 ***
## competencia1 -0.0027755  0.0013679  -2.029  0.04424 *
## competencia2 -0.0031153  0.0011652  -2.674  0.00835 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5353 on 148 degrees of freedom
## Multiple R-squared:  0.8939, Adjusted R-squared:  0.8874
## F-statistic: 138.5 on 9 and 148 DF,  p-value: < 2.2e-16
```

Exploraremos el resultado componente a componente.

3.2.2 Resultados. R2

El R^2 es una medida que cuantifica cuán bien se ajusta el modelo a la realidad. Toma valor de 0 a 1 y se puede interpretar cómo la proporción (o porcentaje) de variabilidad explicada. Dicho de otra forma, si nos preguntamos *¿qué proporción de realidad explica nuestro modelo?*, la respuesta será $R2$.

```
df_consumo %>%
  select(fecha, ventas) %>%
  mutate(ajuste = fit_lm$fitted.values) %>%
  pivot_longer(-fecha) %>%
  ggplot() +
  geom_line(aes(x = fecha, y = value, col = name)) +
  labs(x = "", y = "Ventas")
```



El R^2 se muestra en la salida de `summary()` pero se puede calcular manualmente:

```
1 - sum(fit_lm$residuals ^ 2) / sum((df_consumo$ventas - mean(df_consumo$ventas)) ^ 2)
```

```
## [1] 0.8938834
```

Otras medidas interesantes, que cuantifican si nos hemos equivocado mucho o poco, son el MAPE o el MSE.

```
# MAPE
mean(abs((df_consumo$ventas - fit_lm$fitted.values)) / df_consumo$ventas)
```

```
## [1] 0.07166508
```

```
# MSE
mean((df_consumo$ventas - fit_lm$fitted_values) ^ 2)
```

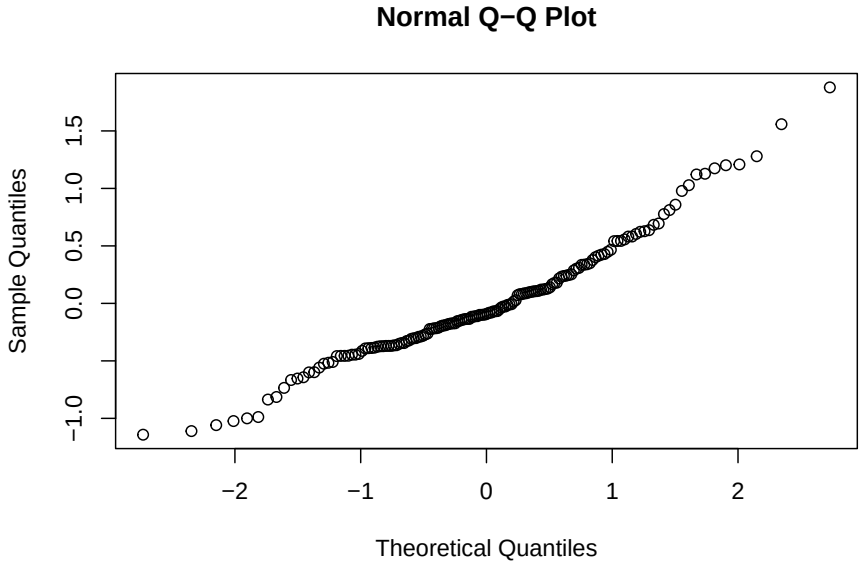
```
## [1] NaN
```

3.2.3 Resultados. Normalidad de residuos

Una de las condiciones que se ponen a la regresión es que los residuos del modelo tienen que ser normales. Con el siguiente gráfico se intenta estudiar si las colas

de la distribución de los residuos no son muy altas (si lo fueran, no se consideraría una distribución normal).

```
qqnorm(fit_lm$residuals)
```



Se puede ejecutar `qqnorm(rnorm(1000, 0, 1))` para tener una referencia de cómo queda este gráfico para una variable normal.

Si tenemos dudas con el gráfico podemos recurrir a algo más robusto, como un test de hipótesis. Con el siguiente código, lanzamos un test que nos ayuda a decidir si, con los datos disponibles, tenemos información para descartar que los residuos sean normales. Si el valor que aparece, el p-valor, es muy bajo (por debajo de 0.05 se suele considerar), rechazamos que sean normales los residuos.

```
shapiro.test(fit_lm$residuals)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  fit_lm$residuals
## W = 0.96295, p-value = 0.00031
```

Como los residuos no son normales, habría que continuar el modelo hasta que se cumpliera esa hipótesis. Se deja como ejercicio para el lector.

3.3 Diagnosis del modelo

Hagamos un ejemplo básico para la diagnosis de un modelo de regresión lineal. Carguemos las librerías `car` y `lmtest`, donde tenemos los tests necesarios.

```
library(car)
library(lmtest)
```

Simulamos un ejemplo muy bueno, donde se cumplan todas las hipótesis:

```
x  <-  1:100
c  <-  sample(c("Madrid","Barcelona"),size=100,replace=T)
c2 <-  ifelse(c=="Madrid",10,3)
y  <-  x*5+rnorm(100,2)+c2
fit<-lm(y~x+c)
```

Observación: Usamos un modelo bueno para saber interpretar correctamente los p-valores de los tests.

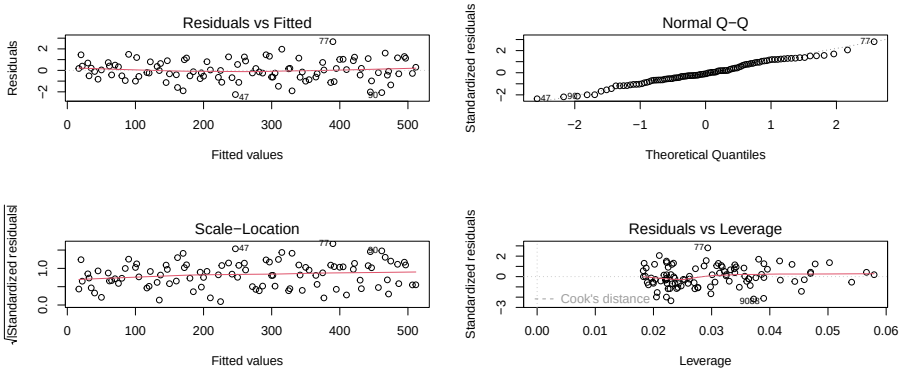
Resumen del modelo:

```
summary(fit)
```

```
##
## Call:
## lm(formula = y ~ x + c)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.25252 -0.62177 -0.09355  0.75163  2.67326
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 5.128783   0.205388   24.97   <2e-16 ***
## x           4.996568   0.003387 1475.13   <2e-16 ***
## cMadrid     6.800535   0.196535   34.60   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.968 on 97 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:      1
## F-statistic: 1.118e+06 on 2 and 97 DF,  p-value: < 2.2e-16
```

Gráficos básicos de diagnosis:

```
par(mfrow=c(2,2))
plot(fit)
```



- Los dos gráficos de la izquierda sirven para ver la homocedasticidad del modelo. No debe haber forma de cono y la recta de tendencia tiene que ser horizontal.
- El gráfico de arriba a la derecha es un gráfico QQ-Plot y sirve para ver si hay normalidad en los residuos. Los puntos tienen que estar cerca de la recta.
- El último gráfico muestra la distancia de Cook y sirve para detectar puntos influyentes.

Veamos cada una de las hipótesis en detalle.

3.3.1 Resultados. Colinealidad

Cuando algunas de las variables regresoras están cerca de ser una combinación lineal, tenemos problemas de multicolinealidad. Una de las maneras de detectar esta multicolinealidad es con el estadístico VIF:

```
library(car)
vif(fit_lm)
table(vif(fit_lm) > 4) # Tabla, no deber?a aparecer TRUE
```

```
##  p2ola1_ad40 p3_ola1_ad40 p4_ola1_ad40 p5_ola1_ad40 p1_ola2_ad40 p1_ola3_ad40
##     1.143835     1.086686     1.018630     1.027647     1.023000     1.078571
##           dp competencia1 competencia2
##     1.289693     1.009968     1.016518
##
```

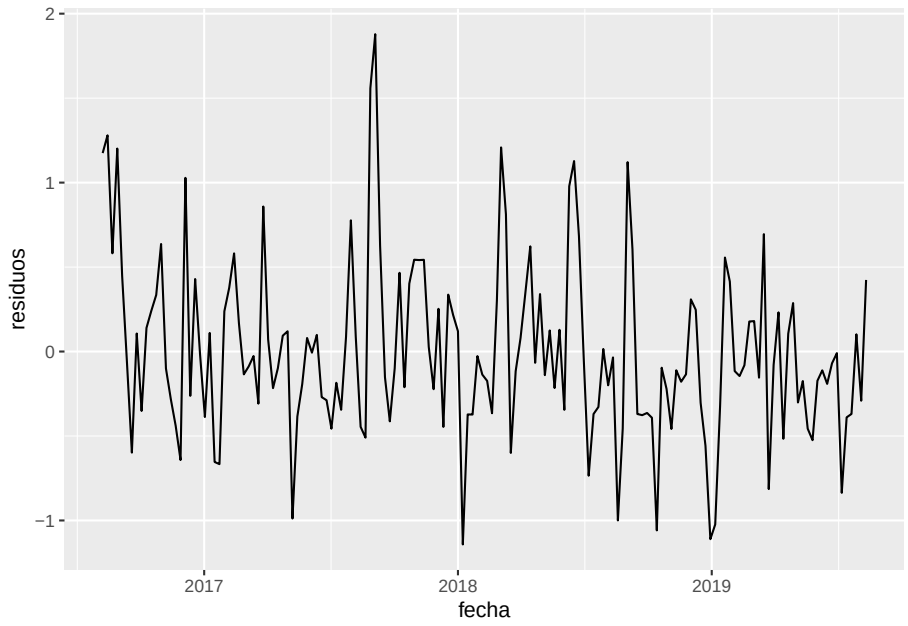
```
## FALSE
##     9
```

Los estadísticos VIF tiene que ser menores que 10 ó 4, de acuerdo con la literatura.

3.3.2 Homocedasticidad

Los residuos del modelo tienen que ser homocedásticos, esto quiere decir que todos tengan la misma variabilidad.Podemos verlo gráficamente con el gráfico general:

```
tibble(fecha = df_consumo$fecha,
       residuos = fit_lm$residuals) %>%
  ggplot() +
  geom_line(aes(x = fecha, y = residuos))
```



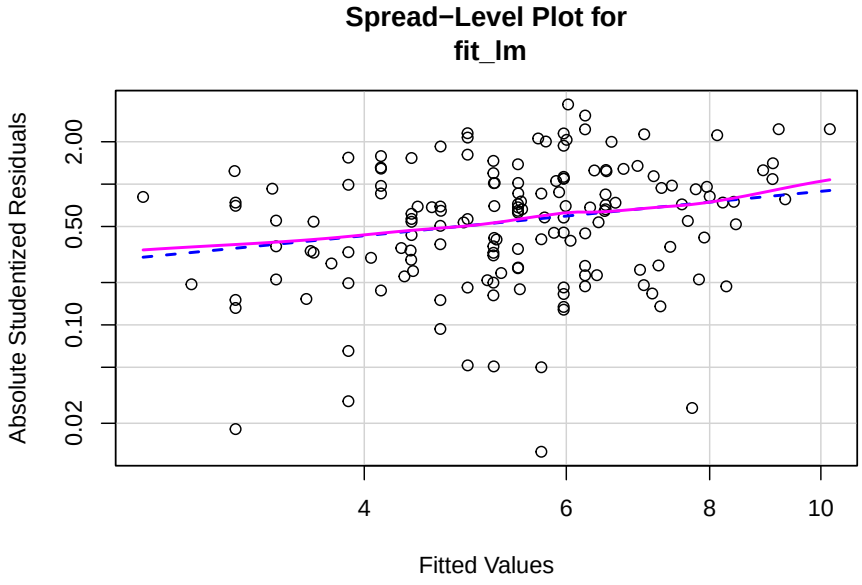
...o con uno más específico

```
library(car)
```

```
## Loading required package: carData
```

```
##
## Attaching package: 'car'

## The following object is masked from 'package:dplyr':
##
##     recode
```

```
spreadLevelPlot(fit_lm)
```




```
##
## Suggested power transformation:  0.2077718
```

En este gráfico tenemos que ver una línea horizontal (así que mal, pero ya dijimos que este no era el modelo definitivo). Además, nos propone una transformación para arreglar el problema. Si la transformación que nos propone está cerca de 1 es bueno. Un 0.5 sería equivalente a realizar una raíz cuadrada. Una transformación habitual es aplicar logaritmos, pero eso llevaría al estudio de modelos multiplicativos, que queda fuera del alcance de este curso.

También tenemos distintos tests para contrastar la hipótesis:

```
library(lmtest)
ncvTest(fit_lm)
```

```
## Non-constant Variance Score Test
## Variance formula: ~ fitted.values
## Chisquare = 6.76172, Df = 1, p = 0.0093134
```

```
bptest(fit_lm)
```

```
##
##  studentized Breusch-Pagan test
##
## data:  fit_lm
## BP = 9.4445, df = 9, p-value = 0.3973
```

El p-valor tiene que ser alto para no rechazar la hipótesis nula y de este modo podremos asumir que se cumple la homocedasticidad.

3.3.3 Autocorrelación

Esta hipótesis hay que testarla sobre todo cuando nuestros datos son longitudinales, *i.e.*, siguen una serie temporal. Por ejemplo, las ventas de un producto durante un año ;)

Usamos el test de Durbin-Watson, en dos versiones implementadas:

```
dwt(fit_lm)
```

```
##  lag Autocorrelation D-W Statistic p-value
##    1       0.3550783       1.25307       0
##  Alternative hypothesis: rho != 0
```

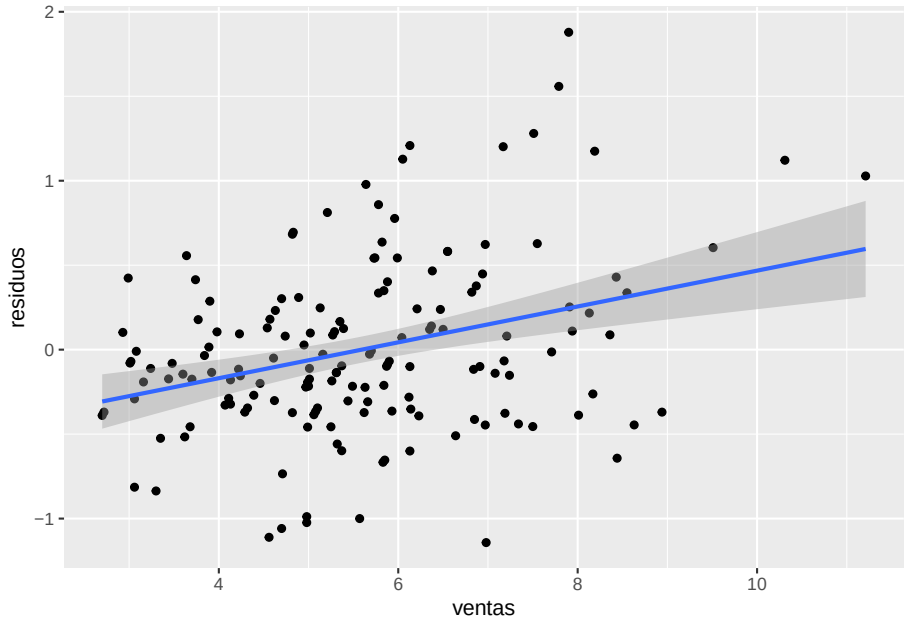
```
dwtest(fit_lm)
```

```
##
##  Durbin-Watson test
##
## data:  fit_lm
## DW = 1.2531, p-value = 4.232e-08
## alternative hypothesis: true autocorrelation is greater than 0
```

Como en los demás tests, p-valores altos nos indican si cumplimos con la hipótesis nula.

También una práctica interesante es dibujar los residuos frente a nuestra variable temporal para encontrar patrones.

```
tibble(ventas = df_consumo$ventas,
       residuos = fit_lm$residuals) %>%
  ggplot(aes(x = ventas, y = residuos)) +
  geom_point() +
  geom_smooth(method = "lm")
```



El gráfico sugiere que a valores más altos de las ventas, el valor de los residuos también aumenta. Por lo tanto, su varianza no se mantiene constante.

3.4 Modelos transversales

Hemos empezado con un modelo sobre una variable que evolucionaba con el tiempo. Es lo que se llama una serie temporal y son modelos habituales en marketing. Se pueden resolver con regresión lineal o con otras técnicas, pero la problemática es frecuente.

La alternativa son los modelos transversales, es decir, que los valores no evolucionan con el tiempo sino que están asociados a otra variable. Por ejemplo, cada valor se asigna a una zona geográfica, a un cliente, a un producto, etc.

Para esta sección vamos a trabajar con `airquality`, un conjunto de datos que podemos cargar así:

```
data(airquality)
```

La documentación de los datos la podemos ver ejecutando `? airquality` en la consola. Contiene métricas asociadas a la calidad del aire en Nueva York en un periodo de tiempo [^1].

[^1] Sí, podríamos tratar estos datos como longitudinales, pero no vamos a tratar la información en su relación con el tiempo, sino entre ellas. De hecho, en el caso de las ventas no hemos llegado a hacer nada temporal tampoco. Análisis temporales suelen incluir dependencias con valores anteriores, estacionalidad (vacaciones, festivos) o acontecimientos históricos (pandemia).

Si echamos un ojo rápido a la cabecera de la tabla nos encontramos con una sorpresa.

```
head(airquality)
```

```
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
## 3    12     149 12.6   74     5   3
## 4    18     313 11.5   62     5   4
## 5    NA      NA 14.3   56     5   5
## 6    28      NA 14.9   66     5   6
```

¡Hay NAs! Horror. Los valores faltantes, *missing* o como se quieran llamar son horribles para los modelos y la mayoría de técnicas no saben trabajar con ellos. En función de los datos disponibles, hay que omitir los registros que tengan algún NA o imputar estos valores (con la media, la mediana, con valores aleatorios de su distribución o haciendo algún modelo en base a otra cosa).

Como nuestro objetivo hoy es ver sintaxis para trabajar con modelos, y no tanto hacer modelos de mucha calidad, hemos optado por eliminarlos. Pero no lo vamos a hacer aún, porque no vamos a trabajar con regresión lineal. ## Árboles de regresión

Una de las técnicas de modelado más fáciles de interpretar son los árboles. Lo que hacen es definir reglas entre variables explicativas que implican un valor en la variable objetivo.

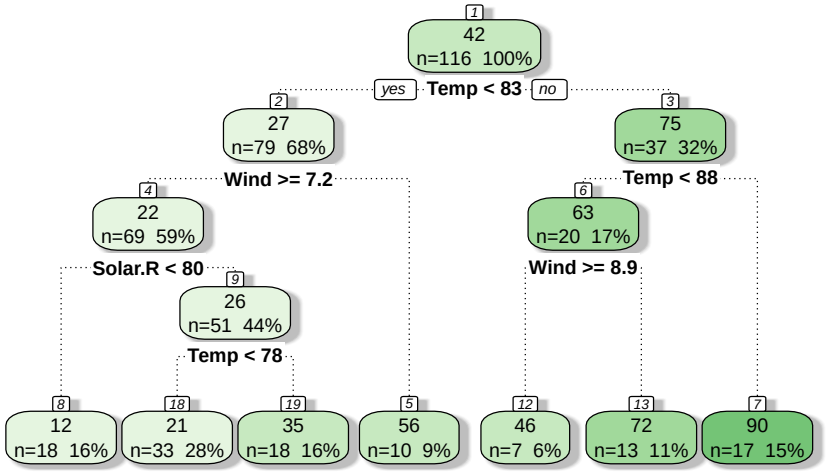
Una forma de plantearlos en R es así:

```
library(rpart)
fit_tree <- rpart(Ozone ~ ., data = airquality)
fit_tree
```

```
## n=116 (37 observations deleted due to missingness)
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 116 125143.1000 42.12931
##    2) Temp< 82.5 79  42531.5900 26.54430
##      4) Wind>=7.15 69  10919.3300 22.33333
##        8) Solar.R< 79.5 18    777.1111 12.22222 *
##        9) Solar.R>=79.5 51   7652.5100 25.90196
##         18) Temp< 77.5 33   2460.9090 21.18182 *
##         19) Temp>=77.5 18   3108.4440 34.55556 *
##      5) Wind< 7.15 10  21946.4000 55.60000 *
##    3) Temp>=82.5 37  22452.9200 75.40541
##      6) Temp< 87.5 20  12046.9500 62.95000
##       12) Wind>=8.9 7    617.7143 45.57143 *
##       13) Wind< 8.9 13   8176.7690 72.30769 *
##      7) Temp>=87.5 17   3652.9410 90.05882 *
```

Aunque queda más visual así:

```
library(rattle)
fancyRpartPlot(fit_tree, caption = NULL)
```



Aunque parezca algo muy básico, realmente hay una buena cantidad de herramientas para ajustar árboles y asegurarse de que se les saca el máximo partido. No obstante, su capacidad predictora no suele ser muy alta, así que vamos directos al siguiente paso.

3.5 Random Forests

Un árbol con muchísimas reglas podría parecer un modelo potente, dado quue si tiene muchas reglas será capaz de predecir bien, ¿no? ¿Eso debería tener sentido? Debiera o no debiera, no ocurre. Porque si creas un árbol con muchas reglas, estás sobreajustado el modelo, es decir, explicando muy detalladamente los datos que tienes delante, pero si lo usas para predecir en un nuevo conjunto de datos, los matices nuevos no los tendrás en tus reglas y tu árbol se quedará corto.

La alternativa que surge aquí es la de hacer muchos árboles y quedarse con una especie de valor medio de los resultados. Si para un registo concreto un árbol me dice 4 y otro me dice 5, me quedo con 4.5 y me conformo.

Esa es la idea de *bagging*, una técnica mediante la cual construimos muchos modelos (pueden no ser árboles, pueden ser regresiones o lo que sea) y como predicción final tomamos alguna agregación (como la media).

Los *random forests* hacen algo así, pero con matices. Concretamente, si se llaman *random*, algo tendrán de aleatorio, ¿no? Tienen dos cosas aleatorias:

- Para cada árbol que los compone (es un bosque) se ha escogido una muestra aleatoria de los registros (esto es habitual en las técnicas *bagging*).
- Para cada árbol, las variables que se usan se escogen aleatoriamente sobre las totales (esto es muy característico de los *random forests*).

```
library(randomForest)
fit_ozone <- randomForest(Ozone ~ ., data = airquality, na.action = na.omit)
fit_ozone
```

```
##
## Call:
##  randomForest(formula = Ozone ~ ., data = airquality, na.action = na.omit)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##           Mean of squared residuals: 318.3277
##                     % Var explained: 70.99
```

```
fit_ozone <- randomForest(Ozone ~ .,
                          data = airquality,
                          ntree = 200,
                          mtry = 4,
                          nodesize = 10,
                          na.action = na.omit)
fit_ozone
```

```
##
## Call:
##  randomForest(formula = Ozone ~ ., data = airquality, ntree = 200,      mtry = 4, nodesize = 1
##                Type of random forest: regression
##                      Number of trees: 200
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 304.3001
##                     % Var explained: 72.27
```

```
fit_ozone$rsq
```

```
##   [1] 0.4271045 0.3051865 0.4710785 0.4947659 0.5611329 0.6057865 0.6221780
##   [8] 0.6144044 0.6360773 0.6439124 0.6099294 0.6078259 0.6221341 0.6297384
##  [15] 0.6436922 0.6562892 0.6764693 0.6765046 0.6783511 0.6876316 0.6910812
##  [22] 0.6888808 0.6892071 0.6933922 0.6936982 0.6891834 0.6903870 0.6914010
##  [29] 0.7000588 0.7004167 0.7074403 0.7050302 0.7078780 0.7121241 0.7115090
##  [36] 0.7156760 0.7115870 0.7150447 0.7135859 0.7137968 0.7156254 0.7142432
##  [43] 0.7147967 0.7166533 0.7154849 0.7185095 0.7187310 0.7191913 0.7198997
##  [50] 0.7185727 0.7173077 0.7173666 0.7168747 0.7183983 0.7155610 0.7146241
##  [57] 0.7196789 0.7173806 0.7135784 0.7099031 0.7087451 0.7109713 0.7087094
##  [64] 0.7076044 0.7051152 0.7062855 0.7089382 0.7095446 0.7088421 0.7081431
##  [71] 0.7092602 0.7068698 0.7060994 0.7046500 0.7045401 0.7085081 0.7100391
##  [78] 0.7069263 0.7079762 0.7072041 0.7075859 0.7090977 0.7056387 0.7059195
##  [85] 0.7059764 0.7056391 0.7082941 0.7082475 0.7089169 0.7091593 0.7105729
##  [92] 0.7115376 0.7115268 0.7143535 0.7153802 0.7184374 0.7195913 0.7192828
##  [99] 0.7215721 0.7211016 0.7203480 0.7206157 0.7203646 0.7183012 0.7186981
## [106] 0.7175039 0.7154235 0.7148883 0.7137017 0.7154745 0.7158340 0.7158273
## [113] 0.7148132 0.7145520 0.7149058 0.7169993 0.7171554 0.7182342 0.7166744
## [120] 0.7160347 0.7165418 0.7175838 0.7168124 0.7168370 0.7152003 0.7164263
## [127] 0.7166811 0.7183138 0.7192527 0.7202600 0.7186683 0.7195385 0.7204451
## [134] 0.7219435 0.7238517 0.7251376 0.7266705 0.7279101 0.7286941 0.7268405
## [141] 0.7254027 0.7255381 0.7257974 0.7262561 0.7261752 0.7266151 0.7266309
## [148] 0.7276260 0.7299314 0.7297511 0.7284091 0.7278561 0.7283203 0.7280814
## [155] 0.7274453 0.7268481 0.7267722 0.7275334 0.7278147 0.7284877 0.7286282
## [162] 0.7286177 0.7288224 0.7280943 0.7288228 0.7282978 0.7295914 0.7293471
```

```
## [169] 0.7285799 0.7281591 0.7275208 0.7280998 0.7277015 0.7277982 0.7277036
## [176] 0.7270212 0.7256882 0.7261799 0.7260694 0.7250974 0.7243521 0.7239569
## [183] 0.7243725 0.7245333 0.7239643 0.7246639 0.7252608 0.7259883 0.7258335
## [190] 0.7255393 0.7248589 0.7246880 0.7245855 0.7254794 0.7248172 0.7240704
## [197] 0.7228591 0.7225510 0.7227847 0.7226865
```

```
airquality_sin_na <- na.omit(airquality)

rf_residuals <- airquality_sin_na$Ozone - fit_ozone$predicted
1 - sum(rf_residuals ^ 2) / sum((airquality_sin_na$Ozone - mean(airquality_sin_na$Ozon
```

```
## [1] 0.7226865
```

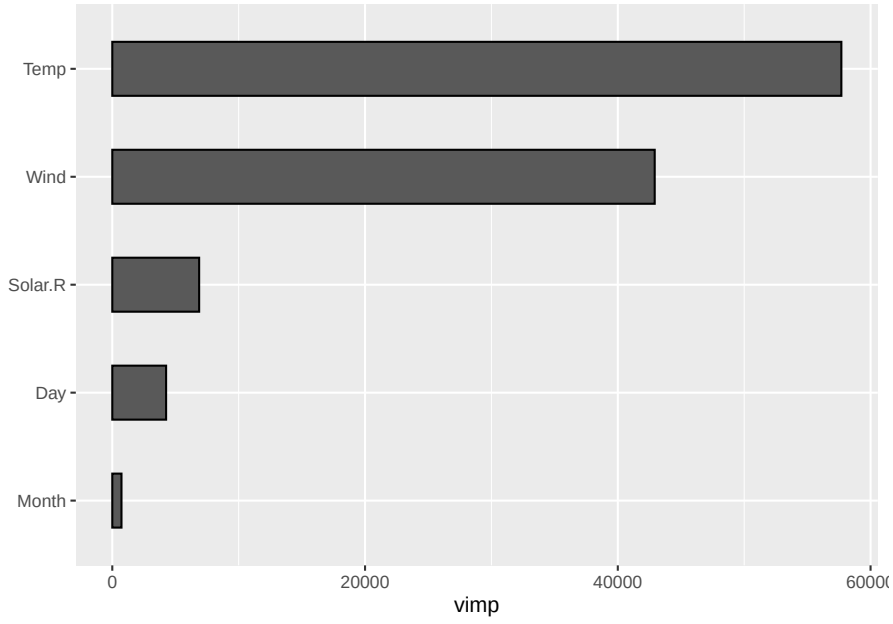
```
# MAPE
mape <- mean(abs((airquality_sin_na$Ozone - fit_ozone$predicted)) / airquality_sin_na$
mape
```

```
## [1] 0.5496248
```

```
# MSE
mse <- mean((airquality_sin_na$Ozone - fit_ozone$predicted) ^ 2)
mse
```

```
## [1] 304.3001
```

```
library(ggRandomForests)
plot(gg_vimp(fit_ozone))
```


Temp
Wind
Solar.R
Day
Month
0
20000
40000
vimp

Chapter 4

Regresión - Ejercicio práctico

Carga los datos `Boston` del paquete MASS. Accede a la documentación de las variables con `? Boston`.

```
data(Boston, package = "MASS")
head(Boston)
```

```
##      crim zn indus chas   nox    rm  age    dis rad tax ptratio  black lstat
## 1 0.00632 18  2.31    0 0.538 6.575 65.2 4.0900   1 296    15.3 396.90  4.98
## 2 0.02731  0  7.07    0 0.469 6.421 78.9 4.9671   2 242    17.8 396.90  9.14
## 3 0.02729  0  7.07    0 0.469 7.185 61.1 4.9671   2 242    17.8 392.83  4.03
## 4 0.03237  0  2.18    0 0.458 6.998 45.8 6.0622   3 222    18.7 394.63  2.94
## 5 0.06905  0  2.18    0 0.458 7.147 54.2 6.0622   3 222    18.7 396.90  5.33
## 6 0.02985  0  2.18    0 0.458 6.430 58.7 6.0622   3 222    18.7 394.12  5.21
##   medv
## 1 24.0
## 2 21.6
## 3 34.7
## 4 33.4
## 5 36.2
## 6 28.7
```

4.0.1 Ejercicio

- Plantea un modelo de regresión lineal, teniendo en cuenta que la variable objetivo es `medv`.
- Plantea un random forest.

Chapter 5

Clasificación

Imaginemos un banco con varios productos: fondo de inversión, plan de pensiones o cuenta de ahorros. Tiene una cartera de clientes que solo han contrado una cuenta corriente y quiere proponer a estos clientes que contraten uno de estos tres productos. El banco tiene información sobre los clientes (cuánto cobra, qué compra, en qué consiste su ocio, si llega a fin de mes, si tiene deudas,... mucha información), por lo que podría intentar averiguar en base a esta información cuál es el producto que le puede interesar más a cada cliente. De esta forma, en lugar de ofrecer los tres productos a todo el mundo, con el riesgo de que la gente se sienta agobiada con tanta propuesta, podrían enfocarse en lo que de verdad puede resultar tentador. Esta variable objetivo, decidir para cada cliente si ofrecer el fondo, el plan o la cuenta de ahorros, es lo que llamamos una variable categórica.

Ha diferencia de las variables continuas, vistas en modelos de regresión, las variables categóricas separan los registros en grupos. Los modelos asociados a estas variables son modelos de clasificación. Con estos modelos, asignaremos a cada registro una clase; en el caso del banco, a cada cliente le asignaremos un producto potencial.

Los modelos de clasificación en los que nos centraremos aquí son los que tratan con variables objetivos que solo tienen 1s y 0s. Esto podría entender como variables de dos categorías pero los problemas que se suelen abordar realmente son aquellos que responden a una pregunta tipo *"¿Pertenece a esta clase o no pertenece?"*

En el caso del banco podríamos traducirlo como tres modelos: uno por cada producto. Estos modelos no solo suelen decir si un registro pertenece a una clase o no, sino cuán propenso es. Por lo tanto, se puede hacer un modelo para cada producto y para cada cliente elegiríamos el caso con mayor propensión.

Las aplicaciones de estos modelos binarios abarcan muchos campos: ver si un cliente comprará un producto o no, si un cliente se dará de baja de su suscrip-

ción, si un prospecto incurrirá en impago de un préstamo, si un paciente sobrevivirá a una intervención, si un análisis se asocia a un resultado positivo de una enfermedad, si un empleado dejará mi empresa, etc…

Pregunta. ¿Qué aplicaciones se te ocurren en tu campo? Piensa en preguntas de respuesta sí o no.

5.1 Regresión logística

Hay muchas técnicas para predecir un *sí* o un *no* en un modelo de clasificación. Nosotros nos vamos a centrar en la regresión logística pero mencionaremos otras.

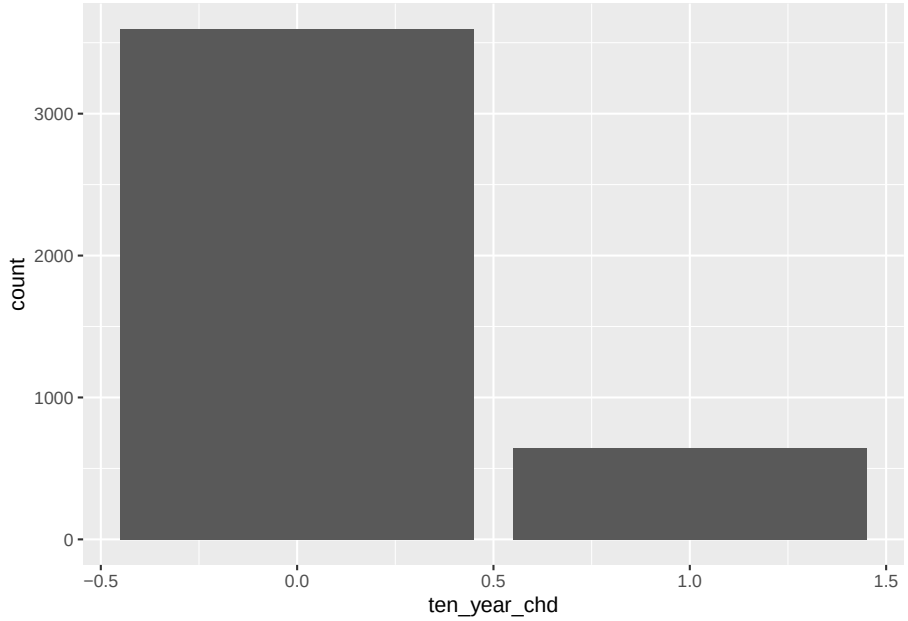
Ejemplificaremos la técnica con el conjunto de datos de riesgo de enfermedades cardiovasculares de Framingham. El conjunto de datos proporciona información sobre 4000 individuos y el riesgo de padecer una enfermedad cardiovascular en diez años (columna `ten_year_chd`). Las columnas:

- `male`. `0`: mujeres; `1`: para varones.
- `age`. Edad en el momento del examen médico.
- `high_school_ged`. `1`: graduado escolar como máximo nivel de estudios.
- `some_college_vocational_school`. `1`: diplomado.
- `college`. `1`: licenciado.
- `current_smoker`. `1`: fumador; `0`: no fumador.
- `cigs_per_day`. Número de cigarrilos al día (media estimada).
- `bp_meds`. `0`: Sin medicamentos por tensión alta; `1`: con medicación contra tensión alta.
- `prevalent_stroke`. `1`: riesgo de derrame.
- `prevalent_hyp`. `1`: riesgo de hipertensión.
- `diabetes`. `1`: diabetes.
- `tot_chol`. Colesterol en mg/dL.
- `sys_bp`. Presión arterial sistólica.
- `dia_bp`. Presión arterial diastólica.
- `bmi`. Índice de masa corporal.
- `heart_rate`. Ritmo cardíaco.
- `glucose`. Índice de glucosa en sangre.
- `ten_year_chd`. `1`: padeció enfermadad CV en los 10 años siguientes al examen.

Un primer vistazo nos muestra la proporción de individuos que padecieron alguna enfermedad cardiovascular en el periodo de diez años de estudio.

```
library(readr)
df_framingham <- read_csv2("data/framingham.csv")
```

```
library(ggplot2)
ggplot(df_framingham) +
  geom_bar(aes(ten_year_chd))
```



La proporción de individuos con enfermedad cardiovascular (CHD por sus siglas en inglés) es:

```
mean(df_framingham$ten_year_chd)
```

```
## [1] 0.1518868
```

Un 15, 19 de los registros padecieron alguna CHD en algún momento. Esa proporción, desde un punto de vista estadístico, es baja. Se suele decir que el modelo estará desbalanceado, es decir, que la proporción de individuos en una clase es muy distinta a la de la otra clase. Informalmente diremos que hay mucha diferencia entre *ceros* y *unos*. Lo ideal para los modelos de clasificación es que las clases tengan proporciones parecidas. Abordaremos este problema más adelante.

Los modelos de regresión logística nos van a permitir calcular la probabilidad de padecer una CHD. Por eso se llama *regresión:* porque da un resultado continuo, pese a buscar un resultado binario. Pero se trata de una regresión especial que, como decimos, dará como resultado una probabilidad. A partir de ella, podremos decir si un individuo padecerá la enfermedad (asignaremos un 1) o no

(un 0). Esta decisión se toma en base a la probabilidad: si esta es *alta*, daremos un 1; si no, un 0.

Más adelante veremos cómo se calcula esta probabilidad en función de los datos de los individuos (edad, origen, estado de saludo, etc.). Por ahora intentemos responder a esta pregunta: **¿qué es una probabilidad alta?** ¿Más de 0,5?

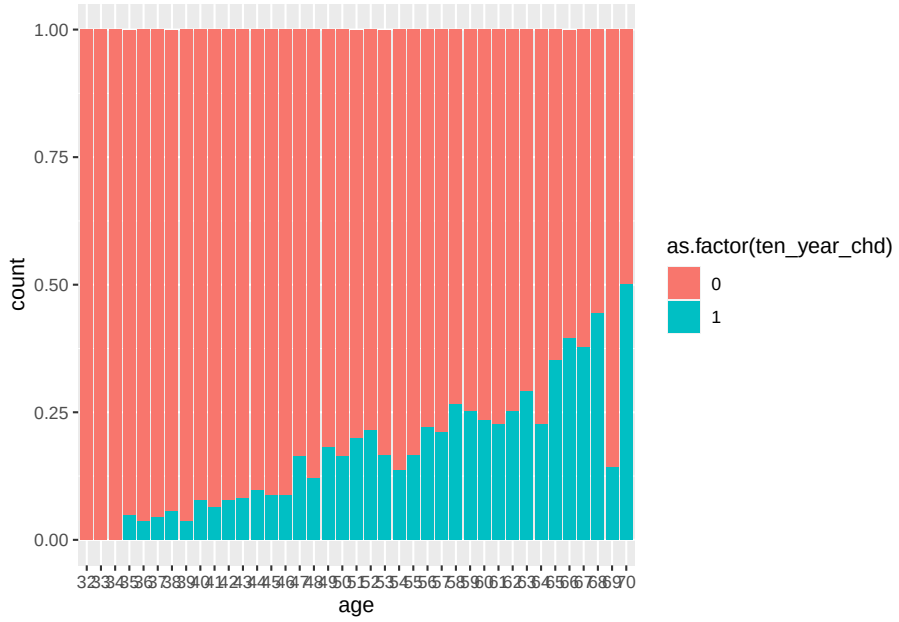
No siempre. En nuestro conjunto de datos, hemos visto que la proporción de individuos afectados por una CHD es 0,15. Esto lo interpretaremos como la probabilidad media de padecer una CHD. Por lo tanto, si un invidividuo tuviera una probabilidad de 0,3 de padecer una CHD, diríamos que su probabilidad es alta y le asignaríamos un 1. ¿Parece que 0,3 es baja? Tengamos en cuenta que tiene el doble de probabilidades que la media de la población…

5.1.1 Regresión lineal frente a logística en dos dimensiones.

Una forma de intentar calcular esta probabilidad (o la etiqueta 0/1), dado que en el fondo vamos a realizar una regresión, podría ser ajustar un modelo de regresión lineal.

Visualicemos la variable objetivo frente a la edad:

```
library(dplyr)
df_framingham %>%
  mutate(age = as.factor(age)) %>%
  ggplot() +
  geom_bar(aes(x = age, fill = as.factor(ten_year_chd)), position = "fill")
```



Podemos ajustar una recta de regresión de estas dos variables con `lm()`.

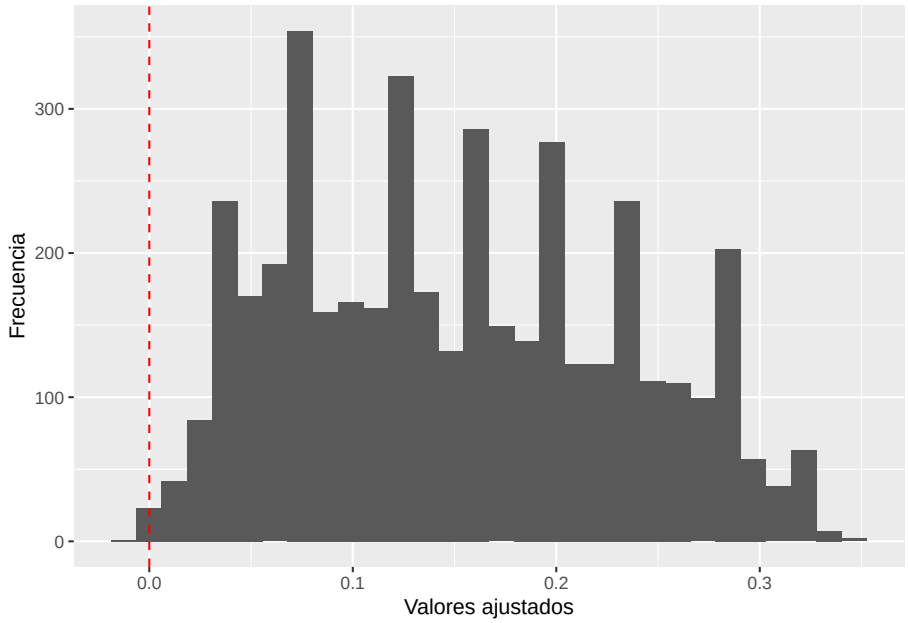
```
fit_linear <- lm(ten_year_chd ~ age, data = df_framingham)
summary(fit_linear)
```

```
##
## Call:
## lm(formula = ten_year_chd ~ age, data = df_framingham)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.34461 -0.19360 -0.10866 -0.05203  0.98572
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.3160479  0.0315289  -10.02   <2e-16 ***
## age          0.0094379  0.0006266   15.06   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3498 on 4238 degrees of freedom
## Multiple R-squared:  0.05081,    Adjusted R-squared:  0.05058
## F-statistic: 226.9 on 1 and 4238 DF,  p-value: < 2.2e-16
```

El coeficiente de la edad es positivo y significativo: algo esperable. Pero con el

intercept se ve algo raro: es negativo. Entremos en detalle en el ajuste:

```
ggplot() +
  geom_histogram(aes(x = fit_linear$fitted.values))  +
  geom_vline(xintercept = 0, color = "red", linetype = 2) +
  labs(x = "Valores ajustados", y = "Frecuencia")
```



Pregunta. ¿Qué salta a la vista, si tenemos en cuenta que estamos intentando etiquetar ceros y unos o, en su defecto, probabilidades?

5.1.2 Fórmula de una regresión logística.

5.1.3 Ajuste y sintaxis

Observación. La regresión logística es una fórmula matemática numérica, por lo que no podremos trabajar con valores missing. Lo ideal sería imputarlos con valores que representaran un valor medio, pero en este caso vamos a omitir las filas que tengan algún valor *missing* para centrarnos en el ajuste del modelo en sí.

```
df_framingham <- na.omit(df_framingham)
```

```
fit_logit0 <- glm(ten_year_chd ~ age, data = df_framingham, family = binomial(link = "logit"))
summary(fit_logit0)
```

```
##
## Call:
## glm(formula = ten_year_chd ~ age, family = binomial(link = "logit"),
##     data = df_framingham)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.0617  -0.6283  -0.4537  -0.3628   2.4689
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -5.719765   0.307777  -18.58   <2e-16 ***
## age          0.077733   0.005702   13.63   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 3121.2  on 3657  degrees of freedom
## Residual deviance: 2920.6  on 3656  degrees of freedom
## AIC: 2924.6
##
## Number of Fisher Scoring iterations: 5
```

Para interpretar el *intercept* recordemos que no estamos modelizando directamente valores *ceros* y *unos*, ni probabilidades, sino

$$\log(\frac{p}{1-p}).$$

El *intercept* negativo está dando valor negativo a esa expresión logarítmica. Si el logaritmo de un número es negativo, significa que ese número es mayor que 0 y menor que 1, por lo que

$$0 < \frac{p}{1-p} < 1.$$

Que sea mayor que cero deriva de que estamos dividiendo una probabilidad por su complementaria, ambos números positivos, por lo que era esperable. Que sea menor que 1 siginifica que $1-p$ es mayor que p (el denominador es mayor que el numerador), y esto encaja con la exploración previa ya que hay un mayor número de *ceros* que de *unos*.

De nuevo, el *intercept* es el valor esperado del ajuste del modelo en el caso de que todas las variables explicativas valgan cero.

Avancemos hacia un modelo completo. Una forma de indicar que queremos ajustar un modelo con todas las variables disponibles es la siguiente:

```
glm(ten_year_chd ~ ., data = df_framingham, family = binomial(link = "logit"))
```

El inconveniente es que se pierde interactividad con el código a la hora de probar a quitar e introducir variables. Es preferible escribirlas todas, aunque podemos hacer un poco de código haga el trabajo por nosotros:

```
cat(names(df_framingham), sep = "\n + ")
```

```
fit_logit <- glm(ten_year_chd ~ 1
 + male
 + age
 + high_school_ged
 + some_college_vocational_school
 + college
 + current_smoker
 + cigs_per_day
 + bp_meds
 + prevalent_stroke
 + prevalent_hyp
 + diabetes
 + tot_chol
 + sys_bp
 + dia_bp
 + bmi
 + heart_rate
 + glucose ,
 data = df_framingham, family = binomial("logit"))

summary(fit_logit)
```

```
##
## Call:
## glm(formula = ten_year_chd ~ 1 + male + age + high_school_ged +
##     some_college_vocational_school + college + current_smoker +
##     cigs_per_day + bp_meds + prevalent_stroke + prevalent_hyp +
##     diabetes + tot_chol + sys_bp + dia_bp + bmi + heart_rate +
```

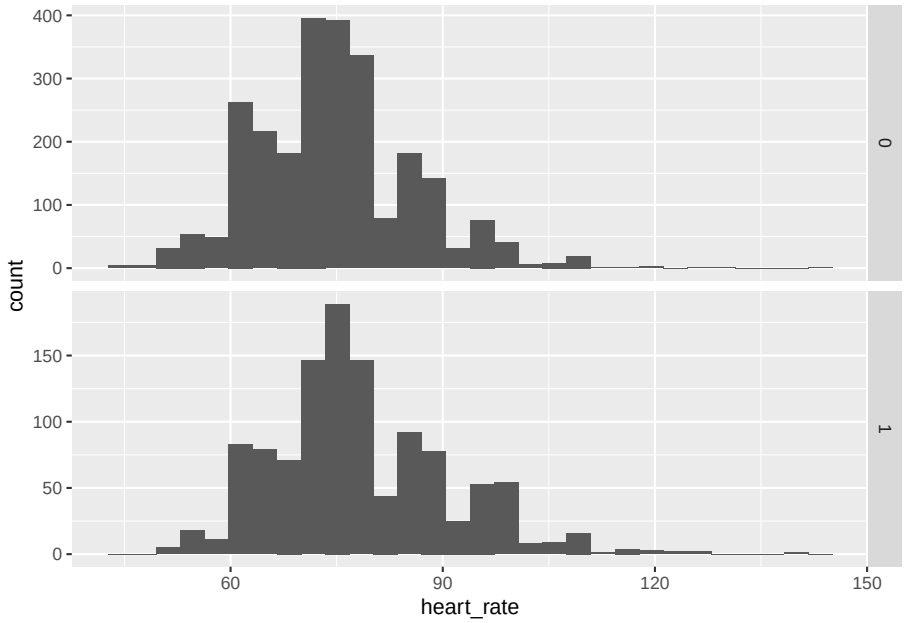
```
##     glucose, family = binomial("logit"), data = df_framingham)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.9144  -0.6035  -0.4239  -0.2876   2.7790
##
## Coefficients:
##                                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)                    -7.018e+00  5.752e-01 -12.200  < 2e-16 ***
## male                            4.901e-01  1.080e-01   4.540 5.63e-06 ***
## age                             6.846e-02  6.481e-03  10.563  < 2e-16 ***
## high_school_ged                -2.106e-01  1.223e-01  -1.722 0.085134 .
## some_college_vocational_school -2.311e-01  1.491e-01  -1.551 0.121016
## college                        -1.132e-01  1.638e-01  -0.691 0.489552
## current_smoker                  4.671e-02  1.553e-01   0.301 0.763618
## cigs_per_day                    1.821e-02  6.218e-03   2.928 0.003411 **
## bp_meds                         3.344e-01  2.284e-01   1.464 0.143098
## prevalent_stroke                7.193e-01  4.906e-01   1.466 0.142619
## prevalent_hyp                   6.326e-01  1.060e-01   5.969 2.38e-09 ***
## diabetes                       -4.669e-03  3.150e-01  -0.015 0.988172
## tot_chol                        2.633e-03  1.123e-03   2.345 0.019018 *
## sys_bp                          1.728e-04  9.999e-05   1.728 0.083947 .
## dia_bp                         -2.907e-04  1.930e-04  -1.506 0.132084
## bmi                            -4.527e-05  6.216e-05  -0.728 0.466447
## heart_rate                     -3.781e-04  4.167e-03  -0.091 0.927709
## glucose                         7.928e-03  2.234e-03   3.549 0.000387 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 3121.2  on 3657  degrees of freedom
## Residual deviance: 2771.1  on 3640  degrees of freedom
## AIC: 2807.1
##
## Number of Fisher Scoring iterations: 5
```

Se observan varias variables no significativas. Las sacaremos del modelo antes de estudiar el ajuste y entender los resultados, pero podemos estudiar también relaciones entre variables significativas de forma que entendamos por qué algunas variables no deberían incluirse.

Por ejemplo, podemos visualizar `prevalent_hyp` frente a `heart_rate`.

```
df_framingham %>%
  select(prevalent_hyp, heart_rate) %>%
```

```
  ggplot() +
  geom_histogram(aes(x = heart_rate)) +
  facet_grid(prevalent_hyp ~ ., scales = "free_y")
```



Explorando el conjunto de datos veremos que información casi duplicada: hay variables continuas y binarias, que van por parejas. Es decir, las variables binarias se explican en gran medida por alguna de las continuas. Por ello, no tiene sentido incluirlas todas a la vez.

En este caso, en lugar de quitar variables según diga el p-valor, puede ser interesante decantarnos por emplear solo las variables binarias, ya que resultará más fácil para la interpretración de los resultados, como veremos más adelante.

```
library(purrr)
vars_binarias <- df_framingham %>%
  map(~length(unique(.))) %>%
  keep(~. == 2) %>%
  names()

vars_binarias
```

```
##  [1] "male"                           "high_school_ged"
##  [3] "some_college_vocational_school" "college"
##  [5] "current_smoker"                 "bp_meds"
##  [7] "prevalent_stroke"               "prevalent_hyp"
```

```
##  [9] "diabetes"                          "ten_year_chd"
```

Podemos reconstruir la fórmula con el código previo y solo con las variables guardadas en el objeto `vars_binarias` o podemos comentar (`#`) manualmente las variables que no nos interesen. Esta segunda forma es especialmente útil para interactuar con el modelo y hacer pruebas rápidas.

Observación. Las variables sobre educación son complementarias y no tiene sentido introducir todas a la vez, así que quitaremos una.

```
fit_logit <- glm(ten_year_chd ~
                   1
                 + male
                 # + age
                 + high_school_ged
                 + some_college_vocational_school
                 # + college
                 + current_smoker
                 # + cigs_per_day
                 + bp_meds
                 + prevalent_stroke
                 + prevalent_hyp
                 + diabetes
                 # + tot_chol
                 # + sys_bp
                 # + dia_bp
                 # + bmi
                 # + heart_rate
                 # + glucose ,
                 ,data = df_framingham, family = binomial("logit"))

summary(fit_logit)
```

```
##
## Call:
## glm(formula = ten_year_chd ~ 1 + male + high_school_ged + some_college_vocational_school +
##     current_smoker + bp_meds + prevalent_stroke + prevalent_hyp +
##     diabetes, family = binomial("logit"), data = df_framingham)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.2673  -0.5909  -0.4734  -0.3781   2.3500
##
## Coefficients:
##                               Estimate Std. Error z value Pr(>|z|)
```

```
## (Intercept)                    -2.29702    0.10402 -22.082  < 2e-16 ***
## male                            0.47546    0.09832   4.836 1.32e-06 ***
## high_school_ged                -0.39886    0.11320  -3.524 0.000426 ***
## some_college_vocational_school -0.30532    0.14074  -2.169 0.030054 *
## current_smoker                  0.16463    0.09851   1.671 0.094678 .
## bp_meds                         0.50707    0.22104   2.294 0.021789 *
## prevalent_stroke                0.80968    0.47804   1.694 0.090313 .
## prevalent_hyp                   0.92195    0.09918   9.295  < 2e-16 ***
## diabetes                        0.94377    0.22339   4.225 2.39e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 3121.2  on 3657  degrees of freedom
## Residual deviance: 2934.5  on 3649  degrees of freedom
## AIC: 2952.5
##
## Number of Fisher Scoring iterations: 5
```

A un nivel de signifación α de 0.1, más que habitual en el ámbito empresarial (aunque no tanto en el académico), todas estas variable resultan significativas.

5.1.4 Ajuste

```
library(caret)
# predict_logit <- predict(fit_logit, newdata = df_framingham, type = "response")
confusionMatrix(as.factor(as.numeric(fit_logit$fitted.values > 0.5)),
                as.factor(df_framingham$ten_year_chd),
                positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3093  544
##          1    8   13
##
##                Accuracy : 0.8491
##                  95% CI : (0.8371, 0.8606)
##     No Information Rate : 0.8477
##     P-Value [Acc > NIR] : 0.42
##
```

```
##                  Kappa : 0.0343
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.023339
##             Specificity : 0.997420
##          Pos Pred Value : 0.619048
##          Neg Pred Value : 0.850426
##              Prevalence : 0.152269
##          Detection Rate : 0.003554
##    Detection Prevalence : 0.005741
##       Balanced Accuracy : 0.510380
##
##        'Positive' Class : 1
##
```

```
confusionMatrix(as.factor(as.numeric(fit_logit$fitted.values > mean(df_framingham$ten_year_chd))),
                as.factor(df_framingham$ten_year_chd),
                positive = "1")
```

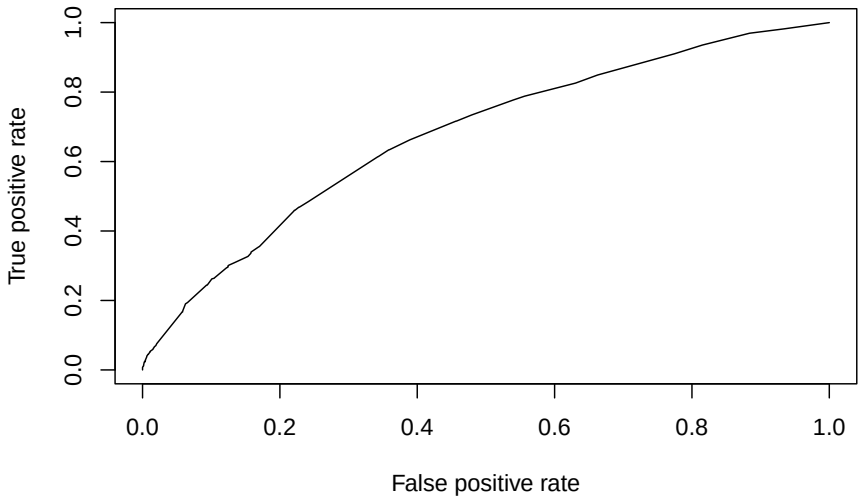
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1989  204
##          1 1112  353
##
##                Accuracy : 0.6402
##                  95% CI : (0.6244, 0.6558)
##     No Information Rate : 0.8477
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1649
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.6338
##             Specificity : 0.6414
##          Pos Pred Value : 0.2410
##          Neg Pred Value : 0.9070
##              Prevalence : 0.1523
##          Detection Rate : 0.0965
##    Detection Prevalence : 0.4005
##       Balanced Accuracy : 0.6376
##
```

```
##         'Positive' Class : 1
##
```

```
library(ROCR)
auc_aux <- performance(prediction(fit_logit$fitted.values,
                        df_framingham$ten_year_chd),
            measure = "auc")

as.numeric(auc_aux@y.values)
```

```
## [1] 0.6760065
```

```
plot(
  performance(prediction(fit_logit$fitted.values,
                         df_framingham$ten_year_chd),
              measure="tpr", x.measure="fpr")
)
```



Pregunta. ¿Cómo sabemos si es bueno y estable este resultado?

5.1.5 Entrenamiento y validación

```
train_size <- floor(0.7 * nrow(df_framingham))
set.seed(31818)
filas_para_train <- sample(seq_len(nrow(df_framingham)), train_size)

df_framingham_train <- df_framingham[filas_para_train, ]
df_framingham_test <- df_framingham[-filas_para_train, ]

fit_logit_split <- glm(ten_year_chd ~
                  1
                + male
                + high_school_ged
                + some_college_vocational_school
                + current_smoker
                + bp_meds
                + prevalent_stroke
                + prevalent_hyp
                + diabetes
                ,data = df_framingham_train, family = binomial("logit"))

summary(fit_logit_split)
```

```
##
## Call:
## glm(formula = ten_year_chd ~ 1 + male + high_school_ged + some_college_vocational_school +
##     current_smoker + bp_meds + prevalent_stroke + prevalent_hyp +
##     diabetes, family = binomial("logit"), data = df_framingham_train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.2799  -0.5596  -0.4506  -0.3749   2.3850
##
## Coefficients:
##                                Estimate Std. Error z value Pr(>|z|)
## (Intercept)                     -2.4276     0.1271 -19.094  < 2e-16 ***
## male                             0.4612     0.1196   3.857 0.000115 ***
## high_school_ged                 -0.3566     0.1372  -2.599 0.009358 **
## some_college_vocational_school  -0.1923     0.1694  -1.135 0.256230
## current_smoker                   0.1914     0.1204   1.589 0.112132
## bp_meds                          0.5462     0.2635   2.073 0.038183 *
## prevalent_stroke                 0.7522     0.5887   1.278 0.201338
## prevalent_hyp                    1.0860     0.1205   9.010  < 2e-16 ***
## diabetes                         0.9269     0.2794   3.318 0.000908 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2146.8  on 2559  degrees of freedom
## Residual deviance: 1996.7  on 2551  degrees of freedom
## AIC: 2014.7
##
## Number of Fisher Scoring iterations: 5
```

```
predicciones_train <- predict(fit_logit_split, newdata = df_framingham_train, type = "
predicciones_test <- predict(fit_logit_split, newdata = df_framingham_test, type = "res

confusionMatrix(as.factor(as.numeric(predicciones_train > mean(df_framingham_train$ten_
                as.factor(df_framingham_train$ten_year_chd),
                positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1590  168
##          1  591  211
##
##                Accuracy : 0.7035
##                  95% CI : (0.6854, 0.7212)
##     No Information Rate : 0.852
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1956
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.55673
##             Specificity : 0.72902
##          Pos Pred Value : 0.26309
##          Neg Pred Value : 0.90444
##              Prevalence : 0.14805
##          Detection Rate : 0.08242
##    Detection Prevalence : 0.31328
##       Balanced Accuracy : 0.64288
##
##        'Positive' Class : 1
##
```

```
confusionMatrix(as.factor(as.numeric(predicciones_test > mean(df_framingham_train$ten_year_chd))),
                as.factor(df_framingham_test$ten_year_chd),
                positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 620  92
##          1 300  86
##
##                Accuracy : 0.643
##                  95% CI : (0.6138, 0.6714)
##     No Information Rate : 0.8379
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1068
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.48315
##             Specificity : 0.67391
##          Pos Pred Value : 0.22280
##          Neg Pred Value : 0.87079
##              Prevalence : 0.16211
##          Detection Rate : 0.07832
##    Detection Prevalence : 0.35155
##       Balanced Accuracy : 0.57853
##
##        'Positive' Class : 1
##
```

De las diferencias entre entrenamiento y validación podemos concluir que el modelo parece sobre ajustado. En cualquier caso, como ya hemos dicho anteriormente, nuestro objetivo hoy es entender la sintaxis, herramientas y flujo de trabajo; no hacer un modelo definitivo.

Problema. Además, tenemos otro frente abierto: no hemos garantizado que la proporción de ceros y unos se mantenga constante entre los conjuntos entrenamiento y validación. Para asegurarnos de que el modelo es generalizable haciendo la comparativa entre ambos conjuntos, necesitamos que estos estén en igualdad de condiciones. Si asumimos que nuestra muestra original es una buena representante de la población general, necesitamos que ambos subconjuntos tengan características similares. Para ello, normalmente se realiza una selección aleatoria de registros para entrenamiento y validación pero que cumpla

determinadas características: principalmente, que la proporción de las clases de la variable objetivo se mantenga, aunque se puede extrapolar esto también a las variables explicativas. Esto no es necesario hacerlo a mano, sino que las principales herramientas de modelado incluyen técnicas para ello, como el entorno **tidymodels** de R. Pero todo eso queda fuera del alcance de este curso.

Otro problema. Supongamos un ejemplo extremo. Una enfermedad muy rara que solo tiene el 0,01% de la población (sí, uno de cada 10.000 habitantes). ¿Cómo podríamos hacer un modelo con una precisión muy alta? La mejor opción, si esa es nuestra prioridad, es decir que nadie tendrá la enfermedad: acertaríamos el 99,99% de las veces. Es una precisión sin rival. Pero un modelo inútil. Aunque suene a caso extremo, en nuestro ejemplo podríamos haber hecho algo parecido y el model tendría una precisión del 85% (mayor que la nuestra actual). Eso no se puede permitir. Hay técnicas de modelado que permiten dar costes a las categorías, o priorizar métricas de bondad de ajuste que eviten esto, pero una recomendación que se debe seguir a menudo es balancear la muestra, con técnicas de remuestreo.

5.1.6 Sobremuestreo

Si tuviéramos una proporción de 50% de ambas categorías en la variable objetivo nos quitaríamos el problema. Una opción para simular esa situación es generar registro extra de la clase que está representada inferiormente, los unos en nuestro caso.

```
filas_con_unos <- which(df_framingham$ten_year_chd == 1)

diferencia_de_ceros_y_unos <- sum(df_framingham$ten_year_chd == 0) - sum(df_framingham$

set.seed(31818)
registros_extra_con_unos <- sample(filas_con_unos, size = diferencia_de_ceros_y_unos, r

library(dplyr)
df_framingham_sobremuestreo <- df_framingham %>%
  mutate(set = "original") %>%
  bind_rows(df_framingham %>%
              slice(registros_extra_con_unos) %>%
              mutate(set = "sobremuestreo"))

# Hemos igualado la proporción de ceros y unos
mean(df_framingham_sobremuestreo$ten_year_chd)
```

```
## [1] 0.5
```

Todo lo que hemos visto anteriormente sigue aplicando con normalidad, aunque hayamos alterado la muestra. Así que procedemos a ajustar nuestro modelo en un conjunto de entrenamiento.

```
# Ahora dividimos train y test

train_size <- floor(0.7 * nrow(df_framingham_sobremuestreo))
set.seed(31818)
filas_para_train <- sample(seq_len(nrow(df_framingham_sobremuestreo)), train_size)

df_framingham_sobre_train <- df_framingham_sobremuestreo[filas_para_train, ]
df_framingham_sobre_test <- df_framingham_sobremuestreo[-filas_para_train, ]


fit_logit_split_sobre <- glm(ten_year_chd ~
                          1
                        + male
                        + high_school_ged
                        + some_college_vocational_school
                        + current_smoker
                        + bp_meds
                        + prevalent_stroke
                        + prevalent_hyp
                        + diabetes
                        ,data = df_framingham_sobre_train,
                        family = binomial("logit"))

summary(fit_logit_split_sobre)
```

```
##
## Call:
## glm(formula = ten_year_chd ~ 1 + male + high_school_ged + some_college_vocational_school +
##     current_smoker + bp_meds + prevalent_stroke + prevalent_hyp +
##     diabetes, family = binomial("logit"), data = df_framingham_sobre_train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0104  -1.0878  -0.8084   1.1212   1.5984
##
## Coefficients:
##                               Estimate Std. Error z value Pr(>|z|)
## (Intercept)                   -0.60258    0.06789  -8.876  < 2e-16 ***
## male                           0.49557    0.06574   7.538 4.76e-14 ***
## high_school_ged               -0.34804    0.07334  -4.745 2.08e-06 ***
## some_college_vocational_school -0.34041    0.09171  -3.712 0.000206 ***
```

```
## current_smoker                0.24069    0.06623   3.634 0.000279 ***
## bp_meds                       0.39121    0.17311   2.260 0.023832 *
## prevalent_stroke              0.60821    0.43685   1.392 0.163846
## prevalent_hyp                 0.84821    0.06836  12.408  < 2e-16 ***
## diabetes                      0.89675    0.17420   5.148 2.64e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 6017.9  on 4340  degrees of freedom
## Residual deviance: 5666.1  on 4332  degrees of freedom
## AIC: 5684.1
##
## Number of Fisher Scoring iterations: 4
```

```
predicciones_train <- predict(fit_logit_split_sobre, newdata = df_framingham_sobre_trai
predicciones_test <- predict(fit_logit_split_sobre, newdata = df_framingham_sobre_test

confusionMatrix(as.factor(as.numeric(predicciones_train > mean(df_framingham_sobre_trai
                as.factor(df_framingham_sobre_train$ten_year_chd),
                positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1413  860
##          1  762 1306
##
##                Accuracy : 0.6264
##                  95% CI : (0.6118, 0.6408)
##     No Information Rate : 0.501
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.2526
##
##  Mcnemar's Test P-Value : 0.01602
##
##             Sensitivity : 0.6030
##             Specificity : 0.6497
##          Pos Pred Value : 0.6315
##          Neg Pred Value : 0.6216
##              Prevalence : 0.4990
##          Detection Rate : 0.3009
```

```
##    Detection Prevalence : 0.4764
##       Balanced Accuracy : 0.6263
##
##        'Positive' Class : 1
##
```

```
confusionMatrix(as.factor(as.numeric(predicciones_test > mean(df_framingham_sobre_train$ten_year_
                as.factor(df_framingham_sobre_test$ten_year_chd),
                positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 625 383
##          1 301 552
##
##                Accuracy : 0.6325
##                  95% CI : (0.6101, 0.6544)
##     No Information Rate : 0.5024
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.2652
##
##  Mcnemar's Test P-Value : 0.001954
##
##             Sensitivity : 0.5904
##             Specificity : 0.6749
##          Pos Pred Value : 0.6471
##          Neg Pred Value : 0.6200
##              Prevalence : 0.5024
##          Detection Rate : 0.2966
##    Detection Prevalence : 0.4584
##       Balanced Accuracy : 0.6327
##
##        'Positive' Class : 1
##
```

Un paso más sería usar test con la proporción original, apoyándonos en la columna extra `set`. Lo dejamos como ejercicio para el lector.

5.1.7 Interpretación de resultados. Odds ratio

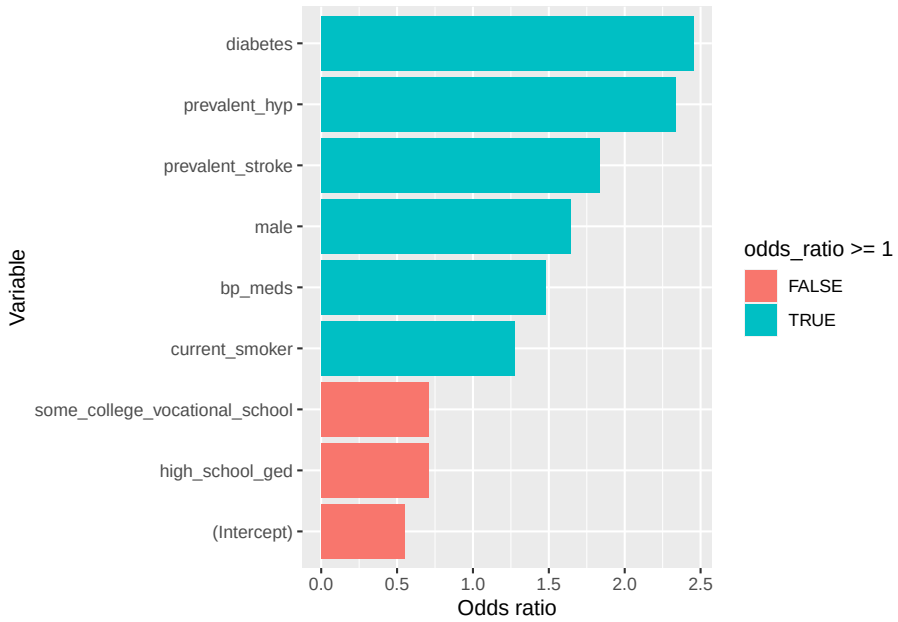
Lo interesante de este número es cuánto difiere de 1:

- Si es menor que 1, entonces la variable influye negativamente.
- Si es mayor que 1, influye positivamente.

```
oddsratio <- exp(fit_logit_split_sobre$coefficients)
oddsratio
```

```
##                   (Intercept)                          male
##                     0.5473976                     1.6414406
##               high_school_ged some_college_vocational_school
##                     0.7060702                     0.7114791
##                 current_smoker                      bp_meds
##                     1.2721210                     1.4787672
##               prevalent_stroke                  prevalent_hyp
##                     1.8371362                     2.3354568
##                      diabetes
##                     2.4516168
```

```
tibble(variable = names(oddsratio),
       odds_ratio = oddsratio) %>%

  ggplot() +
  geom_col(aes(x = reorder(variable, odds_ratio), y = odds_ratio, fill = odds_ratio >=
  labs(x = "Variable", y = "Odds ratio") +
  coord_flip()
```



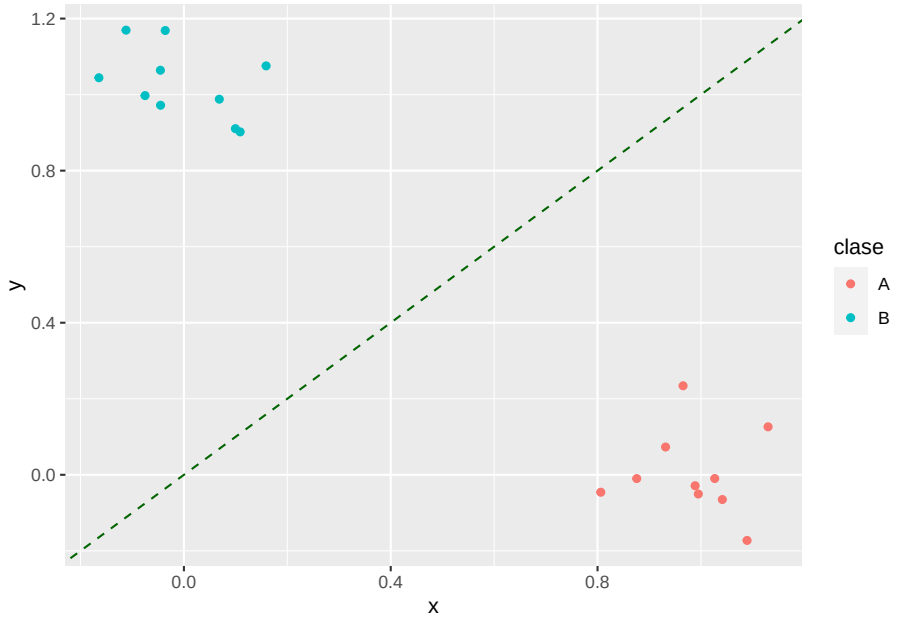
5.2 SVM

Las máquinas de vectores soporte (SVM por sus siglas en inglés) tratan de separar las observaciones en las dos clases a las que pertenecen mediante un hiperplano[^1].

[^1] Hay otros tipos de SVM. Estamos trabajando solo con problemas de clasificación en dos clases, pero se pueden usar como modelos de regresión (variables continuas), detección de atípicos o clasificación multiclase.

Si estamos en dos dimensiones, un hiperplano es una recta.

```
df_ejemplo_svm_lineal <- tibble(
  x = c(rnorm(10, 1, 0.1), rnorm(10, 0, 0.1)),
  y = c(rnorm(10, 0, 0.1), rnorm(10, 1, 0.1)),
  clase = rep(c("A", "B"), each = 10)
)

ggplot(df_ejemplo_svm_lineal) +
  geom_point(aes(x, y, col = clase)) +
  geom_abline(slope = 1, intercept = 0, col = "darkgreen", linetype = 2)
```



Un SVM separará los datos en las dos categorías y punto. No da un resultado probabilístico sobre la pertenencia a una clase. Pero sí se podrían calcular las distancias de los puntos al hiperplano para tener una idea de la lejanía a la clase

contraria: si un punto está muy cerca de la línea divisoria, significaría que es parecido a objetos de la otra clase que también están próximos al hiperplano.

A la hora de ajustar un SVM, es importante destacar que no tendremos información sobre cuán significativas son las variables. No obstante, sí hay que aplicar el sentido común y no incluir información de más: es costoso computacionalmente e, incluso, podríamos incurrir en relaciones espurias, que debemos evitar.

```
library(e1071)

df_framingham <- df_framingham %>%
  mutate(ten_year_chd = as.factor(ten_year_chd))

fit_svm <- svm(
  ten_year_chd ~
    1
  + male
  + age
  # + high_school_ged
  # + some_college_vocational_school
  # + college
  # + current_smoker
  + cigs_per_day
  # + bp_meds
  # + prevalent_stroke
  # + prevalent_hyp
  # + diabetes
  + tot_chol
  + sys_bp
  + dia_bp
  + bmi
  + heart_rate
  + glucose
  ,data = df_framingham,
  type = "C-classification",
  kernel = "radial",
  cost = 10,
  gamma = 0.1
)

summary(fit_svm)
```

```
##
## Call:
## svm(formula = ten_year_chd ~ 1 + male + age + cigs_per_day + tot_chol +
```

```
##     sys_bp + dia_bp + bmi + heart_rate + glucose, data = df_framingham,
##     type = "C-classification", kernel = "radial", cost = 10, gamma = 0.1)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  10
##
## Number of Support Vectors:  1420
##
##  ( 867 553 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```
library(caret)
df_predict <- df_framingham %>%
  select(ten_year_chd, male, age, cigs_per_day, tot_chol, sys_bp, dia_bp, bmi, heart_rate, glucos

# predict(fit_svm, df_predict)
df_predict <- df_predict %>%
  mutate(prediction = fit_svm$fitted)

confusionMatrix(data = df_predict$prediction,
                reference = df_predict$ten_year_chd,
                positive = "1",
                prevalence = 0.15)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3098  453
##          1    3  104
##
##                Accuracy : 0.8753
##                  95% CI : (0.8642, 0.8859)
##     No Information Rate : 0.8477
##     P-Value [Acc > NIR] : 1.049e-06
##
##                   Kappa : 0.2778
```

```
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.18671
##             Specificity : 0.99903
##          Pos Pred Value : 0.97148
##          Neg Pred Value : 0.87439
##              Prevalence : 0.15000
##          Detection Rate : 0.02843
##    Detection Prevalence : 0.02925
##       Balanced Accuracy : 0.59287
##
##        'Positive' Class : 1
##
```

5.2.1 Tuning parameters

Los hiperparámetros del SVM son difíciles de interpretar. Un *coste* bajo tenderá a estimar peor y uno alto sobreajustará y no será útil. ¿Cómo nos decantamos por un término medio? Haciendo pruebas. Parece cutre pero así es la realidad.

Aquí vamos a plantear las pruebas de una manera muy directa, un poco como dictaría la intuición. Realmente la preparación de estas pruebas es un paso importante en los modelos. Se realiza un mallado de posibilidades (el diseño de experimentos es la rama de la estadística que busca optimizar este mallado) y se ejecuta el modelo en todas las combinaciones de hiperparámetros (es decir, en los puntos de la malla). Nosotros vamos a plantear un *grid search* básico, para entender la idea general[^2].

[^2] Para plantear un *grid search* en toda regla, sería preferible meterse en el entorno tidymodels y aprovechar todas las herramientas presentes en sus librerías.

Primero construimos una función que ajuste un SVM en función de los valores de dos hiperpárametros. Esto nos ayudará a organizar el código.

```
ajusta_svm <- function(coste, gamma){
  fit_svm <- svm(
    ten_year_chd ~
    # + male
    + age
    # + cigs_per_day
    # + tot_chol
    # + sys_bp
    # + dia_bp
    # + bmi
```

```
    # + heart_rate
    + glucose
    ,data = df_framingham,
    type = "C-classification",
    kernel = "radial",
    cost = coste,
    gamma = gamma
  )

  return(fit_svm)
}
```

El mallado es la combinación de todos los posibles valores de nuestros parámetros. Si tenemos dos hiperparámetros, con tres posibles valores cada uno, nuestra malla tendrá $3^2 = 9$ valores diferentes, que los podemos calcular con `cross_df` del paquete purrr o con `expand.grid()`.

```
posibles_costes <- c(1, 10, 100)
gammas <- c(0.1, 1, 10)

mallado <- cross_df(list(coste = posibles_costes, gamma = gammas))
```

Si ejecutamos todos estos posibles modelos, podremos ver cuál es el mejor modelo en base a cierta métrica de ajuste, por ejemplo, la precisión.

```
numero_modelos <- nrow(mallado)
vector_metricas <- numeric(numero_modelos)

for(i in seq_len(numero_modelos)){
  fit_auxiliar <- ajusta_svm(mallado$coste[i], mallado$gamma[i])

  matriz_confusion <- confusionMatrix(data = fit_auxiliar$fitted,
                  reference = df_predict$ten_year_chd,
                  positive = "1",
                  prevalence = 0.15)

  vector_metricas[i] <- matriz_confusion$overall["Accuracy"]
}
```

```
which.max(vector_metricas)
```

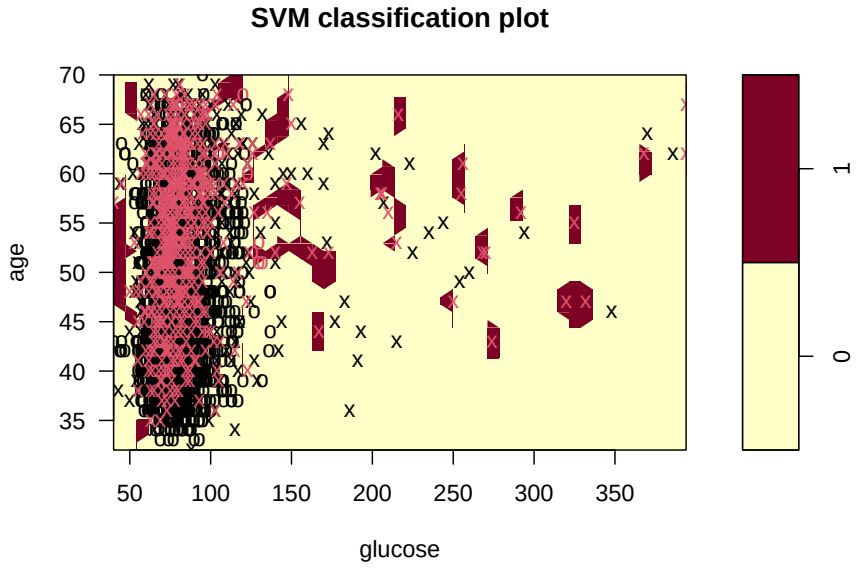
```
## [1] 9
```

Nos queda finalmente ejecutar de nuevo el modelo y explotar sus resultados.

```
fit_svm <- ajusta_svm(mallado$coste[9], mallado$gamma[9])

confusionMatrix(data = fit_svm$fitted,
                reference = df_framingham$ten_year_chd,
                positive = "1",
                prevalence = 0.15)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3084  468
##          1   17   89
##
##                Accuracy : 0.8674
##                  95% CI : (0.856, 0.8782)
##     No Information Rate : 0.8477
##     P-Value [Acc > NIR] : 0.0004096
##
##                   Kappa : 0.231
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.15978
##             Specificity : 0.99452
##          Pos Pred Value : 0.83723
##          Neg Pred Value : 0.87025
##              Prevalence : 0.15000
##          Detection Rate : 0.02433
##    Detection Prevalence : 0.02898
##       Balanced Accuracy : 0.57715
##
##        'Positive' Class : 1
##
```

```
plot(fit_svm, df_predict, age ~  glucose)
```


SVM classification plot
age
glucose
70
65
60
55
50
45
40
35
50
100
150
200
250
300
350
1
0

Chapter 6

Clasificación - Ejercicio práctico

6.1 Lectura

```
library(readr)
library(janitor)
df_churn <- read_csv("data/telecom_churn.csv") %>%
  clean_names()
```

6.2 Tratamiento

Variables numéricas.

```
library(dplyr)
df_churn <- df_churn %>%
  mutate(churn = as.numeric(churn), # logical to 0 and 1
         international_plan = as.numeric(international_plan == "Yes"),
         voice_mail_plan = as.numeric(voice_mail_plan == "Yes"))
```

```
df_churn %>%
  count(state, sort = TRUE) %>%
  slice_head(n = 5)
```

```
## # A tibble: 5 x 2
```

```
##   state     n
##   <chr> <int>
## 1 WV      106
## 2 MN       84
## 3 NY       83
## 4 AL       80
## 5 OH       78
```

Trabajaremos solo con West Virginia (`"WV"`) para evitar el componente geográfico.

```
df_churn_wv <- df_churn %>%
  filter(state == "WV")
```

6.3 Modelo de regresión logística

Hay variables que dependen unas de otras, por lo que no es buena idea introducir todas a la vez en el modelo. Una decisión que habrá que tomar es si trabajamos con variables sobre número de llamadas, sobre minutos hablados o sobre coste.

6.4 Modelo SVM

- Ajusta un SVM al mismo conjunto de datos.

Chapter 7

No supervisado

El conjunto de datos `iris` es un referente en el análisis de datos. Lo usó el estadístico Ronald Fisher en 1936 y desde entonces se recurre a él para ilustrar muchos conceptos de estadística y modelización.

En R está disponible sin necesidad de ninguna librería, con el comando `iris`.

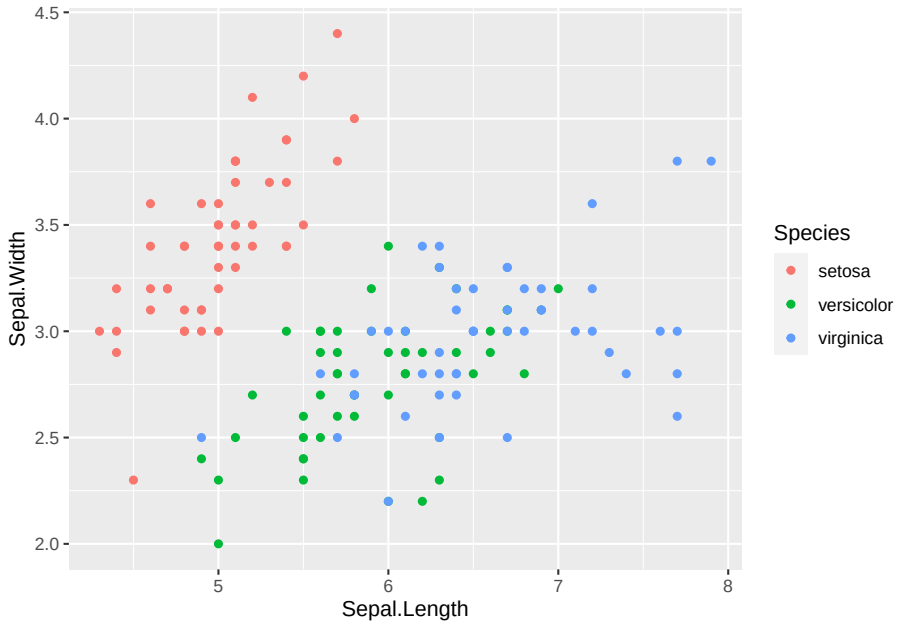
Consta de 150 observaciones con 4 atributos sobre distintas flores de la familia *Iris*. Una quinta columna indica la variedad a la que pertenecen.

El siguiente gráfico muestra los 150 puntos en función de la longitud de sus sépaos y la anchura. Además, están coloreados según la variedad de la flor.

```
library(ggplot2)
head(iris)
```

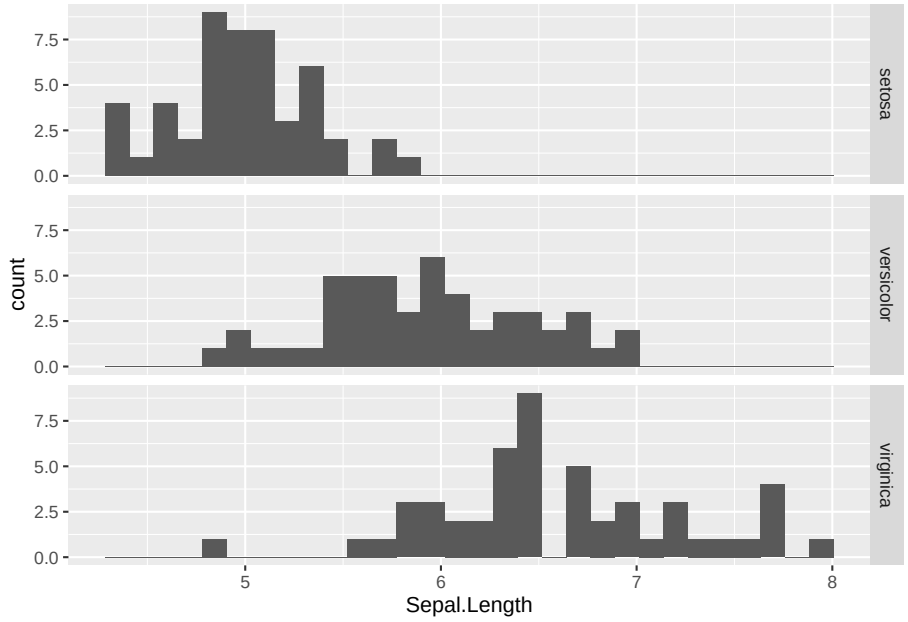
```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
ggplot(iris) +
  geom_point(aes(x = Sepal.Length, y = Sepal.Width, col = Species))
```



En el análisis discriminante `iris` es un ejemplo recurrente ya que se aprecia bien cómo grupos de observaciones pueden distinguirse bien en base a los atributos adecuados. Aunque en el gráfico de dispersión, versicolor y virginica se entremezclan, en el histograma se consigue diferenciar mejor estos grupos.

```
ggplot(iris) +
  geom_histogram(aes(x = Sepal.Length)) +
  facet_grid(Species ~ .)
```



7.1 kmeans

Una obsesión de los humanos es agrupar, crear clases. Es imposible hacer acciones personalizadas: somos muchos y no podemos actuar de manera diferente con cada persona.

Así, si somos una empresa, lanzaremos una campaña de publicidad igual a mucha gente. Quizá el anuncio de televisión cambie si lo emitimos a medio día o por la noche, o quizá la campaña de YouTube vaya dirigida solo a las personas que ven vídeos de autoayuda… pero agrupamos.

Y podemos aprovechar los datos para ello.

Los modelos de *clustering* buscar agrupar observaciones *parecidas entre sí*. Esto se consigue eligiendo los atributos adecuados (variables), de forma que si los atributos numéricamente son parecidos (están cerca), entonces diremos que las observaciones son parecidas.

El algoritmo principal de *clustering* es el llamado k-means. Con él, buscamos generar k grupos de observaciones. Como podemos ver en el gráfico de dispersión de iris, el grupo *setosa* es fácilmente identificable porque todas las observaciones están *cerca entre sí* y lejos de las demás.

Un concepto importante es que estos modelos se llaman **no supervisados.** Con ello nos referimos a que no hay una solución. Aunque vas a trabajar en estas páginas con un dataset conocido, lo normal es que no sepas si una observación

pertenece a un grupo o a otro. El algoritmo te propondrá una solución pero tú no sabrás realmente si está bien o no. Solo puedes comprobar si tiene sentido.

7.1.1 Estandarizar

Algo muy importante es que los atributos elegidos contribuyan adecuadamente. Imagina estas variables de personas: kilocalorías consumidas al día y altura en metros.

id	kcal	mts
1	2000	1,80
2	3000	2,10
3	1800	1,65

¿Qué ocurre con esas variables? Que sus magnitudes no se parecen. Por lo tanto, kcal influirá mucho en alejar y acercar variables (porque su escala se mueve en miles) y la altura apenas en influirá (porque su escala se mueve en torno a 2).

Para solventar eso y que todas las variables que queramos incluir influyan adecuadamente, necesitamos llevarlas a una misma escala.

Una forma de hacer eso es normalizar las variables restando su media y dividiendo por su desviación típica. Así, todas tendrán media 0 y desviación típica 1 (magnitudes comparables).

```
library(dplyr)
df_iris_scaled <- iris %>%
  mutate(across(Sepal.Length:Petal.Width, ~ (. - mean(.)) / sd(.)))
```

7.1.2 Sintaxis y resultados

Para trabajar con el algoritmo kmeans en R necesitas la función `kmeans()` (no hacen falta librerías). Es un algoritmo sencillo, solo necesitas indicar el conjunto de datos y el número de clusters que quieres construir.

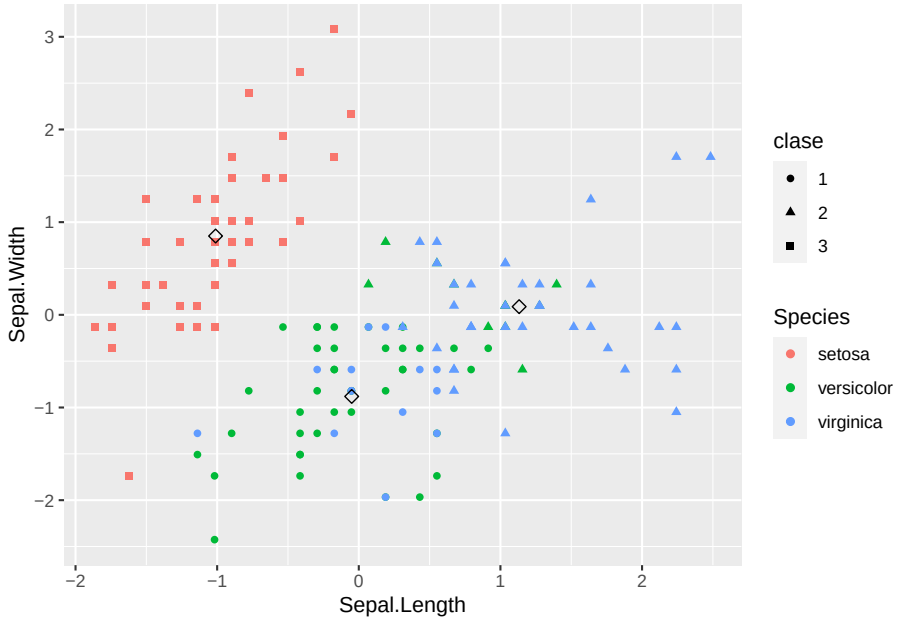
> Sí, en kmeans tienes que indicar el número de grupos. Es lo que se llama un algoritmo no jerárquico. Tú indicas el número de grupos y el algoritmo te los identifica.

```
kmeans_modelo <- kmeans(df_iris_scaled %>% select(-Species), 3)
```

En el siguiente gráfico verás las observaciones del dataset `iris` etiquetadas (por la forma del punto) con su cluster. Los círculos son el cluster 1 y van muy

relacionadas con el color rojo de *setosa*. Los triángulos parecen *virginica* (azul) y los cuadrados *versicolor* (verde).

```
df_iris_scaled %>%
  mutate(clase = as.factor(kmeans_modelo$cluster)) %>%
  ggplot() +
  geom_point(aes(x = Sepal.Length, y = Sepal.Width, col = Species, shape = clase)) +
  geom_point(data = as.data.frame(kmeans_modelo$centers),
             aes(x = Sepal.Length, y = Sepal.Width), size = 2, shape = 5)
```



En este caso, aceptaríamos esta solución y, como conocemos la agrupación real, podríamos comprobar cómo de bien elegida está. Aunque, como hemos dicho más arriba, normalmente no sabremos el grupo original de los individuos.

7.1.3 Número de clusters

Aunque el número de clusters lo tienes que decidir tú, hay técnicas para evaluar si es mejor un número u otro.

Una opción es emplear la métrica *total within-clusters sum of squares*. Sirve para saber cuánta distancia hay entre las observaciones de un mismo cluster. Si las observaciones están muy cerca a su grupo, es bueno porque el cluster estará muy bien agrupado; si es grande, significa que las observaciones distan mucho de su centro del cluster (es un cluster disperso).

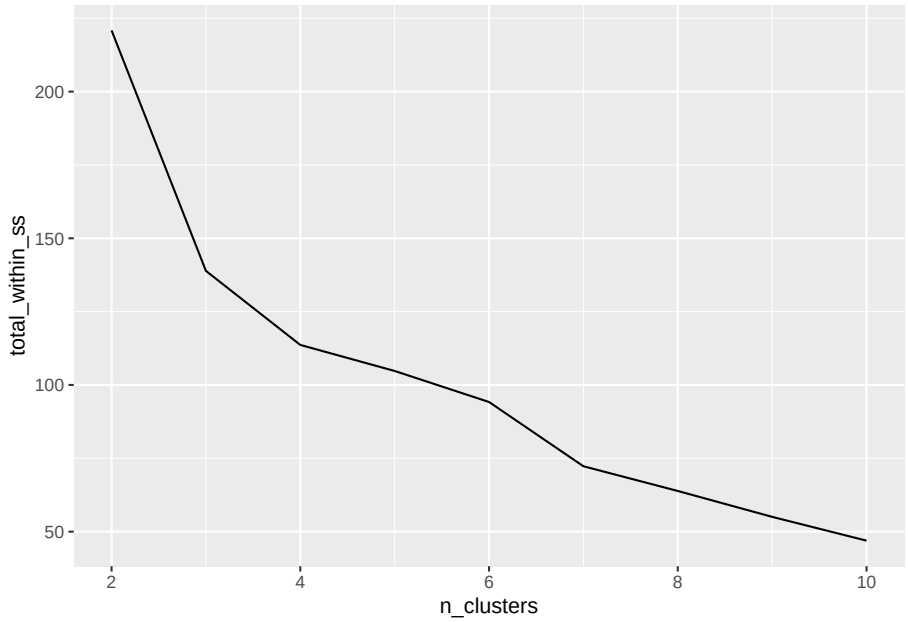
El caso extremo es que cada observación sea un cluster. Pero eso no es práctico.

Con el siguiente código aplicas el algoritmo 9 veces: de 2 a 10 grupos. Verás cómo está métrica mejora (disminuye) a medida que aumentas los grupos.

¿Cuántos grupos eliges?

Pues sinceramente, es a ojo. Lo normal es ver cuándo hay una caída abruta de la métrica y luego no mejora tanto. **Cuidado porque el el caso de iris, recomienda 4 clusters.**

```
df_resultados_varios_clusters <- tibble(
  n_clusters = 2:10,
  total_within_ss = sapply(2:10, \(i){kmeans(df_iris_scaled %>% select(-Species), i)$t
)

ggplot(df_resultados_varios_clusters) +
  geom_line(aes(x = n_clusters, y = total_within_ss))
```




```
# Cuidado: sugiere 4 clusters
```

Al final, el mejor método es calcularlo con el número de grupos que parezca razonable en función del conocimiento de negocio. Y comprobar que los resultados tienen sentido.

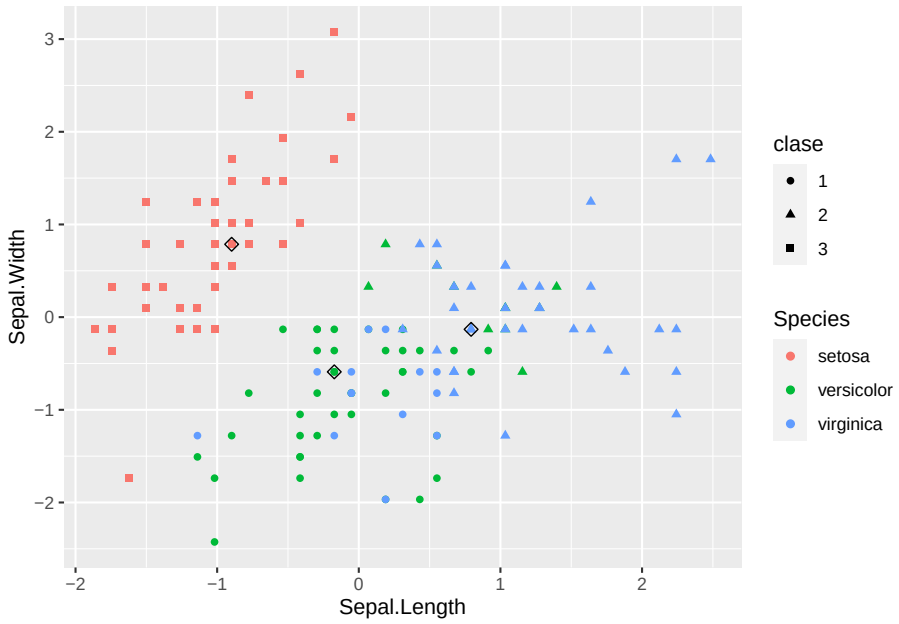
7.2 Clara

El algoritmo kmeans se basa en situar un centro y calcular las distancias de cada observación a ese centro. El centro del cluster es el punto medio de todas las observaciones. Es como el identificador medio de ese cluster: si estamos hablando de clientes, sería el cliente que representa a todos.

¿Problema? Que el centro de un kmeans quizá no sea una observación nuestra. Por lo que el "cliente modelo" no existiría en nuestra base datos.

Un algoritmo alternativo es *PAM*, que busca los grupos fijando como centro una observación real. Pero es un algoritmo muy poco eficiente. *CLARA* es una alternativa común, que ajusta *PAM* sobre muestras de los datos. En R, se necesita cargar la libería *cluster* para usarlo.

```
library(cluster)

clara_iris <- clara(df_iris_scaled %>% select(-Species), k = 3, stand = TRUE)

df_iris_scaled %>%
  mutate(clase = as.factor(kmeans_modelo$cluster)) %>%
  ggplot() +
  geom_point(aes(x = Sepal.Length, y = Sepal.Width, col = Species, shape = clase)) +
  geom_point(data = as.data.frame(clara_iris$medoids),
             aes(x = Sepal.Length, y = Sepal.Width), size = 2, shape = 5)
```



Chapter 8

No supervisado - Ejercicio práctico

```
library(rvest)
library(purrr)

url <- "http://online.wsj.com/public/resources/documents/info-Degrees_that_Pay_you_Back-sort.html
wsj <- read_html(url)

df_url_degrees <- wsj %>%
  html_elements("table") %>%
  html_table(header = TRUE) %>%
  pluck(7)

# write_csv(df_url_degrees, "data/degrees.csv")
```

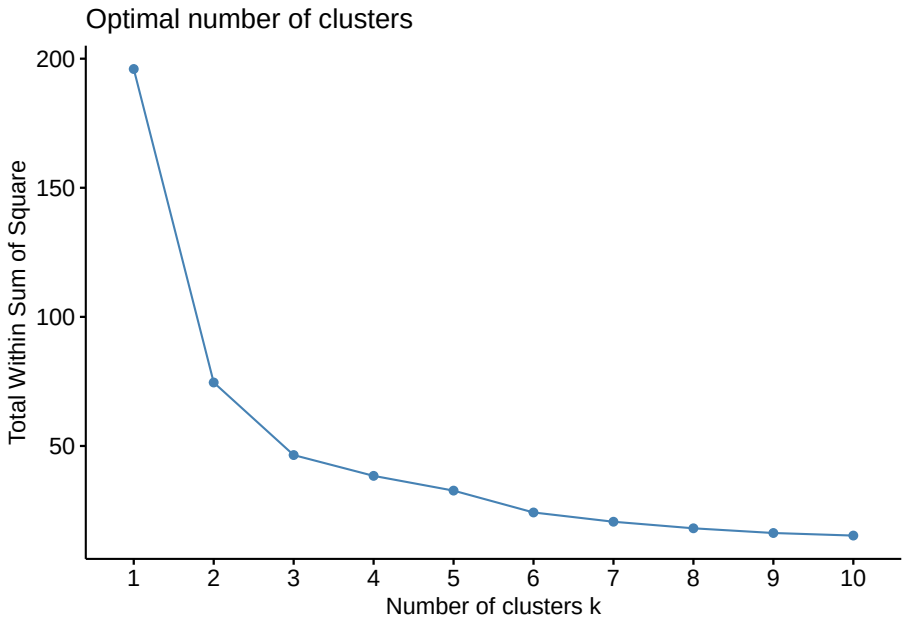
```
library(janitor)
library(dplyr)
library(stringr)

df_url_degrees <- df_url_degrees %>%
  clean_names() %>%
  rename(percent_change = percent_change_from_starting_to_mid_career_salary) %>%
  mutate(across(-c(undergraduate_major, percent_change),
                ~ as.numeric(str_remove_all(., "\\$|\\,"))),
         percent_change = percent_change / 100)

df_cluster_data <- df_url_degrees %>%
```

```
  select(starting_median_salary, mid_career_median_salary, mid_career_10th_percentile_s
  scale()
```

8.0.1 Ejercicio

- Construye un gráfico con un bucle o similar para estudiar cuántos grupos de cluster se deberían hacer. Compáralo con el siguiente gráfico.
- Ajusta un modelo de kmeans o de Clara a los datos.

```
library(factoextra)
fviz_nbclust(df_cluster_data, FUNcluster = kmeans, method = "wss")
```



Bibliography

Xie, Y. (2015). *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition. ISBN 978-1498716963.

Xie, Y. (2022). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.26.