

# PROFESSIONAL CERTIFICATE IN DATA SCIENCE



---

## Notas generales

---

Elaborado por: Marco Méndez Atienza

18 de julio de 2021

# Índice

<b>1. Introducción a R</b>	<b>2</b>
1.1. Básicos, funciones y tipos de datos . . . . .	2
1.1.1. Básicos . . . . .	2
1.1.1.1. Liberías . . . . .	2
1.1.1.2. Creación de objetos . . . . .	2
1.1.1.3. Observación de objetos . . . . .	2
1.1.1.4. Observación de todos los objetos . . . . .	2
1.1.1.5. Resolución de ecuaciones . . . . .	2
1.1.1.6. Funciones . . . . .	2
1.1.1.7. Ayuda . . . . .	3
1.1.1.8. Bases de datos predeterminadas . . . . .	3
1.1.1.9. Nombres de objetos . . . . .	3
1.1.2. Tipos de datos . . . . .	4
1.1.2.1. Clases . . . . .	4
1.1.2.2. Estructura . . . . .	4
1.1.2.3. Accesor . . . . .	4
1.1.2.4. Longitud . . . . .	5
1.1.2.5. Caracteres . . . . .	5
1.1.2.6. Lógicos . . . . .	5
1.1.2.7. Factores . . . . .	5
1.1.2.8. Frecuencia . . . . .	5
1.1.3. Assessment 1 . . . . .	7
1.2. Vectores y acomodo . . . . .	9
1.2.1. Vectores . . . . .	9
1.2.1.1. Subconjuntos . . . . .	9
1.2.2. Coerción de vectores . . . . .	10
1.2.2.1. Datos faltantes . . . . .	10
1.2.3. Acomodo . . . . .	11
1.2.4. Aritmética de vectores . . . . .	12
1.3. Indexado, manipulación de datos y gráficos básicos . . . . .	14
1.3.1. Indexado . . . . .	14
1.3.1.1. Operadores lógicos . . . . .	14
1.3.1.2. Funciones lógicas . . . . .	14
1.3.2. Manipulación de datos básica . . . . .	15
1.3.2.1. dplyr básico . . . . .	15
1.3.2.2. Creando data frames . . . . .	16
1.3.3. Gráficos básicos . . . . .	17
1.3.3.1. Histograma . . . . .	17
1.3.3.2. Boxplot . . . . .	18
1.3.4. Assessment 2 . . . . .	19
1.4. Programación básica . . . . .	25
1.4.1. Condicionales . . . . .	25
1.4.1.1. if/else . . . . .	25
1.4.1.2. ifelse . . . . .	25
1.4.1.2.1. Ejemplo . . . . .	26
1.4.1.3. any/all . . . . .	26
1.4.1.3.1. Ejemplo . . . . .	26
1.4.2. Funciones . . . . .	27
1.4.2.1. Ejemplo . . . . .	27
1.4.3. Bucles . . . . .	29
1.4.3.1. Ejemplo . . . . .	31
1.4.4. Otras funciones . . . . .	32

1.4.5. Assessment 3 . . . . .	32
<b>2. Visualización en R</b> . . . . .	<b>34</b>
2.1. Introducción a la visualización de datos y distribuciones . . . . .	34
2.1.1. Introducción . . . . .	34
2.1.2. Distribuciones . . . . .	34
2.1.3. Tipos de datos . . . . .	34
2.1.4. Introducción a las distribuciones . . . . .	35
2.1.4.1. CDF . . . . .	35
2.1.5. Cuantiles, percentiles y boxplots . . . . .	36
2.1.5.1. Cuantiles . . . . .	36
2.1.5.2. qnorm . . . . .	37
2.1.5.3. Gráficas cuantil-cuantil . . . . .	38
2.1.5.4. Normalizando . . . . .	39
2.1.5.5. Boxplots . . . . .	40
2.1.6. Análisis de datos exploratorio (EDA) . . . . .	41
2.1.6.1. Ejemplo . . . . .	41
2.1.6.2. Impacto de outliers . . . . .	44
2.2. Introducción a ggplot2 . . . . .	45
2.2.1. Introducción . . . . .	45
2.2.2. Componentes del gráfico . . . . .	45
2.2.3. Pasos para crear un gráfico ggplot . . . . .	45
2.2.4. Resultado final . . . . .	59
2.2.5. Versión alternativa . . . . .	60
2.2.6. Otros ejemplos . . . . .	61
2.2.7. Rejillas de gráficos . . . . .	69
2.3. Resumiendo con dplyr . . . . .	71
2.3.1. Introducción . . . . .	71
2.3.2. Marcador de posición de punto . . . . .	71
2.3.3. Group by . . . . .	72
2.3.4. Promedios por grupo . . . . .	73
2.3.5. Clasificación de tablas de datos . . . . .	75
2.4. Gapminder . . . . .	77
2.4.1. Facetado . . . . .	78
2.4.2. Gráficos de series de tiempo . . . . .	81
2.4.3. Transformaciones . . . . .	85
2.4.4. Estratificación y boxplots . . . . .	89
2.4.5. Comparando distribuciones . . . . .	94
2.4.6. Gráficos de densidad . . . . .	99
2.4.7. Falacia ecológica . . . . .	102
2.5. Principios de visualización de datos . . . . .	104
2.5.1. Codificar datos con señales visuales . . . . .	104
2.5.2. Cuándo incluir 0 . . . . .	105
2.5.3. Mostrando los datos . . . . .	105
2.5.4. Ejes consistentes . . . . .	107
2.5.5. Facilitación de comparaciones . . . . .	107
2.5.6. Gráficos de pendiente . . . . .	108
2.5.7. Gráfica Bland-Altman . . . . .	109
2.5.8. Codificando una tercera variable . . . . .	110
2.5.9. Gráficos 3D . . . . .	113
2.5.10. Dígitos significativos . . . . .	113
2.5.11. Assessment 4 . . . . .	114
2.6. Comprehensive assessment . . . . .	123

<b>3. Probabilidad</b>	<b>141</b>
<b>3.1. Probabilidad discreta</b>	141
3.1.1. Introducción	141
3.1.1.1. Notación	141
3.1.1.2. Simulaciones Monte Carlo	141
3.1.1.3. Función mean()	142
3.1.1.4. Distribuciones de probabilidad	142
3.1.1.5. Independencia	142
3.1.2. Combinaciones y permutaciones	142
3.1.2.1. El problema del cumpleaños	145
3.1.2.2. sapply	145
3.1.2.3. ¿Cuántos experimentos Monte Carlo son suficientes?	147
3.1.3. Regla de adición	148
3.1.4. Monty Hall	148
3.1.5. Assessment 5	149
<b>3.2. Probabilidad continua</b>	156
3.2.1. Distribución teórica	156
3.2.2. Densidad de probabilidad	158
3.2.3. Simulaciones Monte Carlo	159
3.2.4. Otras distribuciones continuas	160
3.2.5. Assessment 6	161
<b>3.3. Variable aleatorias, muestreo y Teorema del Límite Central (CLT)</b>	165
3.3.1. Variables aleatorias	165
3.3.2. Muestras aleatorias	165
3.3.3. Distribuciones vs. Distribuciones de probabilidad	167
3.3.4. Notación de variables aleatorias	167
3.3.5. Promedios y proporciones	168
3.3.6. Ley de los grandes números	169
3.3.7. ¿Qué tan grande tiene que ser n?	169
3.3.8. Assessment 7	169
<b>3.4. The Big Short</b>	173
3.4.1. Tasas de interés	173
3.4.2. The Big Short	176
3.4.3. Assessment 8	177
<b>4. Inferencia y modelaje</b>	<b>184</b>
<b>4.1. Parámetros y estimadores</b>	184
4.1.1. Parámetros de un muestreo y estimadores	184
4.1.2. Promedio muestral	188
4.1.3. Sondeo vs. pronóstico	188
4.1.4. Propiedades de los estimadores	189
<b>4.2. EL TLC en práctica</b>	190
4.2.1. Márgenes de error	191
4.2.2. Una simulación Monte Carlo para el TLC	191
4.2.3. Dispersión	192
4.2.4. Tamaño de las encuestas	192
<b>4.3. Intervalos de confianza y valores-p</b>	194
4.3.1. Potencia	195
4.3.2. Valores-p	196
<b>4.4. Modelos estadísticos</b>	197
4.4.1. Agregados de encuestas	197
4.4.2. Modelos multinivel y sesgos	198
4.4.3. Modelos basados en datos	202
<b>4.5. Estadística Bayesiana</b>	204

4.5.1.	Teorema de Bayes . . . . .	204
4.5.2.	Bayes en la práctica . . . . .	205
4.5.3.	El modelo jerárquico . . . . .	205
4.6.	<b>Pronósticos electorales</b> . . . . .	207
4.6.1.	Representación matemática de modelos . . . . .	208
4.6.2.	Prediciendo el Colegio Electoral . . . . .	210
4.6.3.	Pronósticos . . . . .	213
4.6.4.	La distribución t . . . . .	216
4.7.	<b>Pruebas de asociación</b> . . . . .	218
4.7.1.	Test Chi-Cuadrada . . . . .	219
4.8.	<b>Comprehensive Assessment</b> . . . . .	222
<b>5.</b>	<b>Herramientas de productividad</b>	<b>231</b>
5.1.	<b>Instalando el software</b> . . . . .	231
5.1.1.	Introducción a R Studio . . . . .	231
5.1.2.	Introducción a Git y GitHub . . . . .	231
5.2.	<b>Unix Básico</b> . . . . .	232
5.2.1.	Introducción a Unix . . . . .	232
5.2.2.	Trabajando con Unix . . . . .	232
5.3.	<b>Reportes</b> . . . . .	233
5.4.	<b>Git y GitHub</b> . . . . .	234
5.5.	<b>Unix avanzado</b> . . . . .	235
5.5.1.	Argumentos . . . . .	235
5.5.2.	Ayudas y pipas . . . . .	235
5.5.3.	Comodines . . . . .	235
5.5.4.	Variables ambientales y <i>shells</i> . . . . .	235
5.5.5.	Ejecutables, permisos y tipos de archivos . . . . .	235
5.5.6.	Comandos útiles . . . . .	236
<b>6.</b>	<b>Data Wrangling</b>	<b>237</b>
6.1.	<b>Importación de datos</b> . . . . .	237
6.1.1.	Hojas de cálculo . . . . .	237
6.1.2.	Rutas y directorio de trabajo . . . . .	237
6.1.3.	readr y readxl . . . . .	238
6.1.4.	Importación de datos con funciones de R base . . . . .	239
6.1.5.	Archivos del internet . . . . .	239
6.1.6.	Assessment 9 . . . . .	241
6.2.	<b>Ordenación de datos</b> . . . . .	242
6.2.1.	Reorganización de datos . . . . .	242
6.2.2.	Assessment 10 . . . . .	248
6.2.3.	Combinación de tablas . . . . .	251
6.2.4.	Assessment 11 . . . . .	258
6.2.5.	Web scraping . . . . .	260
6.3.	<b>Procesamiento de cadenas</b> . . . . .	262
6.3.1.	Comillas simples y dobles . . . . .	263
6.3.2.	stringr . . . . .	263
6.3.3.	Ejemplo: Asesinatos en EE.UU. . . . .	264
6.3.4.	Ejemplo: Alturas reportadas . . . . .	265
6.3.5.	Regex . . . . .	268
6.3.6.	Clases de caracteres, anclajes y cuantificadores . . . . .	269
6.3.7.	Grupos con Regex . . . . .	272
6.3.8.	Testeo y mejora . . . . .	274
6.3.9.	Separación con Regex . . . . .	275
6.3.9.1.	Resultado final . . . . .	278

6.3.10. Separación de cadenas . . . . .	279
6.3.11. Recodificar . . . . .	282
6.3.12. Assessment 12 . . . . .	283
<b>6.4. Fechas y tiempos . . . . .</b>	<b>286</b>
<b>6.5. Minería de texto . . . . .</b>	<b>290</b>
6.5.1. Caso de estudio: Tweets de Trump . . . . .	290
<b>7. Regresión Lineal . . . . .</b>	<b>304</b>
<b>7.1. Introducción a las regresiones . . . . .</b>	<b>304</b>
7.1.1. Ejemplo: Baseball . . . . .	304
7.1.2. Correlación . . . . .	309
7.1.3. Estratificación y varianza . . . . .	312
7.1.3.1. Distribución normal bivariada . . . . .	318
7.1.4. Assessment 13 . . . . .	321
<b>7.2. Modelos lineales . . . . .</b>	<b>323</b>
7.2.1. Introducción . . . . .	323
7.2.1.1. Regresión multivariada . . . . .	323
7.2.2. MCO . . . . .	326
7.2.3. Assessment 14 . . . . .	332
7.2.4. Tibbles, do y broom . . . . .	339
7.2.4.1. Tibbles . . . . .	339
7.2.4.2. do . . . . .	341
7.2.4.3. broom . . . . .	343
7.2.5. Assessment 15 . . . . .	345
7.2.6. Regresiones y baseball . . . . .	349
7.2.6.1. Falacia de regresión . . . . .	357
7.2.6.2. Modelos de errores de medición . . . . .	360
7.2.7. Assessment 16 . . . . .	362
7.2.8. Assessment 17 . . . . .	365
<b>7.3. Confounding . . . . .</b>	<b>369</b>
7.3.1. Correlación y causalidad . . . . .	369
7.3.1.1. Correlaciones espurias . . . . .	369
7.3.1.2. <i>Outliers</i> . . . . .	371
7.3.1.3. Inversión de causa y efecto . . . . .	373
7.3.1.4. Factores de confusión . . . . .	374
7.3.1.5. Paradoja Simpson . . . . .	378
7.3.2. Assessment 18 . . . . .	378
<b>8. Machine Learning . . . . .</b>	<b>382</b>
<b>8.1. Introducción . . . . .</b>	<b>382</b>
8.1.1. Notación . . . . .	382
8.1.2. Ejemplo . . . . .	382
<b>8.2. Básicos de Machine Learning . . . . .</b>	<b>384</b>
8.2.1. Evaluación de algoritmos de Machine Learning . . . . .	384
8.2.1.1. Matriz de confusión . . . . .	387
8.2.1.2. Precisión balanceada y puntaje $F_1$ . . . . .	389
8.2.1.3. Prevalencia en la práctica . . . . .	391
8.2.1.4. ROC y curvas <i>precision-recall</i> . . . . .	391
8.2.2. Assessment 19 . . . . .	397
8.2.3. Probabilidades condicionales y función de pérdida . . . . .	402
8.2.4. Assessment 20 . . . . .	404
<b>8.3. Regresiones lineales para predicción, suavización y trabajo con matrices . . . . .</b>	<b>407</b>
8.3.1. Regresiones lineales para predicciones . . . . .	407
8.3.2. Assessment 21 . . . . .	408

8.3.2.1.	Regresiones para resultados categóricos . . . . .	412
8.3.2.2.	Regresión logística . . . . .	414
8.3.2.3.	Caso de estudio: 2 o 7 . . . . .	417
8.3.3.	Suavización . . . . .	426
8.3.4.	Assessment 22 . . . . .	434
8.3.5.	Trabajando con matrices . . . . .	439
8.3.5.1.	Notación matricial . . . . .	439
8.3.6.	Assessment 23 . . . . .	448
8.4.	<b>Distancia, kNN, validación cruzada y modelos generativos</b> . . . . .	450
8.4.1.	Vecinos cercanos . . . . .	450
8.4.2.	Assessment 24 . . . . .	454
8.4.3.	kNN . . . . .	456
8.4.4.	Assessment 25 . . . . .	459
8.4.5.	Validación cruzada . . . . .	462
8.4.5.1.	k-fold . . . . .	462
8.4.6.	Assessment 26 . . . . .	464
8.4.6.1.	Bootstrap . . . . .	467
8.4.7.	Assessment 27 . . . . .	471
8.4.8.	Modelos generativos . . . . .	474
8.4.8.1.	Controlando prevalencia . . . . .	475
8.4.8.2.	qda y lda . . . . .	476
8.4.8.3.	Más de tres clases . . . . .	480
8.4.9.	Assessment 28 . . . . .	485
8.5.	<b>Clasificación con más de dos clases y paquete Caret</b> . . . . .	490
8.5.1.	Clasificación con más de dos clases . . . . .	490
8.5.1.1.	Árboles de clasificación . . . . .	501
8.5.1.2.	Bosques aleatorios . . . . .	503
8.5.2.	Assessment 29 . . . . .	506
8.5.3.	Paquete Caret . . . . .	512
8.5.3.1.	Parámetros de afinación . . . . .	512
8.5.4.	Ejercicios del Titanic . . . . .	520
8.6.	<b>Ajuste de modelos y sistemas de recomendación</b> . . . . .	532
8.6.1.	Caso de estudio: MNIST . . . . .	532
8.6.1.1.	kNN . . . . .	534
8.6.2.	Assessment 30 . . . . .	542
8.6.3.	Sistemas de recomendación . . . . .	544
8.6.4.	Assessment 31 . . . . .	553
8.6.5.	Regularización . . . . .	558
8.6.6.	Assessment 32 . . . . .	566
8.6.6.1.	Factorización de matrices . . . . .	572
8.6.6.2.	SVD y PCA . . . . .	577
8.6.7.	Assessment 33 . . . . .	583
8.6.8.	Assessment 34 . . . . .	590
8.6.9.	Comprehensive Assessment . . . . .	606

## 1. Introducción a R

### 1.1. Básicos, funciones y tipos de datos

#### 1.1.1. Básicos

**1.1.1.1. Librerías** Para cargar librerías, se utiliza el comando “library”:

```
library(tidyverse)  
  
library(dslabs)
```

**1.1.1.2. Creación de objetos** Para crear objetos, utilizamos el operador “`<-`”:

```
a <- 1  
b <- 1  
c <- -1
```

**1.1.1.3. Observación de objetos** Para observar los objetos creados, puede escribirse el valor deseado o, de manera más explícita, “`print()` en la consola”:

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

**1.1.1.4. Observación de todos los objetos** Pueden observarse todas los objetos creados mediante “`ls()`”:

```
ls()  
  
## [1] "a" "b" "c"
```

**1.1.1.5. Resolución de ecuaciones** Así, pueden resolverse ecuaciones a partir de los objetos creados:

```
(-b+sqrt(b^2-4*a*c))/(2*a)  
  
## [1] 0.618034  
  
(-b-sqrt(b^2-4*a*c))/(2*a)  
  
## [1] -1.618034
```

**1.1.1.6. Funciones** Las funciones suelen requerir un argumento. Si las funciones están anidadas, se resuelven de adentro hacia afuera:

```
a <- 1  
b <- 1  
c <- -1  
  
log(8)  
  
## [1] 2.079442  
  
log(a)  
  
## [1] 0  
  
exp(1)  
  
## [1] 2.718282
```

```
log(exp(1))
```

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

**1.1.1.7. Ayuda** Los archivos de ayuda son muy útiles para conocer el funcionamiento de las funciones y se utilizan con el comando “help()”:

```
#help("log")
```

Para muchas funciones, puede escribirse “?”:

```
#?log  
#?"+"
```

Para conocer los argumentos de una función:

```
args(log)
```

```
## function (x, base = exp(1))  
## NULL
```

**1.1.1.8. Bases de datos predeterminadas** Existen algunas bases de datos precargadas en R para practicar:

```
data()
```

**1.1.1.9. Nombres de objetos** Algunas reglas para los nombres de objetos son:

1. Tienen que comenzar con una letra; y
2. No pueden contener espacios.

```
solution_1 <- (-b + sqrt(b^2 - 4*a*c))/2*a  
solution_2 <- (-b - sqrt(b^2 - 4*a*c))/2*a
```

```
solution_1
```

```
## [1] 0.618034
```

```
solution_2
```

```
## [1] -1.618034
```

### 1.1.2. Tipos de datos

**1.1.2.1. Clases** Si se define `a = 2`, la función “`class()`” arrojará el tipo de objeto que es:

```
a <- 2
class(a)

## [1] "numeric"

class(ls)

## [1] "function"
```

La forma más común de almacenar datos en R es mediante **dataframes**. Los data frames son tablas que representan observaciones y diferentes variables. Permiten combinar muchos conjuntos de datos en un solo data frame.

```
library(dslabs)
data("murders")
class(murders)

## [1] "data.frame"
```

**1.1.2.2. Estructura** La función “`str()`” muestra la estructura de un data frame.

```
str(murders)

## 'data.frame': 51 obs. of 5 variables:
## $ state      : chr "Alabama" "Alaska" "Arizona" "Arkansas" ...
## $ abb        : chr "AL" "AK" "AZ" "AR" ...
## $ region     : Factor w/ 4 levels "Northeast","South",...: 2 4 4 2 4 4 1 2 2 2 ...
## $ population: num 4779736 710231 6392017 2915918 37253956 ...
## $ total      : num 135 19 232 93 1257 ...
```

Donde “\$” denota cada variable. Similarmente, “`names()`” arroja los nombres de las columnas:

```
names(murders)

## [1] "state"      "abb"        "region"     "population" "total"
```

La función “`head()`” nos muestra las primeras 6 observaciones del data frame.

```
head(murders)

##       state abb region population total
## 1   Alabama  AL  South     4779736   135
## 2    Alaska  AK  West      710231    19
## 3  Arizona  AZ  West     6392017   232
## 4 Arkansas  AR  South     2915918    93
## 5 California  CA  West     37253956  1257
## 6 Colorado  CO  West     5029196    65
```

Donde las filas representan observaciones y las columnas, variables.

**1.1.2.3. Accesor** El símbolo \$ permite acceder a las variables dentro de un data frame. Es importante recalcar que lo resultados de esta función respeta el orden que los datos tienen en el data frame.

```
murders$population

## [1] 4779736 710231 6392017 2915918 37253956 5029196 3574097 897934
## [9] 601723 19687653 9920000 1360301 1567582 12830632 6483802 3046355
```

```
## [17] 2853118 4339367 4533372 1328361 5773552 6547629 9883640 5303925
## [25] 2967297 5988927 989415 1826341 2700551 1316470 8791894 2059179
## [33] 19378102 9535483 672591 11536504 3751351 3831074 12702379 1052567
## [41] 4625364 814180 6346105 25145561 2763885 625741 8001024 6724540
## [49] 1852994 5686986 563626
```

**1.1.2.4. Longitud** También puede saberse el número de observaciones dentro de una variable:

```
pop <- murders$population

length(pop)
```

```
## [1] 51
```

```
class(pop)
```

```
## [1] "numeric"
```

**1.1.2.5. Caracteres** Se utilizan para distinguir entre nombres de variables y las “cuerdas” de caracteres:

```
a <- 1
```

```
a
```

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

```
"a"
```

```
## [1] "a"
```

```
class(murders$state)
```

```
## [1] "character"
```

**1.1.2.6. Lógicos** Los objetos lógicos se basan en un operador racional o lógico.

```
z <- 3 == 2
```

```
z
```

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

```
class(z)
```

```
## [1] "logical"
```

**1.1.2.7. Factores** Los factores son utilizados para almacenar datos categóricos. En el caso del data set de murders, la variable región es un ejemplo de esto. Además, la función “levels()” nos dice cuáles son estas categorías.

```
class(murders$region)
```

```
## [1] "factor"
```

```
levels(murders$region)
```

```
## [1] "Northeast"      "South"          "North Central"   "West"
```

**1.1.2.8. Frecuencia** La función “table()” arroja el número de observaciones para la variable indicada:

```
table(murders$region)
```

```
##          Northeast      South      North      Central      West
##                9             17            12            13
```

### 1.1.3. Assessment 1

Pregunta 1.

```
solucion_1 <- (1 + sqrt(1 + 4*2*(4)))/(2*2)
solucion_2 <- (1 - sqrt(1 + 4*2*(4)))/(2*2)
```

```
solucion_1
```

```
## [1] 1.686141
```

```
solucion_2
```

```
## [1] -1.186141
```

Pregunta 2.

```
log(1024, base = 4)
```

```
## [1] 5
```

Pregunta 3.

```
library(dslabs)
```

```
data("movielens")
```

```
str(movielens)
```

```
## 'data.frame': 100004 obs. of 7 variables:
## $ movieId : int 31 1029 1061 1129 1172 1263 1287 1293 1339 1343 ...
## $ title   : chr "Dangerous Minds" "Dumbo" "Sleepers" "Escape from New York" ...
## $ year    : int 1995 1941 1996 1981 1989 1978 1959 1982 1992 1991 ...
## $ genres  : Factor w/ 901 levels "(no genres listed)",...: 762 510 899 120 762 836 81 762 844 899 ...
## $ userId  : int 1 1 1 1 1 1 1 1 1 ...
## $ rating  : num 2.5 3 3 2 4 2 2 2 3.5 2 ...
## $ timestamp: int 1260759144 1260759179 1260759182 1260759185 1260759205 1260759151 1260759187 1260
```

```
head(movielens)
```

	movieId	title	year
## 1	31	Dangerous Minds	1995
## 2	1029	Dumbo	1941
## 3	1061	Sleepers	1996
## 4	1129	Escape from New York	1981
## 5	1172	Cinema Paradiso (Nuovo cinema Paradiso)	1989
## 6	1263	Deer Hunter, The	1978

	genres	userId	rating	timestamp
## 1	Drama	1	2.5	1260759144
## 2	Animation Children Drama Musical	1	3.0	1260759179
## 3	Thriller	1	3.0	1260759182
## 4	Action Adventure Sci-Fi Thriller	1	2.0	1260759185
## 5	Drama	1	4.0	1260759205
## 6	Drama War	1	2.0	1260759151

```
class(movielens$title)
```

```
## [1] "character"
```

```
class(movielens$genres)
```

```
## [1] "factor"
```

```
niveles <- levels(movielens$genres)
nlevels(movielens$genres)

## [1] 901
```

## 1.2. Vectores y acomodo

### 1.2.1. Vectores

Son la unidad más básica para almacenar datos en R.

Para crear un vector, puede usarse la función de concatenar, “c()”:

```
codes <- c(380, 124, 818)

country <- c("italy", "canada", "egypt")

codes <- c(italy=380, canada =124, egypt = 818)

codes

##  italy canada egypt
##    380      124      818
class(codes)

## [1] "numeric"
```

La función “names()” se utiliza para asignar nombres a los vectores:

```
names(codes) <- country

codes

##  italy canada egypt
##    380      124      818
```

Otra función para crear vectores es la función “seq()”:

```
seq(1,10)

##  [1]  1  2  3  4  5  6  7  8  9 10

seq(1, 10, 0.5)

##  [1]  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0  5.5  6.0  6.5  7.0  7.5  8.0
## [16]  8.5  9.0  9.5 10.0
```

Equivalentemente, si se quieren enteros consecutivos, puede usarse:

```
1:10

##  [1]  1  2  3  4  5  6  7  8  9 10
```

**1.2.1.1. Subconjuntos** Se utilizan los brackets para acceder a elementos de un vector:

```
codes[2]

## canada
##    124

codes[c(1,3)]

## italy egypt
##    380    818

codes[1:2]

##  italy canada
```

```
##      380     124
```

También se puede acceder mediante los nombres de los vectores:

```
codes["canada"]
```

```
## canada
```

```
##     124
```

```
codes[c("egypt", "italy")]
```

```
## egypt italy
```

```
##    818    380
```

### 1.2.2. Coerción de vectores

Los vectores deben ser todos del mismo tipo: así, si se intenta combinar números y caracteres, se arrojará un error, dado que R intentará homologar la clase de los elementos del vector. Por ejemplo, en el siguiente ejemplo, 1 y 3 se vuelven caracteres:

```
x <- c(1, "canada", 3)
```

```
x
```

```
## [1] "1"      "canada" "3"
```

```
class(x)
```

```
## [1] "character"
```

Se dice que R coercionó los datos para volverlos caracteres.

Por otro lado, R ofrece la posibilidad de forzar esta coerción, mediante los siguientes comandos:

```
x <- 1:5
```

```
y <- as.character(x)
```

```
y
```

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

```
z <- as.numeric(y)
```

```
z
```

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

#### 1.2.2.1. Datos faltantes

Cuando R no puede coercionar algo, aparece NA:

```
x <- c("1", "b", "3")
```

```
as.numeric(x)
```

```
## Warning: NAs introduced by coercion
```

```
## [1] 1 NA 3
```

La función “is.na()” determina cuántas observaciones son NA dentro de un vector:

```
library(dslabs)
```

```
data("na_example")
```

```
ind <- is.na(na_example)
sum(ind)
## [1] 145
```

A veces, puede ser necesario remover los NA, dado que algunas funciones como “mean()” arrojarán NA si encuentran al menos 1:

```
mean(na_example[!ind])
## [1] 2.301754
```

### 1.2.3. Acomodo

La función “sort()” acomoda un vector en orden creciente:

```
sort(murders$total)
## [1] 2 4 5 5 7 8 11 12 12 16 19 21 22 27 32
## [16] 36 38 53 63 65 67 84 93 93 97 97 99 111 116 118
## [31] 120 135 142 207 219 232 246 250 286 293 310 321 351 364 376
## [46] 413 457 517 669 805 1257
```

La función “order()” toma un vector y arroja los índices que acomodan ese vector:

```
x <- c(31, 4, 15, 92, 65)
x
```

```
## [1] 31 4 15 92 65
sort(x)
## [1] 4 15 31 65 92
index <- order(x)
```

```
x[index]
```

```
## [1] 4 15 31 65 92
```

Ordenemos los estados a partir del número de homicidios:

```
index <- order(murders$total)

murders$abb[index]
## [1] "VT" "ND" "NH" "WY" "HI" "SD" "ME" "ID" "MT" "RI" "AK" "IA" "UT" "WV" "NE"
## [16] "OR" "DE" "MN" "KS" "CO" "NM" "NV" "AR" "WA" "CT" "WI" "DC" "OK" "KY" "MA"
## [31] "MS" "AL" "IN" "SC" "TN" "AZ" "NJ" "VA" "NC" "MD" "OH" "MO" "LA" "IL" "GA"
## [46] "MI" "PA" "NY" "FL" "TX" "CA"
```

Donde puede verse que VT es el estado con menos homicidios y CA el estado con más.

Existe una manera más fácil de hacer obtener, mediante las funciones “max()” y “min()”:

```
max(murders$total)
## [1] 1257
min(murders$total)
## [1] 2
```

Y si queremos conocer la observación asociada a estos valores, utilizamos “which.max()”, “which.min()”:

```
i_max <- which.max(murders$total)

i_max

## [1] 5

murders$state[i_max]

## [1] "California"

i_min <- which.min(murders$total)

i_min

## [1] 46

murders$state[i_min]

## [1] "Vermont"
```

Otra función útil es “rank()”, que, para un vector dado, arroja el rango de la primera entrada, la segunda, etc:

```
x <- c(31, 4, 15, 92, 65)

rank(x)

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

#### 1.2.4. Aritmética de vectores

¿Es California el estado más violento? Sabemos que es el más poblado:

```
murders$state[which.max(murders$population)]

## [1] "California"

max(murders$population)

## [1] 37253956
```

Es importante recalcar que las operaciones con vectores funcionan “element-wise”:

```
heights <- c(69, 62, 66, 70, 70, 73, 67, 73, 67, 70)

heights*2.54

## [1] 175.26 157.48 167.64 177.80 177.80 185.42 170.18 185.42 170.18 177.80

heights-69

## [1] 0 -7 -3 1 1 4 -2 4 -2 1
```

Para hacer una comparación justa de asesinatos, hay que obtener los asesinatos per cápita.

```
murder_rate <- murders$total/murders$population*100000

murder_rate

## [1] 2.8244238 2.6751860 3.6295273 3.1893901 3.3741383 1.2924531
## [7] 2.7139722 4.2319369 16.4527532 3.3980688 3.7903226 0.5145920
## [13] 0.7655102 2.8369608 2.1900730 0.6893484 2.2081106 2.6732010
```

```
## [19] 7.7425810 0.8280881 5.0748655 1.8021791 4.1786225 0.9992600
## [25] 4.0440846 5.3598917 1.2128379 1.7521372 3.1104763 0.3798036
## [31] 2.7980319 3.2537239 2.6679599 2.9993237 0.5947151 2.6871225
## [37] 2.9589340 0.9396843 3.5977513 1.5200933 4.4753235 0.9825837
## [43] 3.4509357 3.2013603 0.7959810 0.3196211 3.1246001 1.3829942
## [49] 1.4571013 1.7056487 0.8871131
```

Así, podemos ver que California no es el estado con la mayor tasa de asesinatos por habitante:

```
murders$state[order(murder_rate, decreasing=T)]
```

```
## [1] "District of Columbia" "Louisiana"      "Missouri"
## [4] "Maryland"           "South Carolina"   "Delaware"
## [7] "Michigan"          "Mississippi"    "Georgia"
## [10] "Arizona"           "Pennsylvania"   "Tennessee"
## [13] "Florida"          "California"    "New Mexico"
## [16] "Texas"             "Arkansas"       "Virginia"
## [19] "Nevada"            "North Carolina" "Oklahoma"
## [22] "Illinois"          "Alabama"        "New Jersey"
## [25] "Connecticut"        "Ohio"           "Alaska"
## [28] "Kentucky"          "New York"        "Kansas"
## [31] "Indiana"           "Massachusetts" "Nebraska"
## [34] "Wisconsin"         "Rhode Island"   "West Virginia"
## [37] "Washington"        "Colorado"       "Montana"
## [40] "Minnesota"         "South Dakota"   "Oregon"
## [43] "Wyoming"           "Maine"          "Utah"
## [46] "Idaho"             "Iowa"           "North Dakota"
## [49] "Hawaii"            "New Hampshire" "Vermont"
```

### 1.3. Indexado, manipulación de datos y gráficos básicos

#### 1.3.1. Indexado

```
library(dslabs)
data("murders")

murder_rate <- murders$total/murders$population*100000
```

**1.3.1.1. Operadores lógicos** Pueden aplicarse operadores lógicos a vectores:

```
index <- murder_rate <= 0.71

index

## [1] FALSE TRUE
## [13] FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [25] FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE TRUE FALSE
## [37] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE
## [49] FALSE FALSE FALSE

murders$state[index]

## [1] "Hawaii"      "Iowa"        "New Hampshire" "North Dakota"
## [5] "Vermont"

sum(index)

## [1] 5
```

Supóngase que se quieren evaluar el cumplimiento de dos condiciones. Primero, hay que definirlas:

```
west <- murders$region == "West"

safe <- murder_rate <= 1
```

Y luego aplicarlas

```
index <- safe & west

murders$state[index]

## [1] "Hawaii"    "Idaho"     "Oregon"    "Utah"      "Wyoming"
```

```
library(dslabs)

data("murders")

murder_rate <- murders$total/murders$population*100000
```

#### 1.3.1.2. Funciones lógicas

1. **which:** arroja las entradas de un vector lógico que son verdad

```
x <- c(FALSE, TRUE, FALSE, TRUE, TRUE, FALSE)

which(x)

## [1] 2 4 5
```

```
index <- which(murders$state == "Massachusetts")

index

## [1] 22
murder_rate[index]

## [1] 1.802179
```

2. **match**: busca entradas en un vector y regresa el índice necesario para acceder a ellas.

```
index <- match(c("New York", "Florida", "Texas"), murders$state)

index

## [1] 33 10 44
murders$state[index]

## [1] "New York" "Florida" "Texas"
murder_rate[index]

## [1] 2.667960 3.398069 3.201360
```

3. **%in%**: se usa para saber si cada elemento de un primer vector está o no dentro de un segundo vector.

```
x <- c("a", "b", "c", "d", "e")

y <- c("a", "d", "f")

y %in% x

## [1] TRUE TRUE FALSE
c("Boston", "Dakota", "Washington") %in% murders$state

## [1] FALSE FALSE TRUE
```

### 1.3.2. Manipulación de datos básica

```
library(dplyr)

library(dslibs)

data("murders")

murder_rate <- murders$total/murders$population*100000
```

**1.3.2.1. dplyr básico** El paquete dplyr es utilizado para manipular datos y utiliza nombres de funciones relativamente básicas:

1. **mutate()**: se usa para agregar o cambiar una columna en un data frame.

Si queremos agregar la columna de la tasa de homicidios:

```
murders <- mutate(murders, rate=total/population*100000)

head(murders)
```

```
##           state abb region population total      rate
## 1      Alabama  AL   South    4779736   135 2.824424
## 2      Alaska  AK    West     710231    19 2.675186
## 3      Arizona  AZ    West    6392017   232 3.629527
## 4      Arkansas AR   South    2915918    93 3.189390
## 5 California CA    West   37253956  1257 3.374138
## 6 Colorado  CO    West   5029196    65 1.292453
murders <- mutate(murders, rank=rank(-rate))
```

2. `filter()`: se usa para filtrar los datos en subconjuntos de filas.

Supóngase que se quieren filtrar los resultados para aquellas tasas de homicidio menores a 0.71:

```
filter(murders, rate <= 0.71)
```

```
##           state abb      region population total      rate rank
## 1      Hawaii  HI        West    1360301    7 0.5145920 49
## 2      Iowa  IA North Central  3046355   21 0.6893484 47
## 3 New Hampshire NH Northeast 1316470    5 0.3798036 50
## 4 North Dakota ND North Central  672591    4 0.5947151 48
## 5 Vermont  VT Northeast  625741    2 0.3196211 51
```

3. `select()`: se usa para filtrar los datos en subconjuntos de columnas.

A veces, los data frames tienen cientos de columnas cuando solo queremos trabajar con algunas de ellas:

```
new_table <- select(murders, state, region, rate)
```

```
filter(new_table, rate <= 0.71)
```

```
##           state      region      rate
## 1      Hawaii        West 0.5145920
## 2      Iowa North Central 0.6893484
## 3 New Hampshire    Northeast 0.3798036
## 4 North Dakota North Central 0.5947151
## 5 Vermont        Northeast 0.3196211
```

Además, el *pipe operator*, `%>%`, se usa para concatenar funciones. Supóngase que se quieren mostrar las tres variables para los estados con tasa de homicidios menor a 0.71 (es decir, lo mismo que se hizo con anterioridad, pero más compacto):

```
murders %>% select(state, region, rate) %>% filter(rate <= 0.71)
```

```
##           state      region      rate
## 1      Hawaii        West 0.5145920
## 2      Iowa North Central 0.6893484
## 3 New Hampshire    Northeast 0.3798036
## 4 North Dakota North Central 0.5947151
## 5 Vermont        Northeast 0.3196211
```

**1.3.2.2. Creando data frames** Para muchos análisis con dplyr, será necesario crear nuevos data frames con la función “`data.frame()`”:

```
grades <- data.frame(names=c("John", "Juan", "Jean", "Yao"),
                      exam_1 = c(95, 80, 90, 85),
                      exam_2 = c(90, 85, 85, 90),
                      stringsAsFactors = FALSE)
```

```
class(grades$names)
## [1] "character"
```

### 1.3.3. Gráficos básicos

```
library(dslabs)
library(dplyr)

data("murders")
```

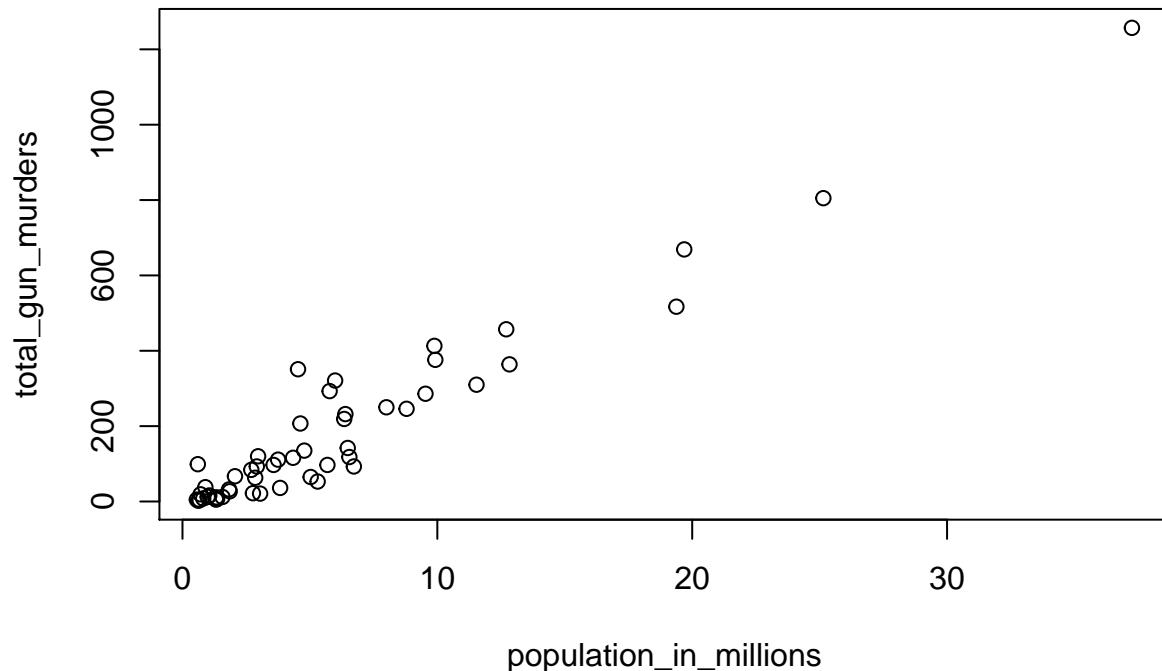
Una gráfica sencilla que relaciona la población en millones y los asesinatos totales es “plot()”:

```
murders <- mutate(murders, rate= murders$total/murders$population*100000)

population_in_millions <- murders$population/10^6

total_gun_murders <- murders$total

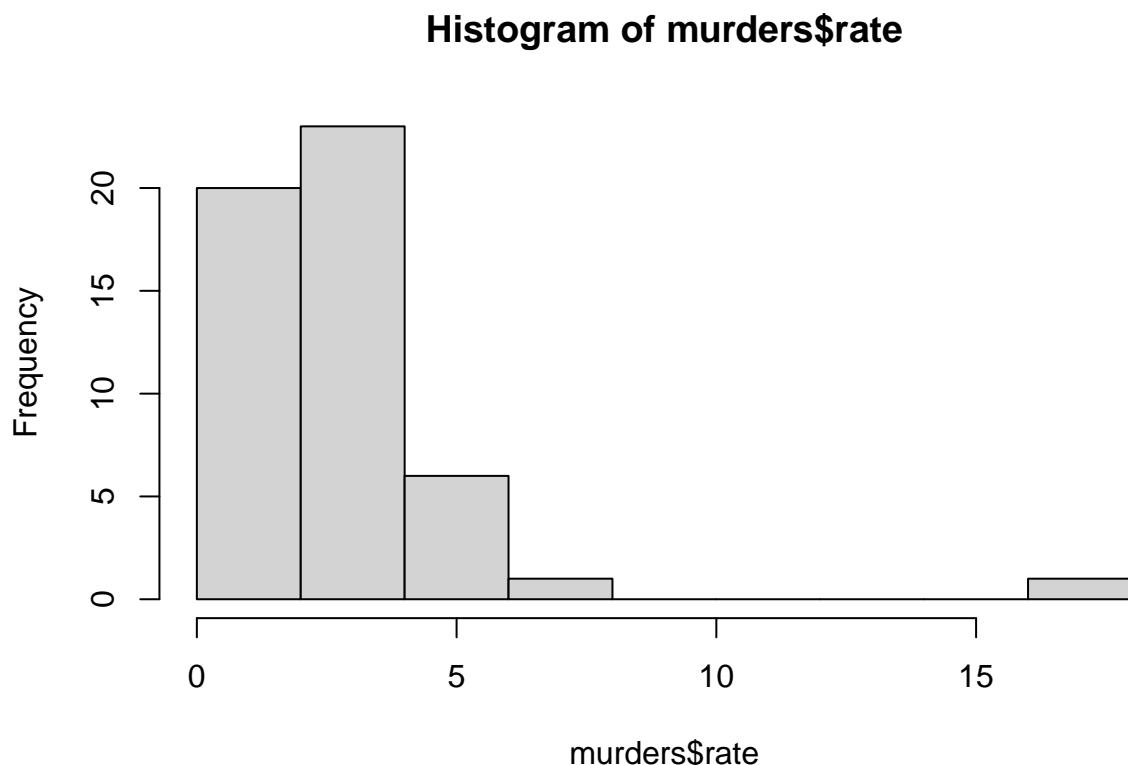
plot(population_in_millions, total_gun_murders)
```



#### 1.3.3.1. Histograma

Para crear un histograma:

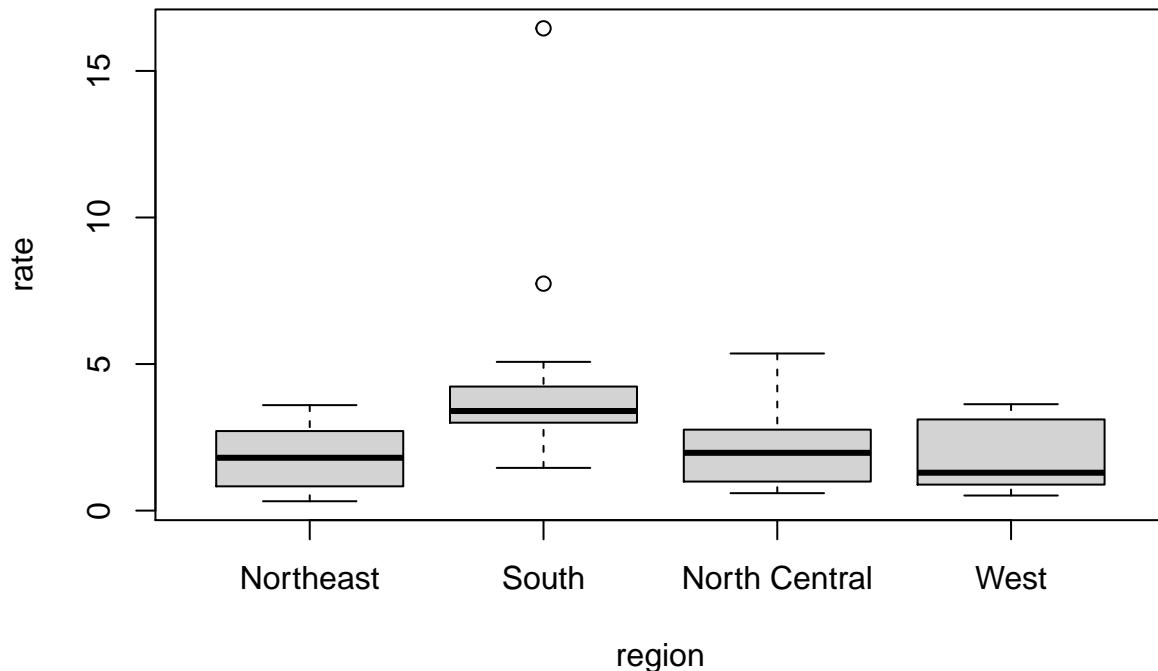
```
hist(murders$rate)
```



#### 1.3.3.2. Boxplot

Para crear un boxplot:

```
boxplot(rate ~ region, data =murders)
```



#### 1.3.4. Assessment 2

```
library(dslabs)
library(dplyr)
data("murders")
data("heights")
options(digits = 3)
```

Pregunta 1.

How many individuals in the dataset are above average height?

```
avg <- mean(heights$height)

ind <- (heights$height) > avg

sum(ind)

## [1] 532
```

Pregunta 2

How many individuals in the dataset are above average height and are female?

```
females <- heights %>% filter(sex == "Female")

ind <- females$height > avg
```

```
sum(ind)
```

```
## [1] 31
```

Pregunta 3.

What proportion of individuals in the dataset are female?

```
mean(heights$sex=="Female")
```

```
## [1] 0.227
```

Pregunta 4.

This question takes you through three steps to determine the sex of the individual with the minimum height.

```
min(heights$height)
```

```
## [1] 50
```

```
index <- match(min(heights$height), heights$height)
```

```
heights$sex[index]
```

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

```
## Levels: Female Male
```

Pregunta 5.

This question takes you through three steps to determine how many of the integer height values between the minimum and maximum heights are not actual heights of individuals in the heights dataset.

```
max(heights$height)
```

```
## [1] 82.7
```

Determine the maximum height.

```
min(heights$height)
```

```
## [1] 50
```

```
max(heights$height)
```

```
## [1] 82.7
```

Which integer values are between the maximum and minimum heights?

```
x <- 50:82
```

How many of the integers in x are NOT heights in the dataset?

```
sum(!x %in% heights$height)
```

```
## [1] 3
```

Pregunta 6.

Using the heights dataset, create a new column of heights in centimeters named ht\_cm. Recall that 1 inch = 2.54 centimeters. Save the resulting dataset as heights2.

```
heights2 <- mutate(heights, ht_cm = heights$height*2.54)
```

What is the height in centimeters of the 18th individual (index 18)?

```
heights2$ht_cm[18]
```

```
## [1] 163
```

What is the mean height in centimeters?

```
mean(heights2$ht_cm)
```

```
## [1] 174
```

Pregunta 7.

Create a data frame females by filtering the heights2 data to contain only female individuals.

```
females <- filter(heights2, sex == "Female")
```

What is the mean height of the females in centimeters?

```
mean(females$ht_cm)
```

```
## [1] 165
```

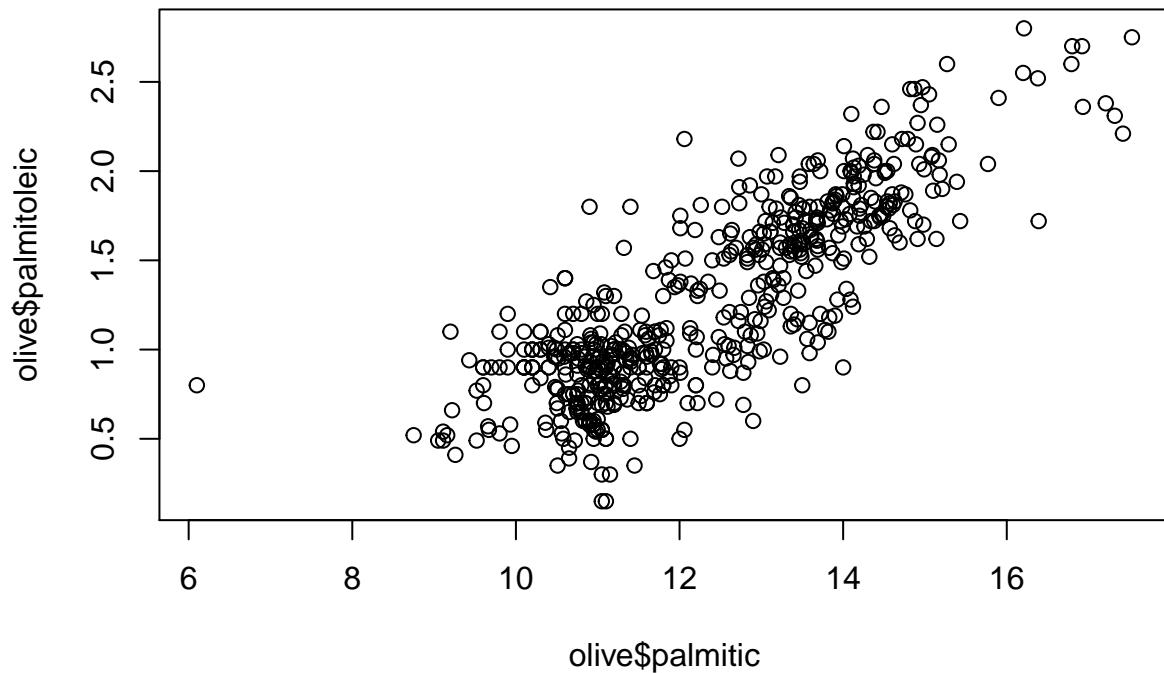
Pregunta 8.

```
data("olive")
head(olive)
```

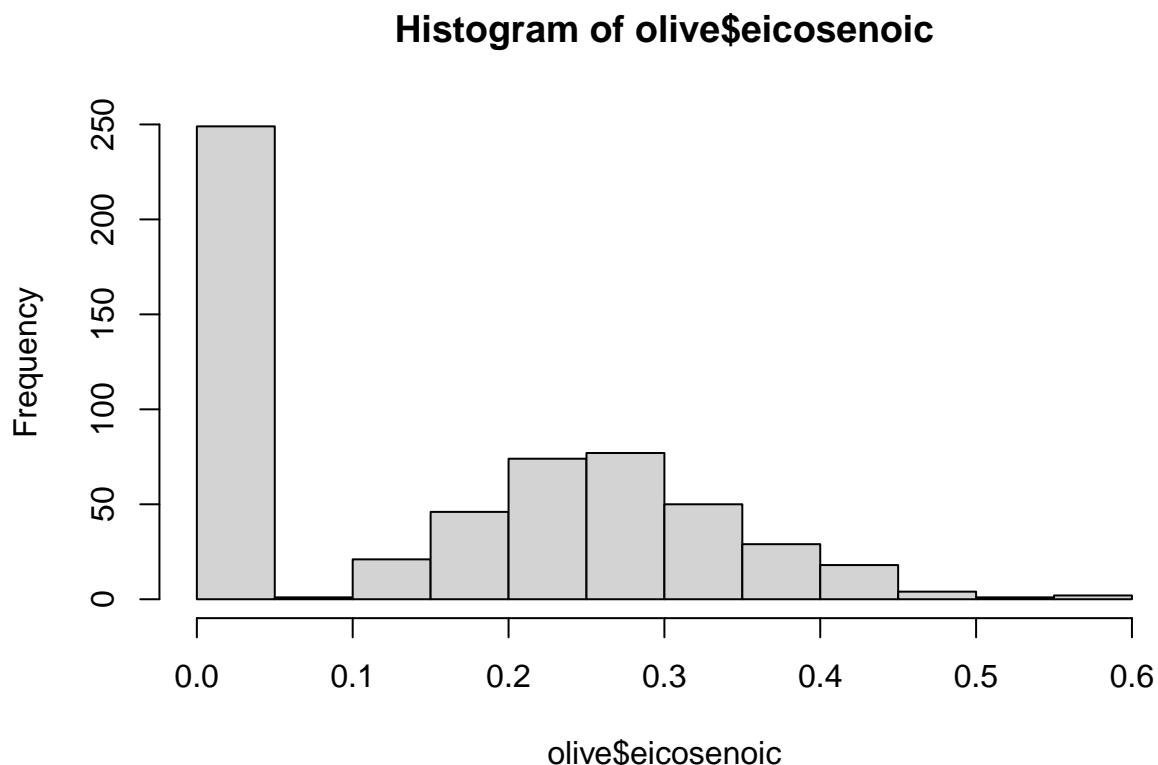
```
##           region      area palmitic palmitoleic stearic oleic linoleic
## 1 Southern Italy North-Apulia    10.75      0.75   2.26  78.2   6.72
## 2 Southern Italy North-Apulia    10.88      0.73   2.24  77.1   7.81
## 3 Southern Italy North-Apulia     9.11      0.54   2.46  81.1   5.49
## 4 Southern Italy North-Apulia     9.66      0.57   2.40  79.5   6.19
## 5 Southern Italy North-Apulia    10.51      0.67   2.59  77.7   6.72
## 6 Southern Italy North-Apulia     9.11      0.49   2.68  79.2   6.78
##   linolenic arachidic eicosenoic
## 1     0.36     0.60     0.29
## 2     0.31     0.61     0.29
## 3     0.31     0.63     0.29
## 4     0.50     0.78     0.35
## 5     0.50     0.80     0.46
## 6     0.51     0.70     0.44
```

Plot the percent palmitic acid versus palmitoleic acid in a scatterplot. What relationship do you see?

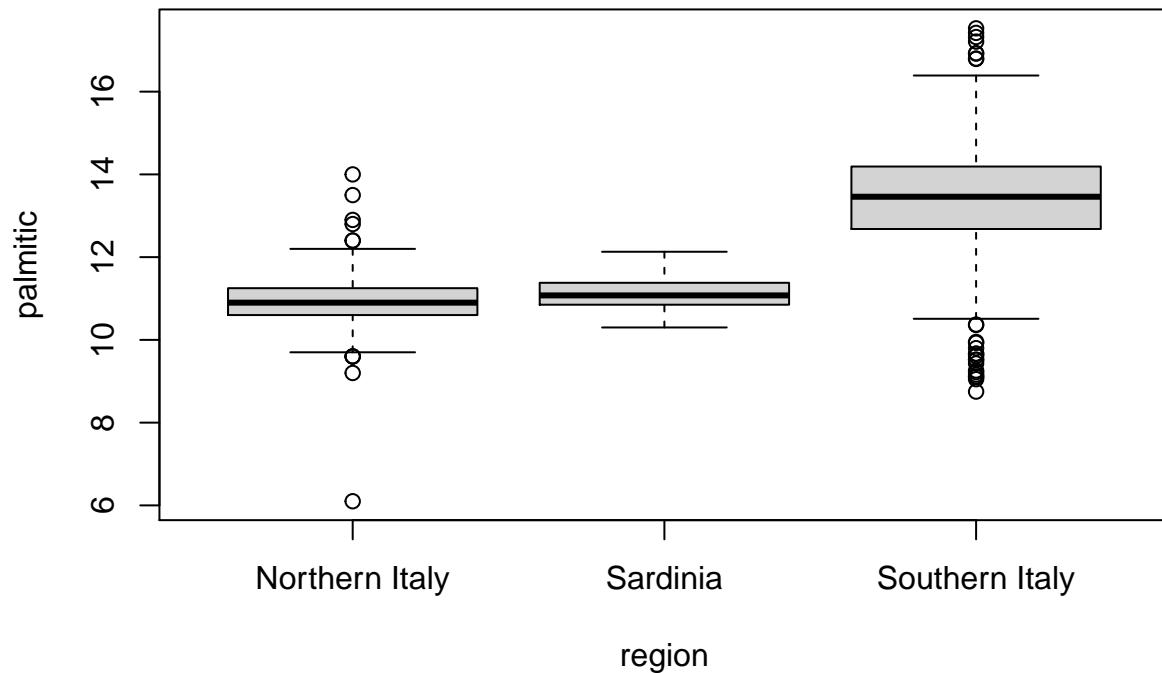
```
plot(olive$palmitic, olive$palmitoleic)
```



```
hist(olive$eicosenoic)
```



```
boxplot(palmitic ~ region, data = olive)
```



## 1.4. Programación básica

### 1.4.1. Condicionales

```
library(dslabs)
data("murders")
murder_rate <- murders$total/murders$population*100000
```

**1.4.1.1. if/else** Las expresiones condicionales más comunes son **if** y **else**.

```
a <- 8

if(a!=0){
  print(1/a)
} else{
  print("No reciprocal for 0.")
}

## [1] 0.125
```

La forma general es:

```
if(condición booleana){
  expresión
} else{
  expresión alternativa
}
```

Otro ejemplo con los datos de murders:

```
ind <- which.min(murder_rate)

if(murder_rate[ind] < 0.5){
  print(murders$state[ind])
} else{
  print("No state has murder rate that low")

## [1] "Vermont"

if(murder_rate[ind] < 0.25){
  print(murders$state[ind])
} else{
  print("No state has murder rate that low")

## [1] "No state has murder rate that low"
```

**1.4.1.2. ifelse** Esta función toma tres argumentos, uno lógico y dos posibles respuestas. Es particularmente útil porque funciona en vectores.

```
a <- 0

ifelse(a > 0, 1/a, NA)

## [1] NA

a <- c(0, 1, 2, -4, 5)
```

```
result <- ifelse( a > 0, 1/a, NA)

result
## [1] NA 1.0 0.5 NA 0.2
```

Esta función suele usarse para reemplazar los NA con otros valores

```
data("na_example")

sum(is.na(na_example))
```

#### 1.4.1.2.1. Ejemplo

```
## [1] 145
no_nas <- ifelse(is.na(na_example), 0, na_example)

sum(is.na(no_nas))
## [1] 0
```

**1.4.1.3. any/all** La función “any()” toma un vector de lógicos y arroja TRUE si cualquiera de las entradas es verdad.

```
z <- c(TRUE, TRUE, FALSE)

any(z)
## [1] TRUE
z <- c(FALSE, FALSE, FALSE)

any(z)
## [1] FALSE
```

La función “all()” toma un vector de lógicos y arroja TRUE si todas las entradas son verdad.

```
z <- c(TRUE, TRUE, FALSE)

all(z)
## [1] FALSE
z <- c(TRUE, TRUE, TRUE)

all(z)
## [1] TRUE
```

**1.4.1.3.1. Ejemplo** Supóngase que queremos que los nombres de las columnas sean, máximo, de 8 caracteres. De lo contrario, se usarán las abreviaciones:

```
# Assign the state abbreviation when the state name is longer than 8 characters
new_names <- ifelse(nchar(murders$state) > 8, murders$abb, murders$state )

new_names
```

```
## [1] "Alabama"  "Alaska"    "Arizona"   "Arkansas"  "CA"        "Colorado"
## [7] "CT"        "Delaware"   "DC"        "Florida"   "Georgia"   "Hawaii"
## [13] "Idaho"     "Illinois"   "Indiana"   "Iowa"      "Kansas"    "Kentucky"
## [19] "LA"        "Maine"      "Maryland"   "MA"        "Michigan"  "MN"
## [25] "MS"        "Missouri"   "Montana"   "Nebraska"  "Nevada"   "NH"
## [31] "NJ"        "NM"        "New York"  "NC"        "ND"        "Ohio"
## [37] "Oklahoma"  "Oregon"    "PA"        "RI"        "SC"        "SD"
## [43] "TN"        "Texas"     "Utah"      "Vermont"   "Virginia"  "WA"
## [49] "WV"        "WI"        "Wyoming"
```

### 1.4.2. Funciones

R tiene algunas funciones ya instaladas, sin embargo, puede que haya necesidad de definir otras de acuerdo a nuestras necesidades:

```
avg <- function(x){
  s <- sum(x)
  n <- length(x)
  s/n
}

x <- 1:100

avg(x)

## [1] 50.5
identical(mean(x), avg(x))

## [1] TRUE
```

Nótese que las funciones que definimos no se guardan en el panel de trabajo. **La forma general es:**

```
ny_function <- function(x){
  operaciones que operen en x, la cual está definida por el usuario al llamar esta función
}
```

Si la función tiene más de un argumento:

```
ny_function <- function(x, y, z){
  operaciones que operen en x, y, z, la cual está definida por el usuario al llamar esta función
}

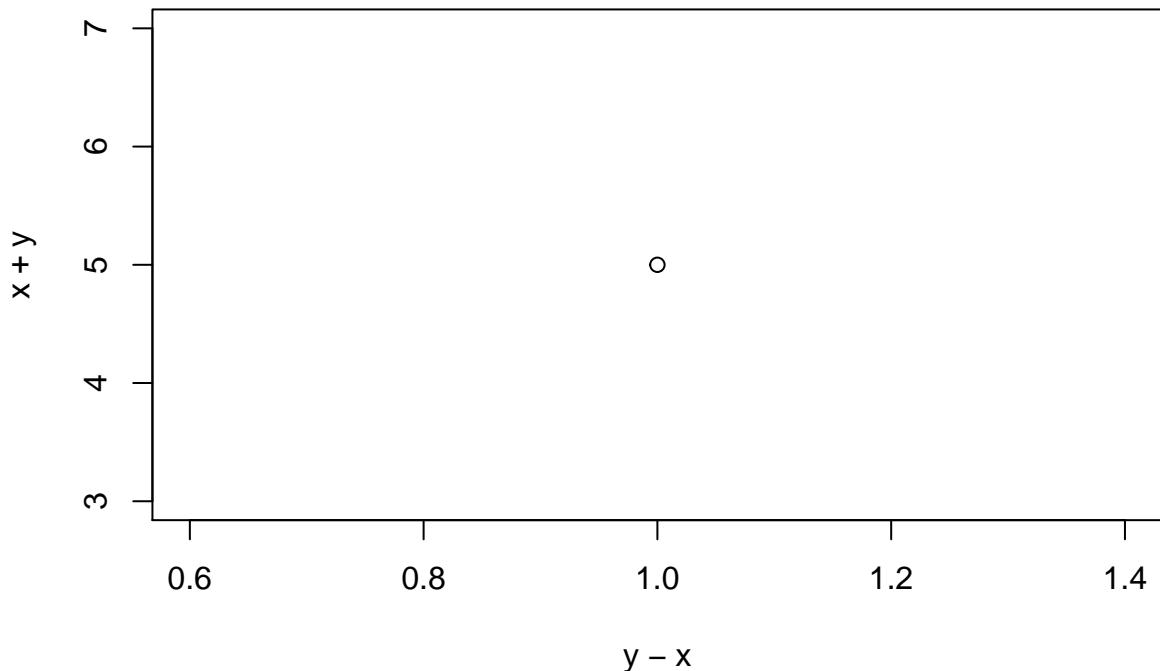
avg <- function(x, arithmetic = TRUE){
  n <- length(x)
  ifelse(arithmetic, sum(x)/n, prod(x)^(1/n))
}
```

#### 1.4.2.1. Ejemplo

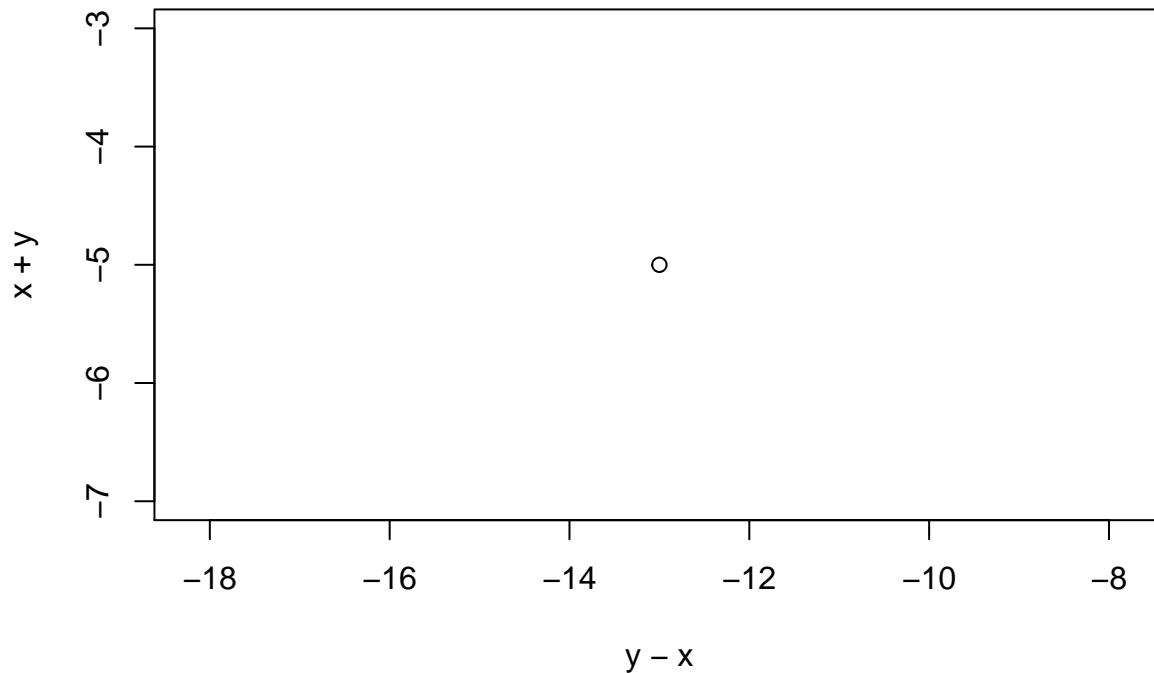
Pueden crearse funciones para graficar:

```
# Create `altman_plot`
altman_plot <- function(x,y){
  plot(y-x, x+y)
}

altman_plot(2,3)
```



```
altman_plot(4, -9)
```



### 1.4.3. Bucles

Considérese la serie matemática:

$$1 + 2 + \dots + n = \frac{n(n + 1)}{2}$$

```
compute_s_n <- function(n){
  x <- 1:n
  sum(x)
}
```

```
compute_s_n(3)
```

```
## [1] 6
compute_s_n(100)
```

```
## [1] 5050
compute_s_n(2021)
```

```
## [1] 2043231
```

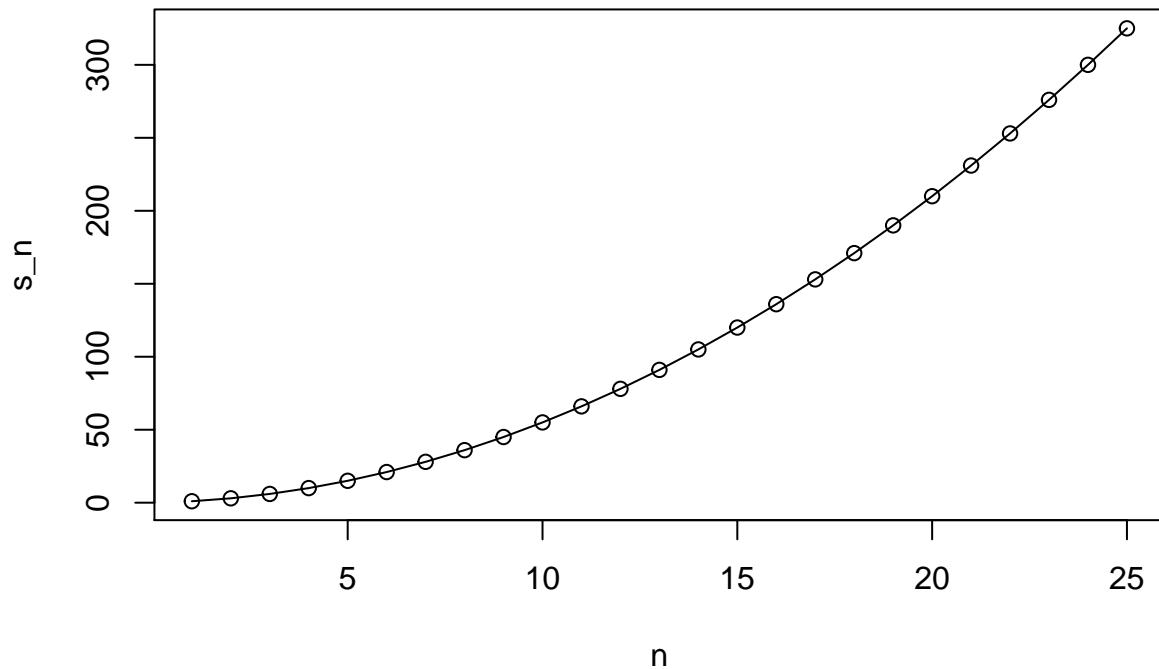
**La forma general de un bucle es:**

```
for(i en algún rango de valores){
  operaciones que usan i, la cual cambia a través del rango de valores
}
```

```
for(i in 1:5){  
  print(i)  
}  
  
## [1] 1  
## [1] 2  
## [1] 3  
## [1] 4  
## [1] 5
```

Supóngase que queremos computar esta suma para  $n = 1, \dots, 25$ . En vez de calcular 25 veces el resultado, utilizamos un bucle:

```
m <- 25  
  
# Creamos un vector vacío para almacenar los resultados que vayamos obteniendo  
  
s_n <- vector(length = m)  
  
for(n in 1:m){  
  s_n[n] <- compute_s_n(n)  
}  
  
s_n  
  
## [1] 1 3 6 10 15 21 28 36 45 55 66 78 91 105 120 136 153 171 190  
## [20] 210 231 253 276 300 325  
n <- 1:m  
  
plot(n, s_n)  
lines(n, n*(n+1)/2)
```



#### 1.4.3.1. Ejemplo

Supóngase que queremos calcular la suma:

$$S_n = 1^2 + 2^2 + 3^2 + \dots + n^2$$

```
# Write a function compute_s_n with argument n that for any given n computes the sum of 1 + 2^2 + ... + n^2

compute_s_n <- function(n){
  x <- 1:n
  sum(x^2)
}

# Report the value of the sum when n=10

compute_s_n(10)

## [1] 385
```

Ahora, se quiere calcular esta suma para cualquier  $n = 1, \dots, 25$ :

```
# Create a vector for storing results
s_n <- vector("numeric", 25)

# write a for-loop to store the results in s_n

m <- 25
for(n in 1:m){
```

```

x <- 1:n
s_n[n] <- compute_s_n(n)
}

s_n

## [1] 1 5 14 30 55 91 140 204 285 385 506 650 819 1015 1240
## [16] 1496 1785 2109 2470 2870 3311 3795 4324 4900 5525

```

#### 1.4.4. Otras funciones

A veces, la utilización de los bucles es reemplazada por otras funciones más poderosas, como:

1. **apply()**
2. **sapply()**
3. **tapply()**
4. **mapply()**
5. **split()**
6. **cut()**
7. **quantile()**
8. **reduce()**
9. **identical()**
10. **unique()**

#### 1.4.5. Assessment 3

```

library(dslabs)
library(dplyr)
data("heights")

```

##### Pregunta 1.

Write an **ifelse()** statement that returns 1 if the sex is Female and 2 if the sex is Male. What is the sum of the resulting vector

```
sex <- ifelse(heights$sex == "Female", 1, 2)
```

```
sum(sex)
```

```
## [1] 1862
```

##### Pregunta 2.

Write an **ifelse()** statement that takes the height column and returns the height if it is greater than 72 inches and returns 0 otherwise. What is the mean of the resulting vector?

```
new_height <- ifelse(heights$height > 72, heights$height, 0)
```

```
mean(new_height)
```

```
## [1] 9.65
```

**Pregunta 3.**

Write a function `inches_to_ft` that takes a number of inches `x` and returns the number of feet. One foot equals 12 inches. What is `inches_to_ft(144)`?

```
feet <- function(n){  
  n/12  
}  
  
feet(144)  
  
## [1] 12  
  
heights <- mutate(heights, ft = feet(height))  
  
sum(heights$ft < 5)  
  
## [1] 20
```

**Pregunta 4.**

Given an integer `x`, the factorial of `x` is called `x!` and is the product of all integers up to and including `x`. The `factorial()` function computes factorials in R. For example, `factorial(4)` returns  $4! = 4 \times 3 \times 2 \times 1 = 24$ .

```
# define a vector of length m  
m <- 10  
f_n <- vector(length = m)  
  
# make a vector of factorials  
for(n in 1:m){  
  f_n[n] <- factorial(n)  
}  
  
# inspect f_n  
f_n  
  
##   [1]      1      2      6     24    120    720   5040  40320 362880  
## [10] 3628800
```

## 2. Visualización en R

### 2.1. Introducción a la visualización de datos y distribuciones

#### 2.1.1. Introducción

```
library(dslabs)
data("murders")

head(murders)

##      state abb region population total
## 1    Alabama  AL   South     4779736   135
## 2     Alaska  AK    West      710231    19
## 3   Arizona  AZ    West     6392017   232
## 4  Arkansas  AR   South     2915918    93
## 5 California  CA    West     37253956  1257
## 6 Colorado  CO    West     5029196    65
```

La visualización de datos puede ser sumamente útil para facilitar el análisis y comparación de información y datos. Se trata de la herramienta más importante del **análisis exploratorio de datos**.

#### 2.1.2. Distribuciones

Normalmente, el promedio y la desviación estándar son valores relevantes para el análisis de datos. Para resumir la información estadística de un conjunto de datos, se utiliza su distribución.

#### 2.1.3. Tipos de datos

Antes de decidir cómo se van a analizar los datos, es importante conocer qué tipos de datos son. Pueden existir dos tipos de variables:

1. Categóricas;
  - I. Ordinales
  - II. No ordinales
2. Numéricas
  - I. Discretas
  - II. Continuas

```
data("heights")
```

Normalmente, las variables categóricas se utilizan con números discretos de un número reducido de grupos diferentes. En el caso de los datos sobre altura de personas, se puede corroborar el número de datos diferentes que hay, para así decidir si la variable puede ser tratada como categórica o numérica continua.

```
unique(heights$height)
```

```
## [1] 75.0 70.0 68.0 74.0 61.0 65.0 66.0 62.0 67.0 72.0 69.0 64.0 60.0 73.0 71.0
## [16] 66.8 63.0 70.5 65.0 64.2 68.5 78.7 69.6 66.5 71.5 76.0 74.5 78.0 62.5 77.0
## [31] 68.9 64.2 59.0 67.7 71.7 70.9 68.1 64.6 51.0 59.1 80.0 65.0 68.9 69.7 72.0
## [46] 72.5 70.1 53.0 64.2 59.1 66.9 70.9 74.8 68.4 69.3 66.5 68.9 61.8 72.0 67.7
## [61] 72.4 79.0 63.8 66.4 69.3 66.9 66.1 74.8 72.4 53.8 65.7 67.7 54.0 68.1 65.0
## [76] 67.7 66.1 70.8 77.2 69.3 67.5 72.8 66.7 68.1 68.5 67.8 67.7 79.0 68.8 70.1
## [91] 73.2 73.6 68.9 66.9 66.9 70.8 72.4 61.3 66.9 58.0 55.0 73.2 66.1 63.0 70.1
## [106] 67.2 72.5 76.0 75.6 70.9 62.4 75.6 71.7 62.6 67.3 64.2 66.1 64.5 70.9 75.4
## [121] 72.8 71.7 64.6 72.8 78.7 64.9 59.8 70.9 62.2 68.1 70.5 52.0 81.0 68.9 62.6
```

```
## [136] 82.7 50.0 73.2 63.4
length(unique(heights$height))

## [1] 139
```

#### 2.1.4. Introducción a las distribuciones

```
head(heights)
```

##### 2.1.4.1. CDF

```
##      sex height
## 1   Male    75
## 2   Male    70
## 3   Male    68
## 4   Male    74
## 5   Male    61
## 6 Female   65
```

Recordemos que la distribución es el resumen estadístico más básico de una lista de objetos o números.

Para obtener la proporción de cada variable categórica (sexo, en este caso):

```
prop.table(table(heights$sex))
```

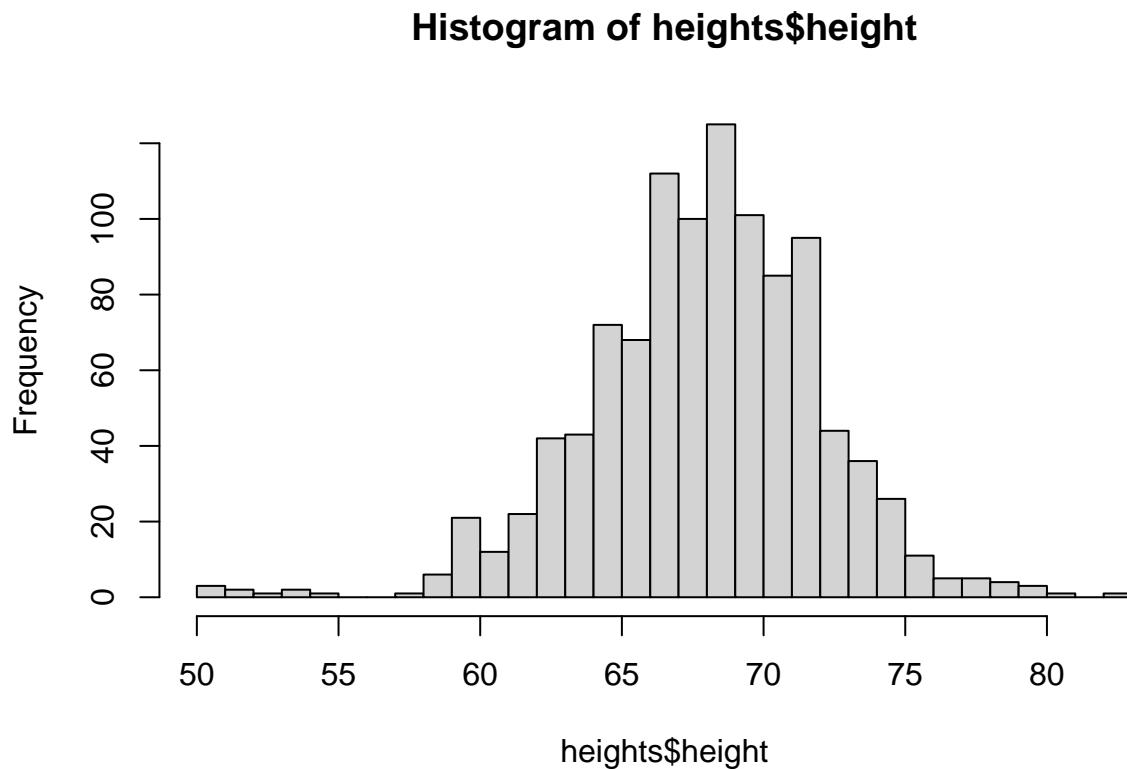
```
##
##      Female     Male
## 0.227 0.773
prop.table(table(murders$region))

##
##      Northeast      South      North      Central      West
## 0.176 0.333 0.235 0.255
```

Cuando los datos no son categóricos, no es útil reportar la frecuencia de los datos, dado que es posible que todos ellos sean únicos. Para este tipo de datos, se utilizará la **Función de Distribución Acumulada (CDF)**:

$$F(a) = P[x \leq a]$$

```
hist(heights$height, breaks = 30)
```



### 2.1.5. Cuantiles, percentiles y boxplots

**2.1.5.1. Cuantiles** Un **cuantil** es una partición que divide a un conjunto de datos en intervalos con probabilidades determinadas. El  $i$ ésimo cuantil es el valor en el cual  $i\%$  de las observaciones son iguales o menores que ese valor. La función es “quantile()”:

```
quantile(murders$total, 0.5)
```

```
## 50%
## 97
```

Por su parte, un **percentil** divide al conjunto de datos en 100 intervalos, cada uno con 1% de probabilidad. Pueden determinarse todos los percentiles de un conjunto de datos así:

```
p <- seq(0.01, 0.99, 0.01)
```

```
quantile(murders$total, p)
```

	1%	2%	3%	4%	5%	6%	7%	8%	9%	10%	11%
##	3.0	4.0	4.5	5.0	5.0	5.0	6.0	7.0	7.5	8.0	9.5
##	12%	13%	14%	15%	16%	17%	18%	19%	20%	21%	22%
##	11.0	11.5	12.0	12.0	12.0	14.0	16.0	17.5	19.0	20.0	21.0
##	23%	24%	25%	26%	27%	28%	29%	30%	31%	32%	33%
##	21.5	22.0	24.5	27.0	29.5	32.0	34.0	36.0	37.0	38.0	45.5
##	34%	35%	36%	37%	38%	39%	40%	41%	42%	43%	44%
##	53.0	58.0	63.0	64.0	65.0	66.0	67.0	75.5	84.0	88.5	93.0
##	45%	46%	47%	48%	49%	50%	51%	52%	53%	54%	55%
##	93.0	93.0	95.0	97.0	97.0	97.0	98.0	99.0	105.0	111.0	113.5

```
##   56%   57%   58%   59%   60%   61%   62%   63%   64%   65%   66%
## 116.0 117.0 118.0 119.0 120.0 127.5 135.0 138.5 142.0 174.5 207.0
## 67%   68%   69%   70%   71%   72%   73%   74%   75%   76%   77%
## 213.0 219.0 225.5 232.0 239.0 246.0 248.0 250.0 268.0 286.0 289.5
## 78%   79%   80%   81%   82%   83%   84%   85%   86%   87%   88%
## 293.0 301.5 310.0 315.5 321.0 336.0 351.0 357.5 364.0 370.0 376.0
## 89%   90%   91%   92%   93%   94%   95%   96%   97%   98%   99%
## 394.5 413.0 435.0 457.0 487.0 517.0 593.0 669.0 737.0 805.0 1031.0
```

Adicionalmente, un **cuartil** divide a los datos en 4 partes, cada una con 25 % de probabilidad. Esto es, son iguales a los percentiles 25to, 50mo y 75to, conocidos como el primer cuartil, la mediana, y el 3er cuartil.

```
summary(murders$total)
```

```
##   Min. 1st Qu. Median   Mean 3rd Qu.   Max.
##     2      24     97    184    268   1257
```

```
p <- seq(0.01, 0.99, 0.01)
```

```
percentiles <- quantile(heights$height, p)
```

```
summary(heights$height)
```

```
##   Min. 1st Qu. Median   Mean 3rd Qu.   Max.
##   50.0   66.0   68.5    68.3    71.0    82.7
```

```
percentiles
```

```
##   1%   2%   3%   4%   5%   6%   7%   8%   9%   10%  11%  12%  13%  14%  15%  16%
## 59.0 60.0 60.0 61.0 62.0 62.0 62.3 63.0 63.0 63.0 63.8 64.0 64.0 64.0 64.2 65.0
## 17% 18% 19% 20% 21% 22% 23% 24% 25% 26% 27% 28% 29% 30% 31% 32%
## 65.0 65.0 65.0 65.0 65.0 65.9 66.0 66.0 66.0 66.0 66.0 66.0 66.0 66.5 66.9 66.9 67.0
## 33% 34% 35% 36% 37% 38% 39% 40% 41% 42% 43% 44% 45% 46% 47% 48%
## 67.0 67.0 67.0 67.0 67.0 67.0 67.3 67.7 67.7 68.0 68.0 68.0 68.0 68.0 68.0 68.0
## 49% 50% 51% 52% 53% 54% 55% 56% 57% 58% 59% 60% 61% 62% 63% 64%
## 68.1 68.5 68.9 68.9 69.0 69.0 69.0 69.0 69.0 69.0 69.0 69.0 69.0 69.6 70.0 70.0 70.0
## 65% 66% 67% 68% 69% 70% 71% 72% 73% 74% 75% 76% 77% 78% 79% 80%
## 70.0 70.0 70.0 70.0 70.0 70.1 70.7 70.9 71.0 71.0 71.0 71.0 71.0 71.1 72.0 72.0 72.0
## 81% 82% 83% 84% 85% 86% 87% 88% 89% 90% 91% 92% 93% 94% 95% 96%
## 72.0 72.0 72.0 72.0 72.0 72.0 72.4 72.8 73.0 73.0 74.0 74.0 74.0 74.8 75.0
## 97% 98% 99%
## 75.0 76.0 78.0
```

```
percentiles[names(percentiles)=="25%"]
```

```
## 25%
## 66
```

```
percentiles[names(percentiles)=="75%"]
```

```
## 75%
## 71
```

**2.1.5.2. qnorm** La función “qnorm()” arroja el valor teórico de un cuantil con una probabilidad  $p$  de observar un valor igual o menor que el valor dado  $y$  suponiendo una distribución normal con media  $\mu$  y desviación estándar  $\sigma$  (por default,  $\mu = 0$  y  $\sigma = 1$ ).

Así, “qnorm()” y “pnorm()” son funciones inversas:

$$pnorm(-1,96) \approx 0,025$$

$$qnorm(0,025) \approx -1,96$$

$$\Rightarrow pnorm(qnorm(0,025)) = 0,025$$

Por tanto, “qnorm()” se usa para determinar los cuantiles teóricos de un conjunto de datos: esto es, el valor teórico de los cuantiles asumiendo que los datos se distribuyen normalmente.

```
index <- heights$sex=="Male"
x <- heights$height[index]
mean(x <= 69.5)
```

### 2.1.5.3. Gráficas cuantil-cuantil

```
## [1] 0.515
```

Esto significa que si  $p = 0,5$ , el valor asociado  $q$  es  $q = 69,5$ . Puede hacerse este cálculo para una serie de valores  $p$  para los valores observados:

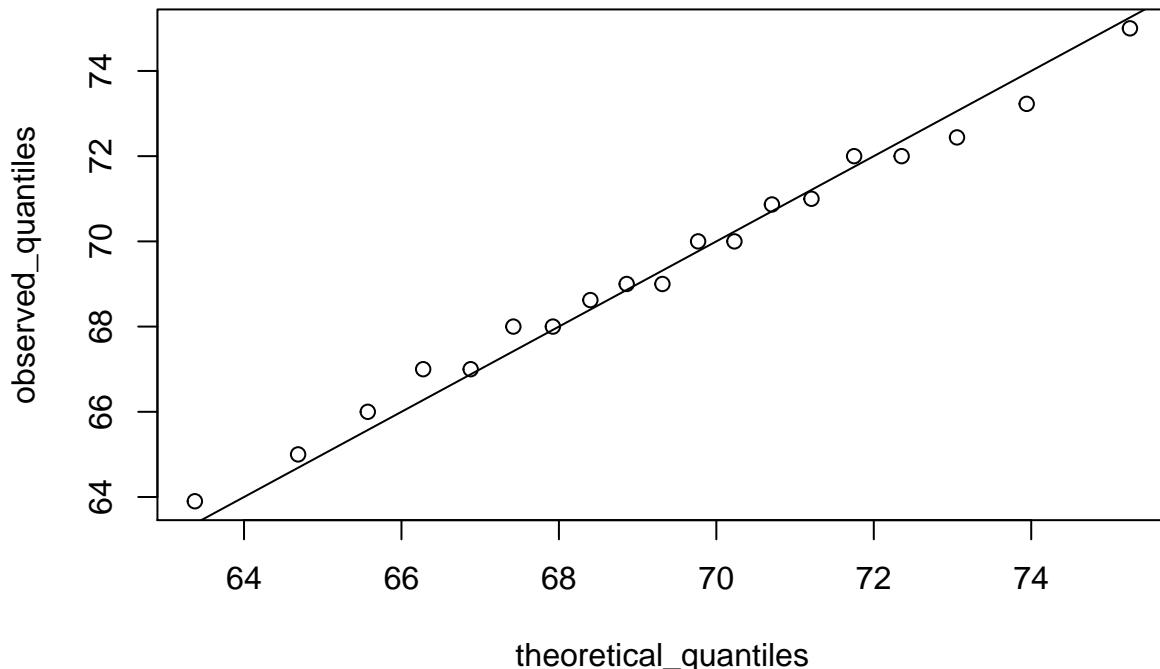
```
p <- seq(0.05,0.95,0.05)
observed_quantiles <- quantile(x,p)
observed_quantiles
##   5% 10% 15% 20% 25% 30% 35% 40% 45% 50% 55% 60% 65% 70% 75% 80%
## 63.9 65.0 66.0 67.0 67.0 68.0 68.0 68.6 69.0 69.0 70.0 70.0 70.9 71.0 72.0 72.0
## 85% 90% 95%
## 72.4 73.2 75.0
```

Y compararlo con los cuantiles teóricos de una distribución normal:

```
theoretical_quantiles <- qnorm(p, mean = mean(x), sd=sd(x))
theoretical_quantiles
## [1] 63.4 64.7 65.6 66.3 66.9 67.4 67.9 68.4 68.9 69.3 69.8 70.2 70.7 71.2 71.8
## [16] 72.4 73.1 73.9 75.3
```

Finalmente, para compararlos, pueden graficarse juntos:

```
plot(theoretical_quantiles, observed_quantiles)
abline(0,1)
```



Lo que indica que la distribución normal es bastante buena para explicar el comportamiento de los datos observados.

#### 2.1.5.4. Normalizando

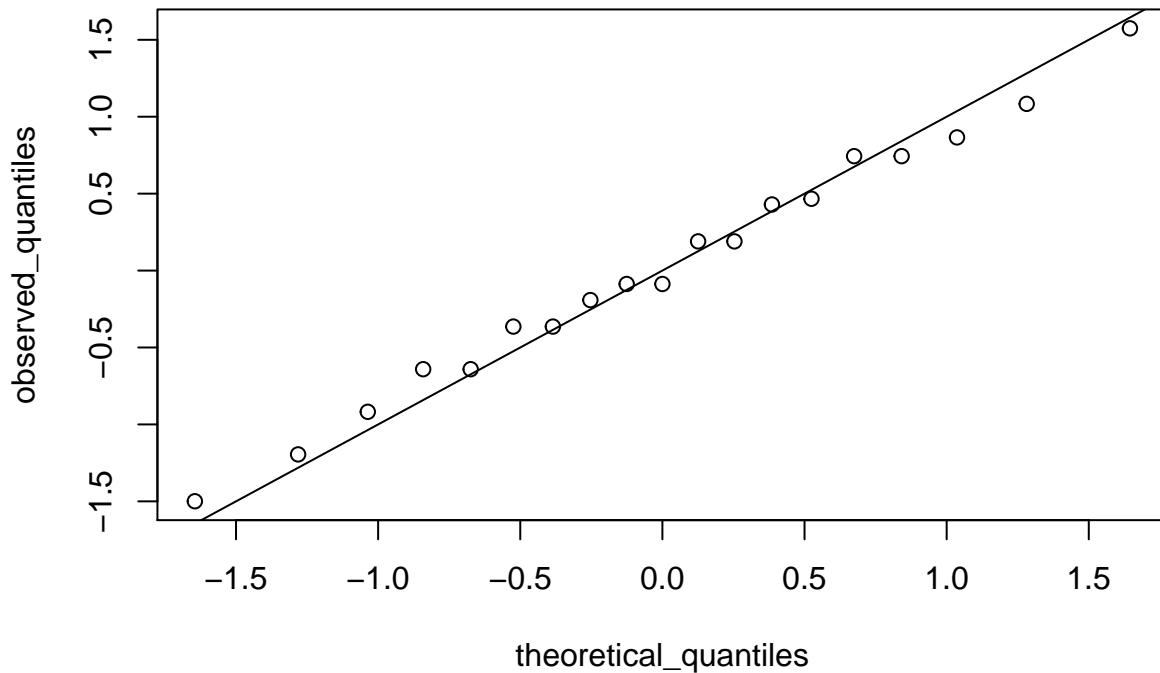
Pueden normalizarse los valores y realizar el mismo análisis que se hizo:

```
z <- scale(x)

observed_quantiles <- quantile(z,p)

theoretical_quantiles <- qnorm(p)

plot(theoretical_quantiles, observed_quantiles)
abline(0,1)
```



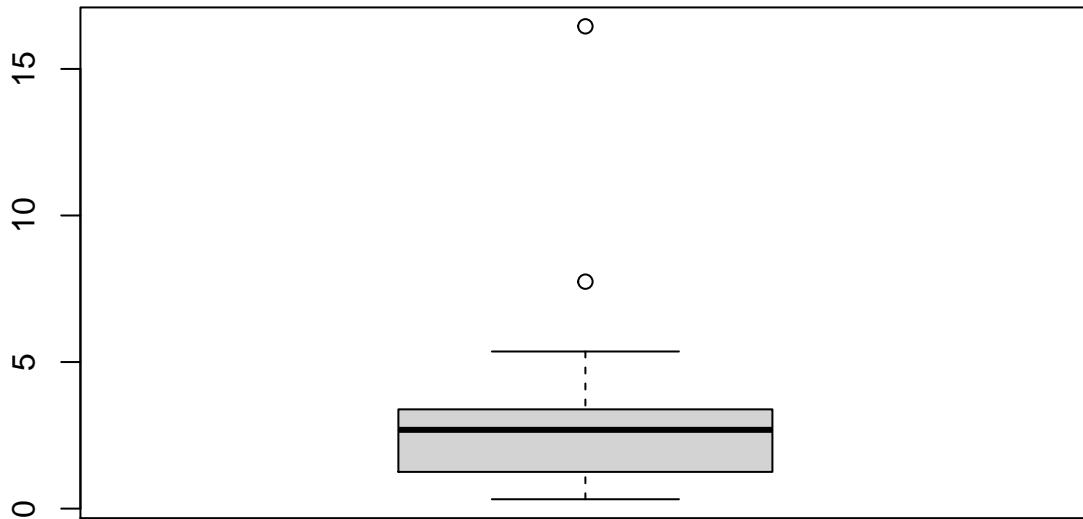
**2.1.5.5. Boxplots** Supóngase que se quiere resumir la distribución de la tasa de homicidios:

```
murder_rate <- murders$total/murders$population*100000
```

**Recomendación:** utilizar un resumen con las siguientes cantidades (y graficando los *outliers* como puntos aparte):

1. Rango
2. Cuartil 25
3. Cuartil 50
4. Cuartil 75

```
boxplot(murder_rate)
```



Donde la caja está delimitada por los cuartiles 25 y 75 (rango intercuartil), la línea negra gruesa es la mediana, las líneas punteadas definen el rango que toman los valores y los puntos son *outliers*.

Entonces, la gráfica indica que:

- La mediana es alrededor de 2.5
- La distribución no es simétrica
- El rango es (0,5) para la mayoría de los estados, exceptuando 2

### 2.1.6. Análisis de datos exploratorio (EDA)

```
library(dslabs)
library(HistData)

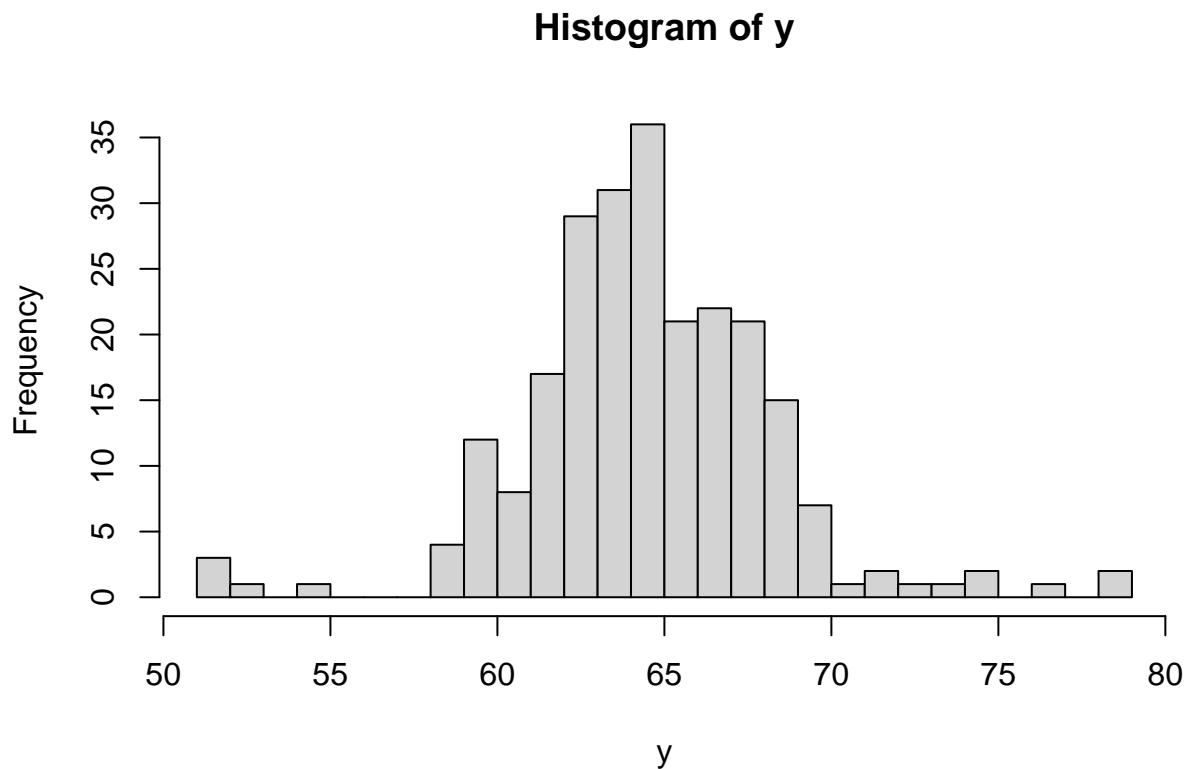
data("heights")
data("Galton")
```

**2.1.6.1. Ejemplo** Se esperaría que la distribución de la altura femenina fuera similar a la masculina. Sin embargo, vemos que no es el caso:

```
index <- heights$sex=="Female"

y <- heights$height[index]

hist(y, breaks =30)
```



```
p <- seq(0.05,0.95,0.05)

observed_quantiles <- quantile(y,p)

observed_quantiles

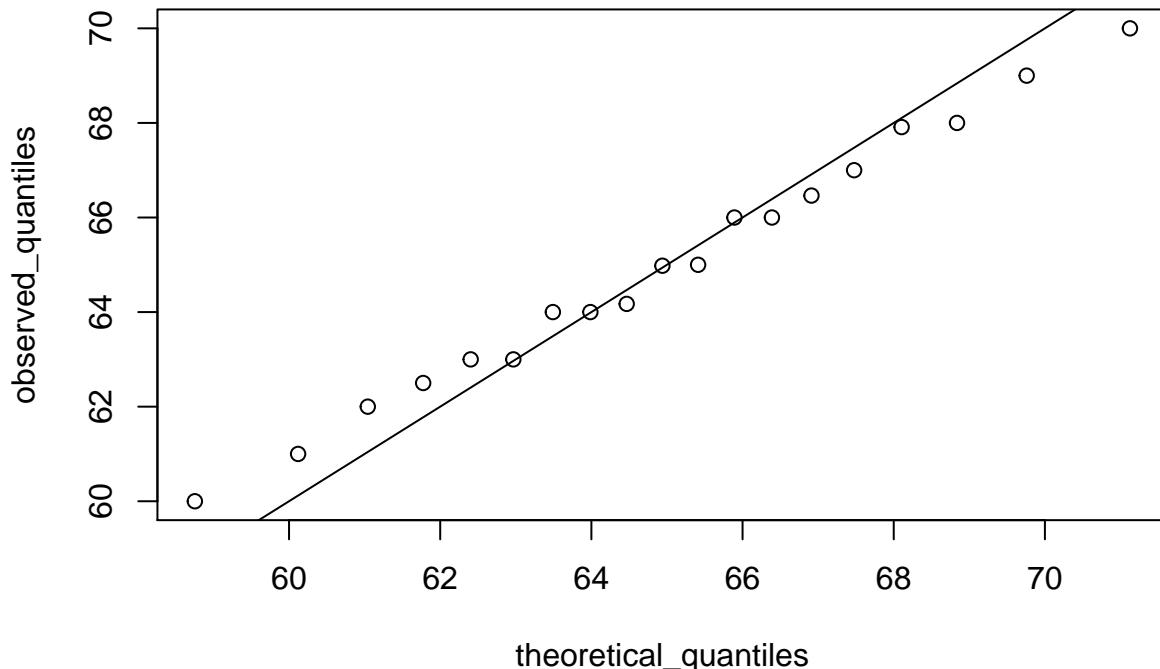
##   5% 10% 15% 20% 25% 30% 35% 40% 45% 50% 55% 60% 65% 70% 75% 80%
## 60.0 61.0 62.0 62.5 63.0 63.0 64.0 64.0 64.2 65.0 65.0 66.0 66.0 66.5 67.0 67.9
## 85% 90% 95%
## 68.0 69.0 70.0

theoretical_quantiles <- qnorm(p, mean = mean(y), sd=sd(y))

theoretical_quantiles

## [1] 58.8 60.1 61.0 61.8 62.4 63.0 63.5 64.0 64.5 64.9 65.4 65.9 66.4 66.9 67.5
## [16] 68.1 68.8 69.8 71.1

plot(theoretical_quantiles, observed_quantiles)
abline(0,1)
```



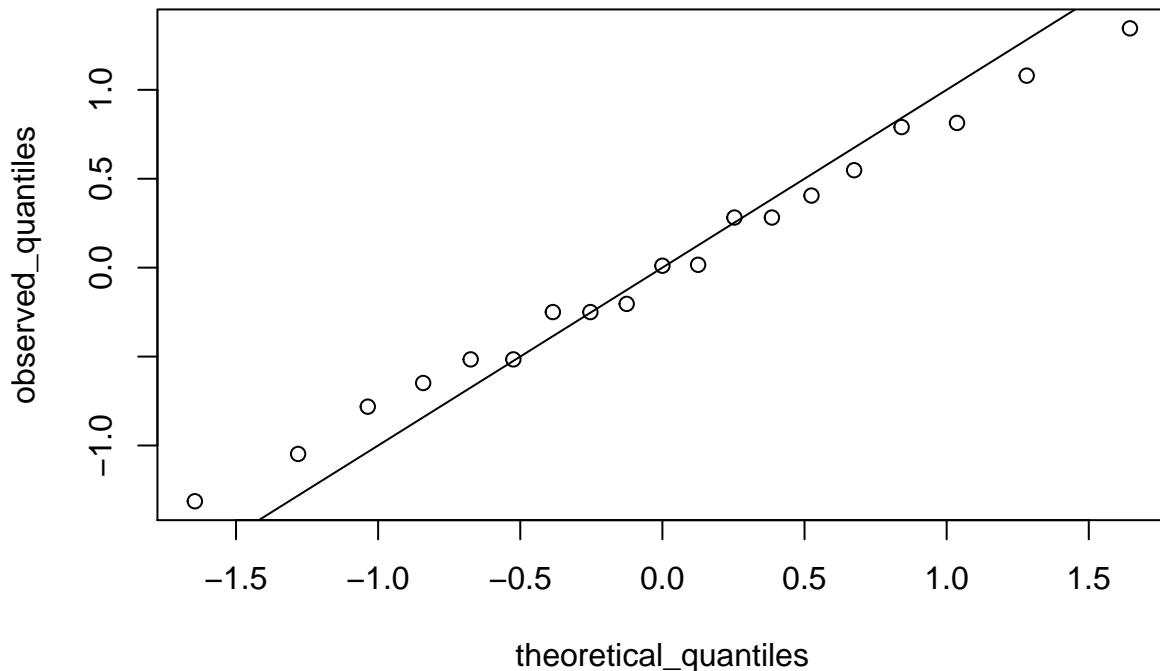
Estandarizando los datos:

```
z <- scale(y)

observed_quantiles <- quantile(z,p)

theoretical_quantiles <- qnorm(p)

plot(theoretical_quantiles, observed_quantiles)
abline(0,1)
```



**2.1.6.2. Impacto de outliers** Para determinar el impacto un *outlier* que pueda estar afectando a los datos, puede escribirse una función simple que tome el valor de éste como input y arroje el promedio:

```
x <- Galton$child
```

```
error_avg <- function(k){  
  x[1] <- k  
  mean(x)  
}
```

```
error_avg(10000)
```

```
## [1] 78.8
```

```
error_avg(-10000)
```

```
## [1] 57.2
```

## 2.2. Introducción a ggplot2

### 2.2.1. Introducción

```
library(pacman)  
  
pacman::p_load(tidyverse, ggplot, grid, lattice, dslabs, dplyr)  
  
data(murders)  
data("heights")
```

Otros paquetes para crear gráficos son **grid** y **lattice**.

Una limitación de ggplot es que está exclusivamente diseñado para trabajar con tablas de datos (renglones como observaciones y columnas como variables).

### 2.2.2. Componentes del gráfico

Los tres componentes básicos de ggplot son:

1. Los datos (data)
2. El tipo de gráfico (geometry)
3. Mapeado estético (mapping -valores de los ejes, texto de puntos o colores categóricos-)

Otros componentes son:

4. Rango y escala de los ejes (scale)
5. Personalización (labels, title, legend, etc.)
6. Temas (theme, style)

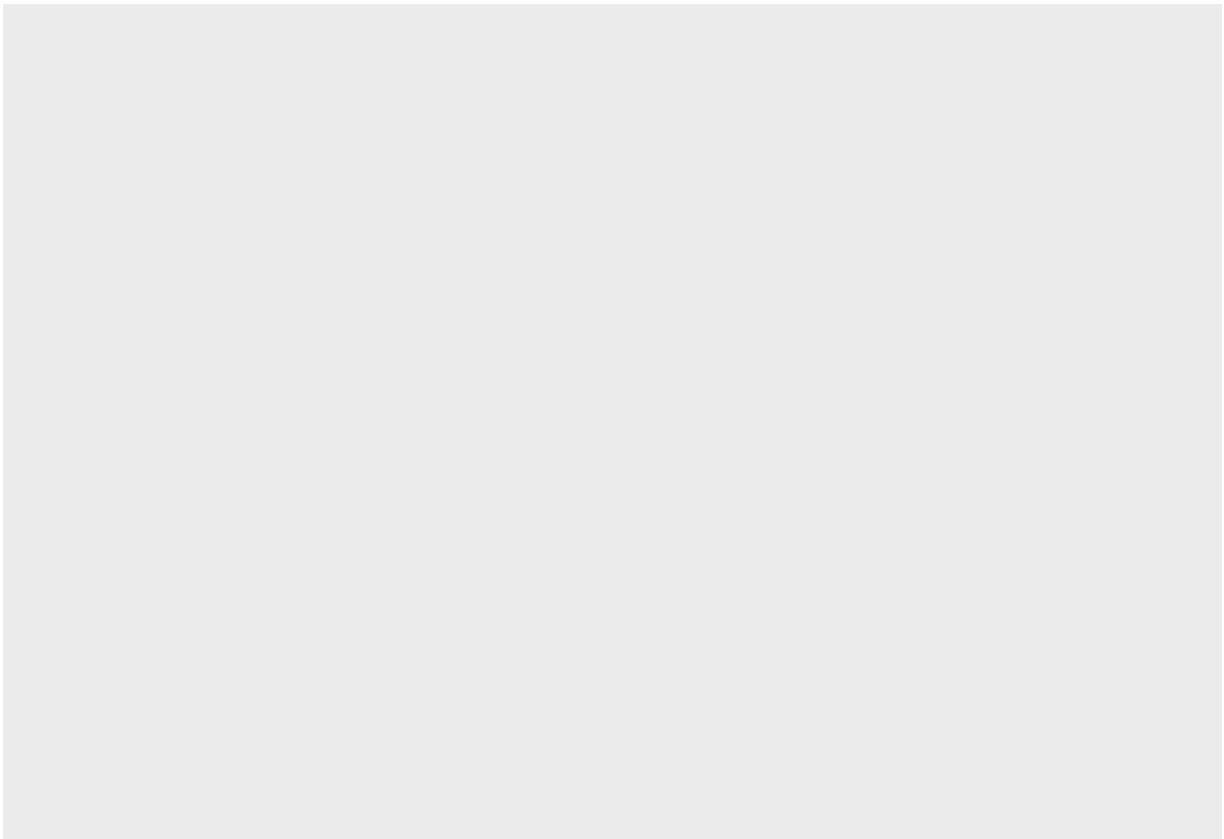
### 2.2.3. Pasos para crear un gráfico ggplot

1. Definición del objeto

**Nota:** ggplot(data = x,...) es equivalente a x %>% ggplot()

```
p <- ggplot(data = murders)
```

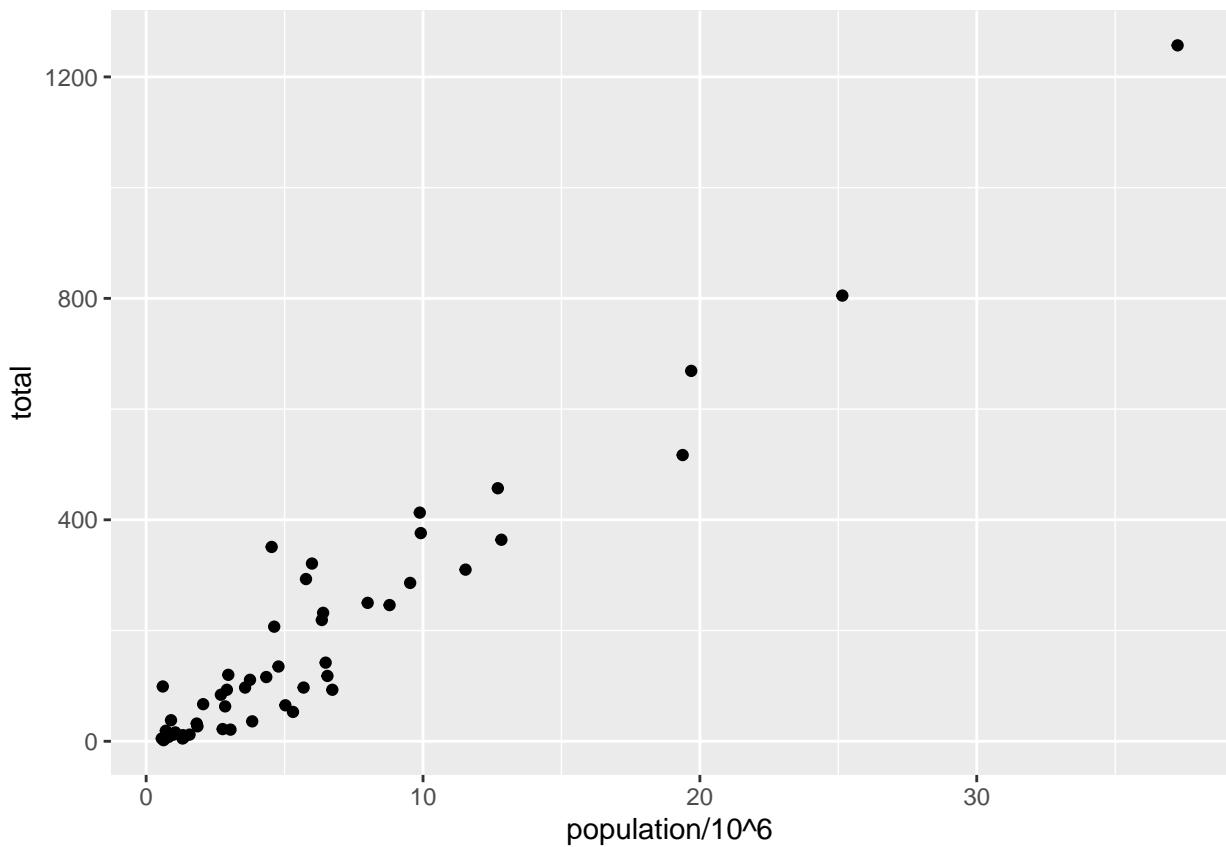
```
p
```



2. Definir el tipo de gráfico

- I. El componente **geometry** requiere dos argumentos (x,y)
- II. Estos se colocan mediante el subcomando “aes()”

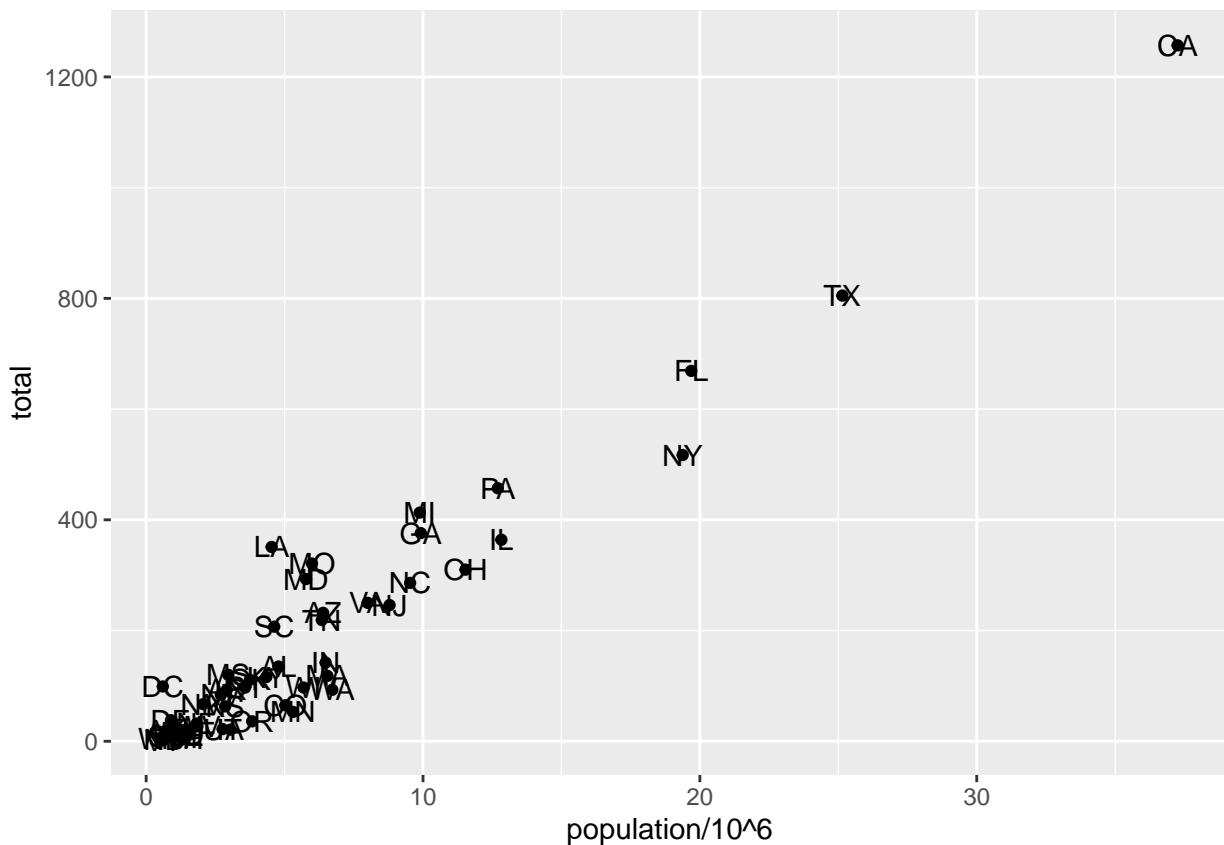
```
murders %>% ggplot() +  
  geom_point(aes(x=population/10^6, y =total))
```



### 3. Etiquetas en puntos

Se utiliza el componente “geom\_label()” (rectángulo pequeño) y “geom\_text()” (solo texto)

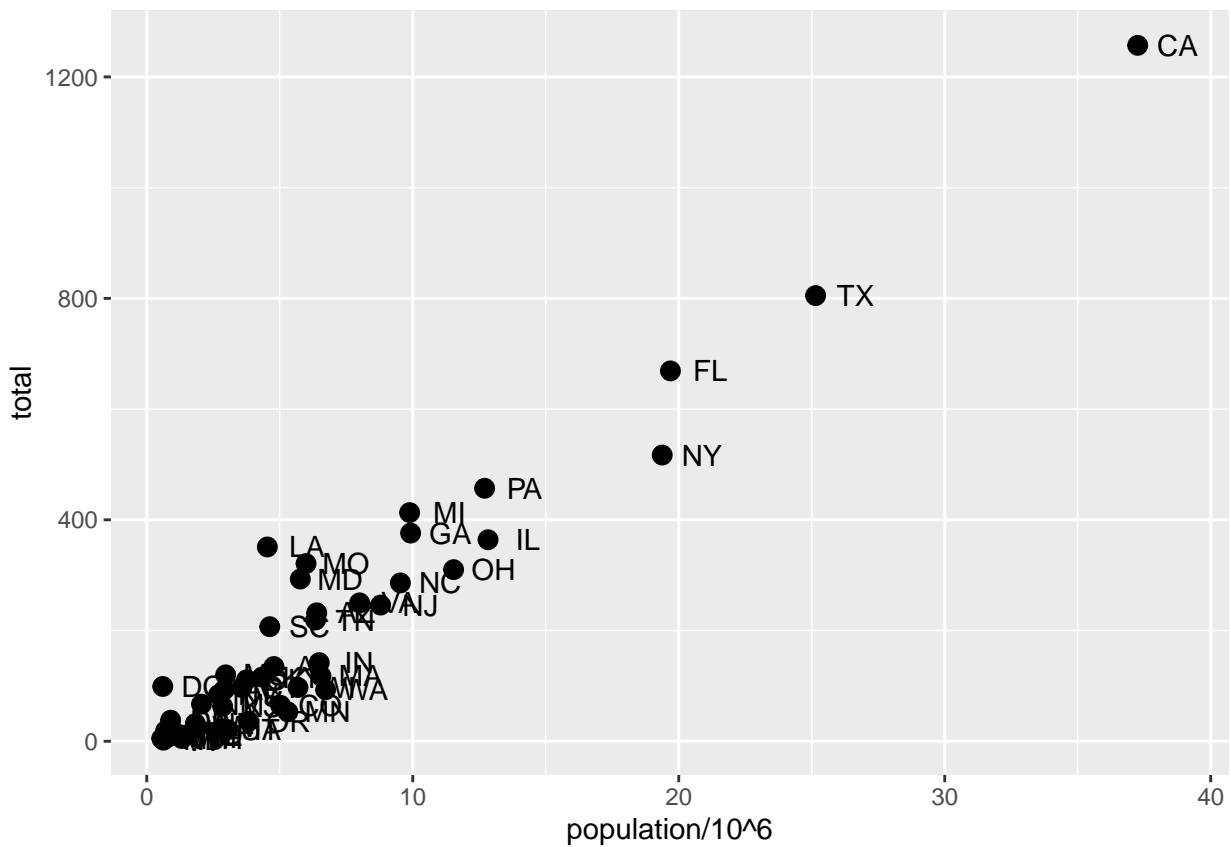
```
murders %>% ggplot() +  
  geom_point(aes(x=population/10^6, y =total)) +  
  geom_text(aes(x=population/10^6, y=total, label =abb))
```



#### 4. Optimizando el gráfico

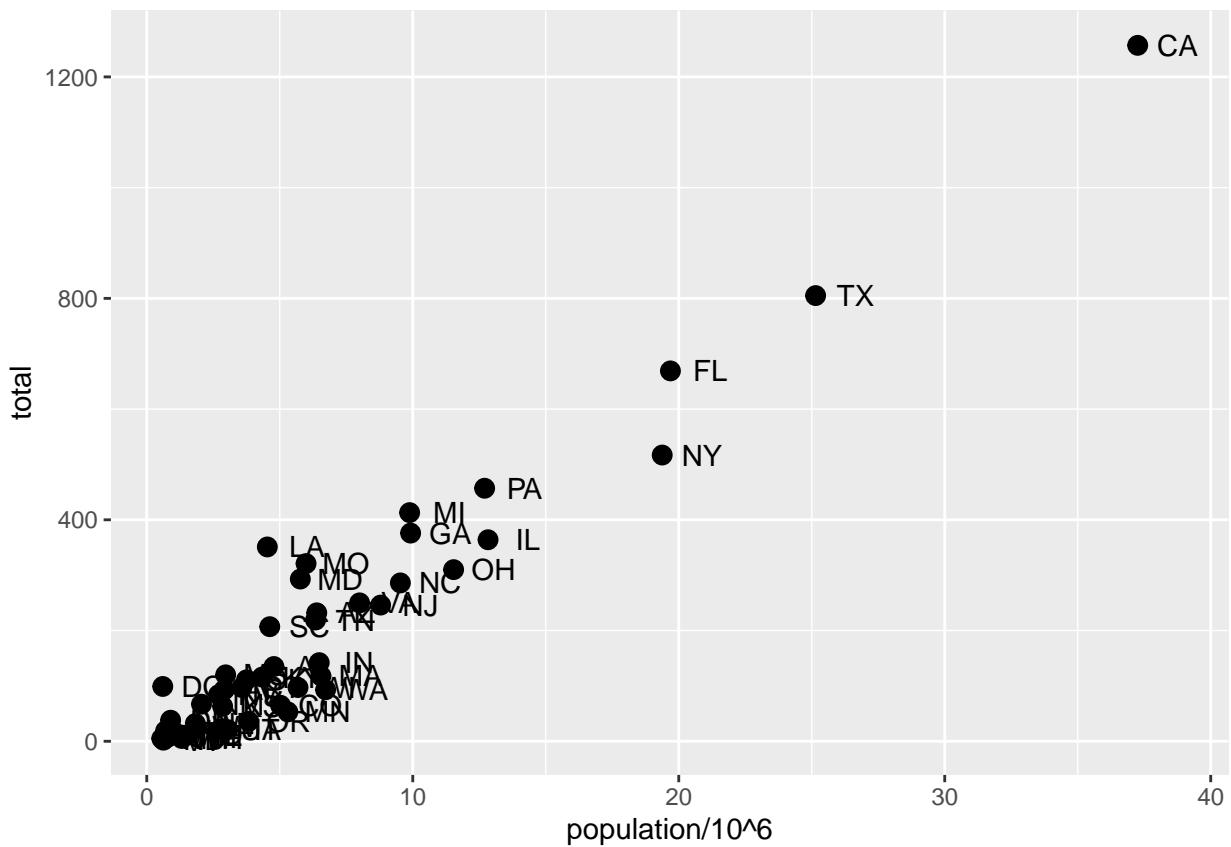
size modifica el tamaño de los puntos dentro de geom\_point y nudge\_x mueve las etiquetas horizontalmente alrededor de los puntos dentro de geom\_text.

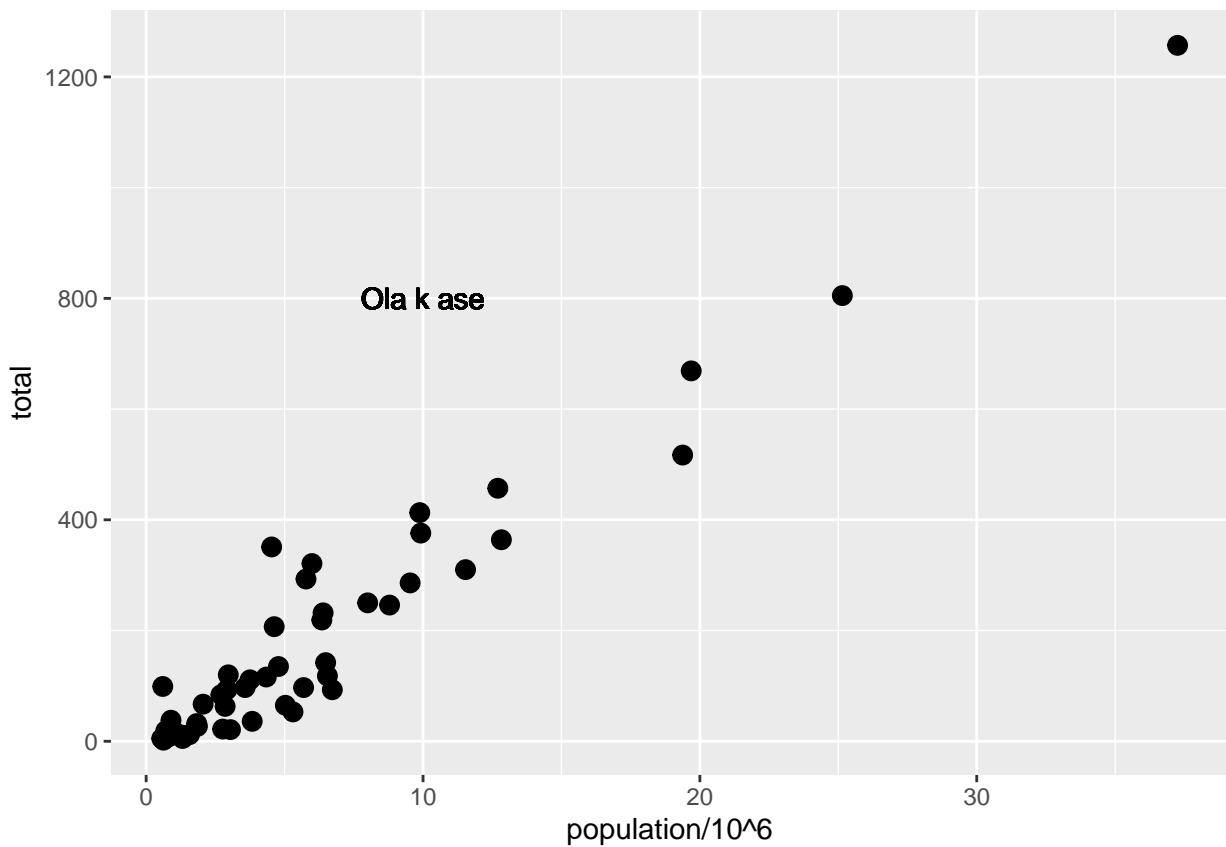
```
murders %>% ggplot() +  
  geom_point(aes(x=population/10^6, y = total), size =3) +  
  geom_text(aes(x=population/10^6, y = total, label = abb), nudge_x = 1.5)
```



**Nota:** Puede definirse el mapping desde el principio, con el objetivo de mantenerlo como predeterminado para las demás capas.

```
murders %>% ggplot(aes(x=population/10^6, y = total, label = abb)) +  
  geom_point(size=3) +  
  geom_text(nudge_x = 1.5)
```

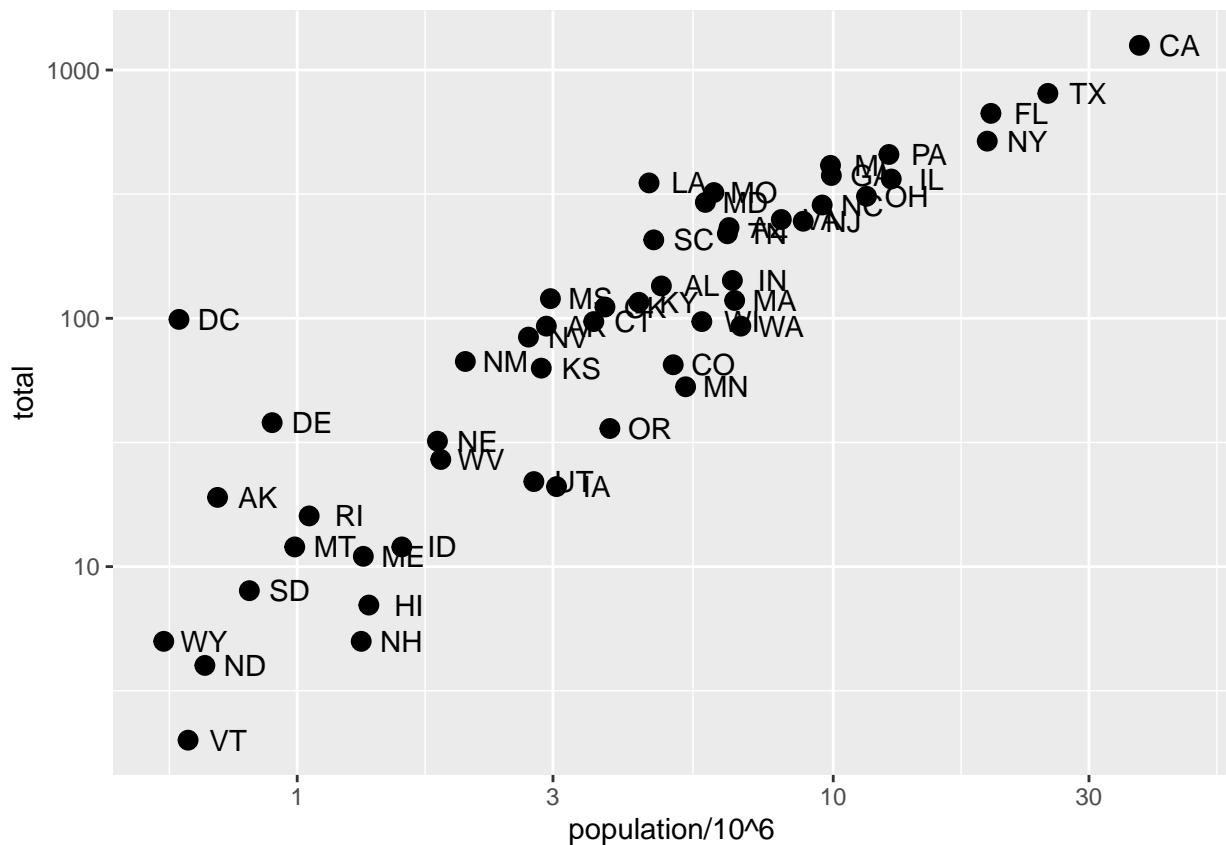




##### 5. Colores, escalas y etiquetas

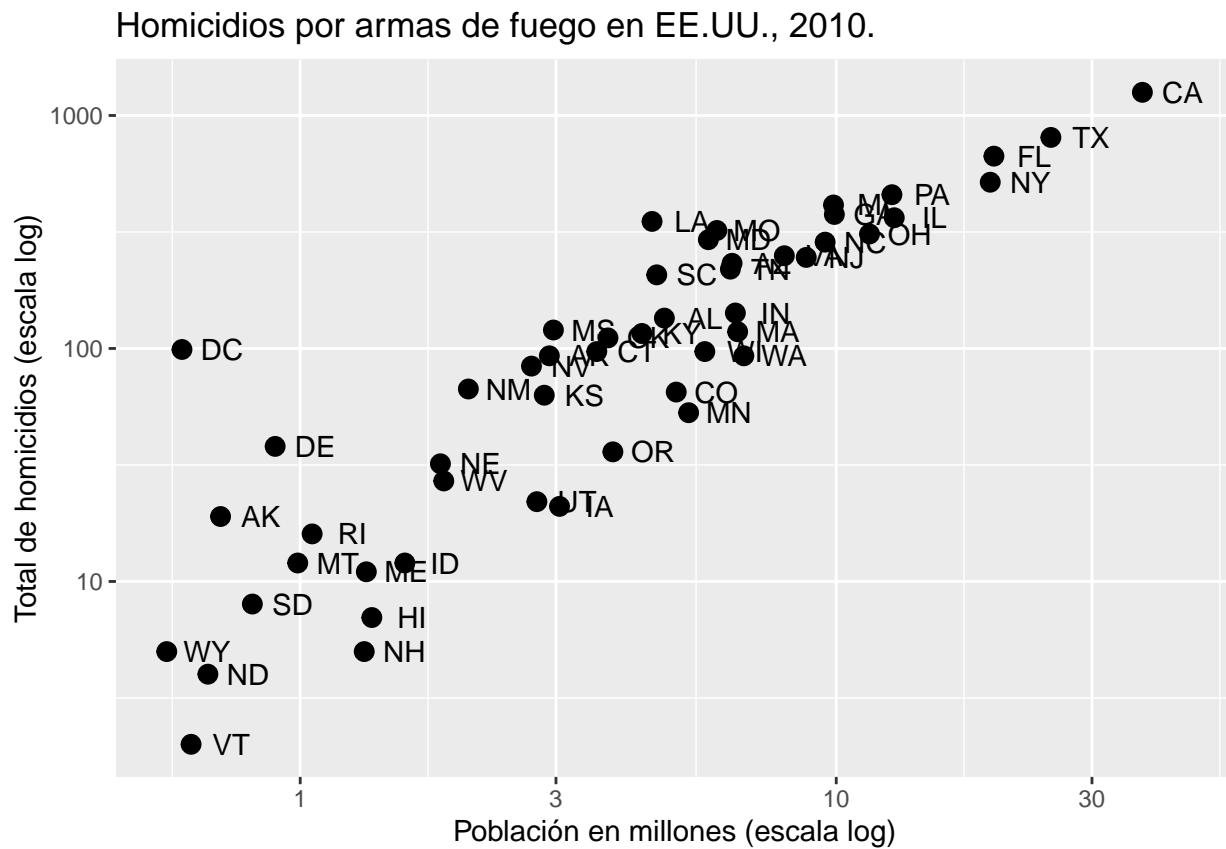
Se puede modificar la escala a logarítmica con “scale\_x\_log10()”

```
murders %>% ggplot(aes(x = population/10^6, y = total, label = abb)) +  
  geom_point(size = 3) +  
  geom_text(nudge_x = 0.075) +  
  scale_x_log10() +  
  scale_y_log10()
```



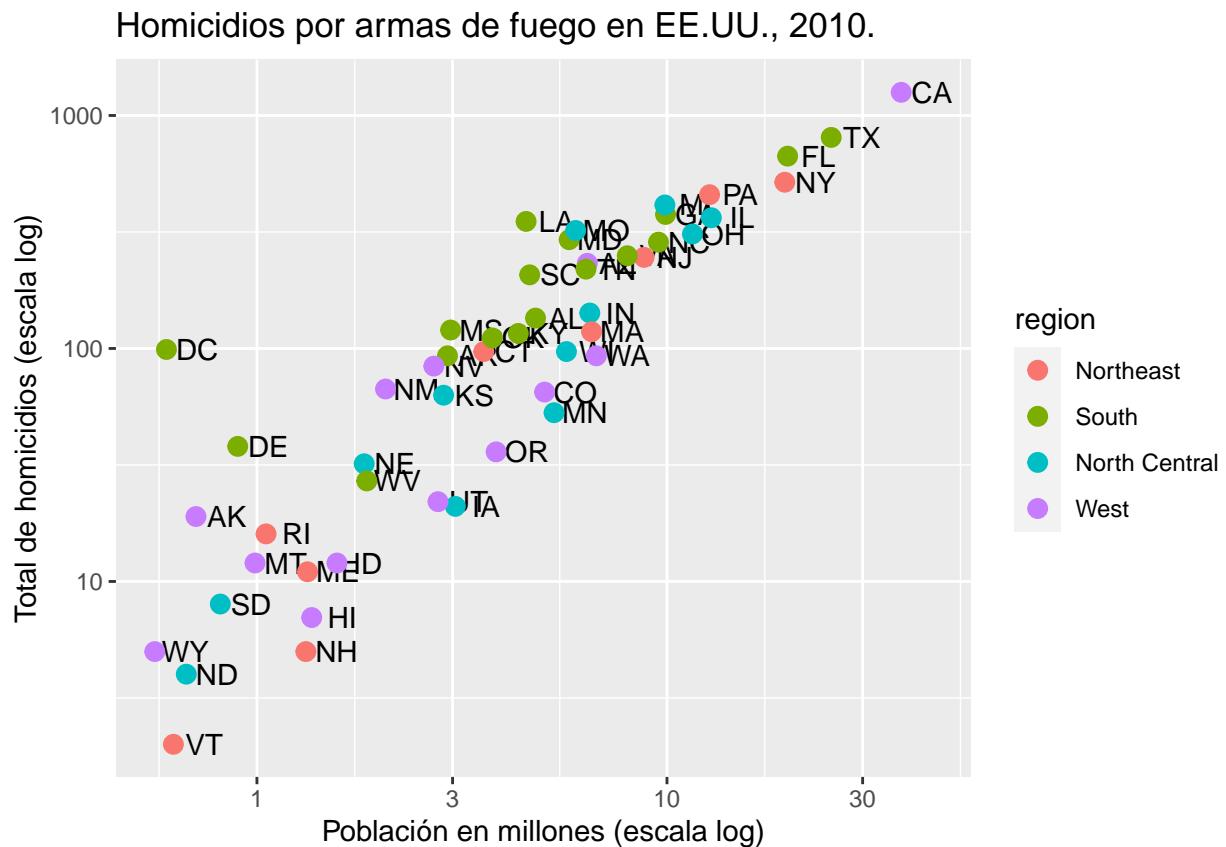
Para agregar etiquetas y títulos se utiliza “xlab()”, “ylab()” y “ggtitle()”

```
murders %>% ggplot(aes(x = population/10^6, y = total, label = abb))+
  geom_point(size = 3) +
  geom_text(nudge_x = 0.075) +
  scale_x_log10() +
  scale_y_log10() +
  xlab("Población en millones (escala log)") +
  ylab("Total de homicidios (escala log)") +
  ggtitle("Homicidios por armas de fuego en EE.UU., 2010.")
```



Para agregar colores a los puntos, se tiene que modificar el componente de `geom_point(color=)`. Además de colores sólidos, puede utilizarse una variable categórica para colorear (esto implica un mapeo, dado que los colores están asociados a los datos que se están utilizando)

```
murders %>% ggplot(aes(x = population/10^6, y = total, label = abb))+
  geom_text(nudge_x = 0.075) +
  geom_point(aes(col=region), size = 3) +
  scale_x_log10() +
  scale_y_log10() +
  xlab("Población en millones (escala log)") +
  ylab("Total de homicidios (escala log)") +
  ggtitle("Homicidios por armas de fuego en EE.UU., 2010.")
```

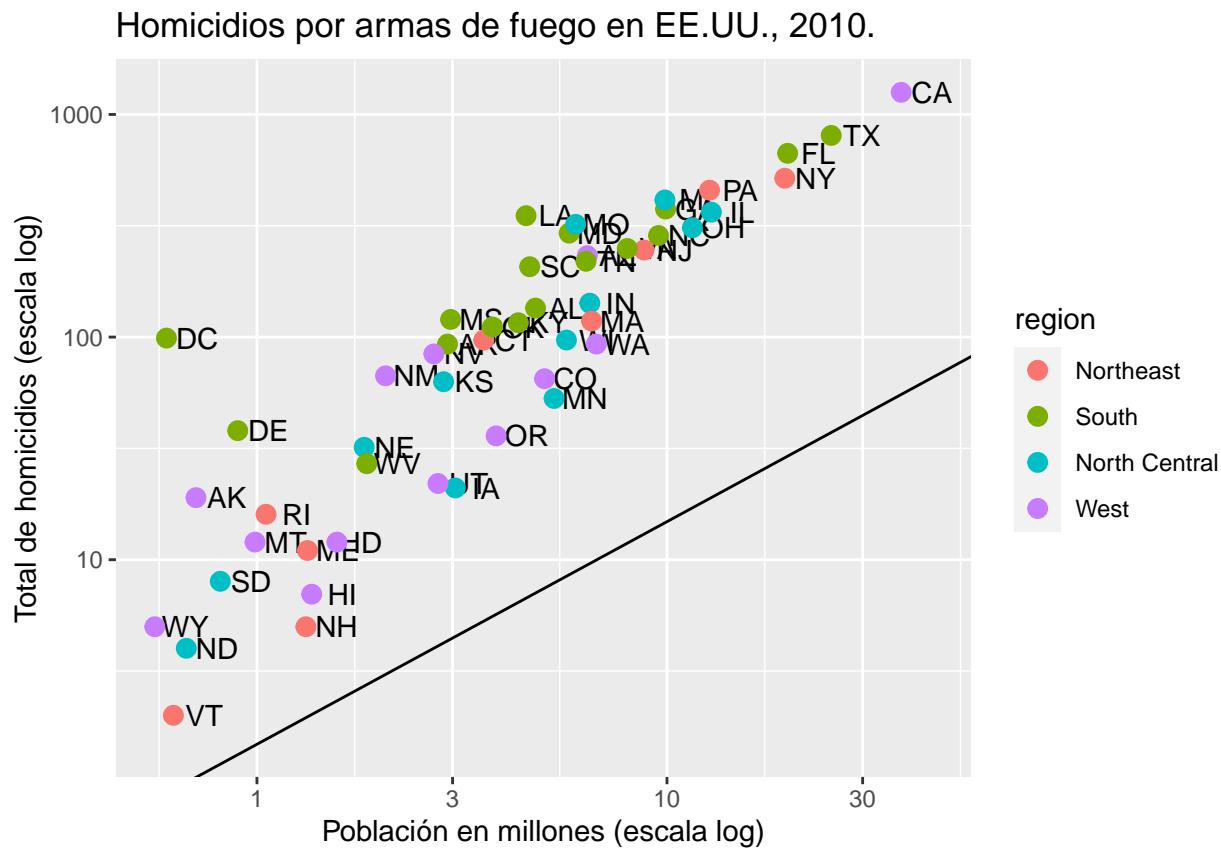


Para agregar una línea que represente la tasa de homicidios promedio para el país, defínase  $r$  como la tasa promedio nacional. Así, la línea estará definida por  $y = r * x$ , dado que si un estado tiene una población  $x$  y tiene una tasa de homicidios igual a la nacional,  $r$ , se multiplican para obtener el total de homicidios.

La línea se agrega mediante el comando “geom\_abline()”; por default, esta línea tiene ordenada 0 y pendiente 1

```
r <- murders %>%
  summarize(rate = sum(total) / sum(population)*10^6) %>% .$rate

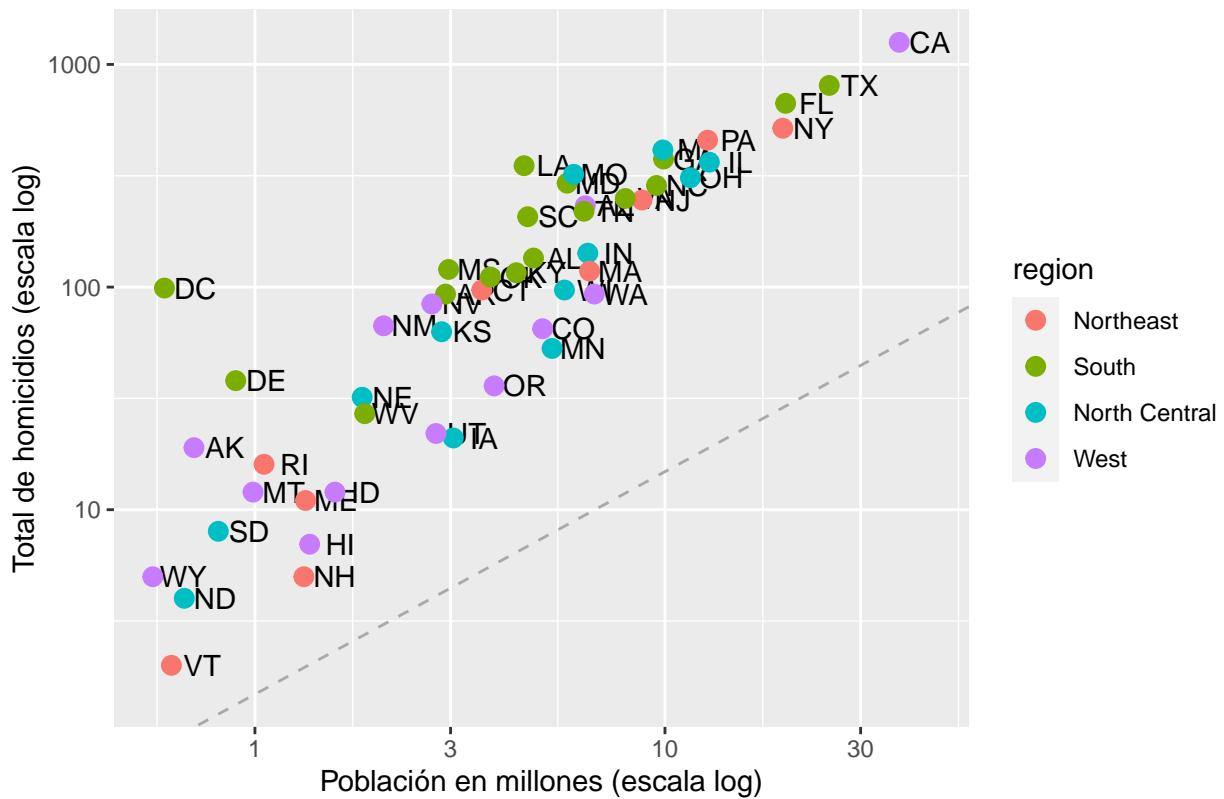
murders %>% ggplot(aes(x = population/10^6, y = total, label = abb)) +
  geom_text(nudge_x = 0.075) +
  geom_point(aes(col=region), size = 3) +
  geom_abline(intercept = log10(r)) +
  scale_x_log10() +
  scale_y_log10() +
  xlab("Población en millones (escala log)") +
  ylab("Total de homicidios (escala log)") +
  ggtitle("Homicidios por armas de fuego en EE.UU., 2010.")
```



Se puede modificar la línea para que sea punteada, gris y que esté “por debajo” de la demás información:

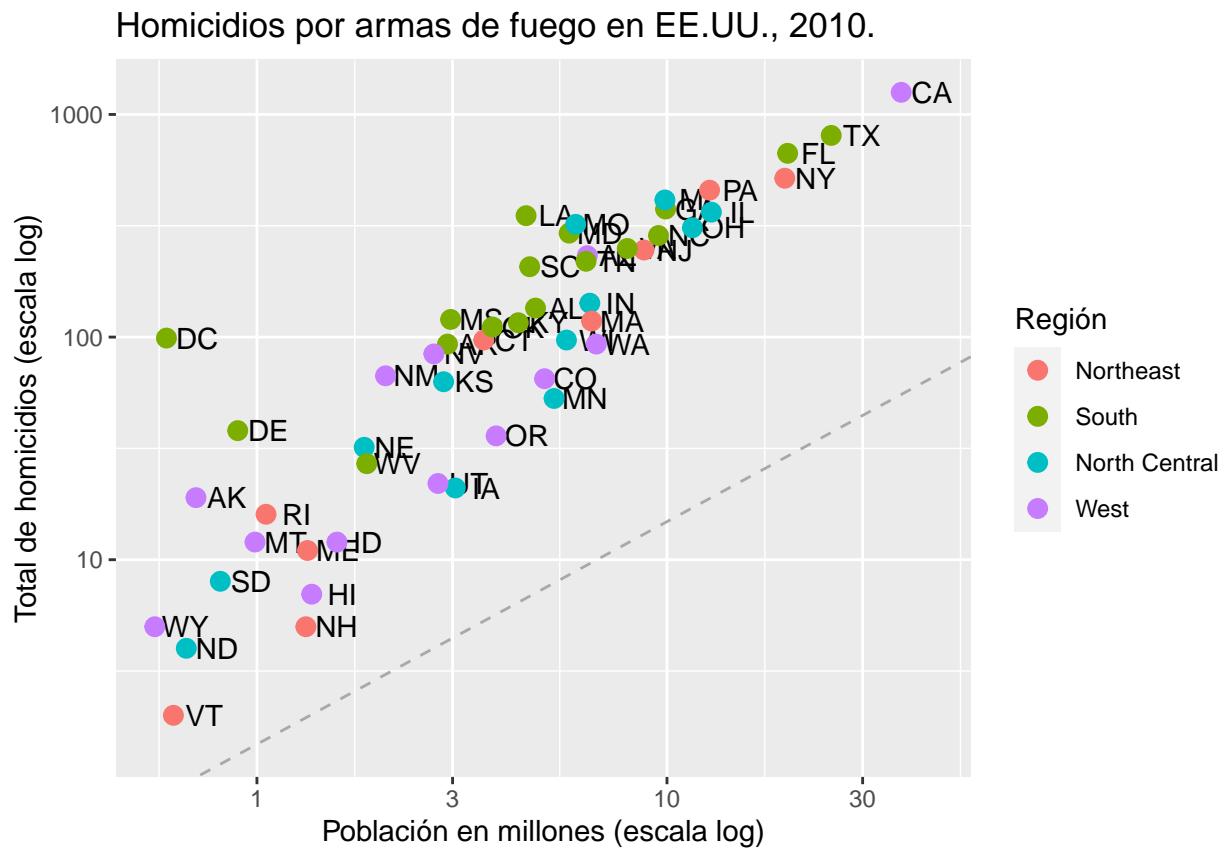
```
murders %>% ggplot(aes(x = population/10^6, y = total, label = abb))+
  geom_text(nudge_x = 0.075) +
  geom_abline(intercept = log10(r), lty = 2, color = "darkgrey") +
  geom_point(aes(col=region), size = 3) +
  scale_x_log10() +
  scale_y_log10() +
  xlab("Población en millones (escala log)") +
  ylab("Total de homicidios (escala log)") +
  ggtitle("Homicidios por armas de fuego en EE.UU., 2010.")
```

## Homicidios por armas de fuego en EE.UU., 2010.



Además, queremos que “region” empiece con mayúscula:

```
murders %>% ggplot(aes(x = population/10^6, y = total, label = abb))+
  geom_text(nudge_x = 0.075) +
  geom_abline(intercept = log10(r), lty = 2, color = "darkgrey") +
  scale_color_discrete(name = "Región") +
  geom_point(aes(col=region), size = 3) +
  scale_x_log10() +
  scale_y_log10() +
  xlab("Población en millones (escala log)") +
  ylab("Total de homicidios (escala log)") +
  ggtitle("Homicidios por armas de fuego en EE.UU., 2010.")
```

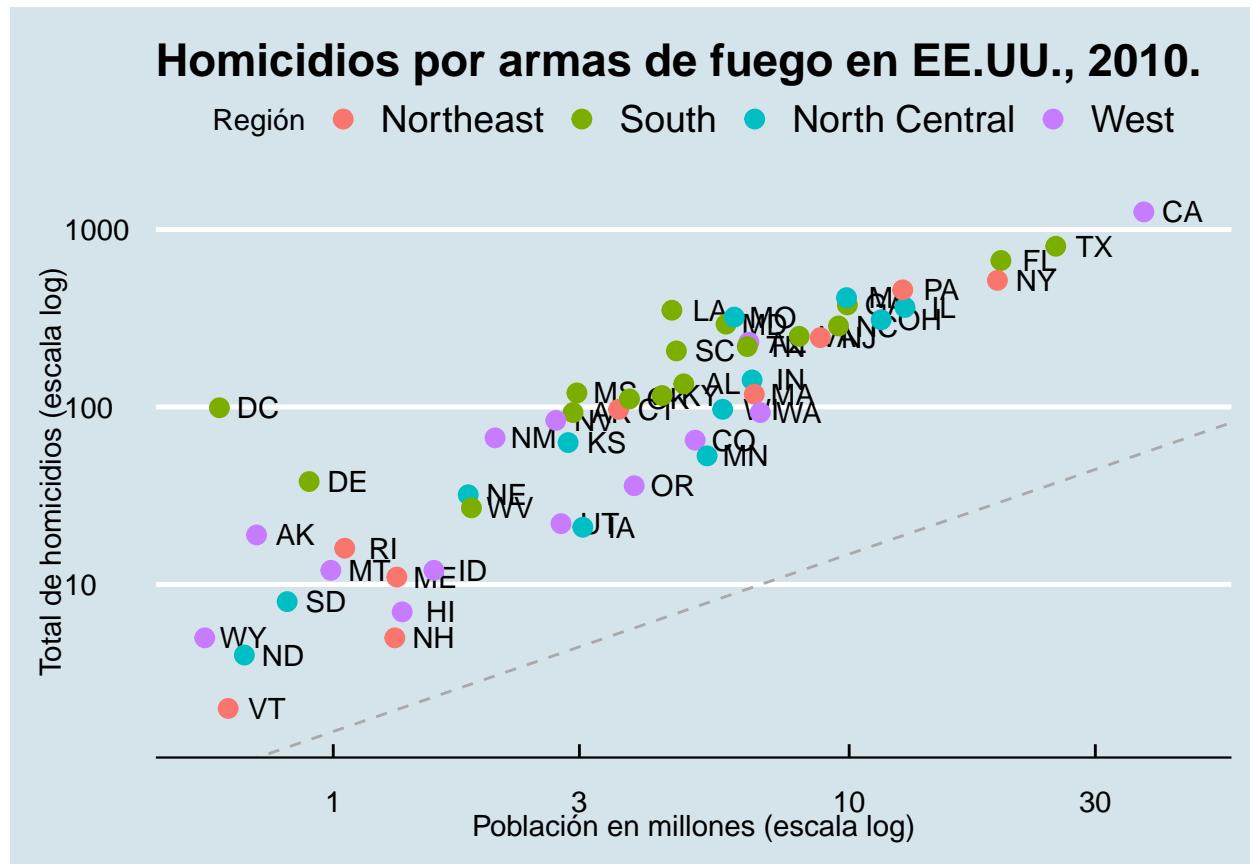


## 6. Paquetes add-on

Se utilizarán los paquetes ggthemes y ggrepel:

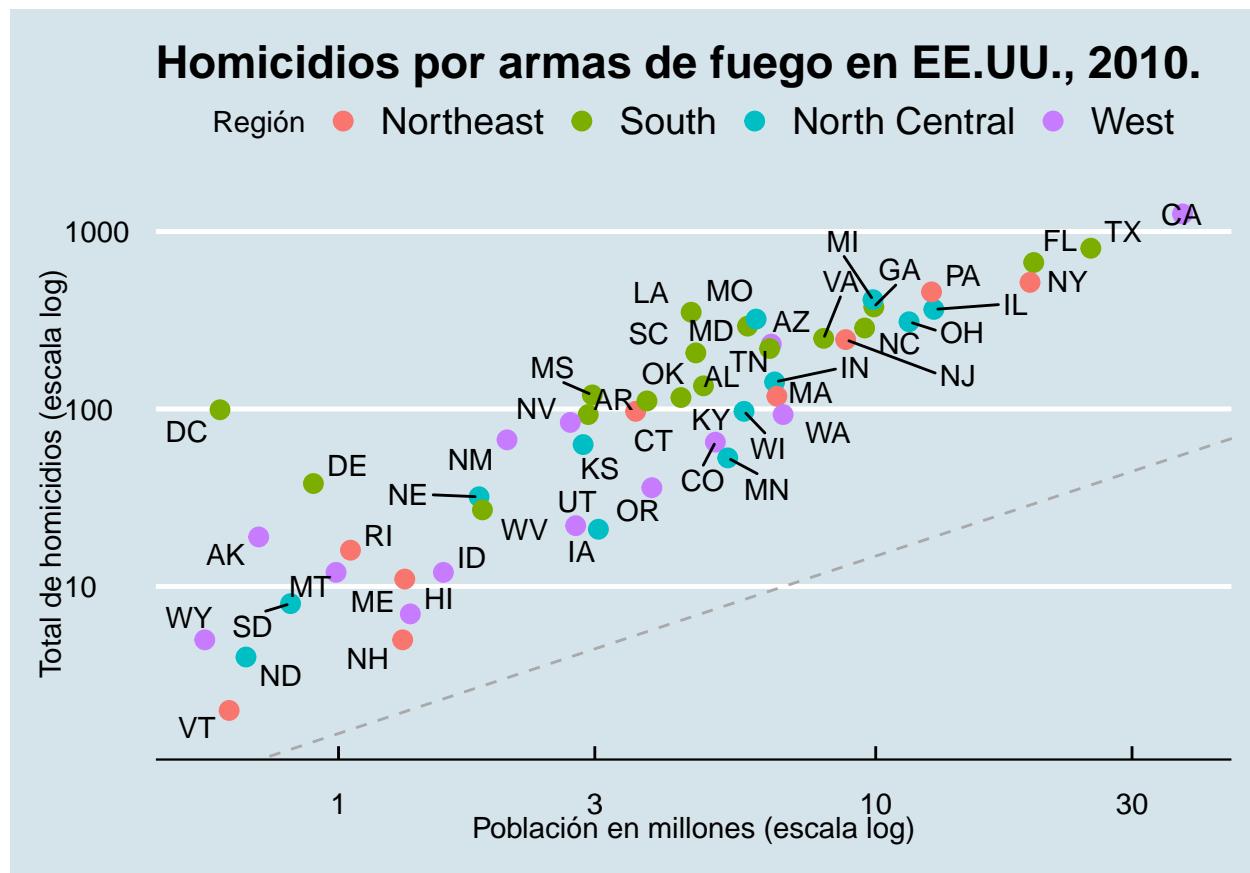
```
library(ggthemes)
library(ggrepel)

murders %>% ggplot(aes(x = population/10^6, y = total, label = abb))+
  geom_text(nudge_x = 0.075) +
  geom_abline(intercept = log10(r), lty = 2, color = "darkgrey") +
  scale_color_discrete(name = "Región") +
  geom_point(aes(col=region), size = 3) +
  scale_x_log10() +
  scale_y_log10() +
  xlab("Población en millones (escala log)") +
  ylab("Total de homicidios (escala log)") +
  ggtitle("Homicidios por armas de fuego en EE.UU., 2010.") +
  theme_economist()
```



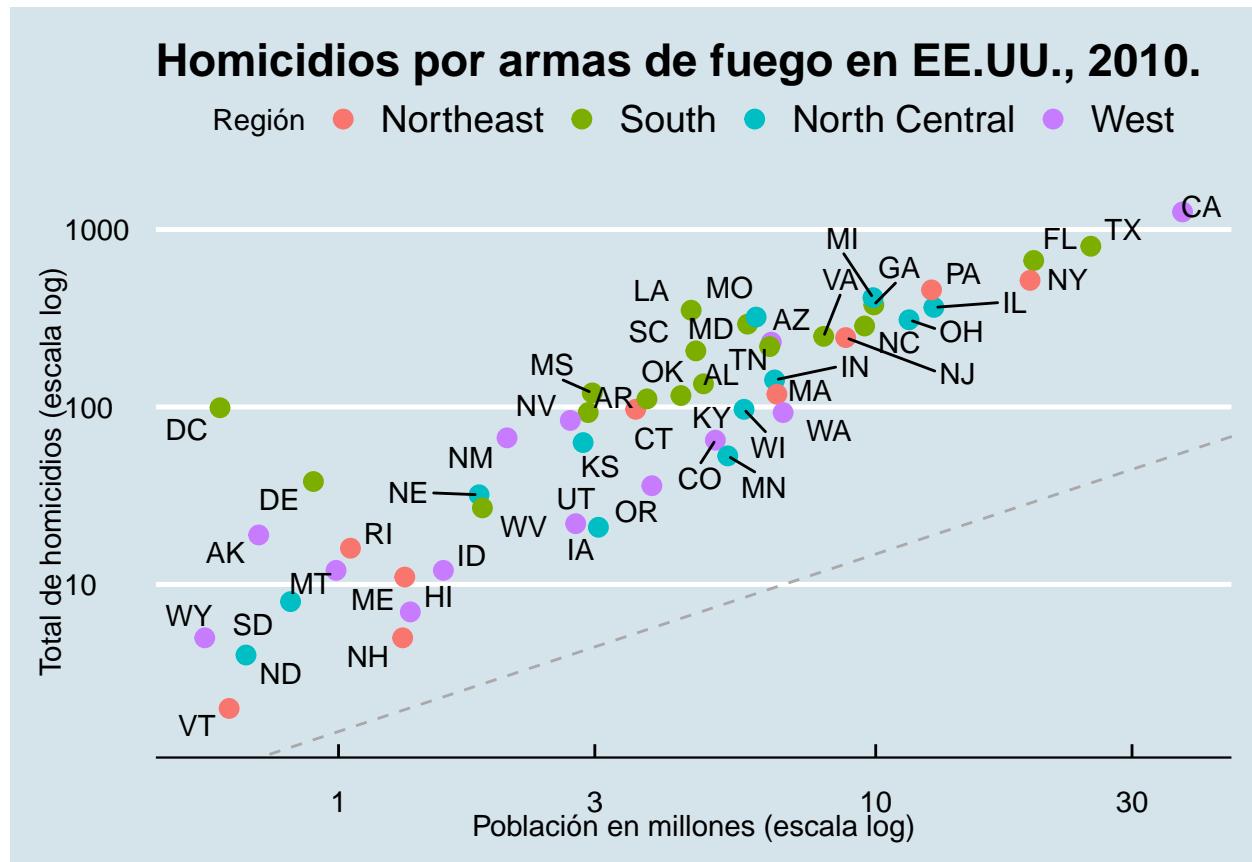
Para cambiar las posiciones de las etiquetas y evitar que se empalmen, se utiliza “`geom_text_repel()`”:

```
murders %>% ggplot(aes(x = population/10^6, y = total, label = abb))+
  geom_abline(intercept = log10(r), lty = 2, color = "darkgrey") +
  geom_point(aes(col=region), size = 3) +
  geom_text_repel() +
  scale_x_log10() +
  scale_y_log10() +
  xlab("Población en millones (escala log)") +
  ylab("Total de homicidios (escala log)") +
  ggtitle("Homicidios por armas de fuego en EE.UU., 2010.") +
  scale_color_discrete(name = "Región") +
  theme_economist()
```



#### 2.2.4. Resultado final

```
murders %>% ggplot(aes(x = population/10^6, y = total, label = abb))+
  geom_abline(intercept = log10(r), lty = 2, color = "darkgrey") +
  geom_point(aes(col=region), size = 3) +
  geom_text_repel() +
  scale_x_log10() +
  scale_y_log10() +
  xlab("Población en millones (escala log)") +
  ylab("Total de homicidios (escala log)") +
  ggtitle("Homicidios por armas de fuego en EE.UU., 2010.") +
  scale_color_discrete(name = "Región") +
  theme_economist()
```

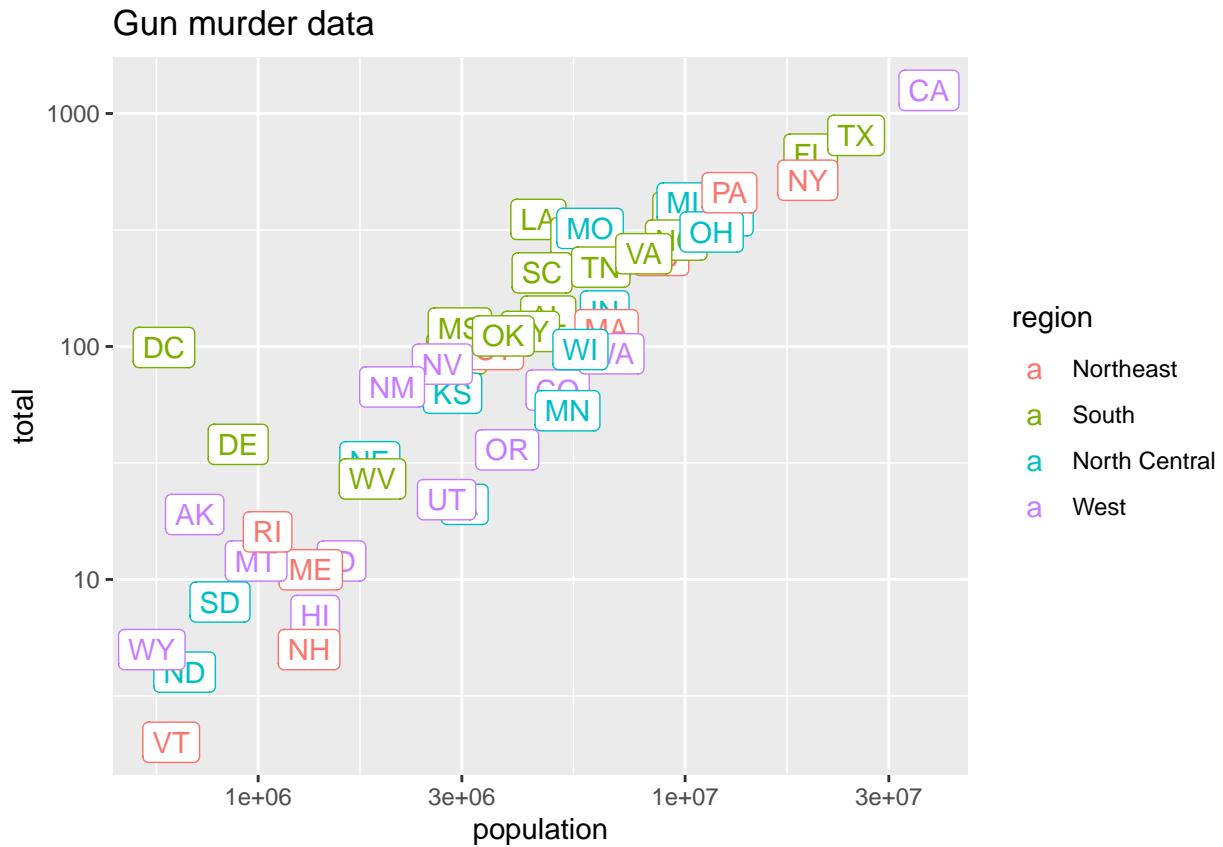


### 2.2.5. Versión alternativa

Si en vez de puntos queremos que las abreviaciones de los estados aparezcan, ya no se utiliza “geom\_point()”:

```
p <- murders %>% ggplot(aes(population, total, label = abb, color = region)) +
  geom_label()

p + scale_x_log10() +
  scale_y_log10() +
  ggtitle("Gun murder data")
```



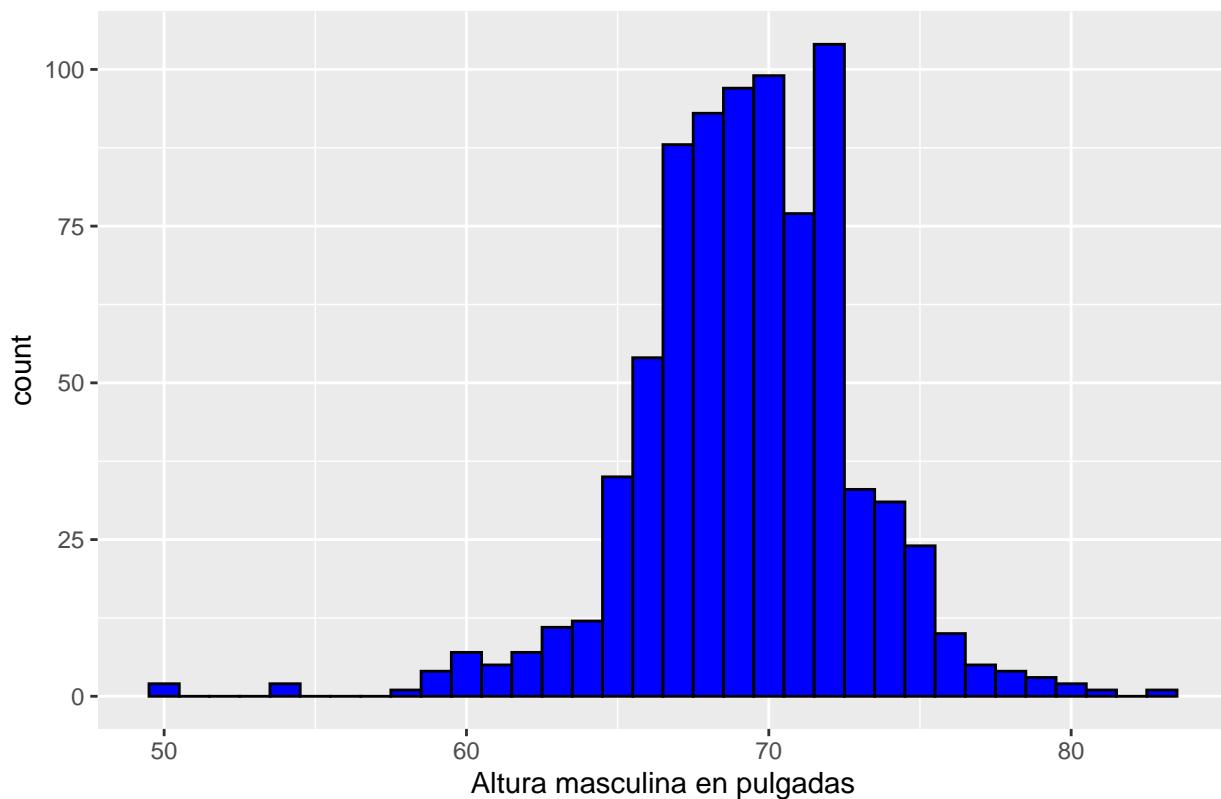
### 2.2.6. Otros ejemplos

Se quiere hacer un histograma de las alturas masculinas. Primero, hay que filtrar las alturas para solo trabajar con las de hombres:

Para este ejemplo, se utilizará “geom\_histogram()”:

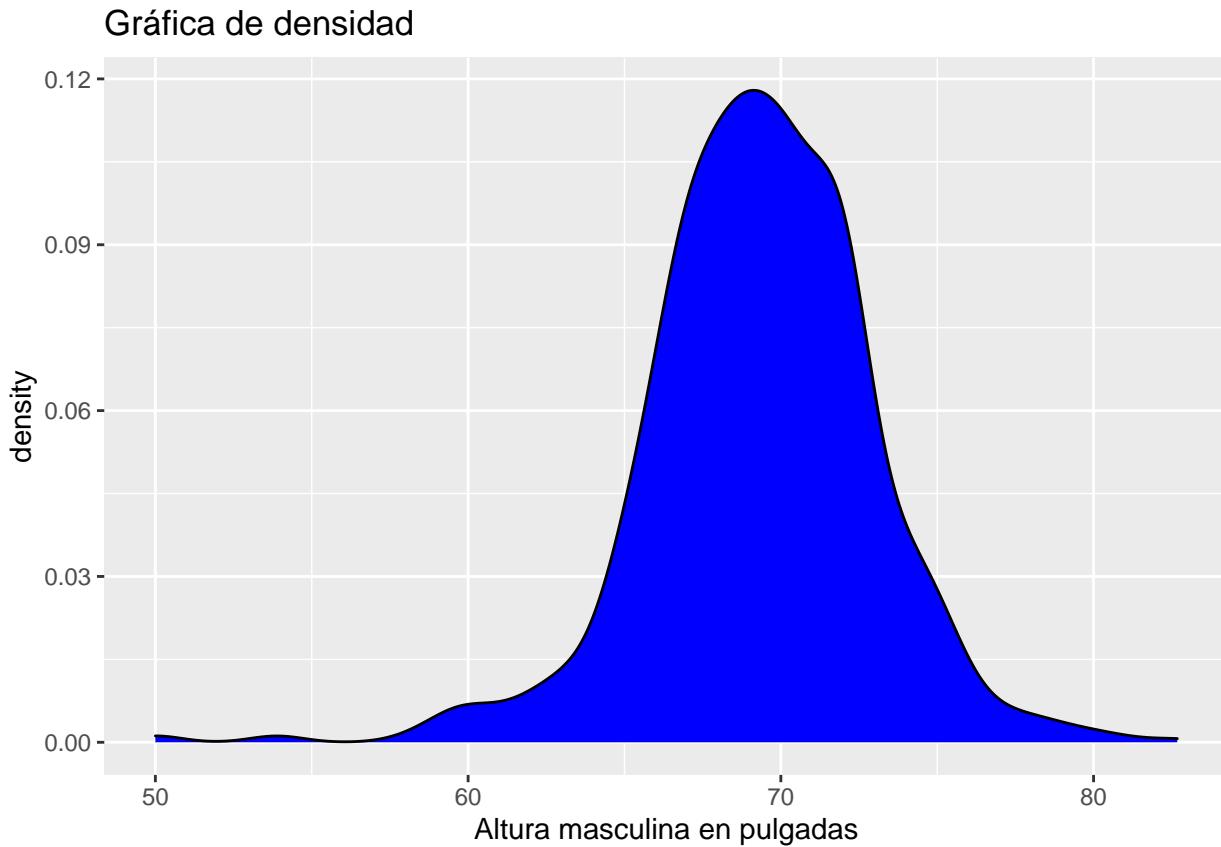
```
heights %>%
  filter(sex == "Male") %>%
  ggplot(aes(x = height)) +
  geom_histogram(binwidth = 1, fill = "blue", col = "black") +
  xlab("Altura masculina en pulgadas") +
  ggtitle("Histograma")
```

## Histograma



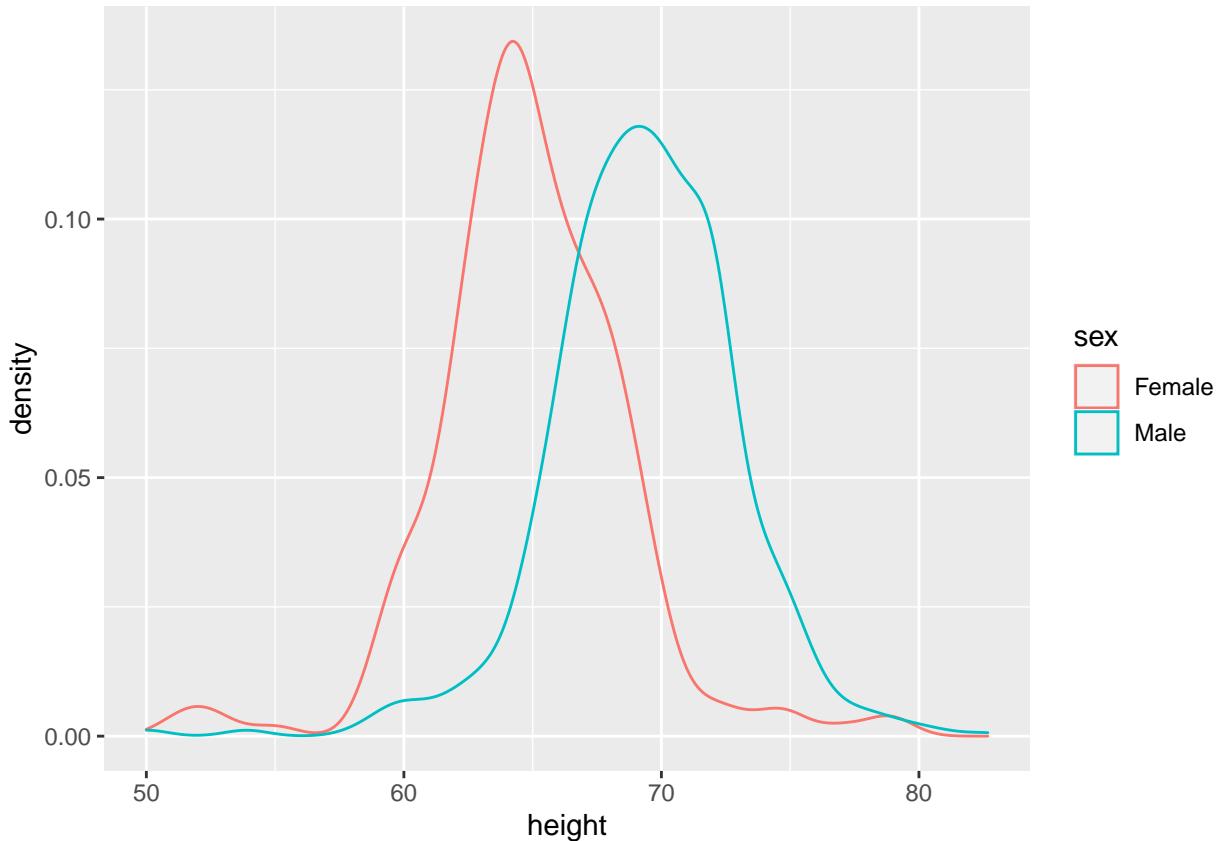
Se agregará una densidad suavizada con “geom\_density()”:

```
heights %>%
  filter(sex == "Male") %>%
  ggplot(aes(x = height)) +
  geom_density(fill = "blue") +
  xlab("Altura masculina en pulgadas") +
  ggtitle("Gráfica de densidad")
```

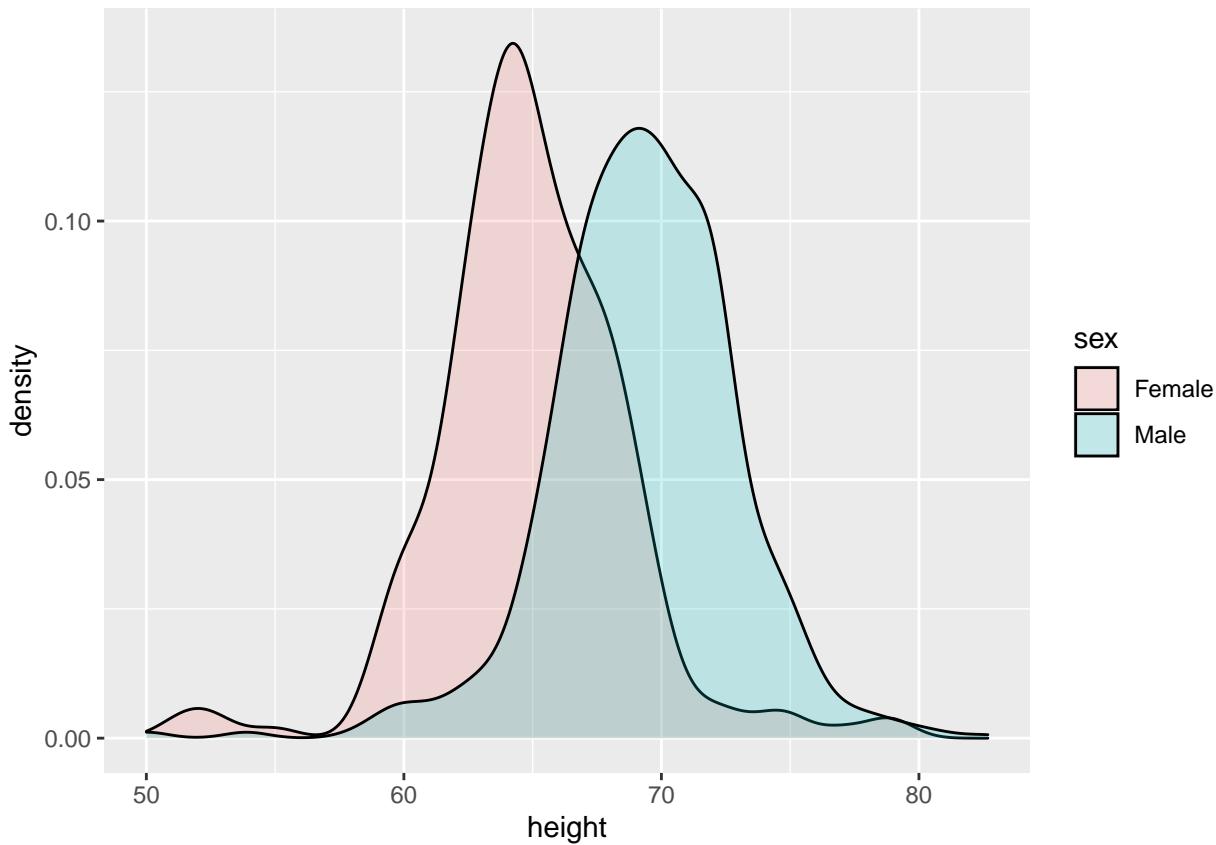


Si se quieren presentar dos gráficas de densidad simultáneamente, se utiliza el argumento “group=”:

```
heights %>%
  ggplot(aes(height, group = sex, color = sex)) +
  geom_density()
```



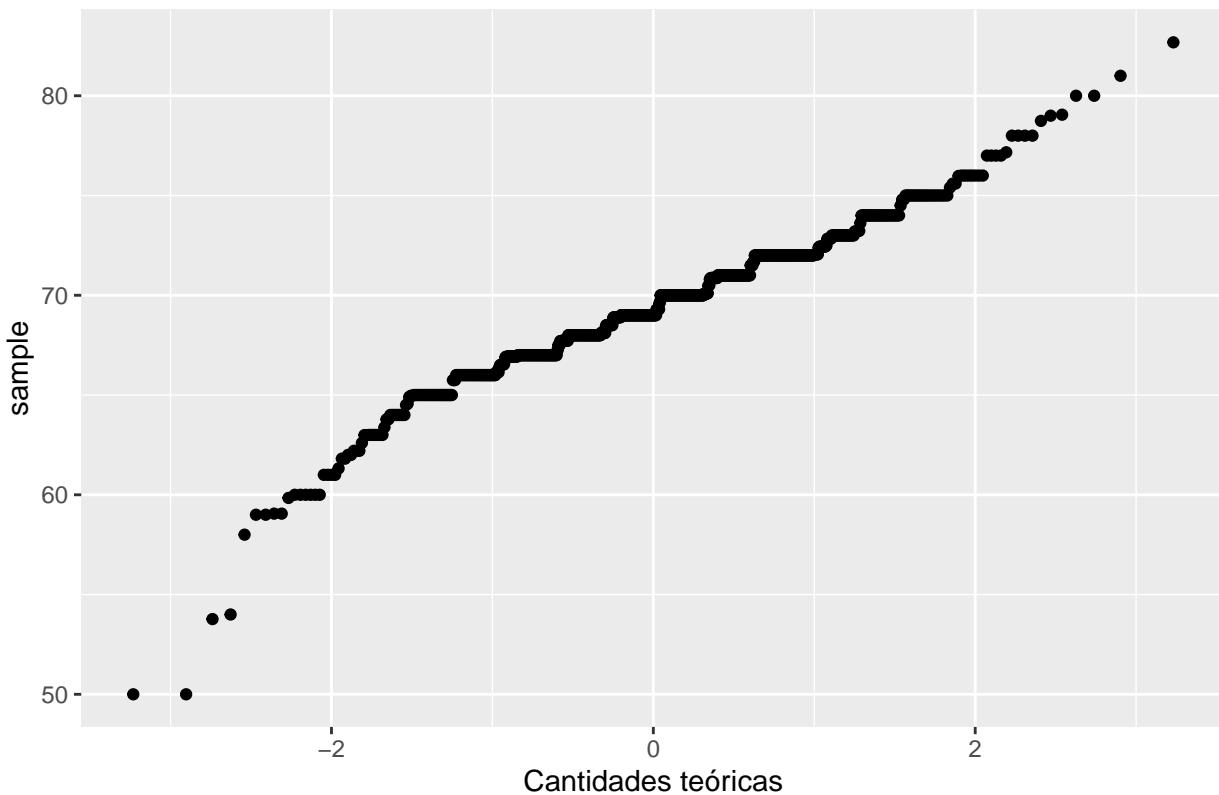
```
heights %>%
  ggplot(aes(height, group = sex, fill = sex)) +
  geom_density(alpha = 0.2)
```



Para un gráfico Q-Q, se usa “geom\_qq()”, donde hay que especifica una muestra y el default es una distribución normal estándar.

```
heights %>%
  filter(sex == "Male") %>%
  ggplot(aes(sample = height)) +
  geom_qq() +
  xlab("Cantidades teóricas") +
  ggtitle("Gráfica Q-Q")
```

Gráfica Q-Q

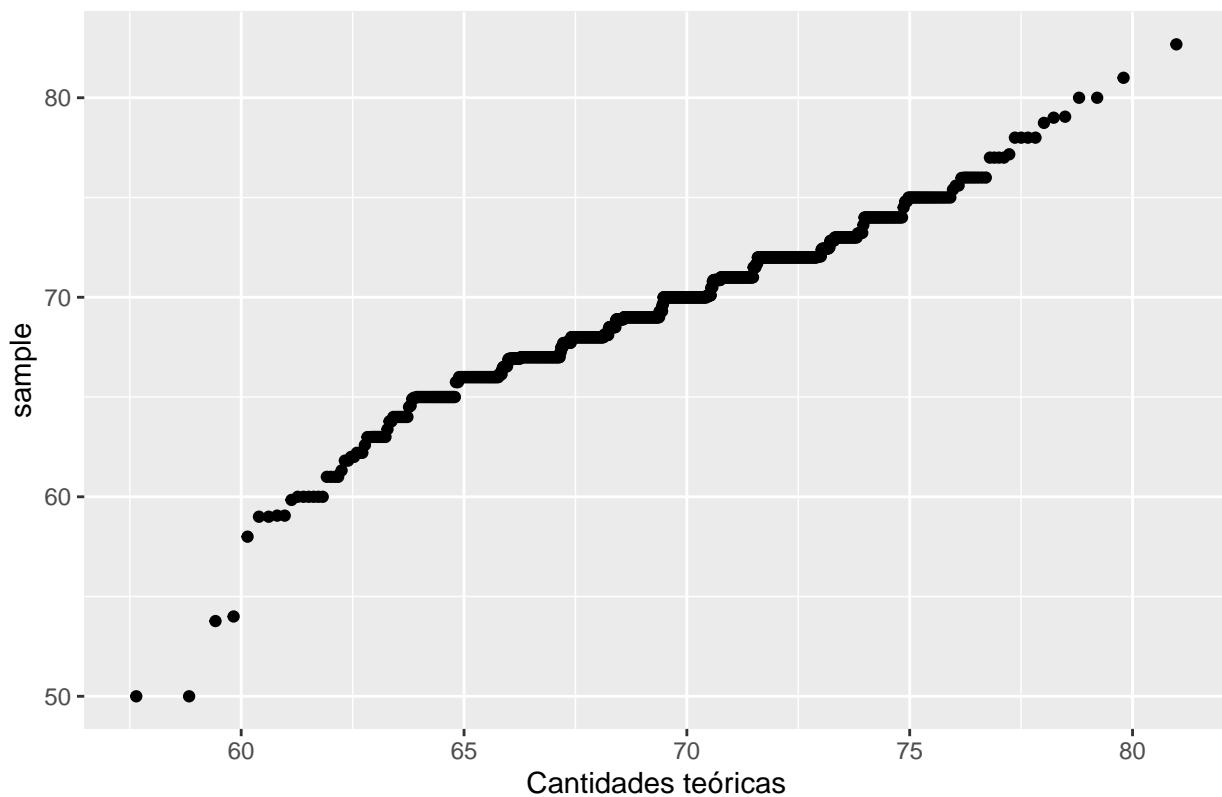


Puede definirse un objeto con la media y desviación estándar de nuestros datos con el objetivo de que las cantidades teóricas sean concistentes con los datos.

```
params <- heights %>%
  filter(sex == "Male") %>%
  summarize(mean = mean(height), sd = sd(height))

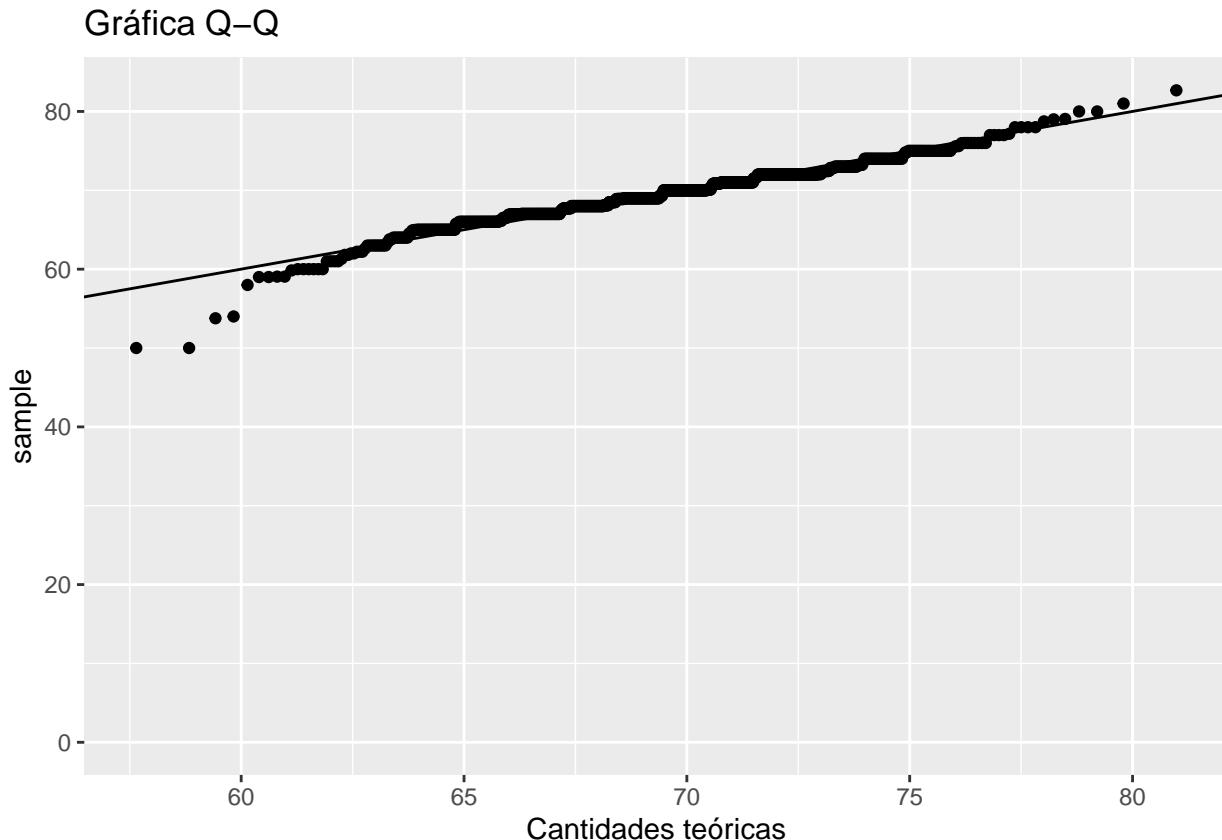
heights %>%
  filter(sex == "Male") %>%
  ggplot(aes(sample = height)) +
  geom_qq(dparams = params) +
  xlab("Cantidades teóricas") +
  ggtitle("Gráfica Q-Q")
```

Gráfica Q-Q



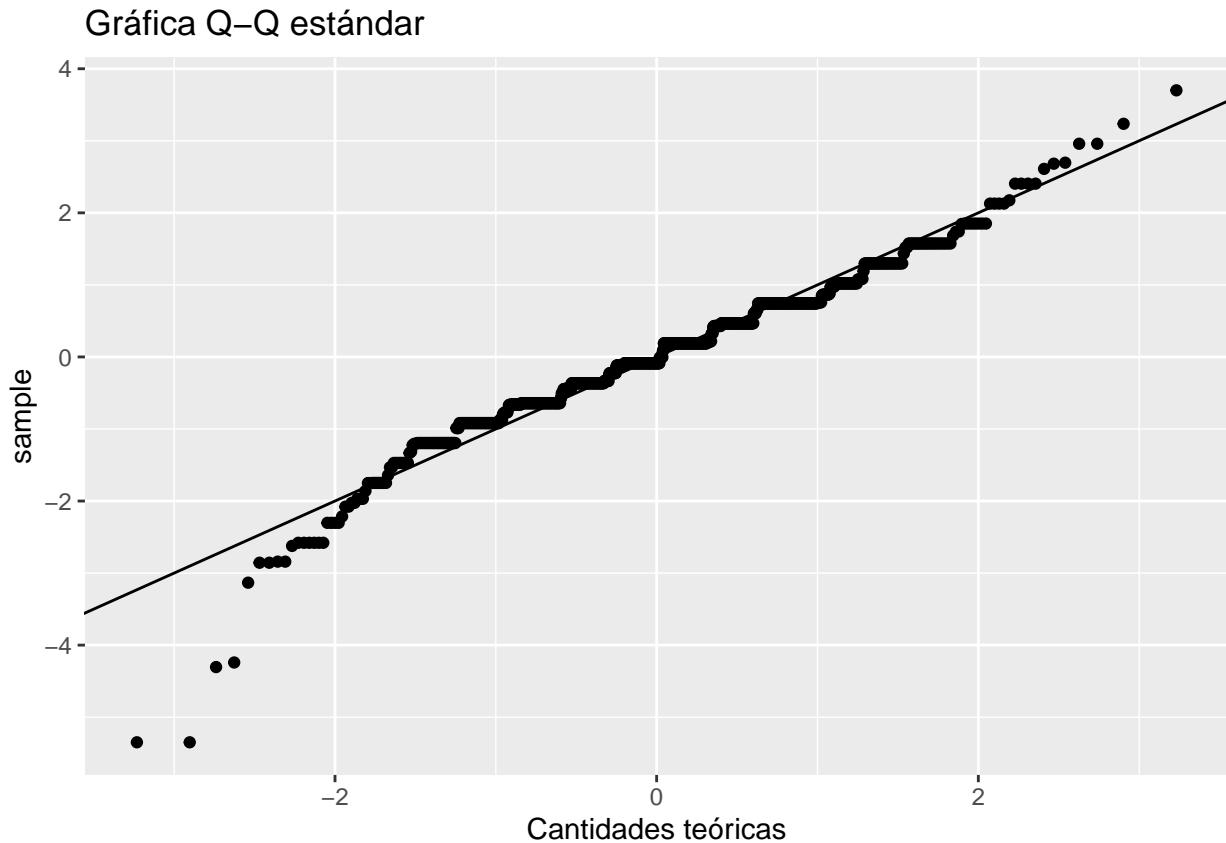
Se pueden agregar líneas de identidad para ver qué tanto se acerca a una distribución normal:

```
heights %>%
  filter(sex == "Male") %>%
  ggplot(aes(sample = height)) +
  geom_qq(dparams = params) +
  geom_abline() +
  xlab("Cantidades teóricas") +
  ggtitle("Gráfica Q-Q")
```



Se puede realizar el mismo análisis pero normalizando los datos:

```
heights %>%
  filter(sex == "Male") %>%
  ggplot(aes(sample = scale(height))) +
  geom_qq() +
  geom_abline() +
  xlab("Cantidades teóricas") +
  ggtitle("Gráfica Q-Q estandar")
```



### 2.2.7. Rejillas de gráficos

Para esto, se puede utilizar el paquete “gridExtra”

Hay que asignar los gráficos a objetos:

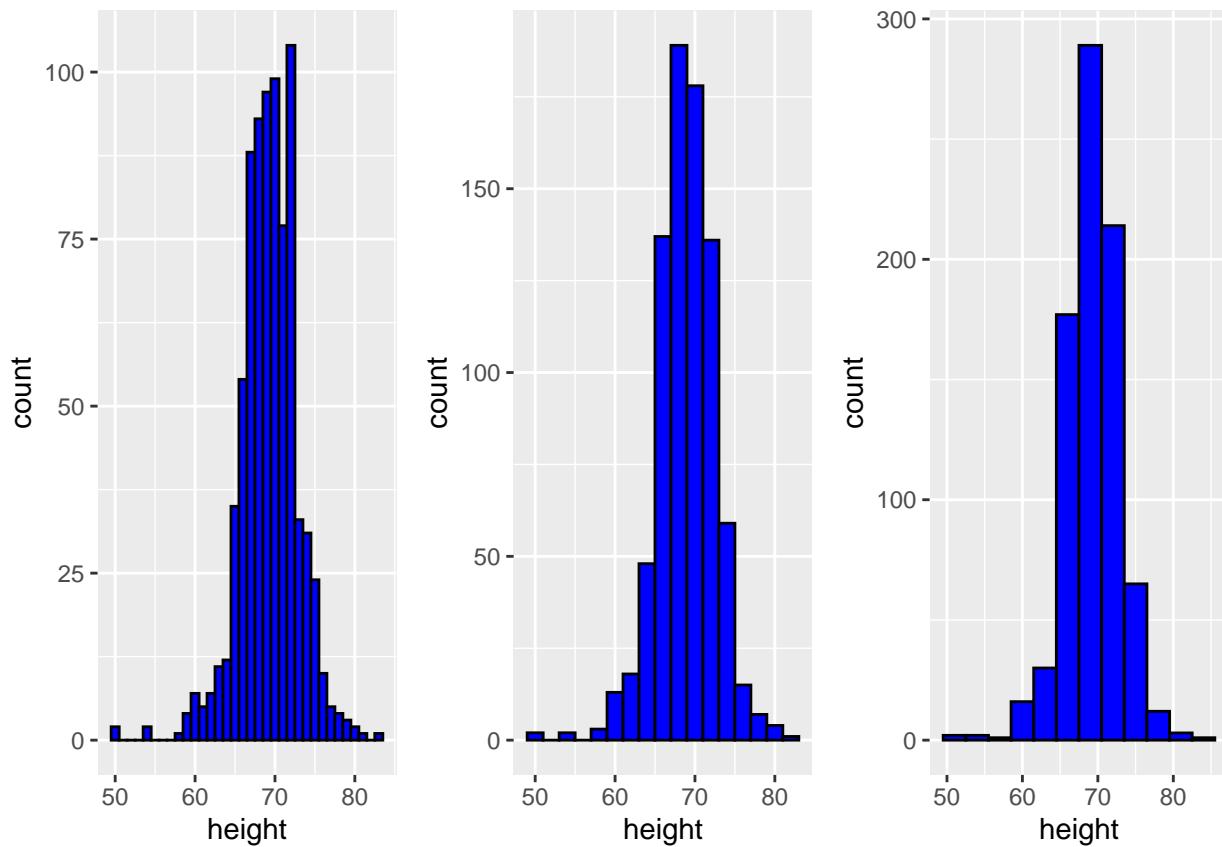
```
p <- heights %>% filter(sex == "Male") %>% ggplot(aes(x = height))

p1 <- p + geom_histogram(binwidth = 1, fill = "blue", col = "black")

p2 <- p + geom_histogram(binwidth = 2, fill = "blue", col = "black")

p3 <- p + geom_histogram(binwidth = 3, fill = "blue", col = "black")

grid.arrange(p1, p2, p3, ncol=3)
```



## 2.3. Resumiendo con dplyr

### 2.3.1. Introducción

Obtengamos el promedio y la desviación estándar para las alturas masculinas:

```
s <- heights %>%
  filter(sex == "Male") %>%
  summarize(promedio = mean(height), desviación_estándar = sd(height))

s

##   promedio desviación_estándar
## 1      69.3             3.61
s$promedio

## [1] 69.3
s$desviación_estándar

## [1] 3.61
```

Ahora, obtengamos la mediana, el máximo y el mínimo de las alturas masculinas:

```
heights %>%
  filter(sex == "Male") %>%
  summarize(mediana = median(height),
            mínimo = min(height),
            máximo = max(height))

##   mediana mínimo máximo
## 1      69     50    82.7
```

Se pueden obtener los mismos resultados mediante la función quantile:

```
heights %>%
  filter(sex == "Male") %>%
  summarize(range = quantile(height, c(0, .5, 1)))

##   range
## 1 50.0
## 2 69.0
## 3 82.7
```

### 2.3.2. Marcador de posición de punto

Con este operador, dplyr arrojará vectores en vez de dataframes.

Obtengamos el promedio de la tasa de homicidios (nótese que no se calcula la tasa promedio nacional, ya que la siguiente operación pondera igualmente a todos los estados, chicos o grandes):

```
murders <- murders %>% mutate(murder_rate = total/population*100000)

summarize(murders, mean(murder_rate))

##   mean(murder_rate)
## 1           2.78
```

Para calcular la tasa de homicidios correcta, hay que dividir el número total de homicidios entre la población total:

```

us_murder_rate <- murders %>%
  summarize(rate = sum(total) / sum(population)*100000)

us_murder_rate

##   rate
## 1 3.03
class(us_murder_rate)

## [1] "data.frame"

```

Donde el objeto us\_murder\_rate es de clase dataframe. Si queremos utilizar solo el valor, hay que utilizar el accesor de punto:

```

us_murder_rate %>% .$rate

## [1] 3.03
class(us_murder_rate %>% .$rate)

## [1] "numeric"

```

Así, resumiendo el procedimiento descrito, y si queremos crear un objeto con el VALOR de la tasa de homicidios:

```

us_murder_rate <- murders %>%
  summarize(rate = sum(total) / sum(population)*100000) %>%
  .$rate

us_murder_rate

## [1] 3.03
class(us_murder_rate)

## [1] "numeric"

```

### 2.3.3. Group by

Una operación muy común en el análisis de datos es separar a los datos por grupos, y después resumir cada uno.

Por ejemplo, puede quererse computar la media y desviación estándar de las alturas femenina y masculina por separado:

```

heights %>% group_by(sex)

## # A tibble: 1,050 x 2
## # Groups:   sex [2]
##       sex     height
##   <fct>    <dbl>
## 1 Male      75
## 2 Male      70
## 3 Male      68
## 4 Male      74
## 5 Male      61
## 6 Female    65
## 7 Female    66
## 8 Female    62

```

```
## 9 Female      66
## 10 Male       67
## # ... with 1,040 more rows
```

Donde la tabla arrojada se conoce como **group data frame**.

```
heights %>%
  group_by(sex) %>%
  summarize(promedio = mean(height), desviación_estándar = sd(height))

## # A tibble: 2 x 3
##   sex     promedio desviación_estándar
##   <fct>    <dbl>          <dbl>
## 1 Female    64.9           3.76
## 2 Male      69.3           3.61
```

Ahora, calcularemos la tasa de homicidios para cada región de EE.UU.:

```
murders %>%
  group_by(region) %>%
  summarize(median_rate = median(murder_rate))

## # A tibble: 4 x 2
##   region      median_rate
##   <fct>        <dbl>
## 1 Northeast    1.80
## 2 South        3.40
## 3 North Central 1.97
## 4 West         1.29
```

### 2.3.4. Promedios por grupo

Para calcular la presión sanguínea promedio por grupo de edad:

```
NHANES %>%
  filter(Gender == "female") %>%
  group_by(AgeDecade) %>%
  summarize(average = mean(BPSysAve, na.rm = TRUE), standard_deviation = sd(BPSysAve, na.rm = TRUE))

## # A tibble: 9 x 3
##   AgeDecade average standard deviation
##   <fct>      <dbl>          <dbl>
## 1 "0-9"       100.           9.07
## 2 "10-19"     104.           9.46
## 3 "20-29"     108.          10.1
## 4 "30-39"     111.          12.3
## 5 "40-49"     115.          14.5
## 6 "50-59"     122.          16.2
## 7 "60-69"     127.          17.1
## 8 "70+"       134.          19.8
## 9 <NA>        142.          22.9

NHANES %>%
  filter(Gender == "male") %>%
  group_by(AgeDecade) %>%
  summarize(average = mean(BPSysAve, na.rm = TRUE), standard deviation = sd(BPSysAve, na.rm = TRUE))

## # A tibble: 9 x 3
```

```
##   AgeDecade average standard_deviation
##   <fct>      <dbl>          <dbl>
## 1 "0-9"       97.4           8.32
## 2 "10-19"     110.            11.2
## 3 "20-29"     118.            11.3
## 4 "30-39"     119.            12.3
## 5 "40-49"     121.            14.0
## 6 "50-59"     126.            17.8
## 7 "60-69"     127.            17.5
## 8 "70+"       130.            18.7
## 9 <NA>        136.            23.5
```

Pueden presentarse los resultados simultáneamente:

```
NHANES %>% group_by(AgeDecade, Gender) %>%
  summarize(average = mean(BPSysAve, na.rm = TRUE),
            standard_deviation = sd(BPSysAve, na.rm=TRUE))

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

## # A tibble: 18 x 4
## # Groups:   AgeDecade [9]
##   AgeDecade Gender average standard deviation
##   <fct>     <fct>    <dbl>          <dbl>
## 1 "0-9"      female    100.            9.07
## 2 "0-9"      male      97.4           8.32
## 3 "10-19"    female   104.            9.46
## 4 "10-19"    male     110.            11.2
## 5 "20-29"    female   108.            10.1
## 6 "20-29"    male     118.            11.3
## 7 "30-39"    female   111.            12.3
## 8 "30-39"    male     119.            12.3
## 9 "40-49"    female   115.            14.5
## 10 "40-49"   male     121.            14.0
## 11 "50-59"   female   122.            16.2
## 12 "50-59"   male     126.            17.8
## 13 "60-69"   female   127.            17.1
## 14 "60-69"   male     127.            17.5
## 15 "70+"     female   134.            19.8
## 16 "70+"     male     130.            18.7
## 17 <NA>       female   142.            22.9
## 18 <NA>       male     136.            23.5
```

Finalmente, para presentar los promedios por raza, de los hombres entre 40-49 años:

```
NHANES %>% filter(Gender == "male", AgeDecade == "40-49") %>%
  group_by(Race1) %>%
  summarize(average = mean(BPSysAve, na.rm = TRUE),
            standard deviation = sd(BPSysAve, na.rm=TRUE)) %>%
  arrange(average)

## # A tibble: 3 x 3
##   Race1   average standard deviation
##   <fct>    <dbl>          <dbl>
## 1 White    120.            13.4
## 2 Other    120.            16.2
## 3 Hispanic 122.            11.1
```

```
## 4 Mexican      122.          13.9
## 5 Black        126.          17.1
```

### 2.3.5. Clasificación de tablas de datos

Cuando se trabaja con una base de datos, puede ser conveniente clasificarla mediante diferentes columnas. Para el ordenamiento de tablas completas, la función “arrange()” de dplyr es muy útil. Por ejemplo, si se quiere ordenar a los estados a partir de su tamaño poblacional:

```
murders %>% arrange(population) %>% head()
```

```
##           state abb      region population total murder_rate
## 1         Wyoming WY        West    563626     5     0.887
## 2 District of Columbia DC       South   601723    99    16.453
## 3         Vermont VT    Northeast  625741     2     0.320
## 4       North Dakota ND  North Central 672591     4     0.595
## 5         Alaska AK        West   710231    19     2.675
## 6    South Dakota SD  North Central 814180     8     0.983
```

En cambio, si se quieren ordenar los estados con base en su tasa de homicidios de manera descendente, se utiliza la función “desc()”:

```
murders %>% arrange(desc(murder_rate)) %>% head()
```

```
##           state abb      region population total murder_rate
## 1 District of Columbia DC       South   601723    99    16.45
## 2         Louisiana LA       South  4533372   351     7.74
## 3         Missouri MO  North Central 5988927   321     5.36
## 4         Maryland MD       South  5773552   293     5.07
## 5    South Carolina SC       South  4625364   207     4.48
## 6         Delaware DE       South  897934    38     4.23
```

Supóngase que existen valores empatados. Un criterio de desempate puede ser con base en una segunda, tercera o cuarta columna. Así, pueden ordenarse los datos por región, y dentro de cada región, por tasa de homicidios:

```
murders1 <- murders %>% arrange(region, desc(murder_rate))
```

Otra función muy útil es “top\_n()”, la cual permite observar una proporción más grande de los datos que con “head()”:

```
murders1 %>% arrange(desc(murder_rate)) %>% top_n(10)
```

```
## Selecting by murder_rate
##           state abb      region population total murder_rate
## 1 District of Columbia DC       South   601723    99    16.45
## 2         Louisiana LA       South  4533372   351     7.74
## 3         Missouri MO  North Central 5988927   321     5.36
## 4         Maryland MD       South  5773552   293     5.07
## 5    South Carolina SC       South  4625364   207     4.48
## 6         Delaware DE       South  897934    38     4.23
## 7         Michigan MI  North Central 9883640   413     4.18
## 8     Mississippi MS       South  2967297   120     4.04
## 9         Georgia GA       South 9920000   376     3.79
## 10        Arizona AZ        West  6392017   232     3.63
```

**NOTA:** Pueden removverse los NA de un dataframe mediante el argumento “na.rm=TRUE”.

```
library(dslabs)
data(na_example)
mean(na_example)

## [1] NA
sd(na_example)

## [1] NA
mean(na_example, na.rm = TRUE)

## [1] 2.3
sd(na_example, na.rm = TRUE)

## [1] 1.22
```

## 2.4. Gapminder

Gapminder es una iniciativa que busca desbancar creencias sensacionalistas sobre fenómenos como pobreza, equidad de género, desigualdad, etc., mediante datos y gráficos sumamente interactivos y verosímiles.

Estudiaremos diferencias en la mortalidad infantil a través de diferentes países:

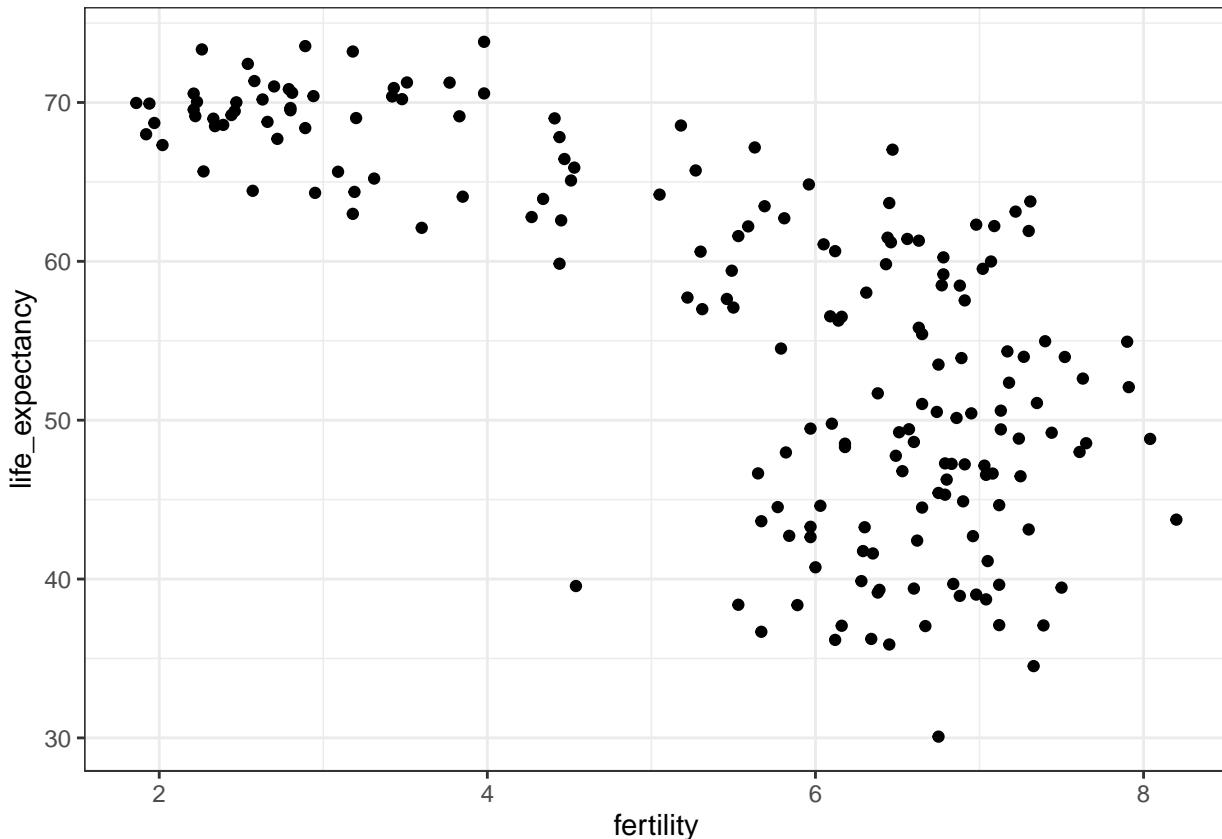
```
gapminder %>%
  filter(year == 2015 &
        country %in% c("Sri Lanka", "Poland", "Turkey", "Malaysia", "Pakistan", "Thailand")) %>%
  select(country, infant_mortality)

##      country infant_mortality
## 1  Malaysia              6.0
## 2  Pakistan             65.8
## 3   Poland              4.5
## 4 Sri Lanka              8.4
## 5  Thailand             10.5
## 6    Turkey             11.6
```

Ahora, grafiquemos la expectativa de vida contra las tasas de fertilidad en 1962.

```
ds_theme_set()

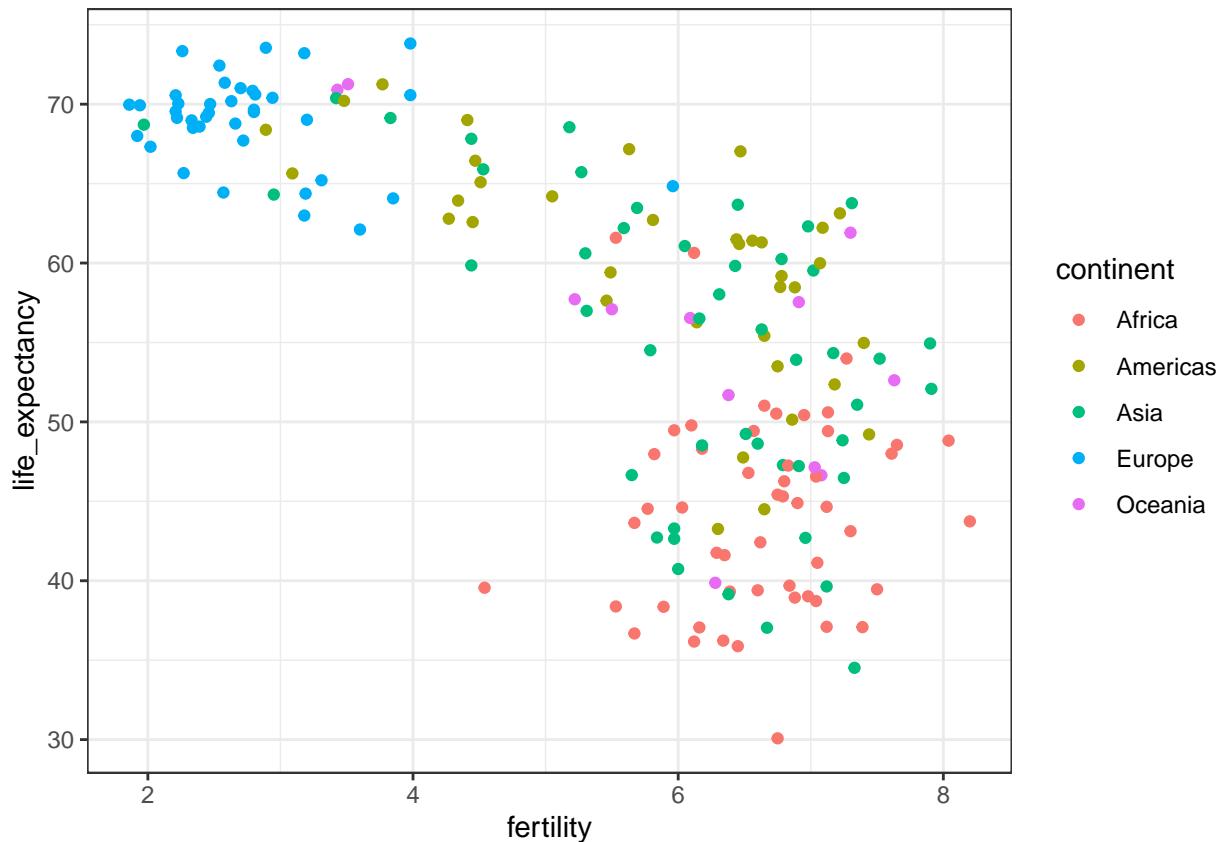
filter(gapminder, year == 1962) %>%
  ggplot(aes(fertility, life_expectancy)) +
  geom_point()
```



Puede verse que, a grandes rasgos, se forman dos conjuntos de datos: aquellos con expectativa de vida de alrededor de 70 años y con fertilidad más baja, y aquellos con expectativas de vida de menos de 65 años y

fertilidad más alta. ¿Representan los puntos a países desarrollados y en vías de desarrollo, como se esperaría?

```
filter(gapminder, year == 1962) %>%
  ggplot(aes(fertility, life_expectancy, color = continent)) +
  geom_point()
```

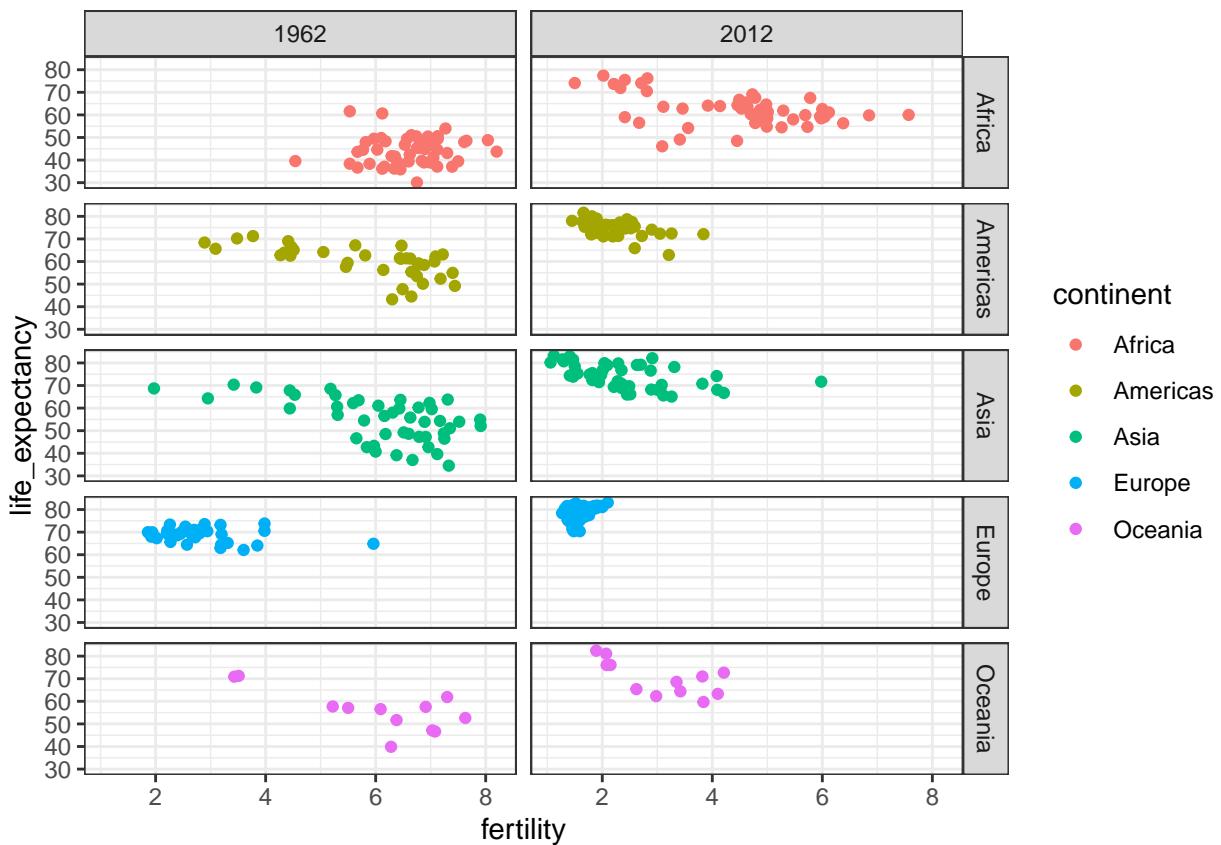


Efectivamente, se ve que el primer conjunto lo conforman, principalmente, países europeos, mientras que el segundo grupo está conformado por África, Latinoamérica y Asia.

#### 2.4.1. Facetado

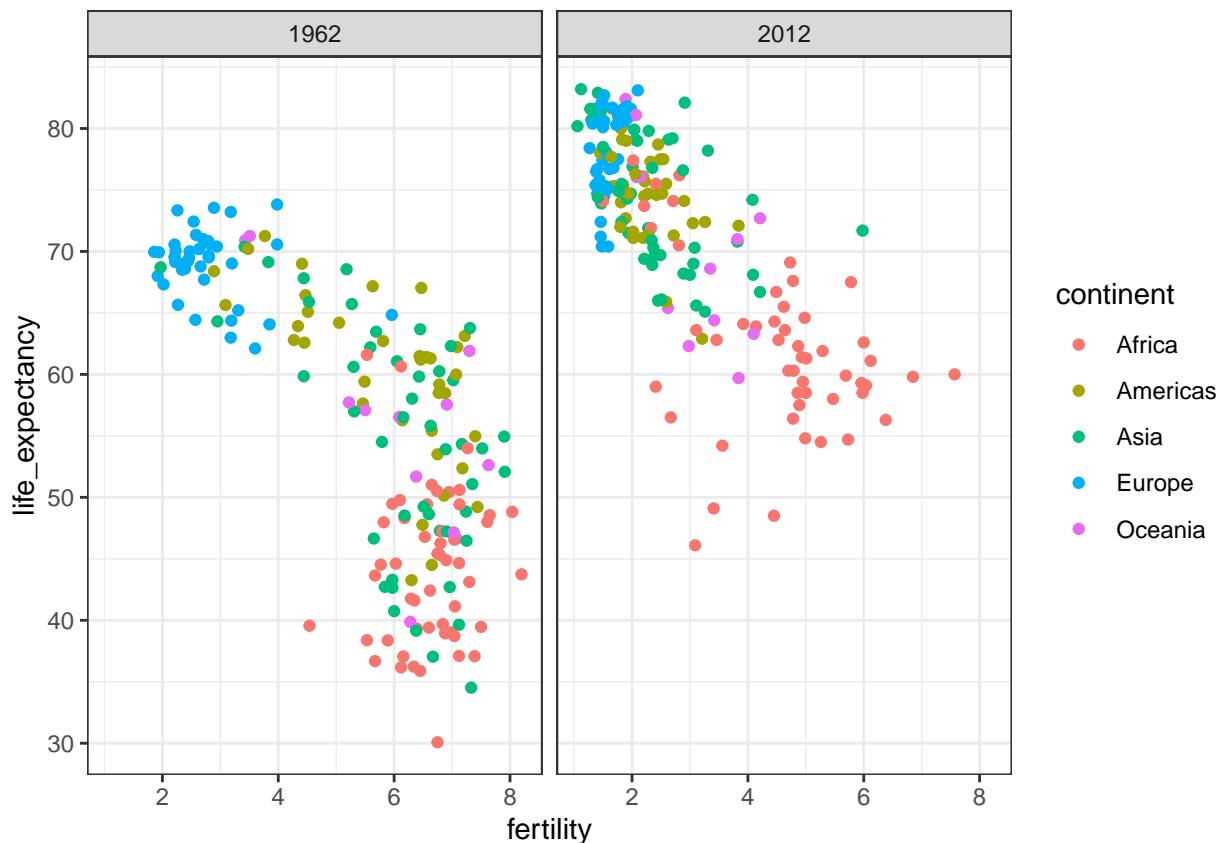
¿Esto se cumple 50 años después? Puede graficarse cada año por separado; sin embargo, es preferible graficarlos simultáneamente, para fines de mejor comparabilidad. En ggplot, esto se puede lograr mediante el **facetado de variables** con la función “facet\_grid()”:

```
filter(gapminder, year %in% c(1962, 2012)) %>%
  ggplot(aes(fertility, life_expectancy, col = continent)) +
  geom_point() +
  facet_grid(continent ~ year)
```



Si queremos comparar la situación mundial en los dos años (esto es, no estamos usando ninguna variables como filas):

```
filter(gapminder, year %in% c(1962, 2012)) %>%
  ggplot(aes(fertility, life_expectancy, col = continent)) +
  geom_point() +
  facet_grid(.~year)
```

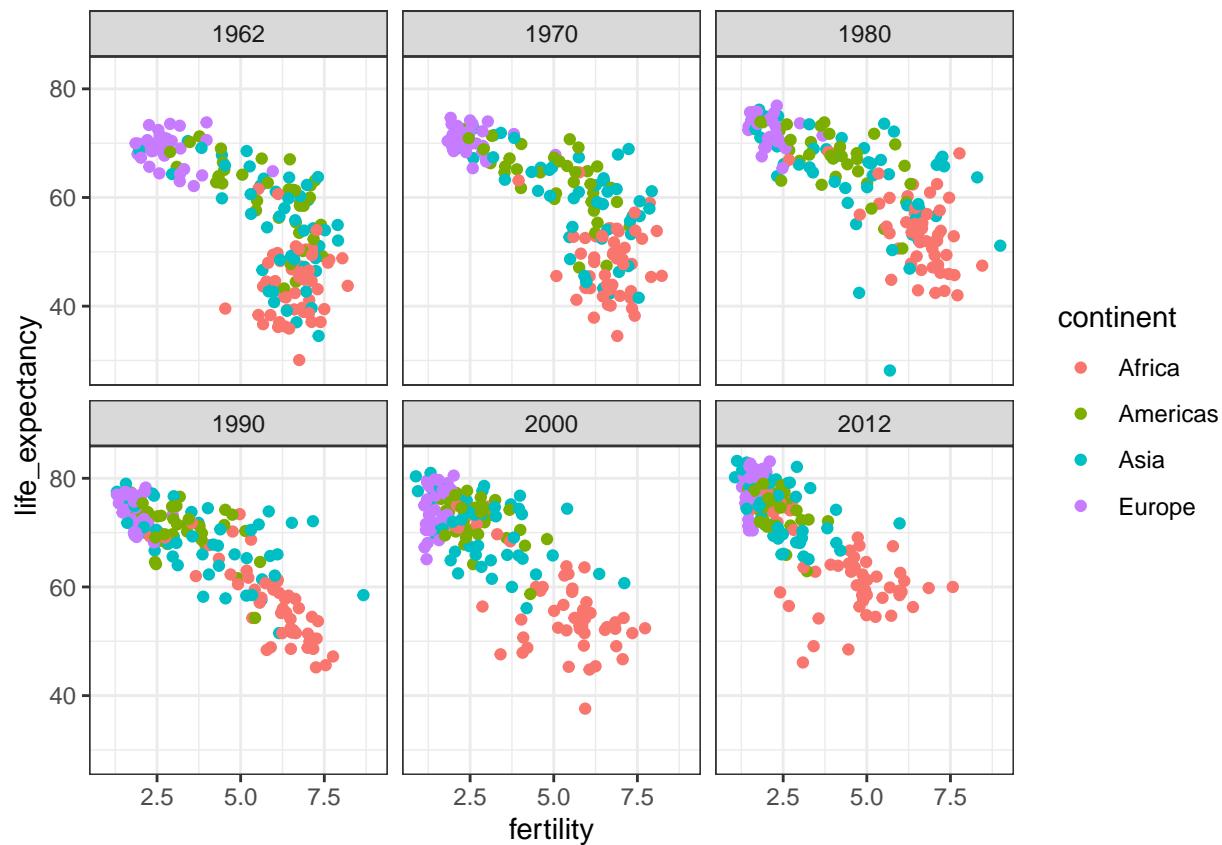


Donde puede verse que, en general, América y Asia se ha movido hacia los valores de Europa, mientras que África continúa con un comportamiento similar al de 1962.

Para analizar la evolución de esta transformación, puede hacerse el gráfico para diferentes años. Por ejemplo, supóngase que queremos analizar los años 1962, 1970, 1980, 1990, 2000 y 2012; no obstante, no queremos que todos aparezcan en la misma fila. Para esto, se utiliza la función “facet\_wrap()”

```
years <- c(1962, 1970, 1980, 1990, 2000, 2012)
continents <- c("Europe", "Asia", "Americas", "Africa")

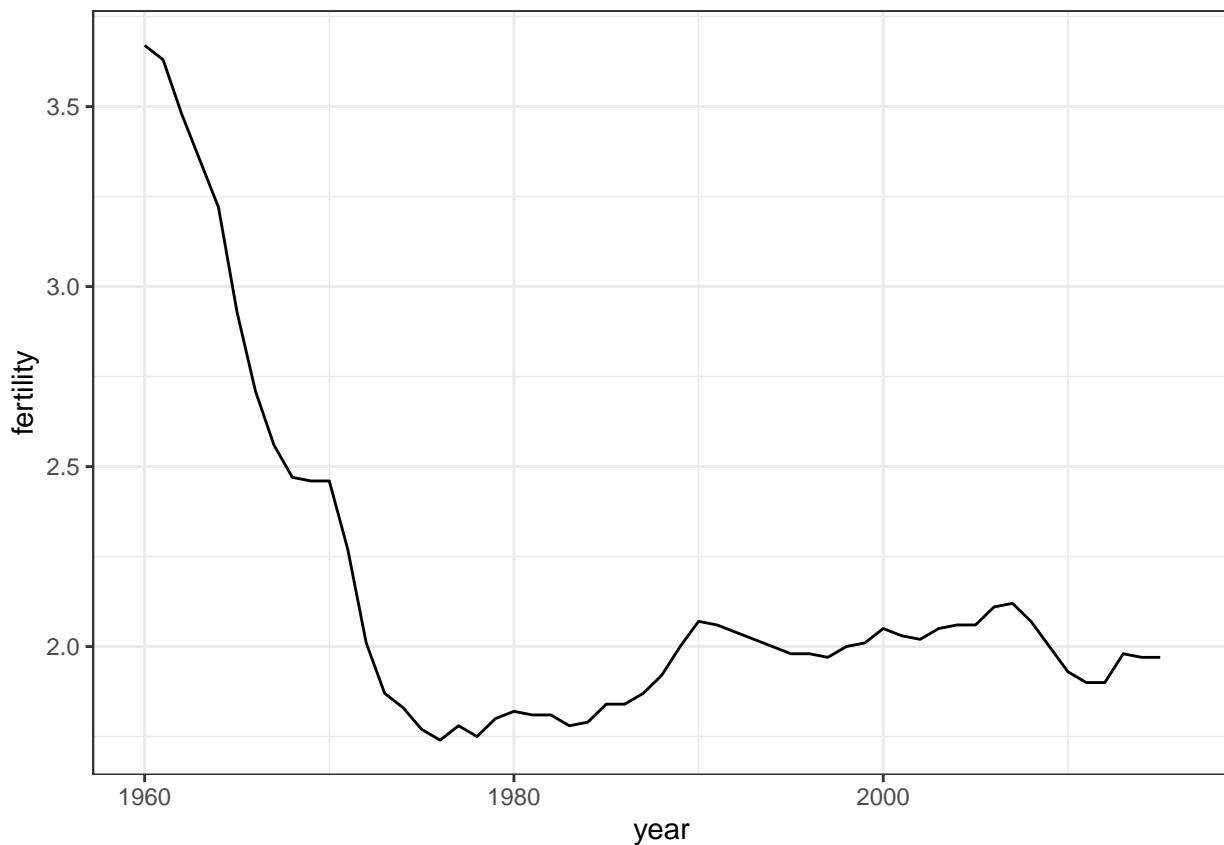
filter(gapminder, year %in% years & continent %in% continents) %>%
  ggplot(aes(fertility, life_expectancy, col = continent)) +
  geom_point() +
  facet_wrap(~year)
```



#### 2.4.2. Gráficos de series de tiempo

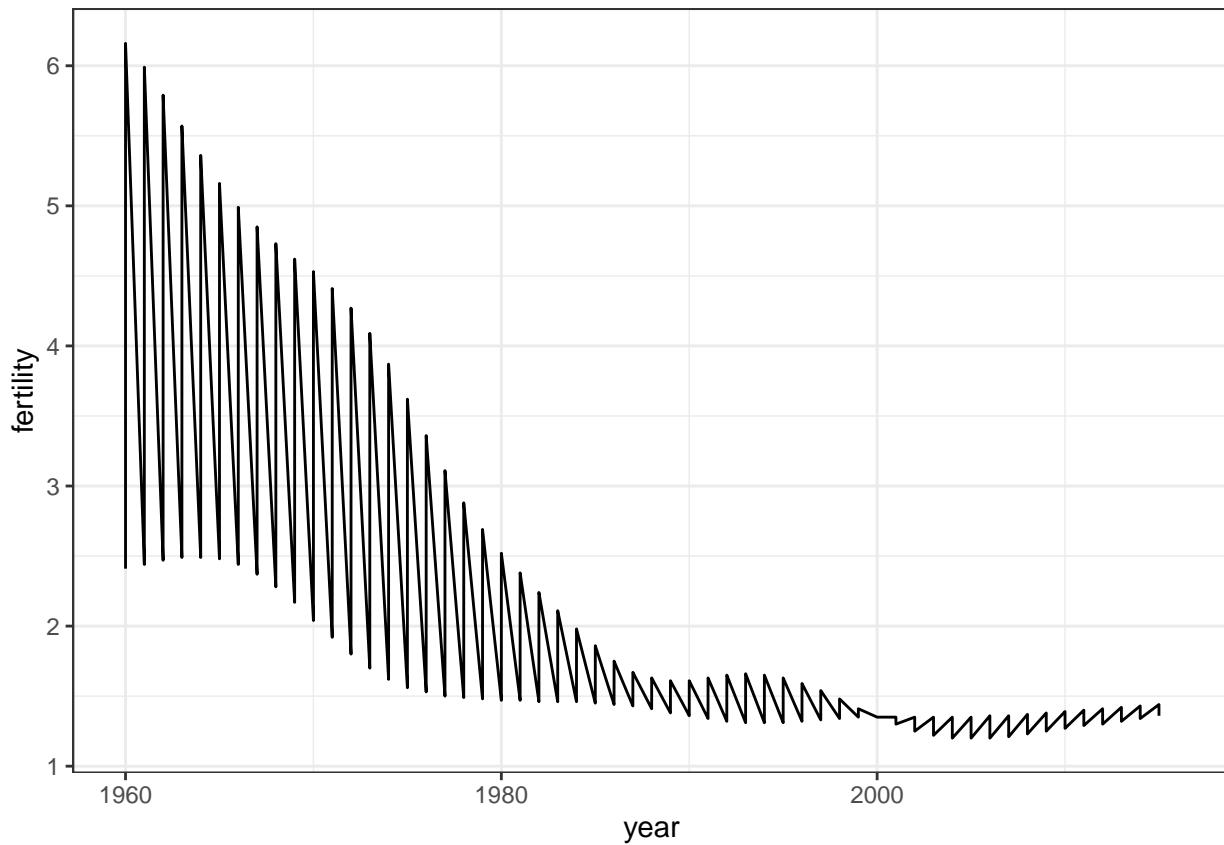
Puede analizarse con más detalle la evolución de estas variables en los países con una gráfica de series de tiempo, donde el eje x es el tiempo y el eje y la variable de interés. Por ejemplo, considérese el caso de EE.UU.:

```
gapminder %>% filter(country == "United States") %>%
  ggplot(aes(year, fertility)) +
  geom_line()
## Warning: Removed 1 row(s) containing missing values (geom_path).
```



Ahora, considérense los casos de Corea del Sur y Alemania:

```
countries <- c("South Korea", "Germany")  
  
gapminder %>% filter(country %in% countries) %>%  
  ggplot(aes(year, fertility)) +  
  geom_line()  
  
## Warning: Removed 2 row(s) containing missing values (geom_path).
```

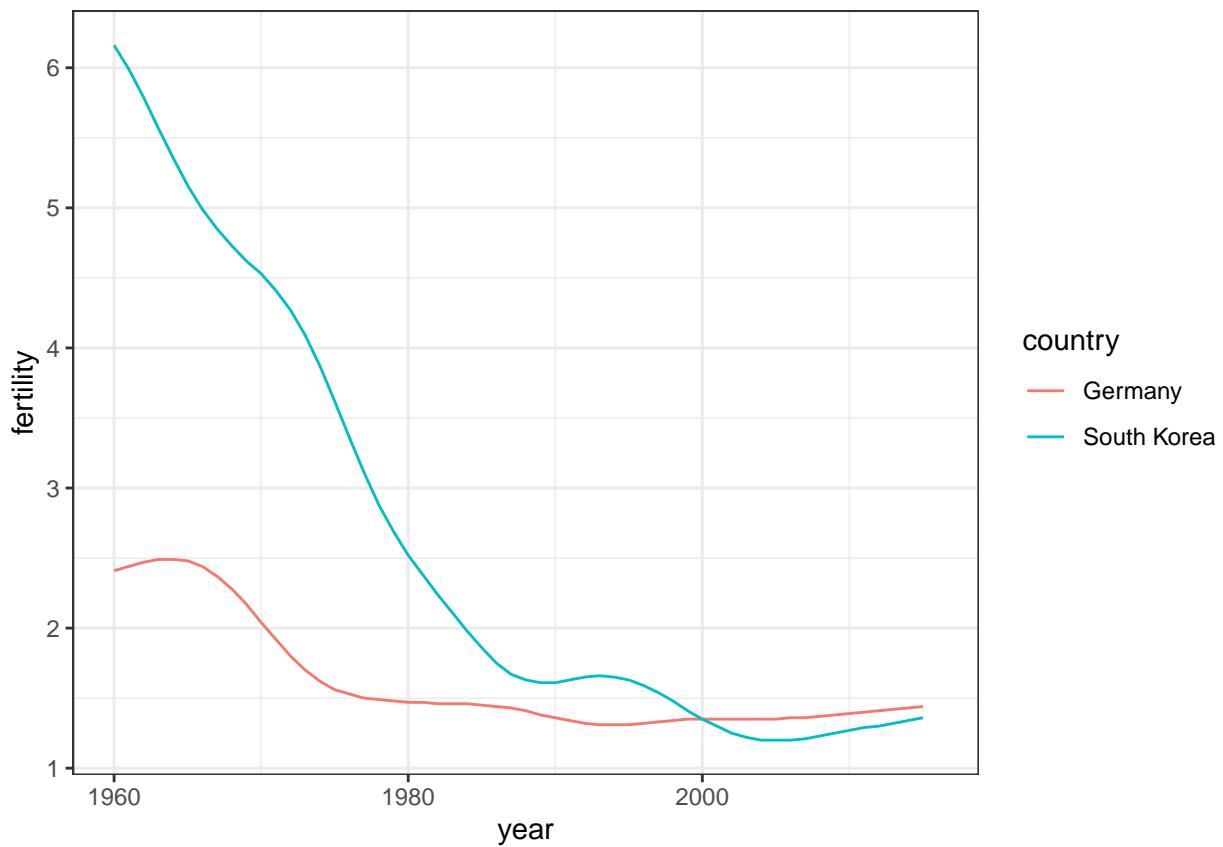


Pero esto no es lo que queremos. El gráfico anterior arroja una línea que pasa por las observaciones de ambos países, pero las queremos separadas.

```
countries <- c("South Korea", "Germany")

gapminder %>% filter(country %in% countries) %>%
  ggplot(aes(year, fertility, group = country, color = country)) +
  geom_line()

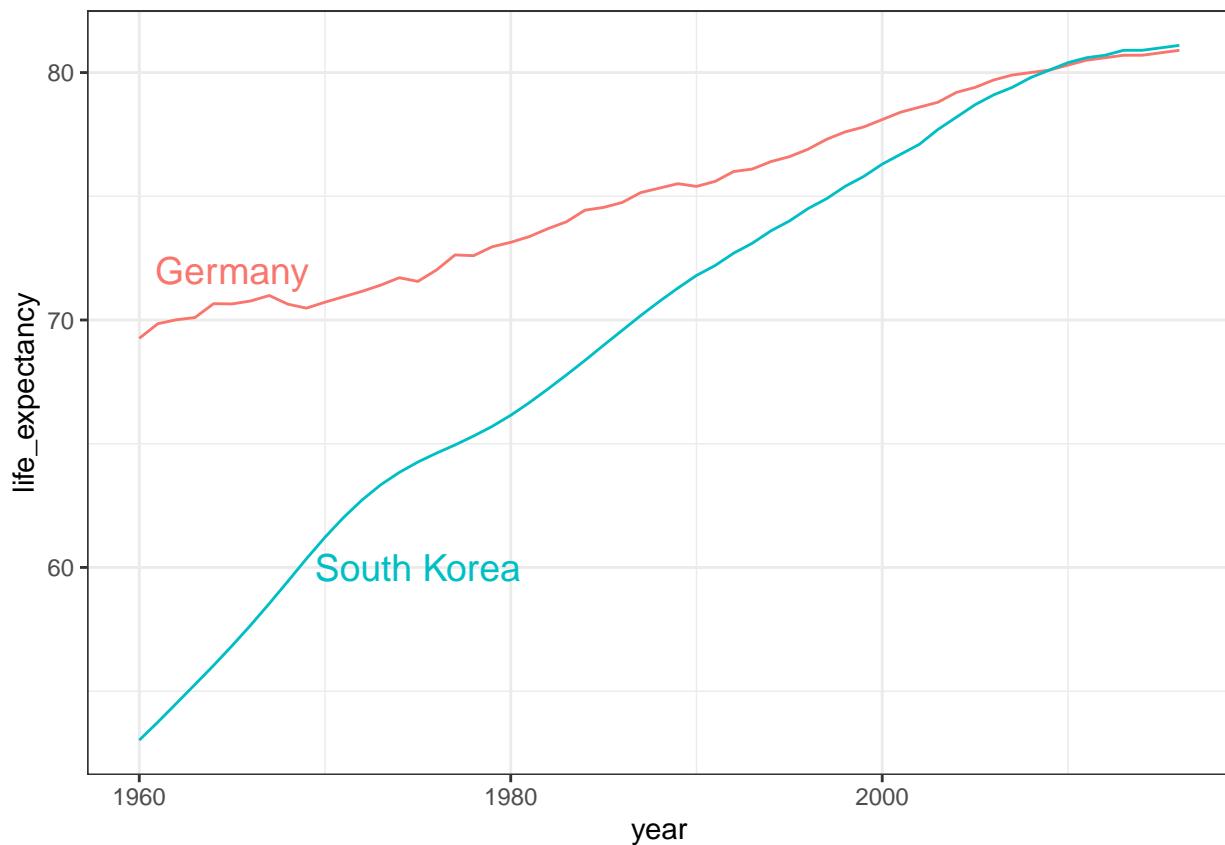
## Warning: Removed 2 row(s) containing missing values (geom_path).
```



Para agregar etiquetas a un gráfico de serie de tiempo:

```
labels <- data.frame(country = countries, x = c(1975, 1965), y = c(60, 72))

gapminder %>% filter(country %in% countries) %>%
  ggplot(aes(year, life_expectancy, col = country)) +
  geom_line() +
  geom_text(data = labels, aes(x, y, label = country), size = 5) +
  theme(legend.position = "none")
```



#### 2.4.3. Transformaciones

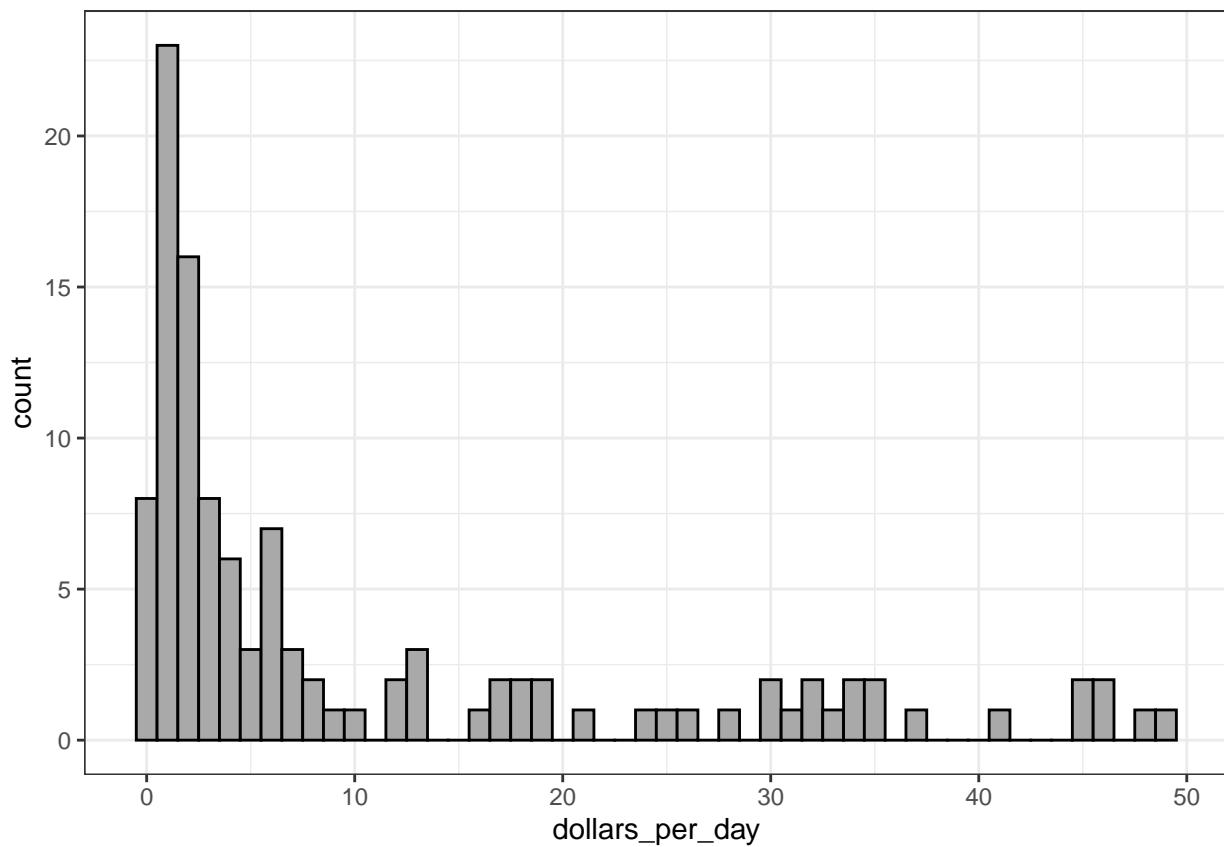
Se analizará la variable del ingreso de los países. Primero, se agregará una columna del ingreso por persona diario a la base de datos:

```
gapminder <- gapminder %>%
  mutate(dollars_per_day = gdp/population/365)
```

Graficaremos un histograma de la evolución del ingreso diario per cápita en 1970:

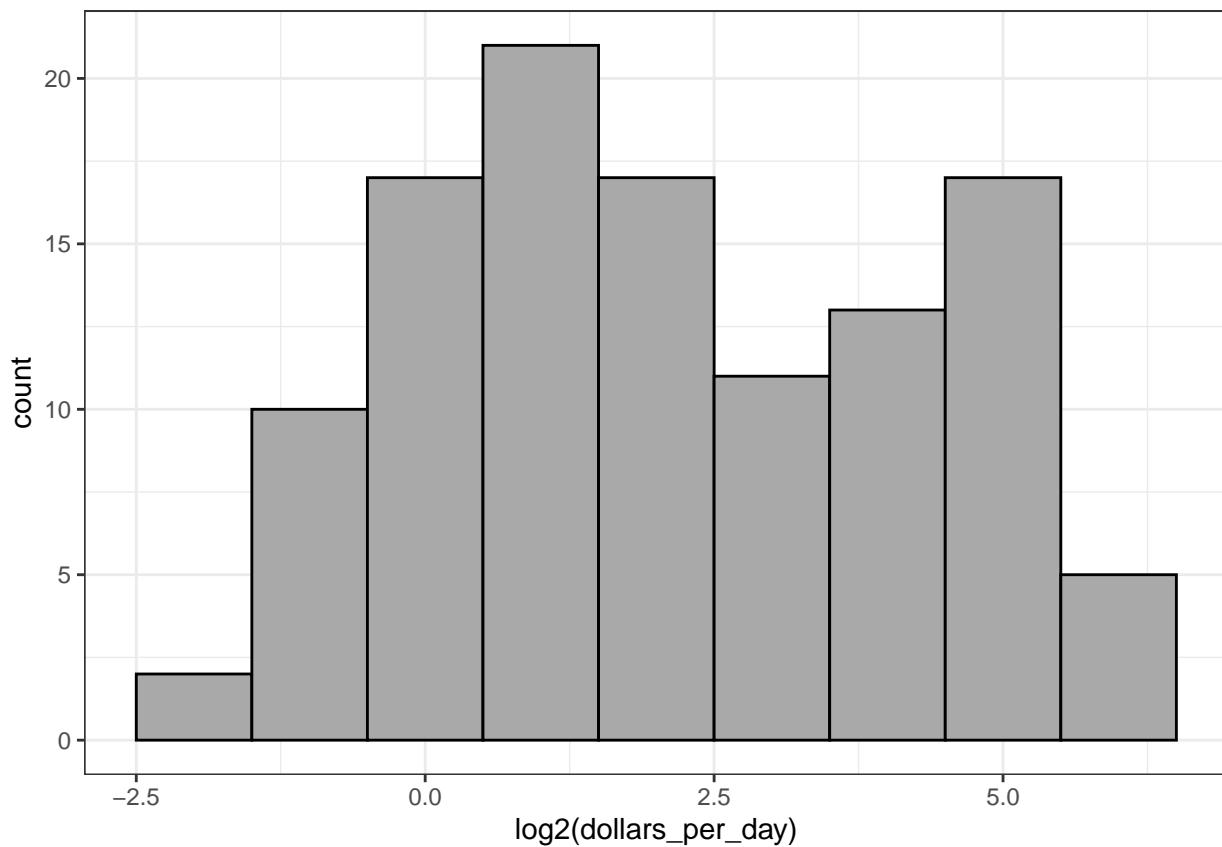
```
past_year <- 1970

gapminder %>%
  filter(year == past_year & !is.na(gdp)) %>%
  ggplot(aes(dollars_per_day)) +
  geom_histogram(binwidth = 1, fill = "darkgray", color = "black")
```



Ahora, obtendremos la distribución logarítmica base 2:

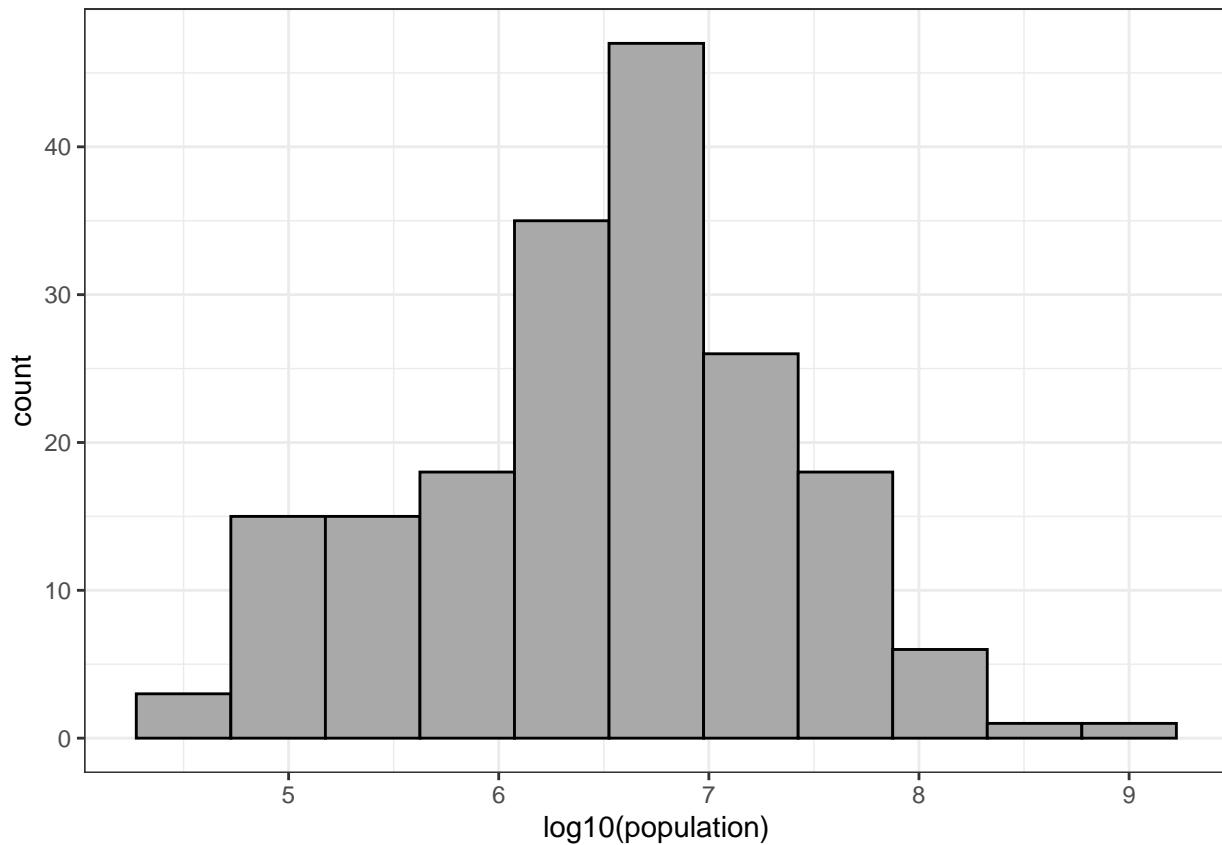
```
gapminder %>%
  filter(year == past_year & !is.na(gdp)) %>%
  ggplot(aes(log2(dollars_per_day))) +
  geom_histogram(binwidth = 1, fill = "darkgray", color = "black")
```



Donde se observa que existen dos modas locales (1 y 5 = \$2 y \$32), que representa la dicotomía de ingresos en el mundo en ese año.

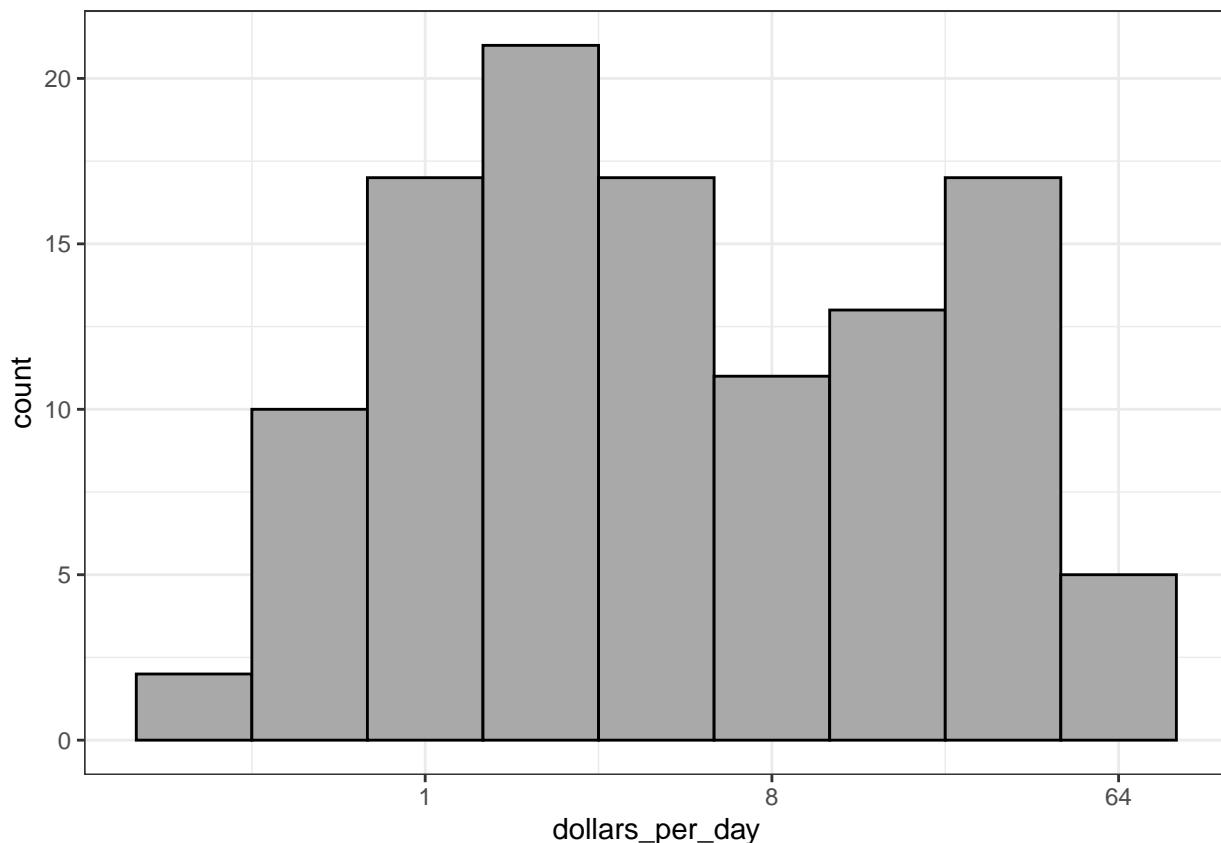
Opuestamente, al analizar la población, tiene más sentido utilizar una transformación log base 10:

```
gapminder %>%
  filter(year == past_year & !is.na(population)) %>%
  ggplot(aes(log10(population))) +
  geom_histogram(binwidth = 0.45, fill = "darkgray", color = "black")
```



Alternativamente, las transformaciones pueden realizarse en los ejes mediante la función “`scale_x_continuous()`”, lo que permite conservar la escala original en el eje x:

```
gapminder %>%
  filter(year == past_year & !is.na(gdp)) %>%
  ggplot(aes(dollars_per_day)) +
  geom_histogram(binwidth = 1, fill = "darkgray", color = "black") +
  scale_x_continuous(trans = "log2")
```

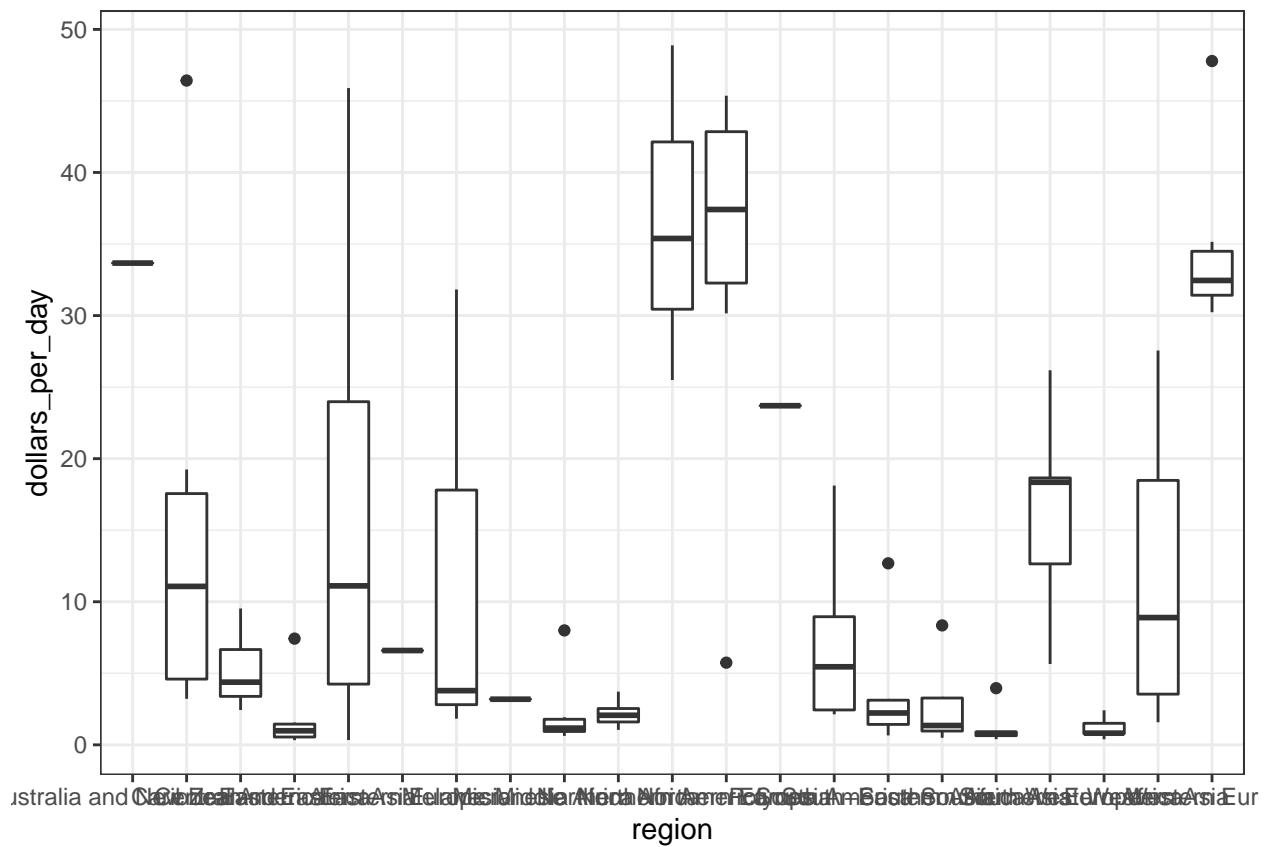


#### 2.4.4. Estratificación y boxplots

La gráfica inmediata anterior no muestra si los países son desarrollados o en vías de desarrollo. Para observar la distribución del ingreso a través de regiones, primero hay que estratificar los datos en regiones y luego examinar la distribución para cada una. Dado que la base de datos tiene muchas regiones (22), muchos histogramas o densidades suavizadas no serán útiles: en su lugar, pueden graficarse varios boxplots:

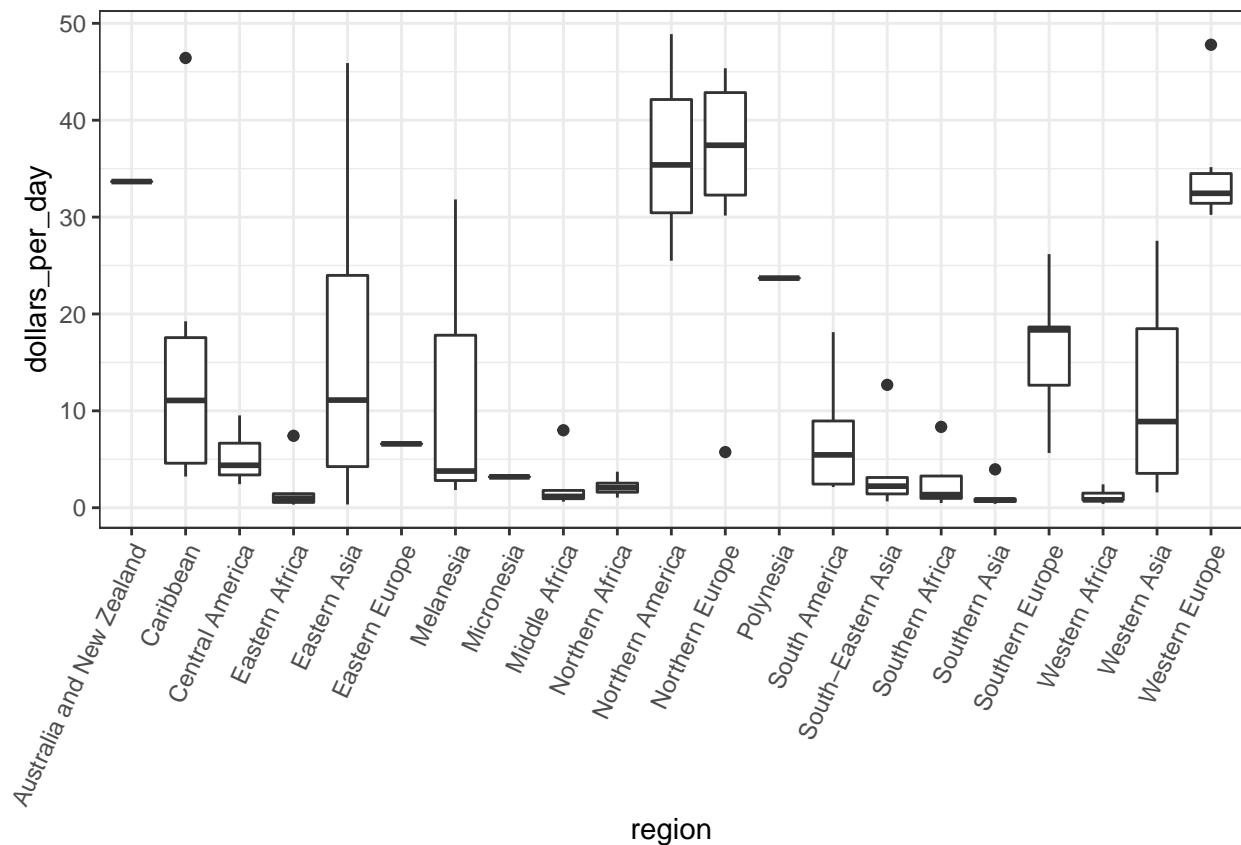
```
p <- gapminder %>%
  filter(year == past_year & !is.na(gdp)) %>%
  ggplot(aes(region, dollars_per_day)) +
  scale_color_discrete(name = "Continente")

p + geom_boxplot()
```



Para escribir las etiquetas verticalmente con el argumento “element\_text()”:

```
p + geom_boxplot()+
  theme(axis.text.x = element_text(angle = 65, hjust = 1))
```



Se pueden agregar algunos ajustes, como ordenar los países en un orden diferente al alfabético. Para esto, se utilizará la función “reorder()”. Un ejemplo sencillo, supóngase el siguiente vector de factores, cada uno asociado con valores arbitrarios; si queremos ordenar los factores con base en la media asociada a cada nivel:

```
fac <- factor(c("Asia", "Asia", "West", "West", "West"))
```

```
levels(fac)
```

```
## [1] "Asia" "West"
```

```
value <- c(10, 11, 12, 6, 4)
```

```
fac <- reorder(fac, value, FUN = mean)
```

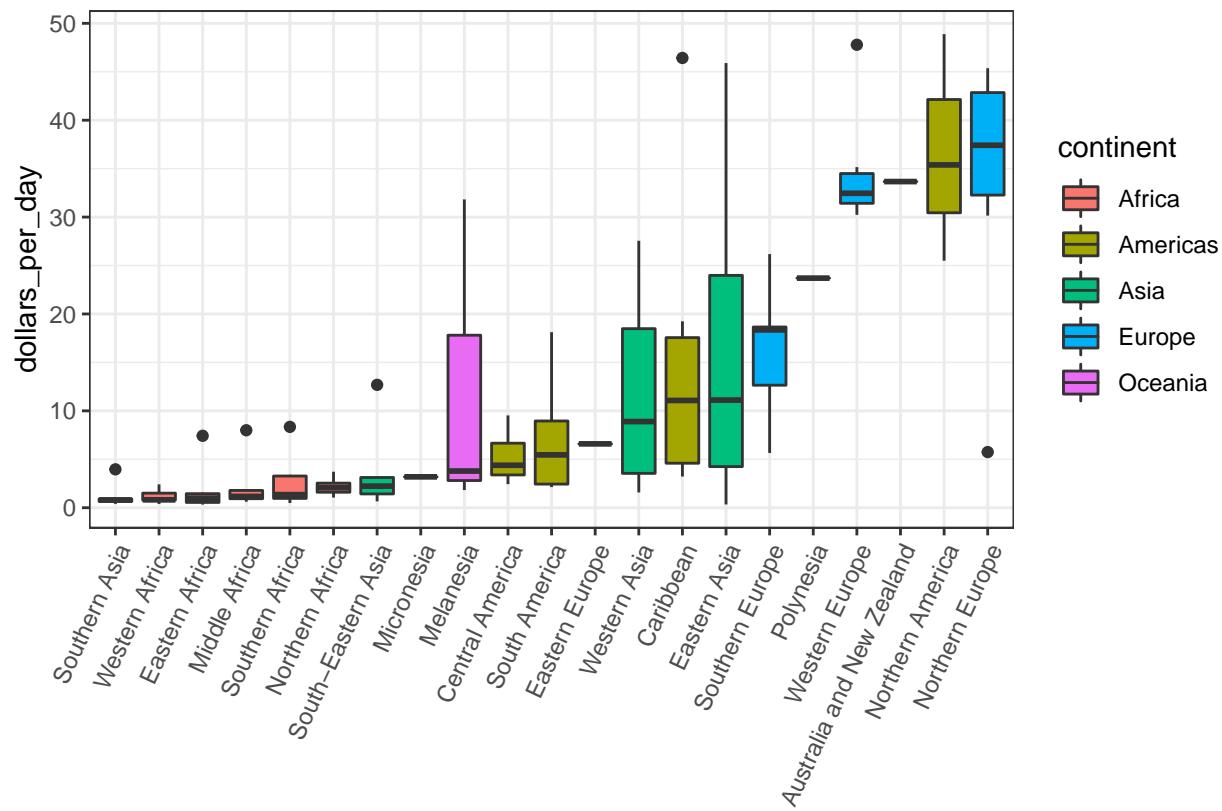
```
levels(fac)
```

```
## [1] "West" "Asia"
```

Así, aplicando lo anterior al ejemplo y reordenando los países con base en su nivel de ingreso mediano:

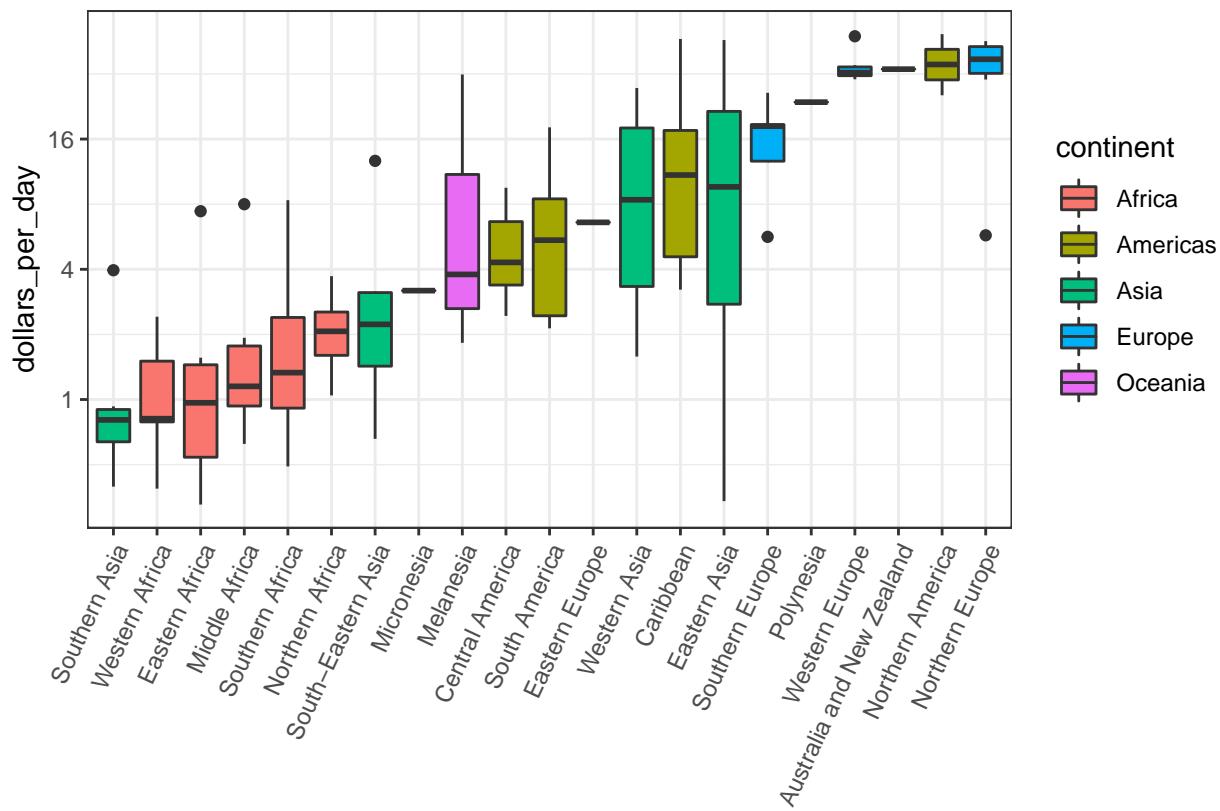
```
p <- gapminder %>%
  filter(year == past_year & !is.na(gdp)) %>%
  mutate(region = reorder(region, dollars_per_day, FUN = median)) %>%
  ggplot(aes(region, dollars_per_day, fill = continent)) +
  geom_boxplot() +
  theme(axis.text.x = element_text(angle = 65, hjust = 1)) +
  xlab("")
```

```
p
```



Aplicando una transformación logarítmica base 2, con el objetivo de visualizar de mejor manera las diferencias entre los países con menos ingreso diario per cápita:

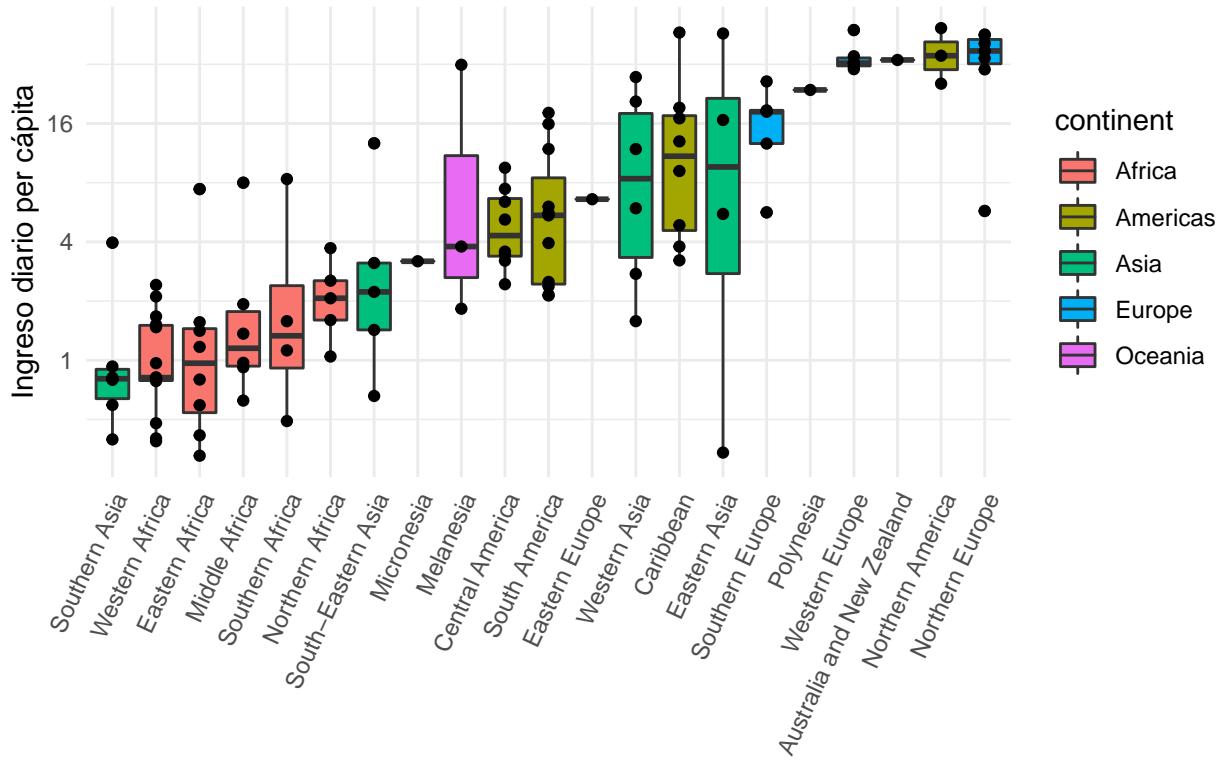
```
p + scale_y_continuous(trans = "log2")
```



Finalmente, pueden mostrarse los datos particulares, dado que no son muchos:

```
p + scale_y_continuous(trans = "log2") +
  geom_point(show.legend = FALSE) +
  ylab("Ingreso diario per cápita") +
  ggtitle("Ingreso diario per cápita por país, 1970, en dólares.") +
  theme_minimal() +
  theme(plot.title = element_text(hjust=0.5)) +
  theme(axis.text.x = element_text(angle = 65, hjust = 1))
```

### Ingreso diario per cápita por país, 1970, en dólares.

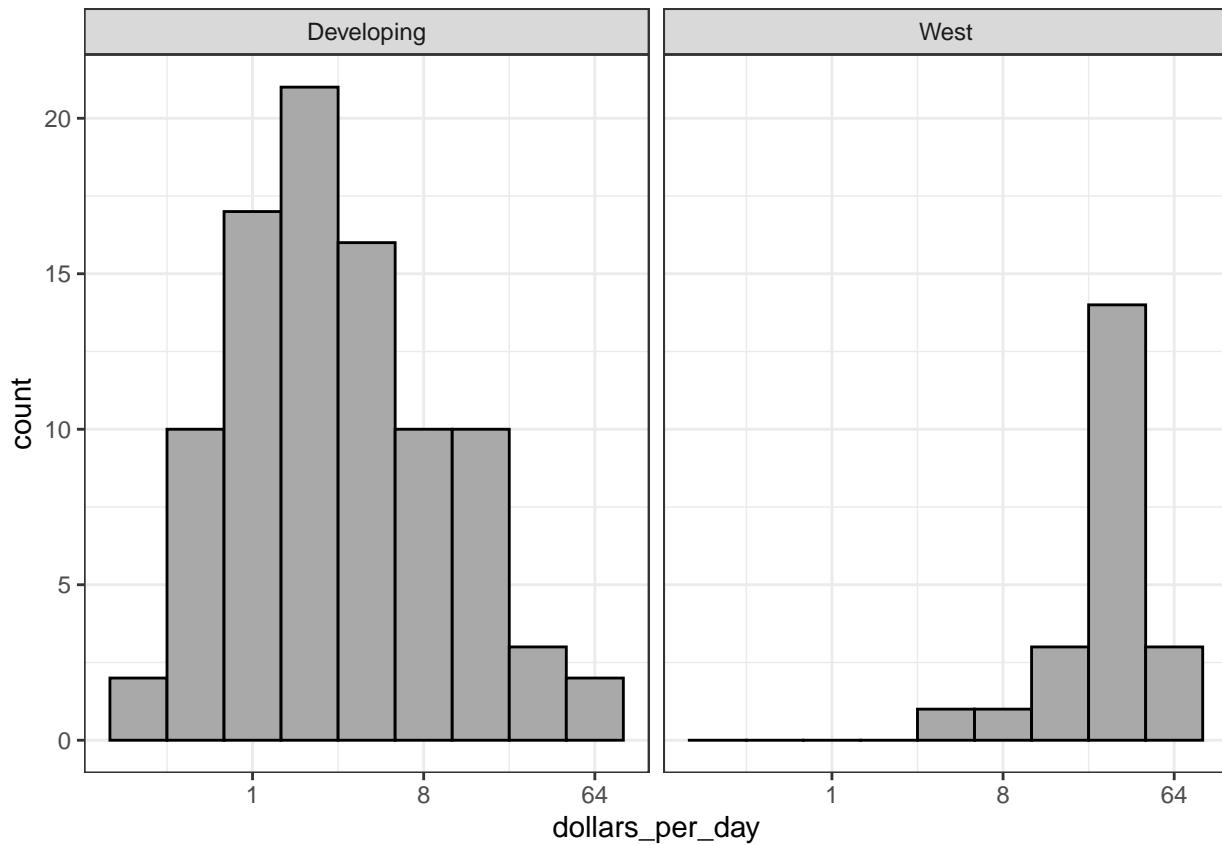


#### 2.4.5. Comparando distribuciones

Defínase un vector con las regiones en el Occidente, con el objetivo de comparar sus ingresos en 1970 con el resto del mundo:

```
west <- c("Western Europe", "Northern Europe", "Southern Europe", "Northern America", "Australia and New Zealand")

gapminder %>%
  filter(year == past_year & !is.na(gdp)) %>%
  mutate(group = ifelse(region %in% west, "West", "Developing")) %>%
  ggplot(aes(dollars_per_day)) +
  geom_histogram(binwidth = 1, fill = "darkgray", color = "black") +
  scale_x_continuous(trans = "log2") +
  facet_grid(.~group)
```

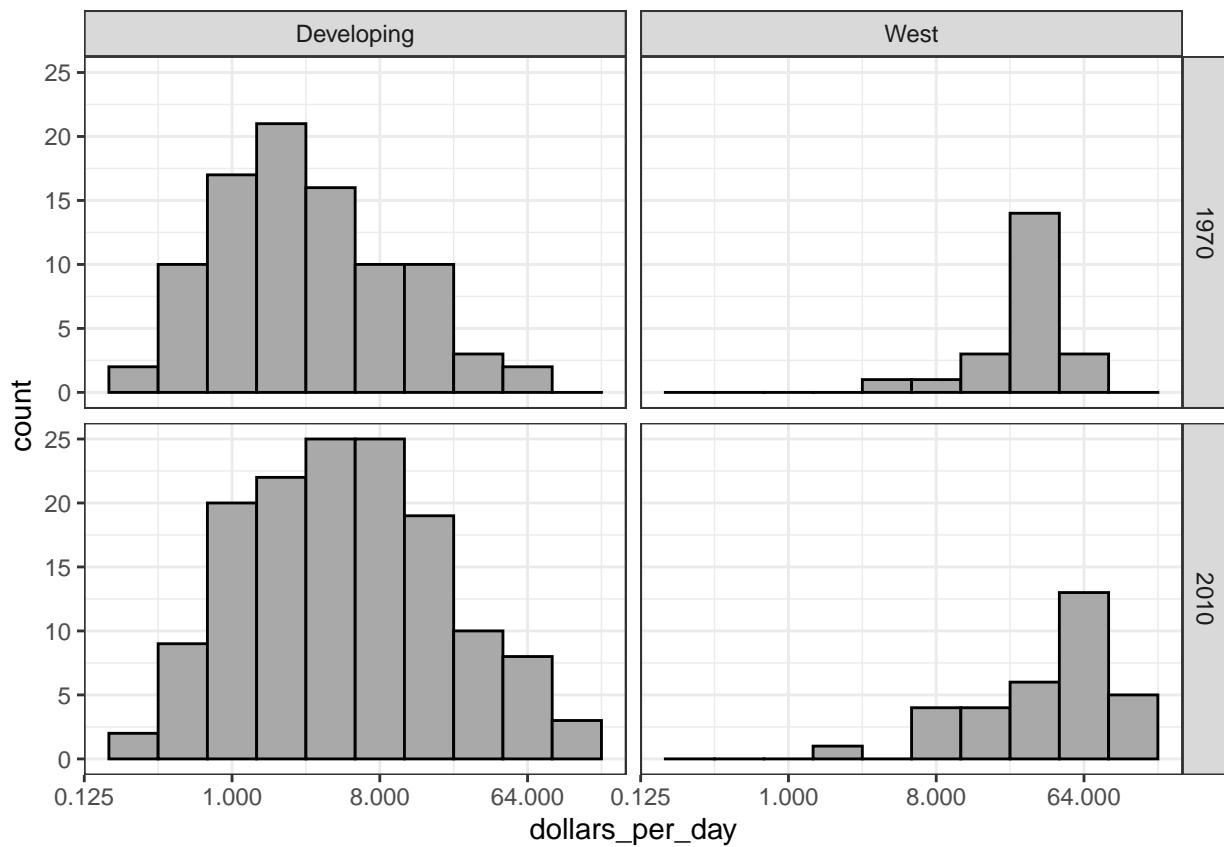


Ahora, hay que comparar lo anterior con la situación de 2010, facetando simultáneamente por región y año:

```
past_year <- 1970
```

```
present_year <- 2010
```

```
gapminder %>%
  filter(year %in% c(past_year, present_year) & !is.na(gdp)) %>%
  mutate(group = ifelse(region %in% west, "West", "Developing")) %>%
  ggplot(aes(dollars_per_day)) +
  geom_histogram(binwidth = 1, fill = "darkgray", color = "black") +
  scale_x_continuous(trans = "log2") +
  facet_grid(year ~ group)
```



Es pertinente apuntar que existen más observaciones para 2010 que para 1970. Así, volveremos a hacer el gráfico pero solo para aquellos países con observaciones tanto en 1970 como en 2010. Primero, hay que obtener la lista de estos países, mediante la función “intersect()”:

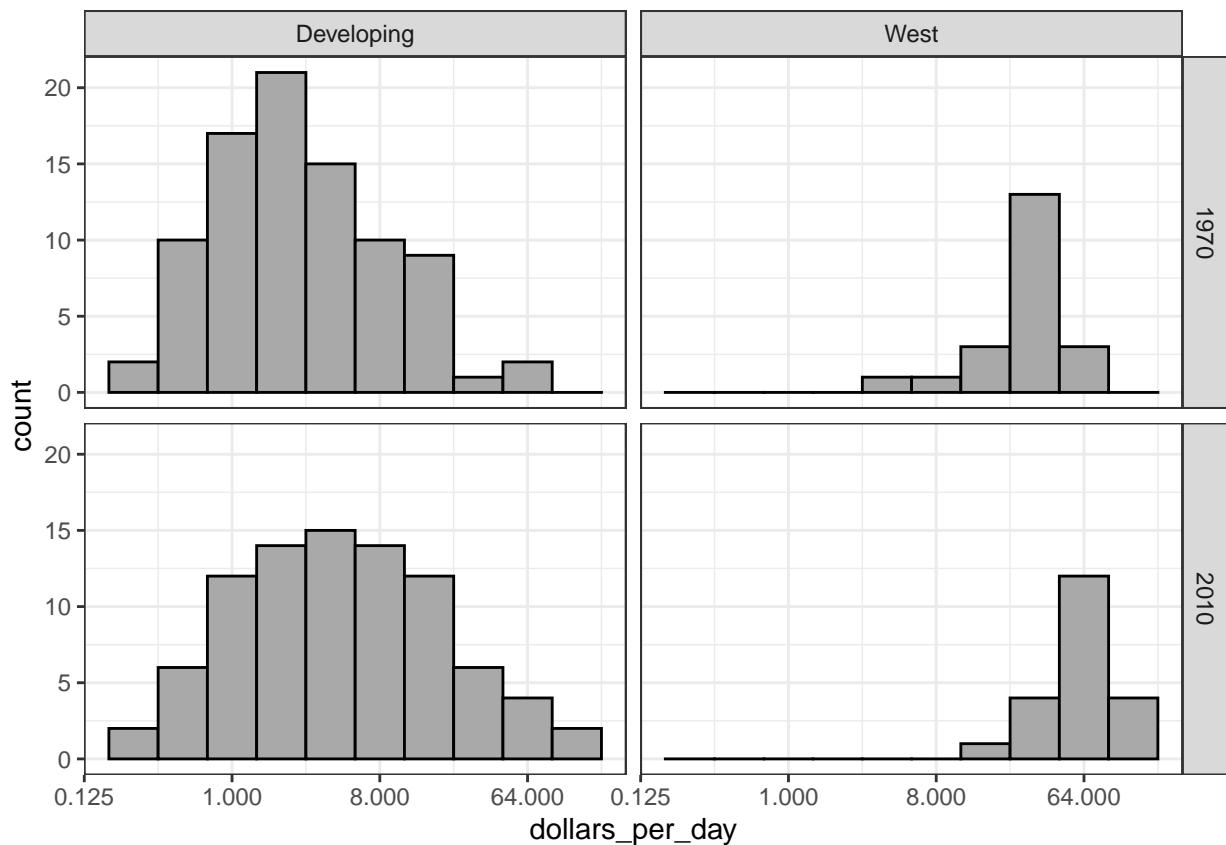
```
country_list_1 <- gapminder %>%
  filter(year == past_year & !is.na(dollars_per_day)) %>% .$country

country_list_2 <- gapminder %>%
  filter(year == present_year & !is.na(dollars_per_day)) %>% .$country

country_list <- intersect(country_list_1, country_list_2)
```

Así, el gráfico resultante es:

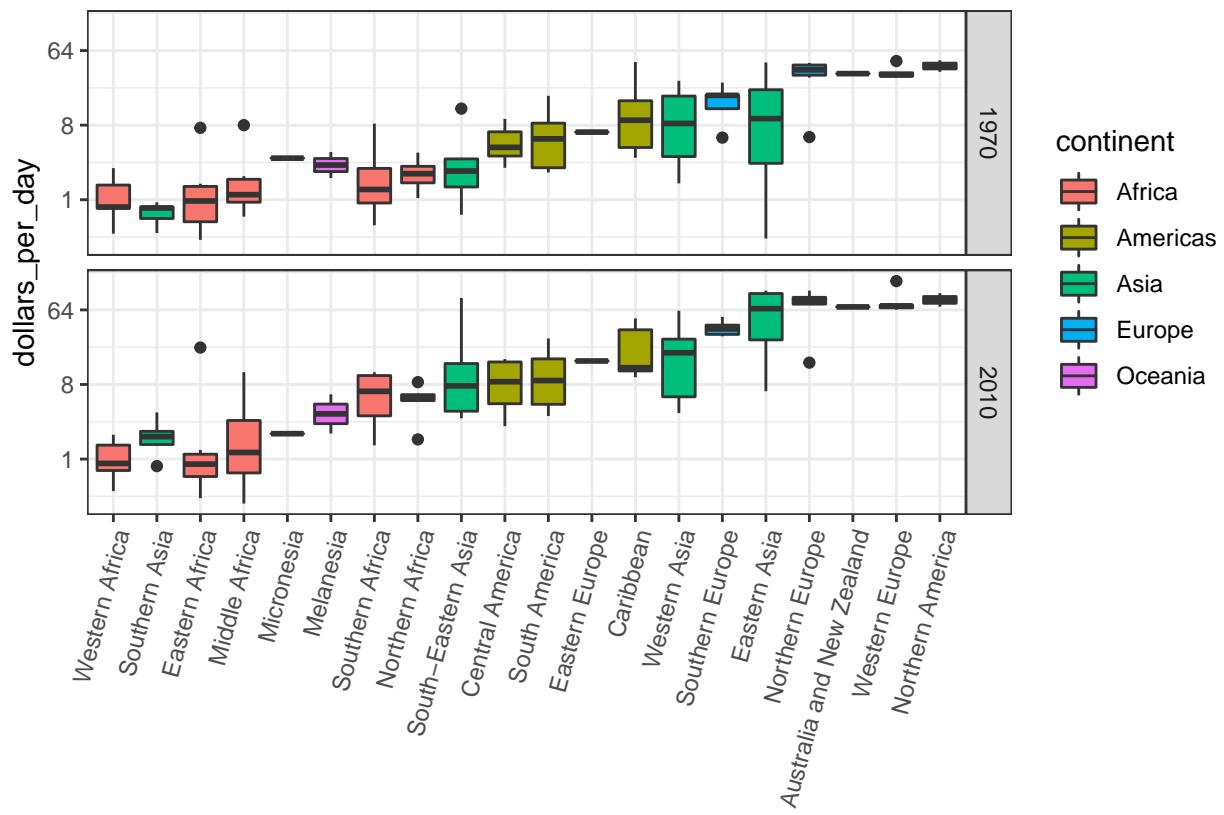
```
gapminder %>%
  filter(year %in% c(past_year, present_year) & country %in% country_list) %>%
  mutate(group = ifelse(region %in% west, "West", "Developing")) %>%
  ggplot(aes(dollars_per_day)) +
  geom_histogram(binwidth = 1, fill = "darkgray", color = "black") +
  scale_x_continuous(trans = "log2") +
  facet_grid(year ~ group)
```



Para obtener las regiones que han mejorado más, pueden volverse a hacer los boxplots anteriores, pero ahora agregando 2010:

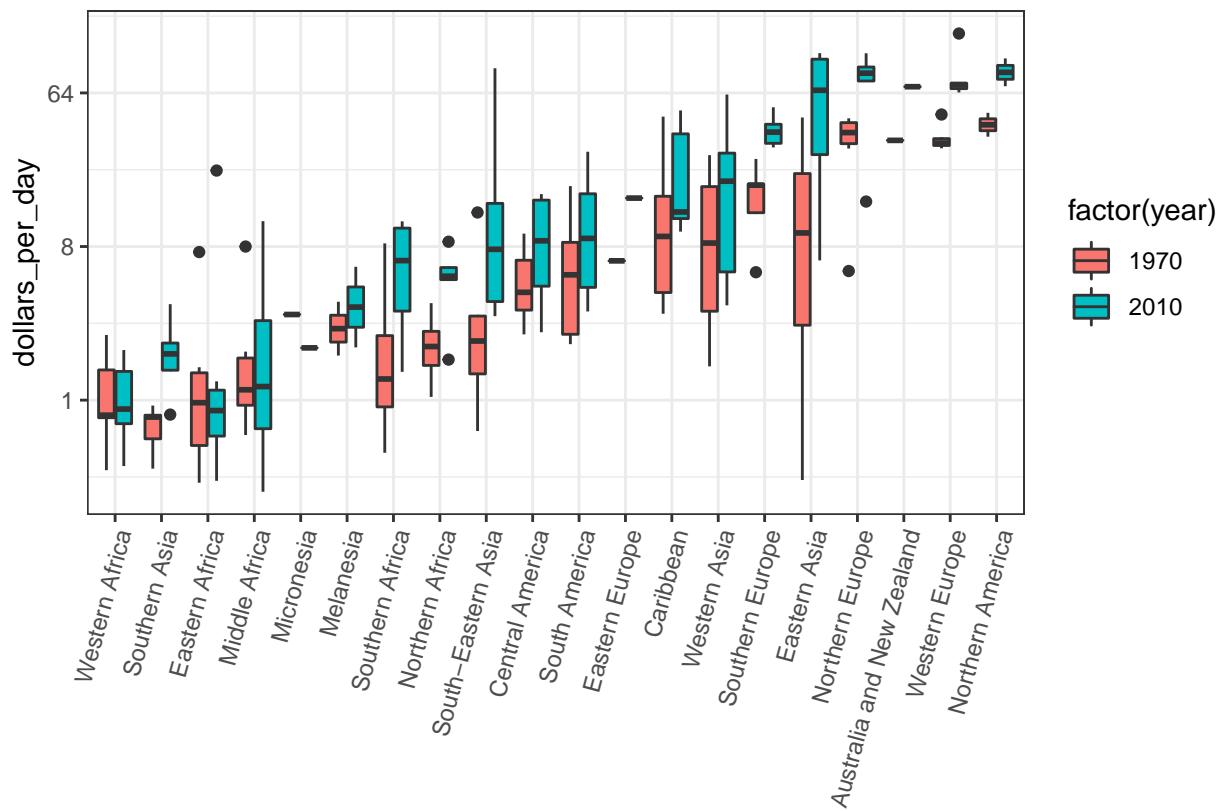
```
p <- gapminder %>%
  filter(year %in% c(past_year, present_year) & country %in% country_list) %>%
  mutate(region = reorder(region, dollars_per_day, FUN = median)) %>%
  ggplot() +
  theme(axis.text.x = element_text(angle = 75, hjust = 1)) +
  xlab("") +
  scale_y_continuous(trans = "log2")

p + geom_boxplot(aes(region, dollars_per_day, fill = continent)) +
  facet_grid(year~.)
```



Si se quiere compararlas horizontalmente en una misma gráfica:

```
p + geom_boxplot(aes(region, dollars_per_day, fill = factor(year)))
```



#### 2.4.6. Gráficos de densidad

Primero, hay que apuntar que el número de países en vías de desarrollo es sustantivamente mayor al de los Occidentales:

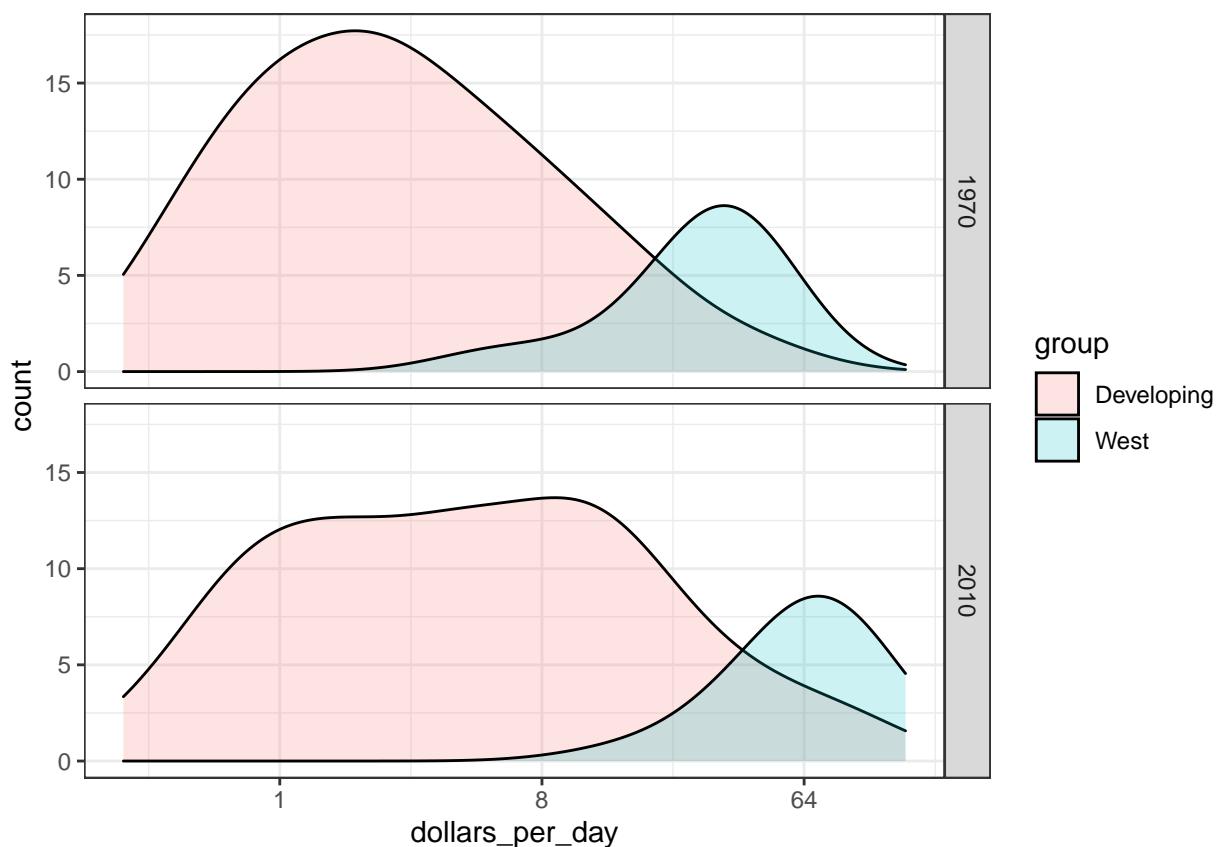
```
gapminder %>%
  filter(year == past_year & country %in% country_list) %>%
  mutate(group = ifelse(region %in% west, "West", "Developing")) %>% group_by(group) %>%
  summarize(n = n()) %>% knitr::kable()
```

group	n
Developing	87
West	21

Esto puede no ser conveniente, dado que las gráficas de densidad representan un área que suma 1, independientemente del número de observaciones. Así, para que estas áreas sean proporcionales al tamaño de los grupos se usa el argumento “count()”:

```
p <- gapminder %>%
  filter(year %in% c(past_year, present_year) & country %in% country_list) %>%
  mutate(group = ifelse(region %in% west, "West", "Developing")) %>%
  ggplot(aes(dollars_per_day, y = ..count.., fill = group)) +
  scale_x_continuous(trans = "log2")

p + geom_density(alpha = 0.2, bw = 0.75) + facet_grid(year~.)
```



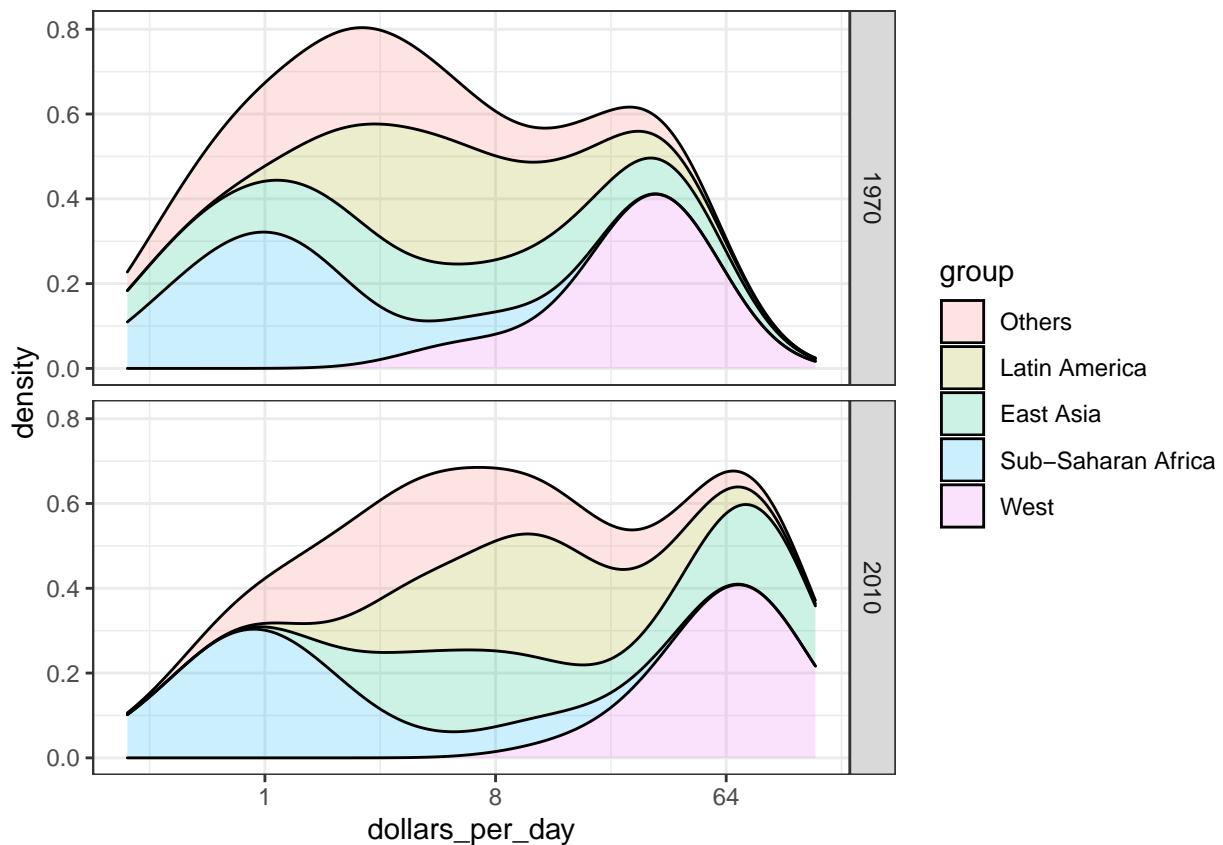
Puede verse que la densidad de los países en vías de desarrollo en 2010 es multimodal, principalmente debido a que muchos países asiáticos mejoraron sus niveles de ingresos. Así, el gráfico puede alterarse para mostrar regiones de manera separada con la función “case\_when()”. Primero, creamos los grupos pertinentes y después los transformamos en factores con el orden deseado:

```
gapminder <- gapminder %>%
  mutate(group = case_when(
    .$region %in% west ~ "West",
    .$region %in% c("Eastern Asia", "South-Eastern Asia") ~ "East Asia",
    .$region %in% c("Caribbean", "Central America", "South America") ~ "Latin America",
    .$continent == "Africa" &
    .$region != "Northern Africa" ~ "Sub-Saharan Africa", TRUE ~ "Others"))

gapminder <- gapminder %>%
  mutate(group = factor(group,
                        levels = c("Others", "Latin America", "East Asia", "Sub-Saharan Africa", "West")))

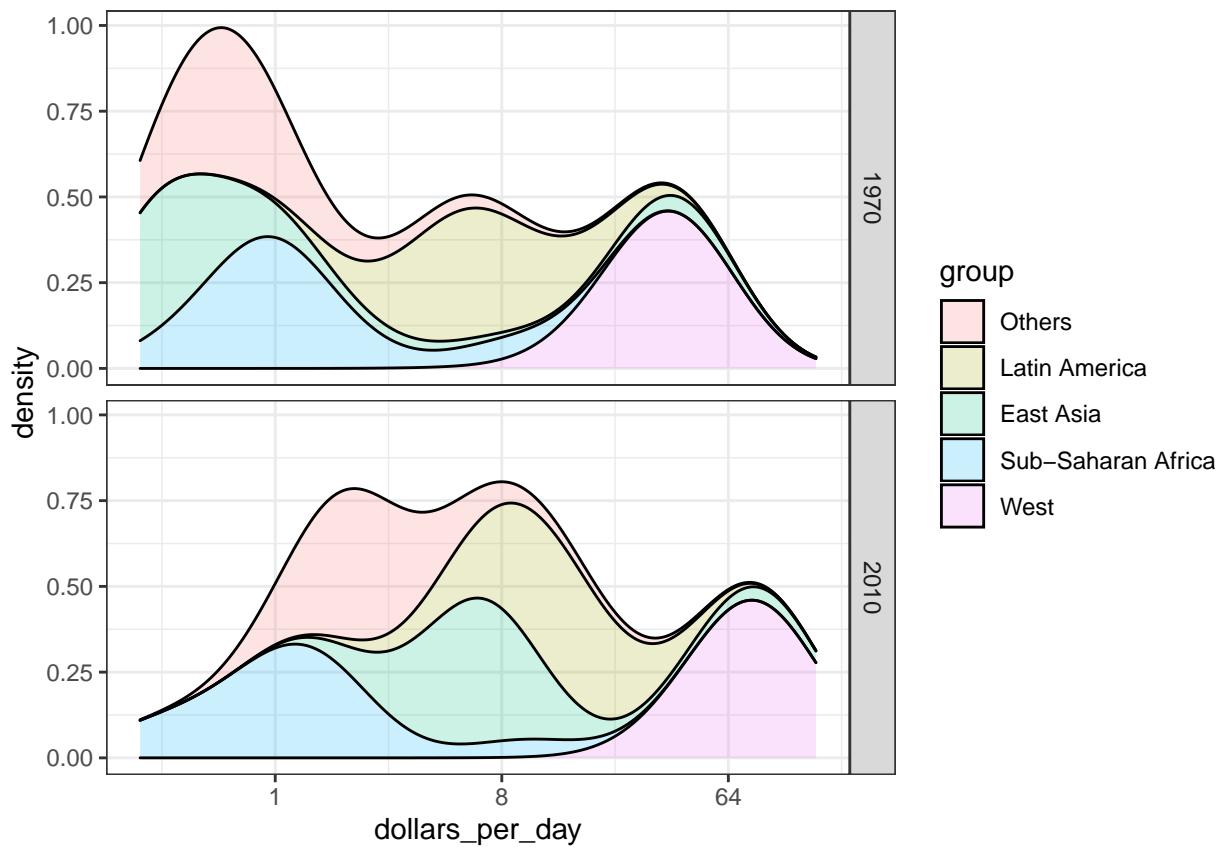
p <- gapminder %>%
  filter(year %in% c(past_year, present_year) & country %in% country_list) %>%
  ggplot(aes(dollars_per_day, fill = group)) +
  scale_x_continuous(trans = "log2")

p + geom_density(alpha = 0.2, bw = 0.75, position = "stack") +
  facet_grid(year~.)
```



Finalmente, pueden ponderarse las densidades mediante el argumento “weight” en el mapeado:

```
gapminder %>%
  filter(year %in% c(past_year, present_year) & country %in% country_list) %>%
  group_by(year) %>%
  mutate(weight = population/sum(population)*2) %>%
  ungroup() %>%
  ggplot(aes(dollars_per_day, fill = group, weight = weight)) +
  scale_x_continuous(trans = "log2") +
  geom_density(alpha = 0.2, bw = 0.75, position = "stack") +
  facet_grid(year~.)
```



#### 2.4.7. Falacia ecológica

Analizaremos la relación entre la tasa de supervivencia infantil y el ingreso promedio por país. Para ello, se compararán varias regiones, por lo que es necesario definirlas primero:

```
gapminder <- gapminder %>%
  mutate(group = case_when(
    .region %in% west ~ "The West",
    .region %in% "Northern Africa" ~ "Northern Africa",
    .region %in% c("Eastern Asia", "South-Eastern Asia") ~ "East Asia",
    .region == "Southern Asia" ~ "Southern Asia",
    .region %in% c("Central America", "South America", "Caribbean") ~ "Latin America",
    .continent == "Africa" & .region != "Northern Africa" ~ "Sub-Saharan Africa",
    .region %in% c("Melanesia", "Micronesia", "Polynesia") ~ "Pacific Islands"))

surv_income <- gapminder %>%
  filter(year %in% present_year & !is.na(gdp) & !is.na(infant_mortality) & !is.na(group)) %>%
  group_by(group) %>%
  summarize(income = sum(gdp)/sum(population)/365,
            infant_survival_rate = 1-sum(infant_mortality/1000*population)/sum(population))
surv_income %>% arrange(income)

## # A tibble: 7 x 3
##   group           income infant_survival_rate
##   <chr>        <dbl>             <dbl>
## 1 Sub-Saharan Africa  1.76            0.936
## 2 Southern Asia     2.07            0.952
```

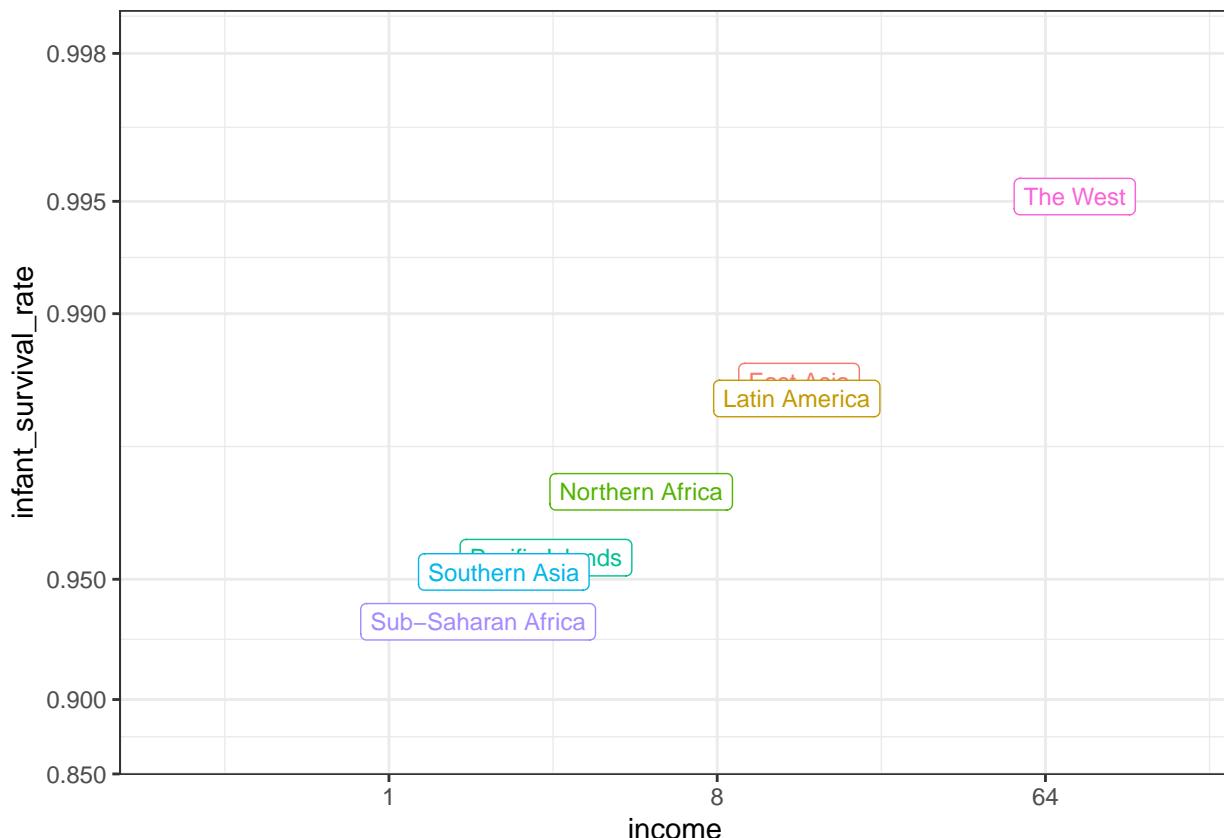
```
## 3 Pacific Islands      2.70      0.956
## 4 Northern Africa     4.94      0.970
## 5 Latin America        13.2      0.983
## 6 East Asia            13.4      0.985
## 7 The West              77.1      0.995
```

Así, podemos graficar las diferentes tasas de supervivencia infantil por regiones. Nótese que se utilizó una **transformación logística (logit)**: para una tasa proporcional  $p$ , se define:

$$f(p) = \log\left(\frac{p}{1-p}\right)$$

$f(p)$  puede entenderse como las “probabilidades” de que un niño sobreviva y es útil para analizar cantidades cercanas a 0 o 1.

```
surv_income %>% ggplot(aes(income, infant_survival_rate, label = group, color = group)) +
  scale_x_continuous(trans = "log2", limit = c(0.25, 150)) +
  scale_y_continuous(trans = "logit", limit = c(0.875, 0.9981),
                     breaks = c(0.85, 0.90, 0.95, 0.99, 0.995, 0.998)) +
  geom_label(size = 3, show.legend = FALSE)
```



¿Se puede concluir que un país con bajos ingresos tiene una tasa de supervivencia infantil baja? Asumir que todas las tasas son más bajas en África Sub-Sahariana que en Asia Sudoriental, que en las Islas del Pacífico, etc. es llamada una falacia ecológica. Esto es, la relación casi perfecta que se observa es solo válida para los promedios regionales; en lo particular, los datos presentan mucho mayor variabilidad.

## 2.5. Principios de visualización de datos

```
library(dslabs)
library(ggplot2)
library(dplyr)
library(ggthemes)
library(ggrepel)
library(knitr)
library(tidyverse)
library(RColorBrewer)
data("murders")
data("heights")
```

### 2.5.1. Codificar datos con señales visuales

Algunas opciones para configurar estas señales son:

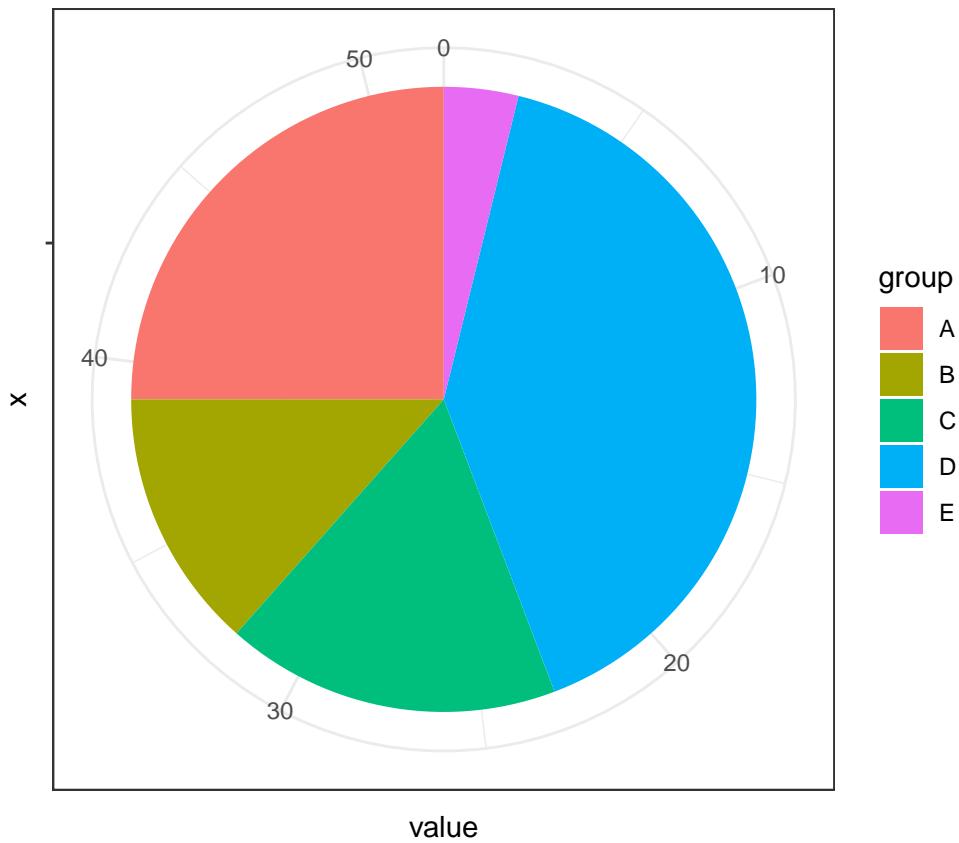
- Posición
- Alineación
- Ángulos
- Áreas
- Brillo
- Colores

Supóngase que se quieren reportar los resultados de una encuesta que pregunta qué navegador usa la gente y realizada en 2000 y 2015.

```
data <- data.frame(
  group = LETTERS[1:5],
  value = c(13, 7, 9, 21, 2)

)

ggplot(data, aes(x = "", y = value, fill = group)) +
  geom_bar(stat = "identity", width = 1) +
  coord_polar("y", start =0)
```



En general, las gráficas de pastel o de dona no son recomendables, dado que no es fácil calcular el área o ángulos que se presentan. Como alternativa, es preferible utilizar gráficas de barras si se quiere graficar cantidades.

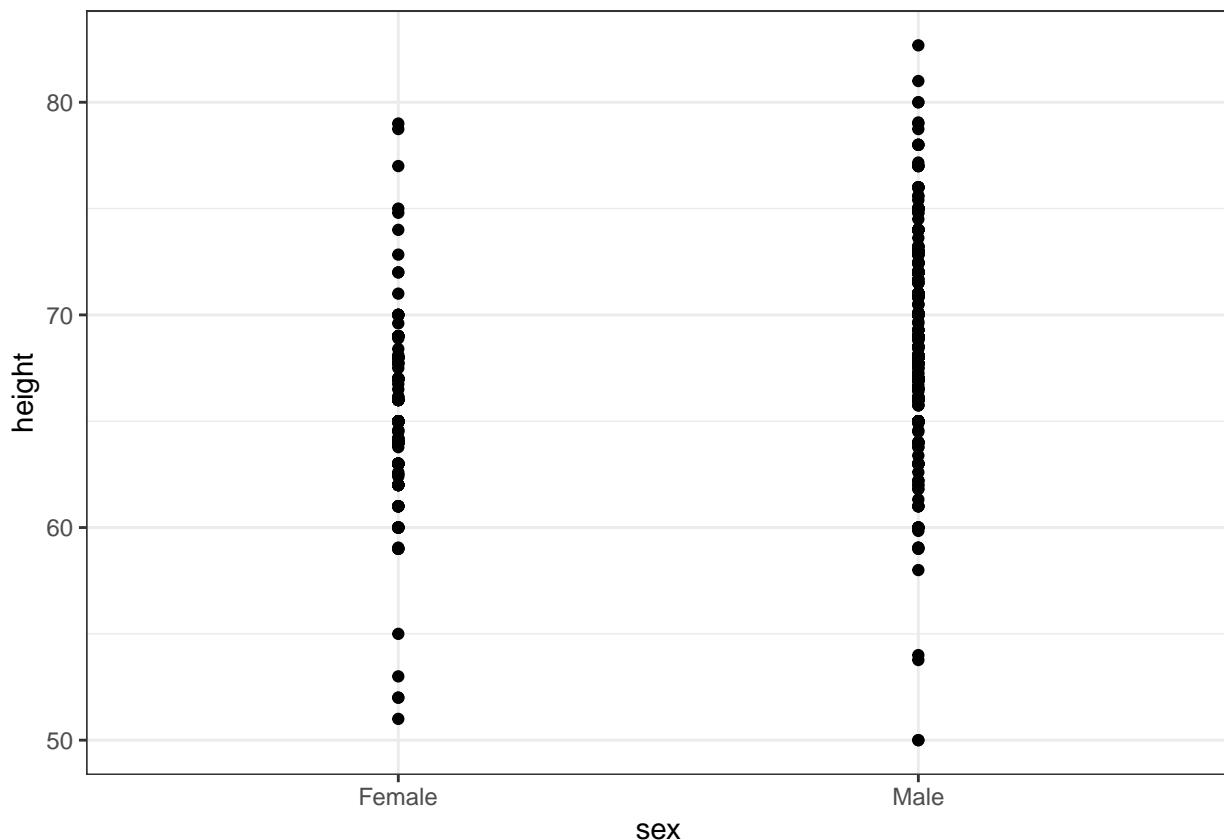
### 2.5.2. Cuándo incluir 0

Si no se agrega el 0 e una gráfica, la interpretación de esta puede ser manipulada para aparentar diferencias entre cantidades más grandes o pequeñas de lo que en realidad son.

### 2.5.3. Mostrando los datos

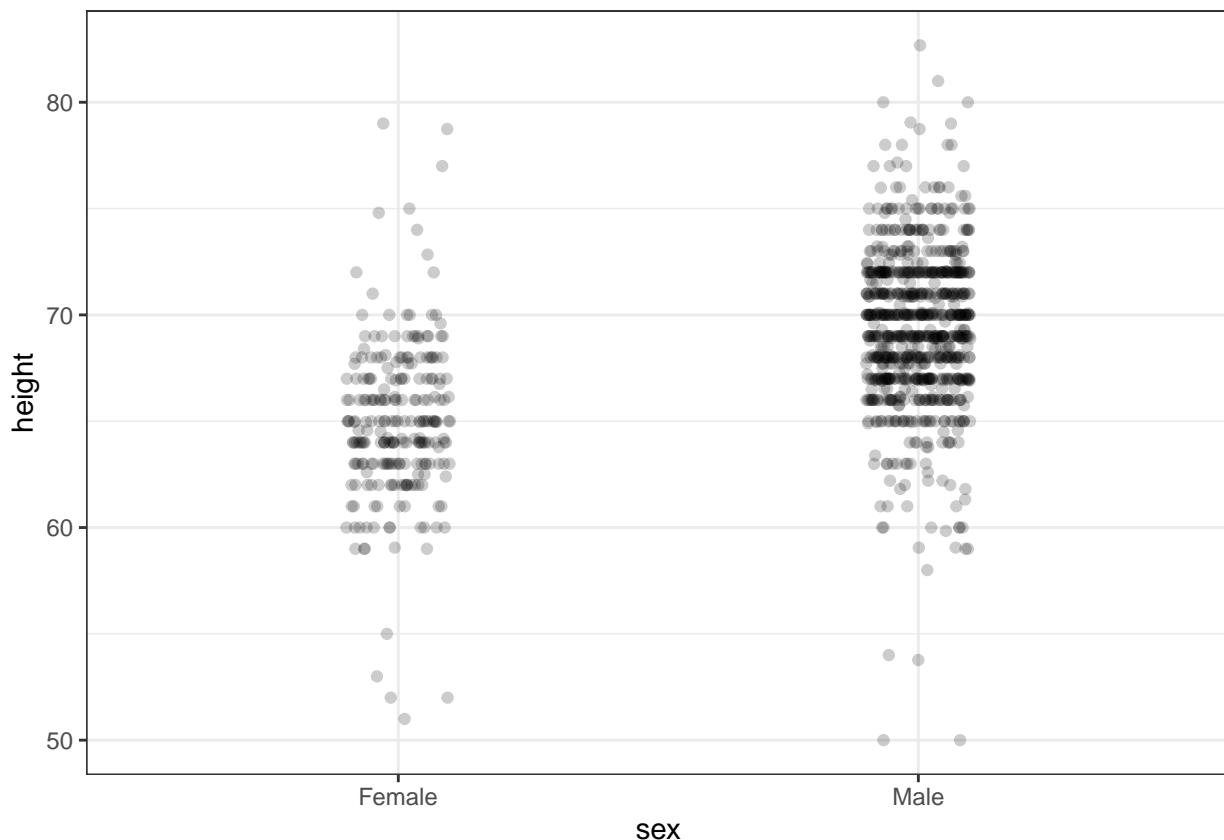
Para mostrar una distribución, puede no ser recomendable utilizar el siguiente gráfico:

```
heights %>% ggplot(aes(sex, height)) + geom_point()
```



En su lugar, puede utilizarse la función “geom\_jitter()”:

```
heights %>%
  ggplot(aes(sex, height)) +
  geom_jitter(width = 0.1, alpha = 0.2)
```



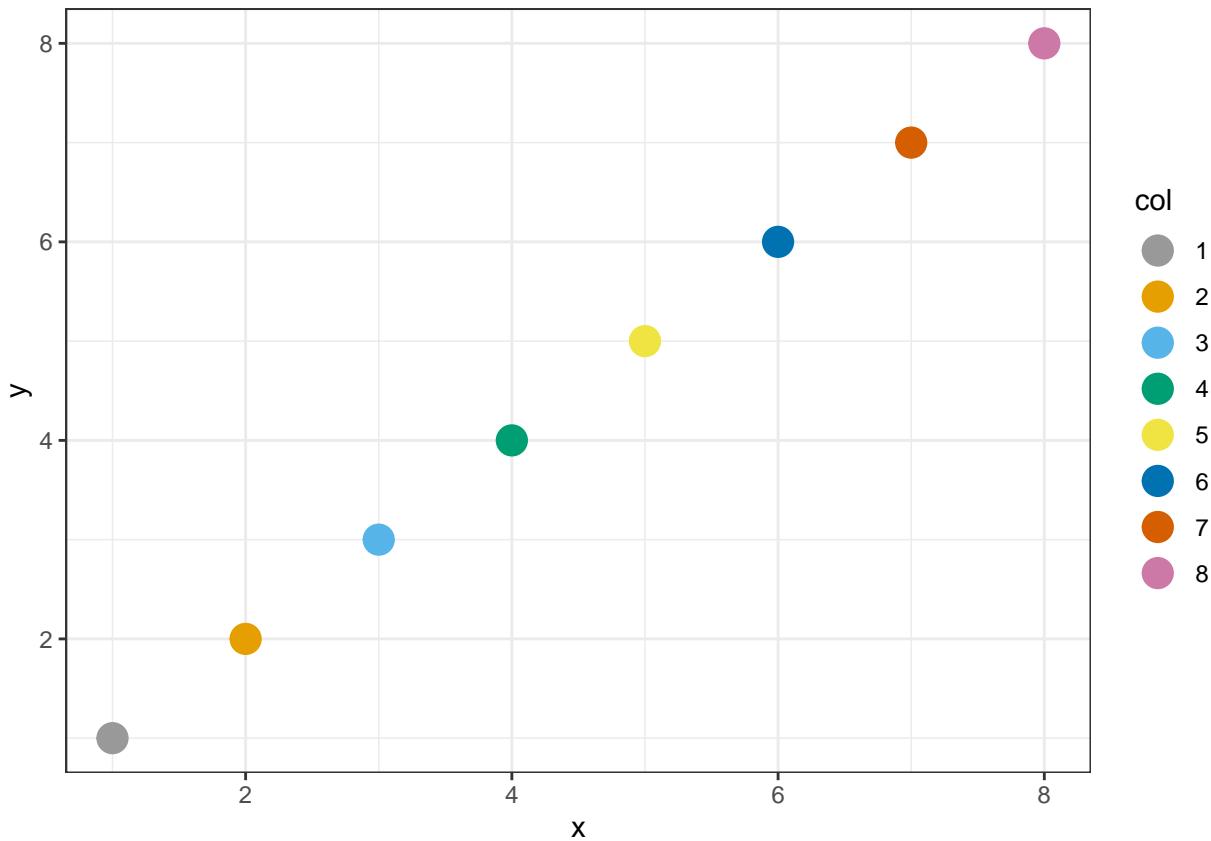
#### 2.5.4. Ejes consistentes

Si se están comparando datos, hay que mantener la escala de los ejes igual en los gráficos que se están comparando, con el objetivo de facilitar la comparabilidad. Asimismo, y con el mismo objetivo, mantener igualmente alineados los gráficos de manera vertical.

#### 2.5.5. Facilitación de comparaciones

```
color_blind_friendly_cols <- c("#999999", "#E69F00", "#56B4E9", "#009E73",
                                "#FOE442", "#0072B2", "#D55E00", "#CC79A7")

p1 <- data.frame(x = 1:8, y = 1:8, col = as.character(1:8)) %>%
  ggplot(aes(x, y, color = col)) +
  geom_point(size = 5)
p1 + scale_color_manual(values = color_blind_friendly_cols)
```



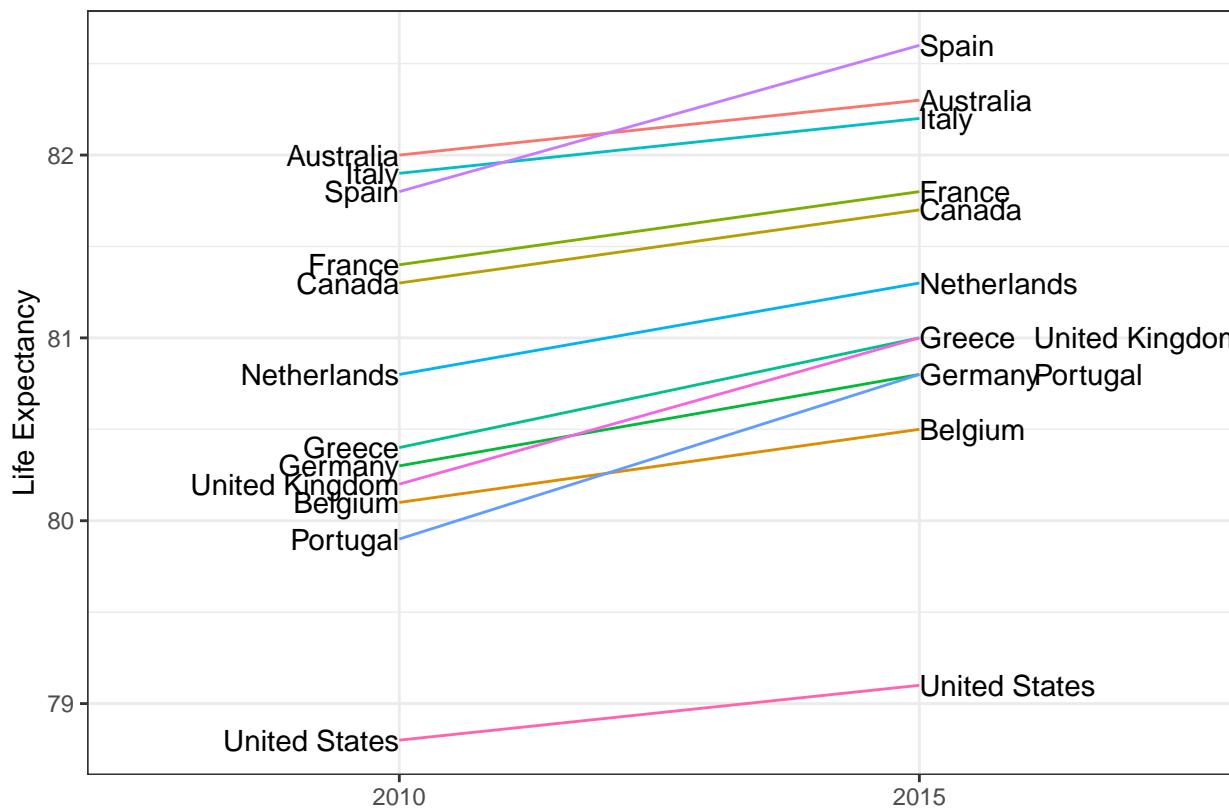
### 2.5.6. Gráficos de pendiente

Al comparar dos variables (homicidios vs. población, fertilidad vs. expectativa de vida o supervivencia infantil vs. ingreso), y si estas son del mismo tipo, puede ser preferible hacerlo en diferentes puntos de tiempo. Por ejemplo, compárese la expectativa de vida de 2010 y 2015:

```
west <- c("Western Europe", "Northern Europe", "Southern Europe", "Northern America",
        "Australia and New Zealand")

dat <- gapminder %>%
  filter(year %in% c(2010, 2015) & region %in% west & !is.na(life_expectancy) & population > 10^7)

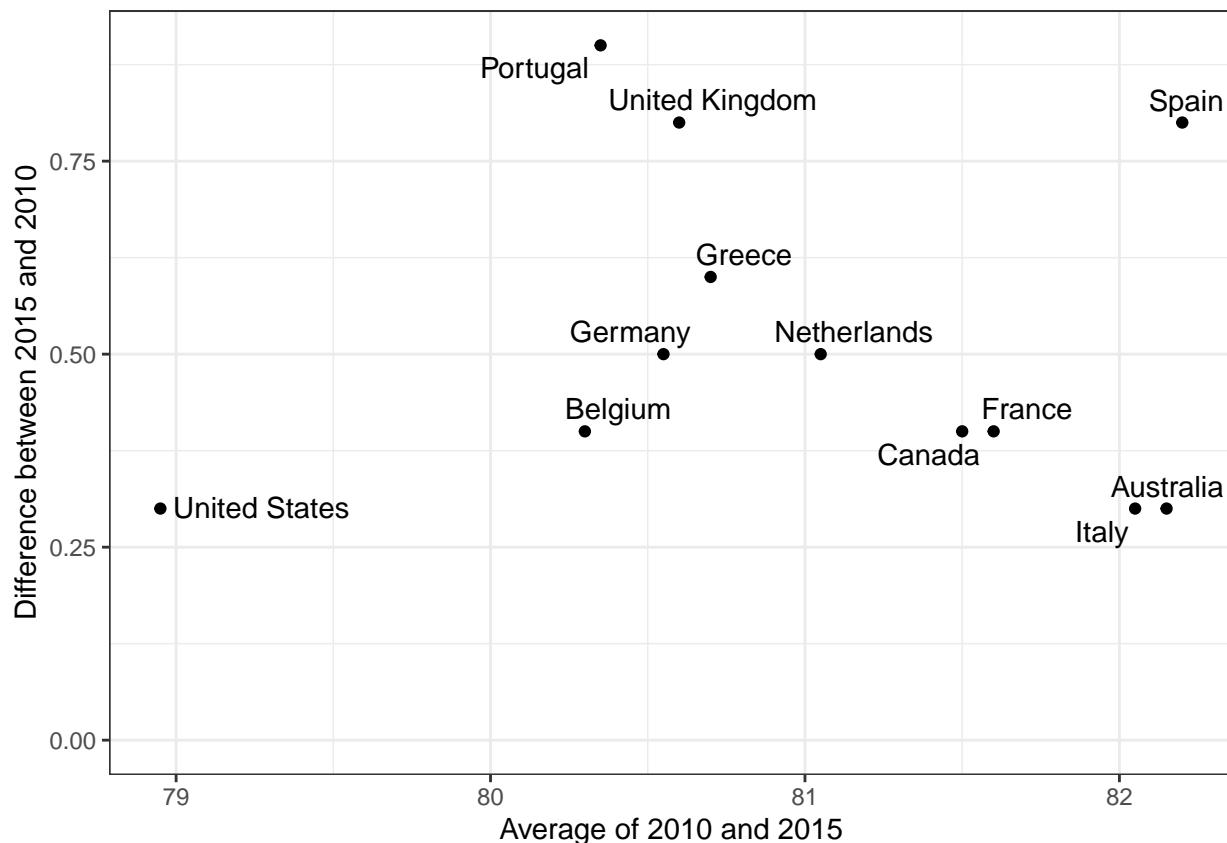
dat %>%
  mutate(location = ifelse(year == 2010, 1, 2),
         location = ifelse(year == 2015 & country %in% c("United Kingdom", "Portugal"),
                           location + 0.22, location), hjust =
         ifelse(year == 2010, 1, 0)) %>%
  mutate(year = as.factor(year)) %>%
  ggplot(aes(year, life_expectancy, group = country)) +
  geom_line(aes(color = country), show.legend = FALSE) +
  geom_text(aes(x = location, label = country, hjust = hjust), show.legend = FALSE) +
  xlab("") +
  ylab("Life Expectancy")
```



### 2.5.7. Gráfica Bland-Altman

Este tipo de gráfico se utiliza cuando se compara una o más variables en dos puntos del tiempo, especialmente para un número de observaciones pequeño. Muestra la diferencia entre las condiciones del eje y y la media entre las condiciones en el eje x

```
dat %>%
  mutate(year = paste0("life_expectancy_", year)) %>%
  select(country, year, life_expectancy) %>% spread(year, life_expectancy) %>%
  mutate(average = (life_expectancy_2015 + life_expectancy_2010)/2,
        difference = life_expectancy_2015 - life_expectancy_2010) %>%
  ggplot(aes(average, difference, label = country)) +
  geom_point() +
  geom_text_repel() +
  geom_abline(lty = 2) +
  xlab("Average of 2010 and 2015") +
  ylab("Difference between 2015 and 2010")
```



### 2.5.8. Codificando una tercera variable

```
data("us_contagious_diseases")
str(us_contagious_diseases)

## 'data.frame': 16065 obs. of 6 variables:
## $ disease      : Factor w/ 7 levels "Hepatitis A",...: 1 1 1 1 1 1 1 1 1 ...
## $ state        : Factor w/ 51 levels "Alabama","Alaska",...: 1 1 1 1 1 1 1 1 1 ...
## $ year         : num  1966 1967 1968 1969 1970 ...
## $ weeks_reporting: num  50 49 52 49 51 51 45 45 45 46 ...
## $ count         : num  321 291 314 380 413 378 342 467 244 286 ...
## $ population    : num  3345787 3364130 3386068 3412450 3444165 ...
```

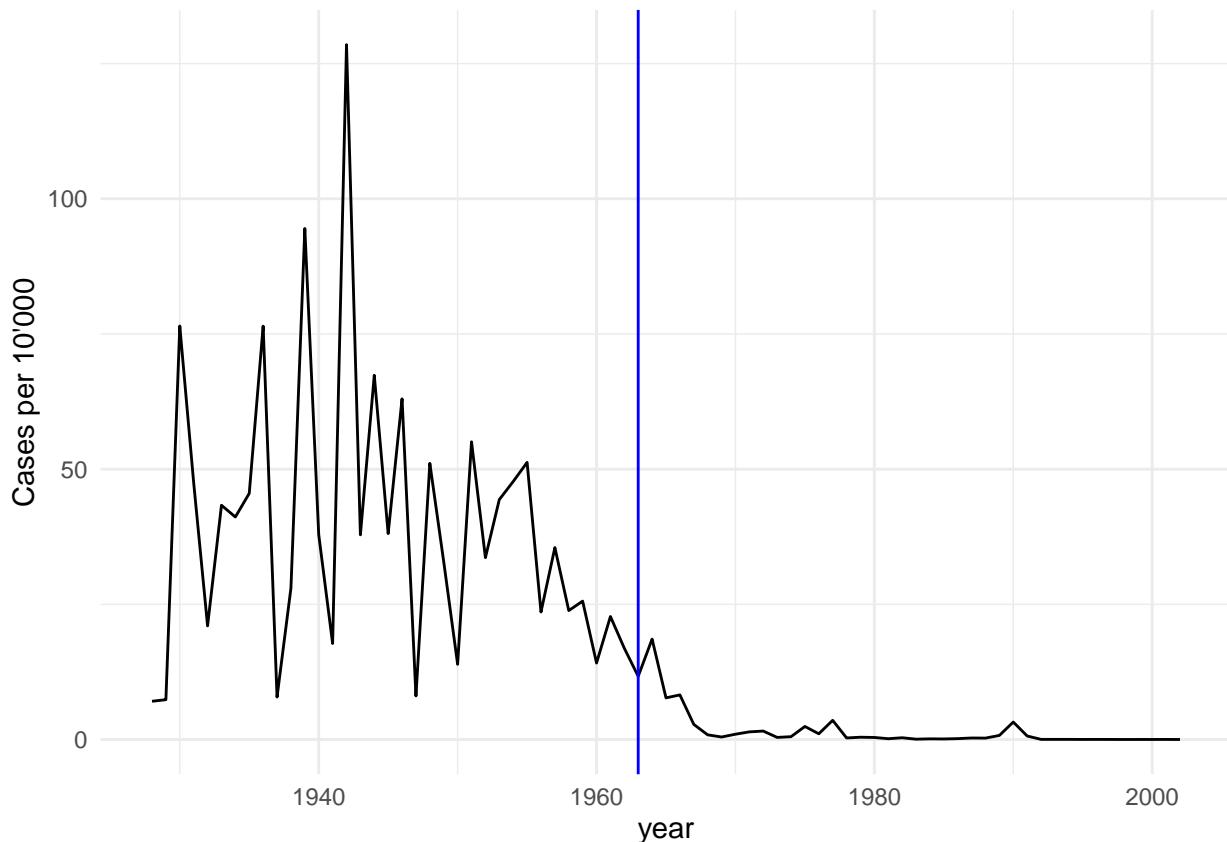
Se creará un objeto para contener la información sobre rubeola, donde se incluirá la tasa por 100'000 habitantes, ordenando los estados por el valor promedio de la enfermedad y excluyendo Alaska y Hawaii:

```
the_disease <- "Measles"

dat <- us_contagious_diseases %>%
  filter(!state %in% c("Hawaii", "Alaska") & disease == the_disease) %>%
  mutate(rate = count/population*10000) %>%
  mutate(state = reorder(state, rate))
```

Así, fácilmente se pueden graficar la tasa de incidencia de rubeola por año en California (donde la línea azul representa el año donde se introdujo la vacuna):

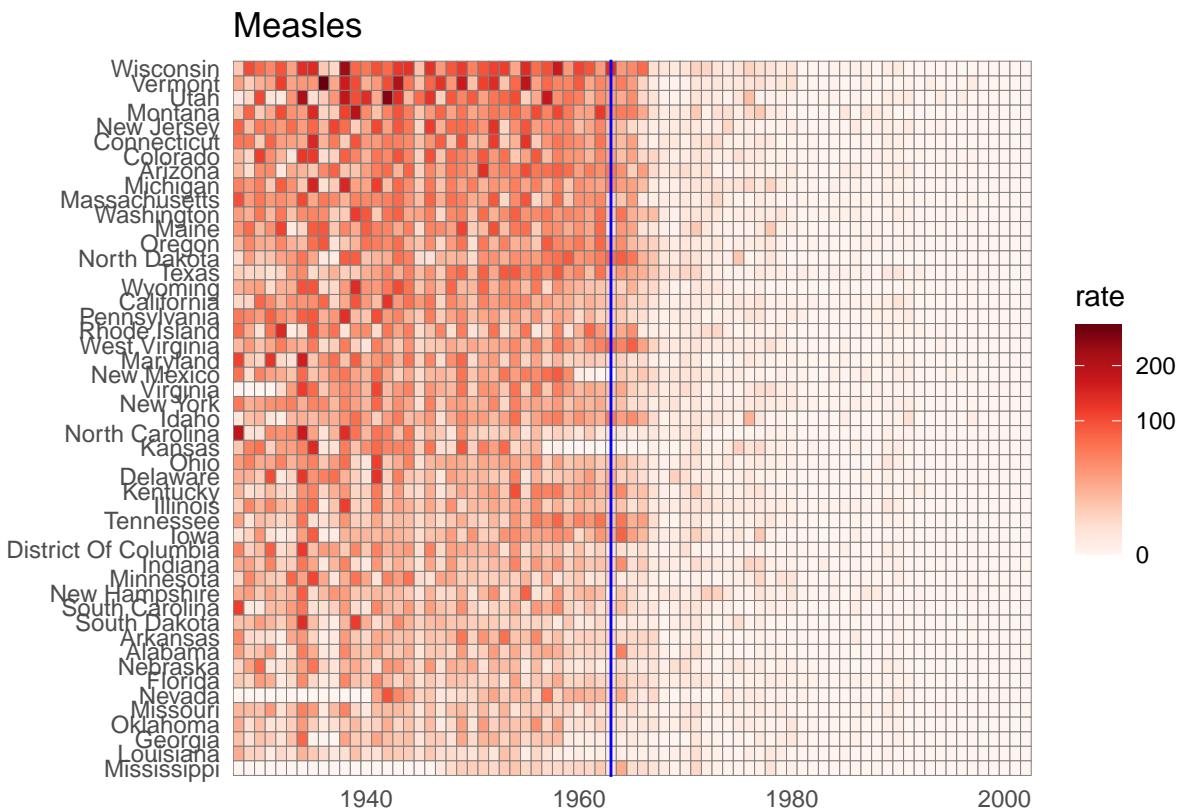
```
dat %>%
  filter(state == "California") %>%
  ggplot(aes(year, rate)) +
  geom_line() +
  ylab("Cases per 10'000") +
  geom_vline(xintercept = 1963, col = "blue") +
  theme_minimal()
```



¿Cómo mostrar los datos de todos los estados en un solo gráfico? Considérese que hay tres variables de interés: año, estado y tasa de incidencia. Por ejemplo, en el WSJ, se usa el eje x para el año, el eje y para el estado, y cuadros de colores para las tasas. Al escoger un color para la escala y paleta de una variable numérica, puede ser **secuencial** (preferible para datos que van de mayor a menor) o **divergente** (mejor para datos que divergen del centro).

Así, se usará el comando “geom\_tile()” para colorear las regiones que representen distintas tasas

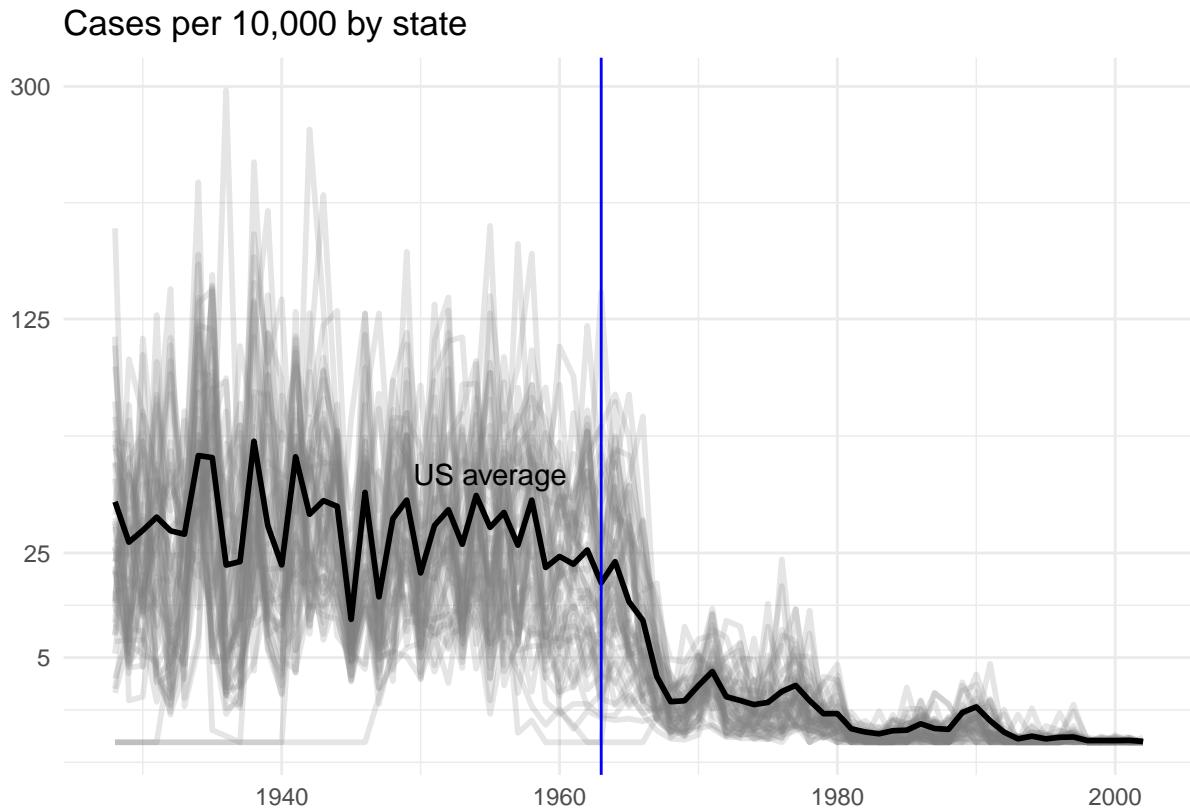
```
dat %>%
  ggplot(aes(year, state, fill = rate)) +
  geom_tile(color = "grey50") +
  scale_x_continuous(expand = c(0,0)) +
  scale_fill_gradientn(colors = RColorBrewer::brewer.pal(9, "Reds"), trans = "sqrt") +
  geom_vline(xintercept = 1963, col = "blue") +
  theme_minimal() + theme(panel.grid = element_blank()) +
  ggtitle(the_disease) +
  ylab("") +
  xlab("")
```



Alternativamente, puede hacerse un gráfico que muestra las tasas promedio de cada estado y la nacional

```
avg <- us_contagious_diseases %>%
  filter(disease == the_disease) %>% group_by(year) %>%
  summarize(us_rate = sum(count, na.rm = TRUE)/sum(population, na.rm = TRUE)*10000)

dat %>%
  filter(!is.na(rate)) %>%
  ggplot() +
  geom_line(aes(year, rate, group = state), color = "grey50",
            show.legend = FALSE, alpha = 0.2, size = 1) +
  geom_line(mapping = aes(year, us_rate), data = avg, size = 1, col = "black") +
  scale_y_continuous(trans = "sqrt", breaks = c(5, 25, 125, 300)) +
  ggtitle("Cases per 10,000 by state") +
  xlab("") +
  ylab("") +
  geom_text(data = data.frame(x = 1955, y = 50),
            mapping = aes(x, y, label = "US average"), color = "black") +
  geom_vline(xintercept = 1963, col = "blue") +
  theme_minimal()
```



#### 2.5.9. Gráficos 3D

En general, hay que evitar pseudo gráficos 3D.

#### 2.5.10. Dígitos significativos

El default de R es mostrar 7 dígitos significativos. Lo recomendable es usar 3 o 4, lo cual se puede configurar desde un inicio mediante la función “options(digits=n)” o individualmente con “signif()” y “round()”

```
options(digits = 4)
```

### 2.5.11. Assessment 4

```
options(digits =3)
library(tidyverse)
library(titanic)
library(gridExtra)

titanic <- titanic_train %>%
  select(Survived, Pclass, Sex, Age, SibSp, Parch, Fare) %>%
  mutate(Survived = factor(Survived), Pclass = factor(Pclass), Sex = factor(Sex))
```

**Pregunta 1.**

Inspect the data and also use ?titanic\_train to learn more about the variables in the dataset. Match these variables from the dataset to their variable type. There is at least one variable of each type (ordinal categorical, non-ordinal categorical, continuous, discrete).

```
head(titanic)

##   Survived Pclass     Sex Age SibSp Parch Fare
## 1        0     3 male  22     1     0  7.25
## 2        1     1 female 38     1     0 71.28
## 3        1     3 female 26     0     0  7.92
## 4        1     1 female 35     1     0 53.10
## 5        0     3 male  35     0     0  8.05
## 6        0     3 male  NA     0     0  8.46
```

**Pregunta 2.**

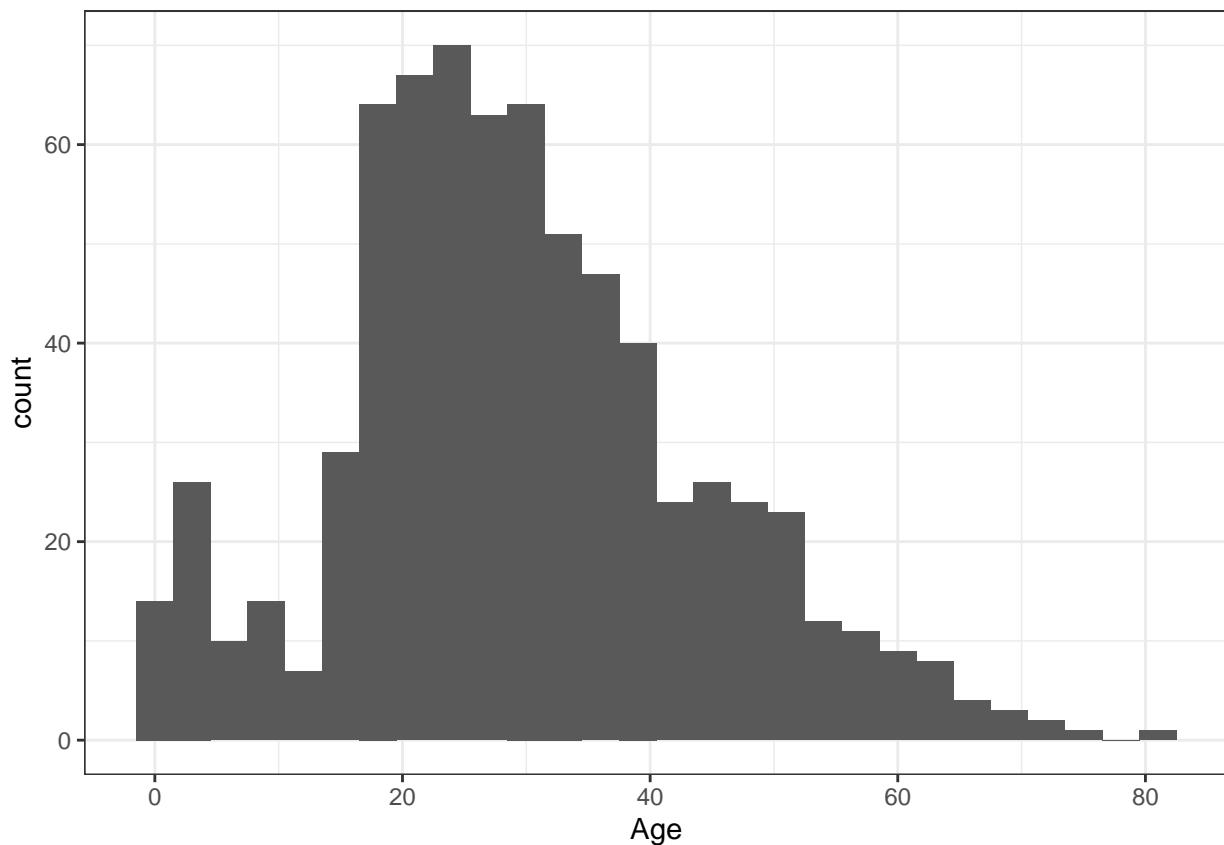
Make density plots of age grouped by sex. Try experimenting with combinations of faceting, alpha blending, stacking and using variable counts on the y-axis to answer the following questions. Some questions may be easier to answer with different versions of the density plot. Which of the following are true?

```
total <- titanic %>%
  filter(Sex %in% c("female", "male")) %>%
  count(Sex)

total

##      Sex    n
## 1 female 314
## 2 male  577

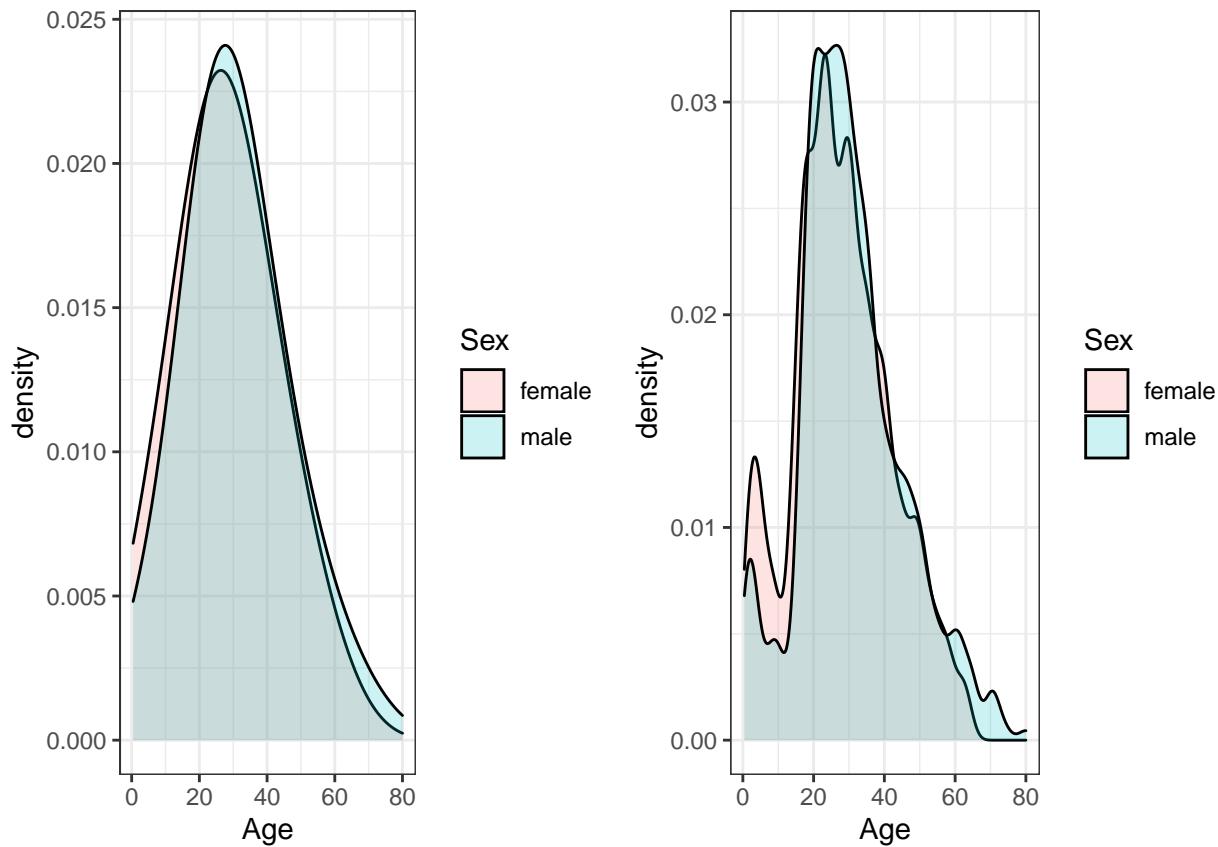
titanic %>%
  filter(!is.na(Age)) %>%
  ggplot(aes(Age, group = Sex)) +
  geom_histogram(binwidth = 3)
```



```
p1 <- titanic %>%
  filter(!is.na(Age)) %>%
  ggplot(aes(Age, group = Sex, fill = Sex)) +
  geom_density(alpha = 0.2, bw = 10)

p2 <- titanic %>%
  filter(!is.na(Age)) %>%
  ggplot(aes(Age, group = Sex, fill = Sex)) +
  geom_density(alpha = 0.2, bw = 2)

grid.arrange(p1, p2, ncol=2)
```



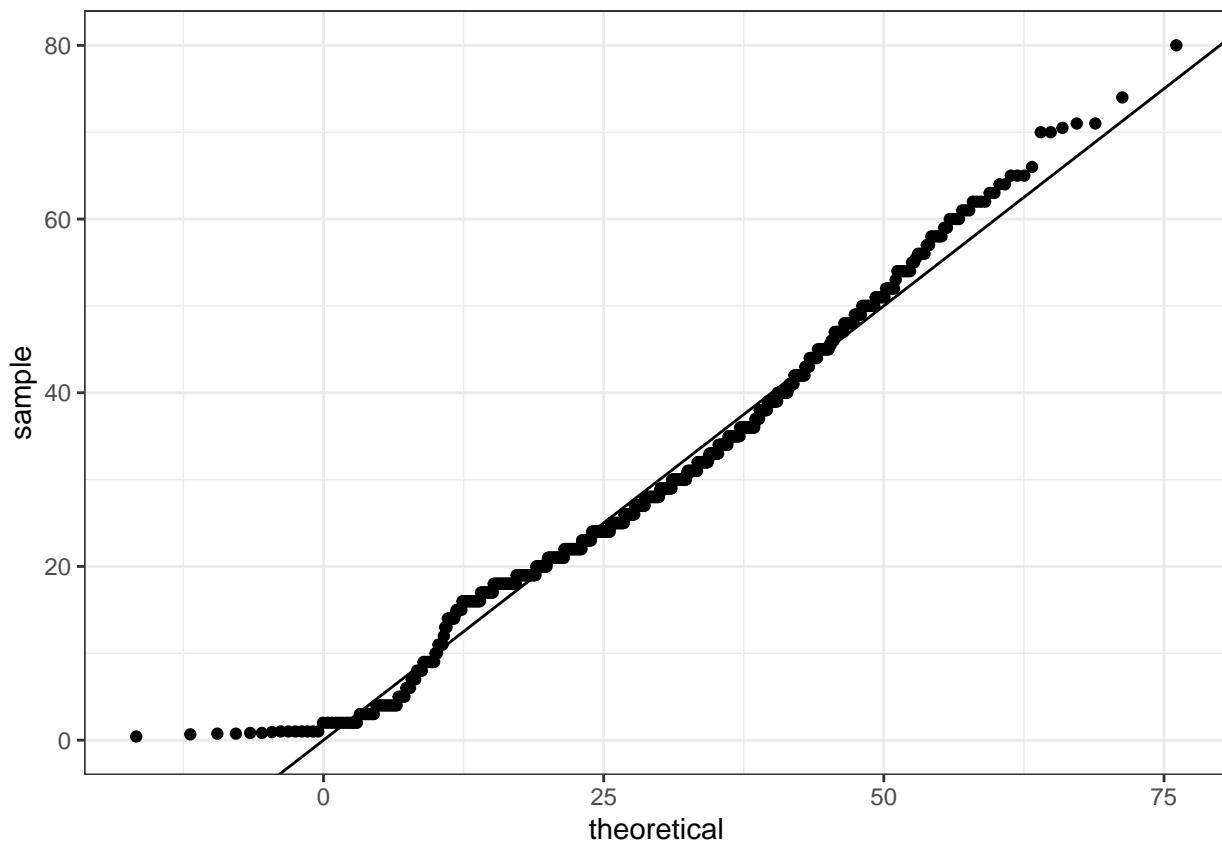
### Pregunta 3.

Use `geom_qq()` to make a QQ-plot of passenger age and add an identity line with `geom_abline()`. Filter out any individuals with an age of `NA` first. Use the following object as the `dparams` argument in `geom_qq()`:

```
params <- titanic %>%
  filter(!is.na(Age)) %>%
  summarize(mean = mean(Age), sd = sd(Age))
```

```
titanic %>%
  ggplot(aes(sample = Age)) +
  geom_qq(dparams = params) +
  geom_abline()
```

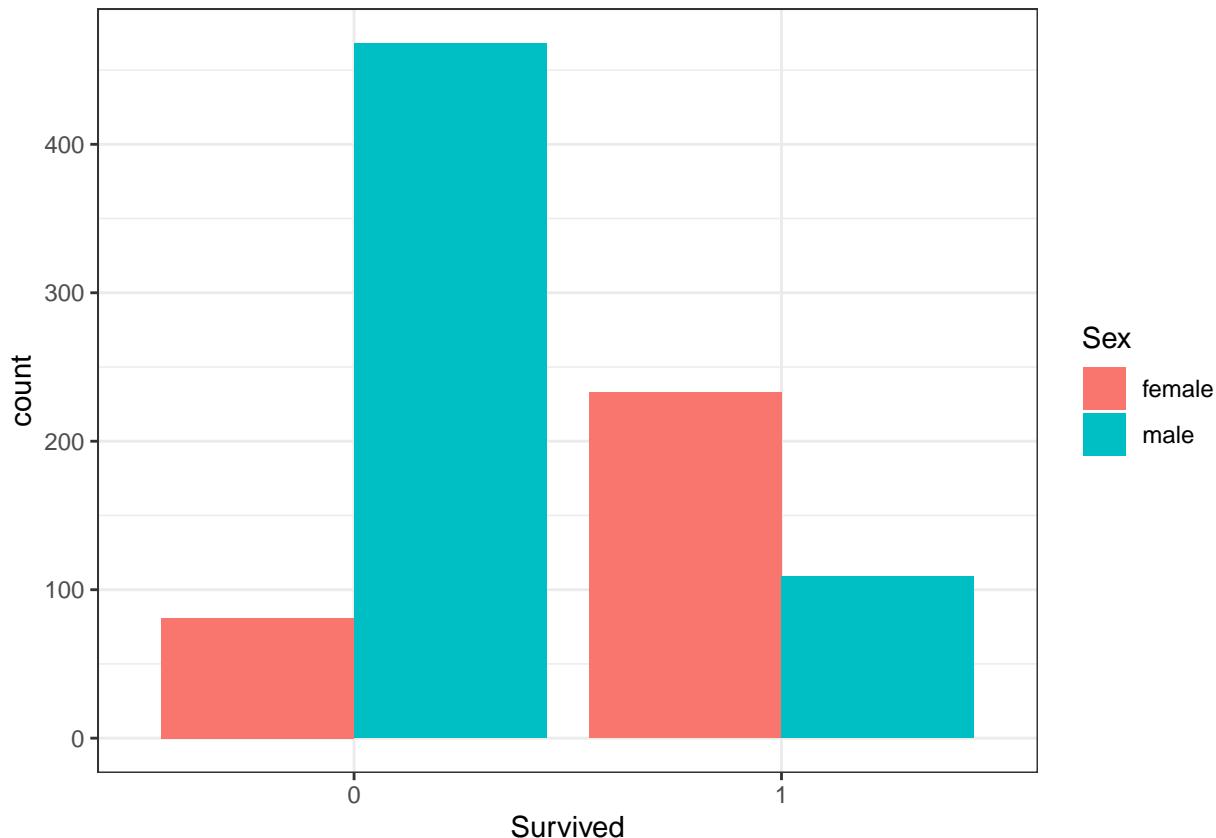
```
## Warning: Removed 177 rows containing non-finite values (stat_qq).
```



#### Pregunta 4.

To answer the following questions, make barplots of the Survived and Sex variables using `geom_bar()`. Try plotting one variable and filling by the other variable. You may want to try the default plot, then try adding `position = position_dodge()` to `geom_bar()` to make separate bars for each group.

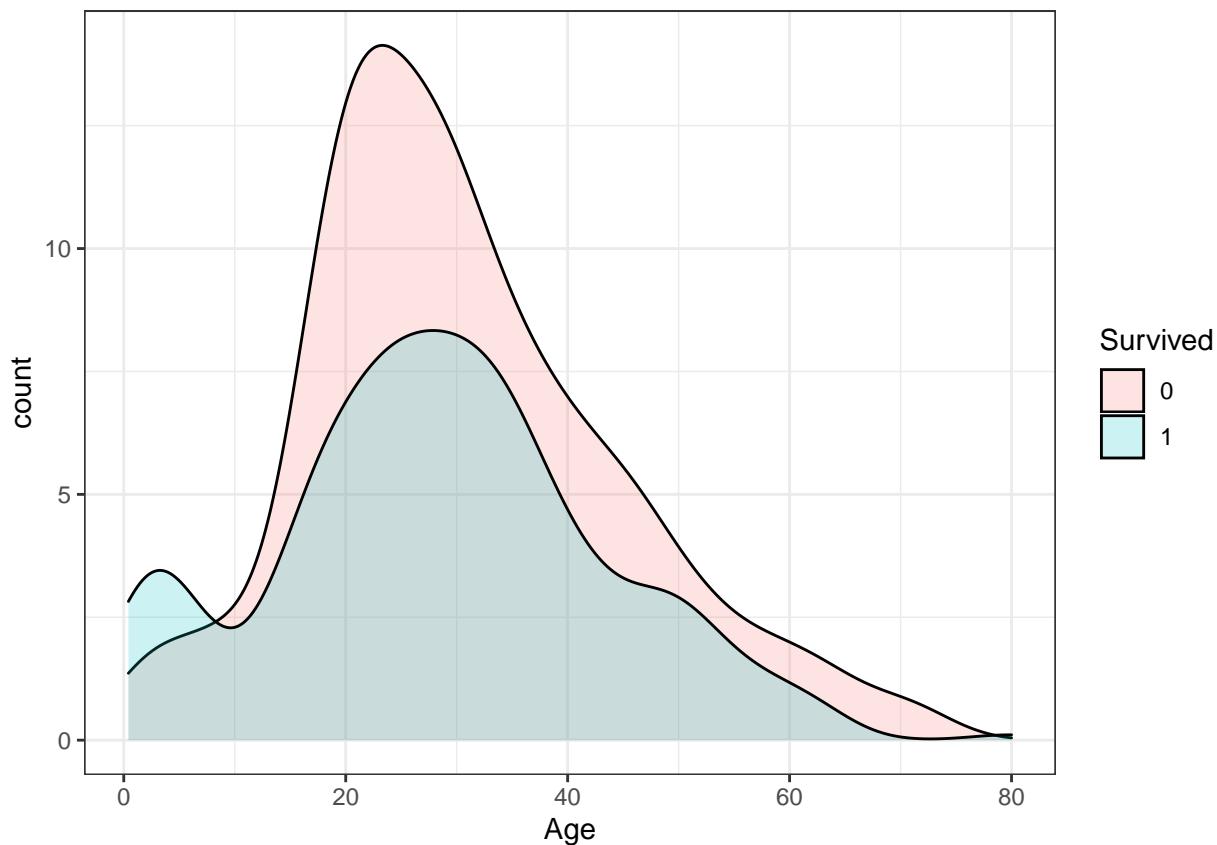
```
titanic %>%
  ggplot(aes(Survived, fill = Sex)) +
  geom_bar(position = position_dodge())
```

**Pregunta 5.**

Make a density plot of age filled by survival status. Change the y-axis to count and set alpha = 0.2.

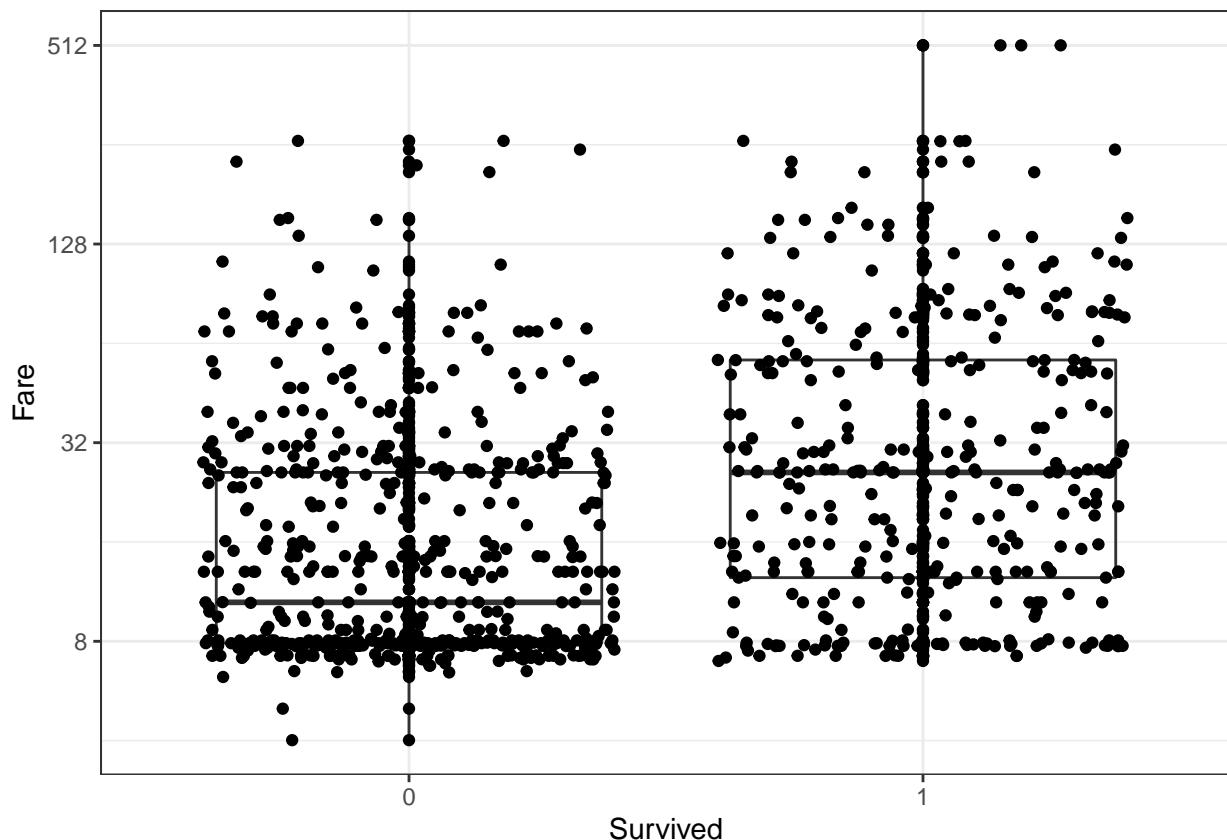
```
titanic %>%
  ggplot(aes(Age, y = ..count.., fill = Survived)) +
  geom_density(alpha = 0.2)

## Warning: Removed 177 rows containing non-finite values (stat_density).
```

**Pregunta 6.**

Filter the data to remove individuals who paid a fare of 0. Make a boxplot of fare grouped by survival status. Try a log2 transformation of fares. Add the data points with jitter and alpha blending.

```
titanic %>%
  filter(Fare > 0) %>%
  ggplot(aes(Survived, Fare)) +
  geom_boxplot(alpha = 0.2) +
  scale_y_continuous(trans = "log2") +
  geom_point(show.legend = FALSE) +
  geom_jitter()
```



#### Pregunta 7.

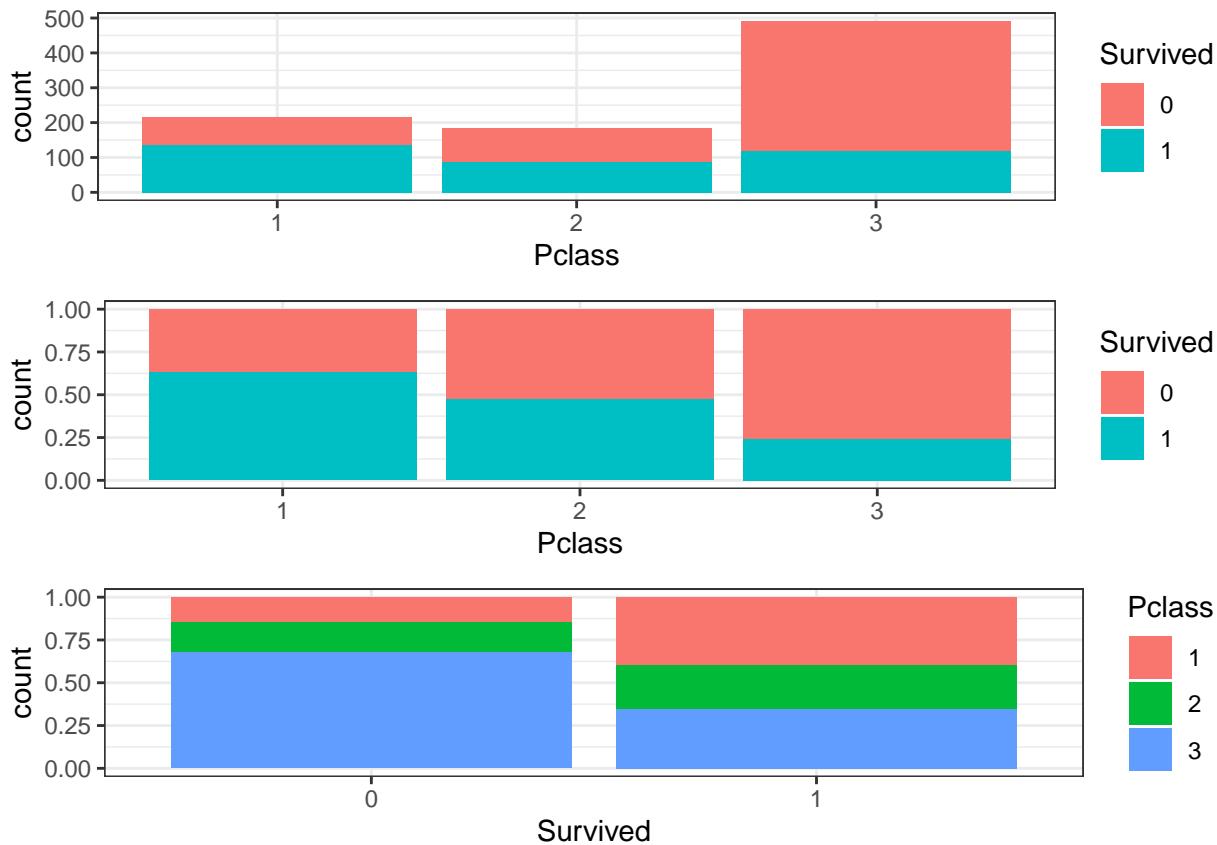
The Pclass variable corresponds to the passenger class. Make three barplots. For the first, make a basic barplot of passenger class filled by survival. For the second, make the same barplot but use the argument position = position\_fill() to show relative proportions in each group instead of counts. For the third, make a barplot of survival filled by passenger class using position = position\_fill().

```
p3 <- titanic %>%
  ggplot(aes(Pclass, fill = Survived)) +
  geom_bar()

p4 <- titanic %>%
  ggplot(aes(Pclass, fill = Survived)) +
  geom_bar(position = position_fill())

p5 <- titanic %>%
  ggplot(aes(Survived, fill = Pclass)) +
  geom_bar(position = position_fill())

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

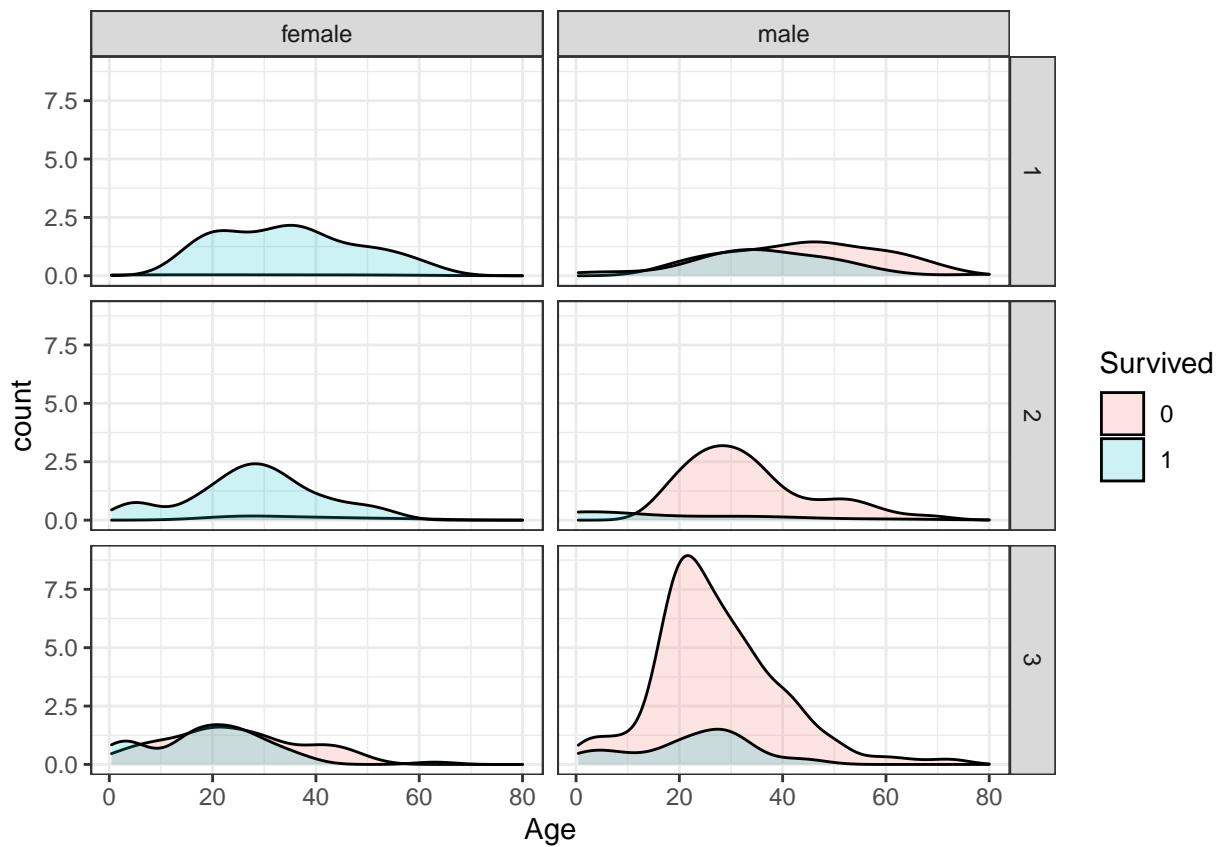


#### Pregunta 8.

Create a grid of density plots for age, filled by survival status, with count on the y-axis, faceted by sex and passenger class.

```
titanic %>%
  ggplot(aes(Age, y = ..count.., fill = Survived)) +
  geom_density(alpha=0.2) +
  facet_grid(Pclass~Sex)
```

```
## Warning: Removed 177 rows containing non-finite values (stat_density).
```



## 2.6. Comprehensive assessment

```
library(tidyverse)
library(ggrepel)
library(dslabs)
library(ggthemes)
library(gridExtra)
data("stars")
data("temp_carbon")
data("greenhouse_gases")
data("historic_co2")
options(digits = 3)
```

Pregunta 1.

Load the stars data frame from dslabs. This contains the name, absolute magnitude, temperature in degrees Kelvin, and spectral class of selected stars. Absolute magnitude (shortened in these problems to simply “magnitude”) is a function of star luminosity, where negative values of magnitude have higher luminosity.

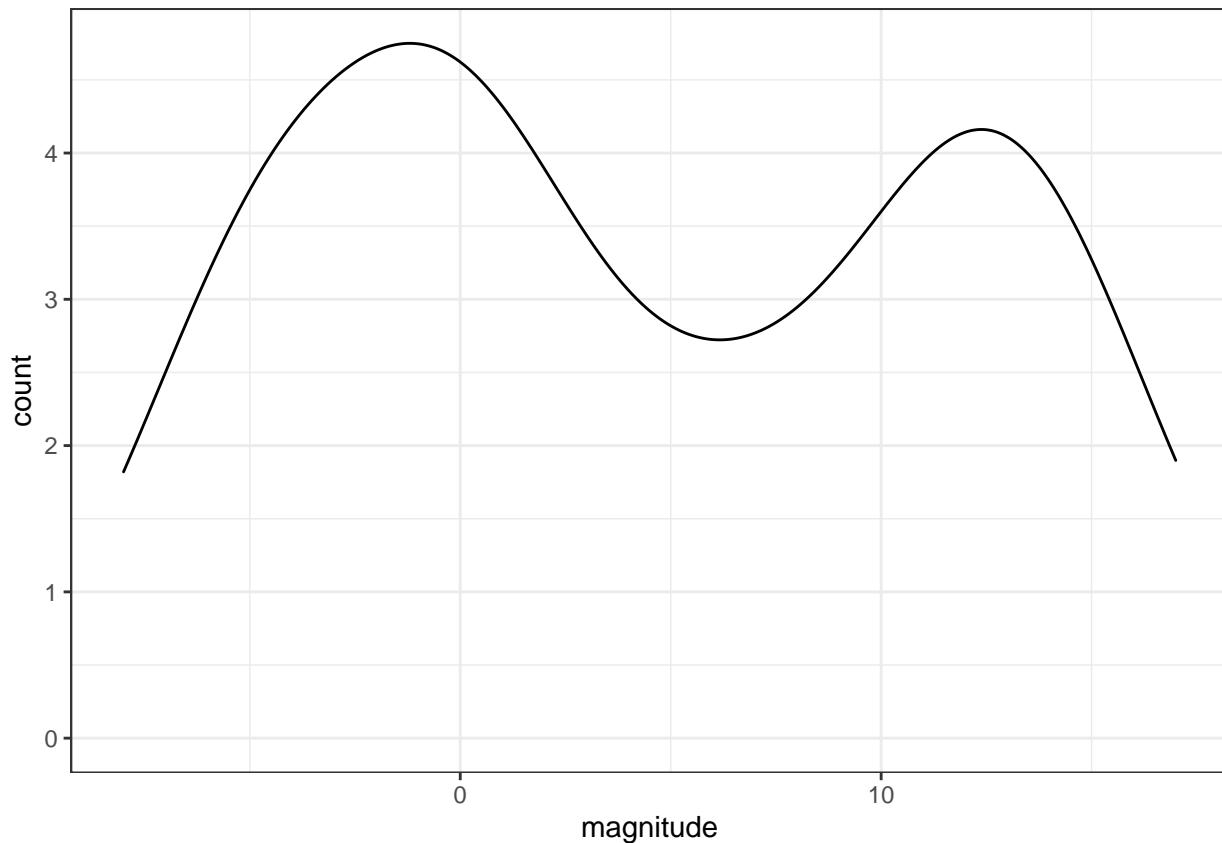
```
summarize(stars, Media = mean(magnitude), DS = sd(magnitude))
```

```
##   Media    DS
## 1  4.26 7.35
```

Pregunta 2.

Make a density plot of the magnitude.

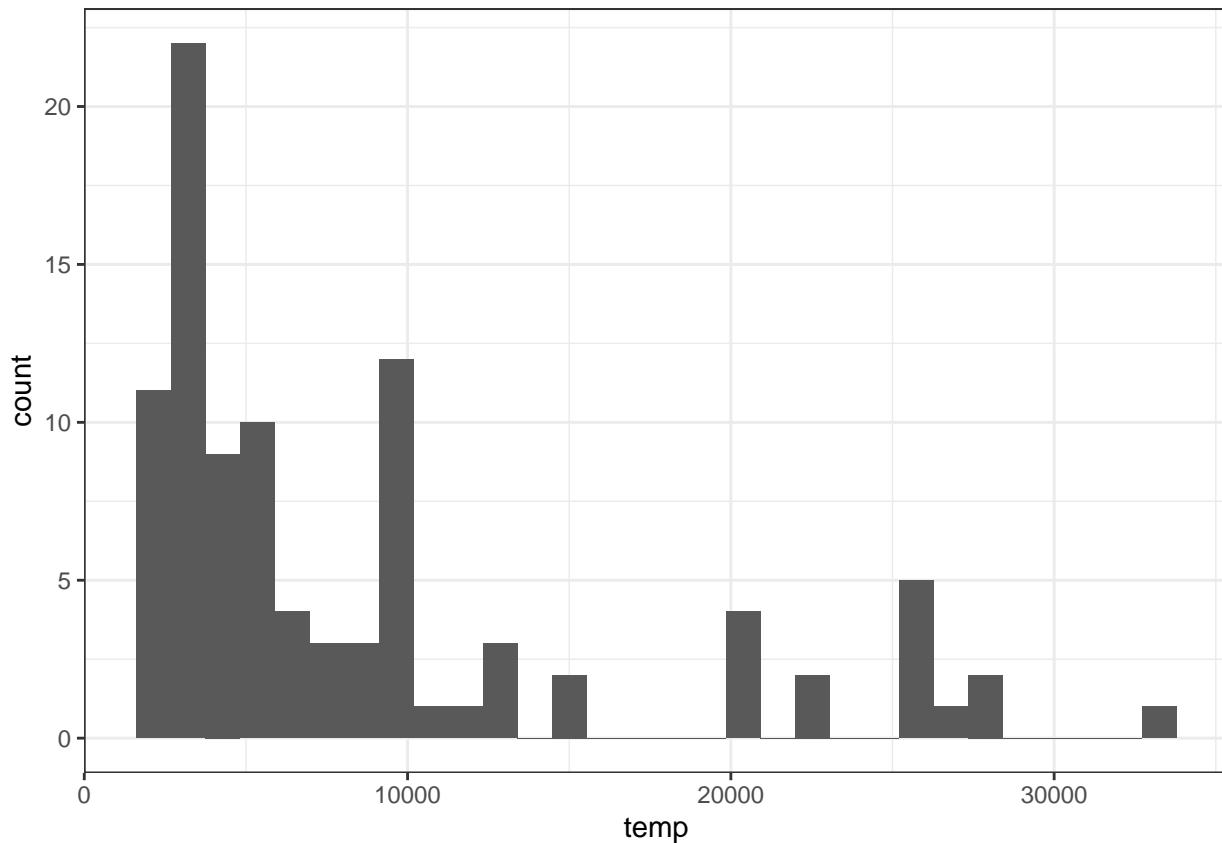
```
stars %>%
  ggplot(aes(magnitude, y = ..count..)) +
  geom_density()
```

**Pregunta 3.**

Examine the distribution of star temperature.

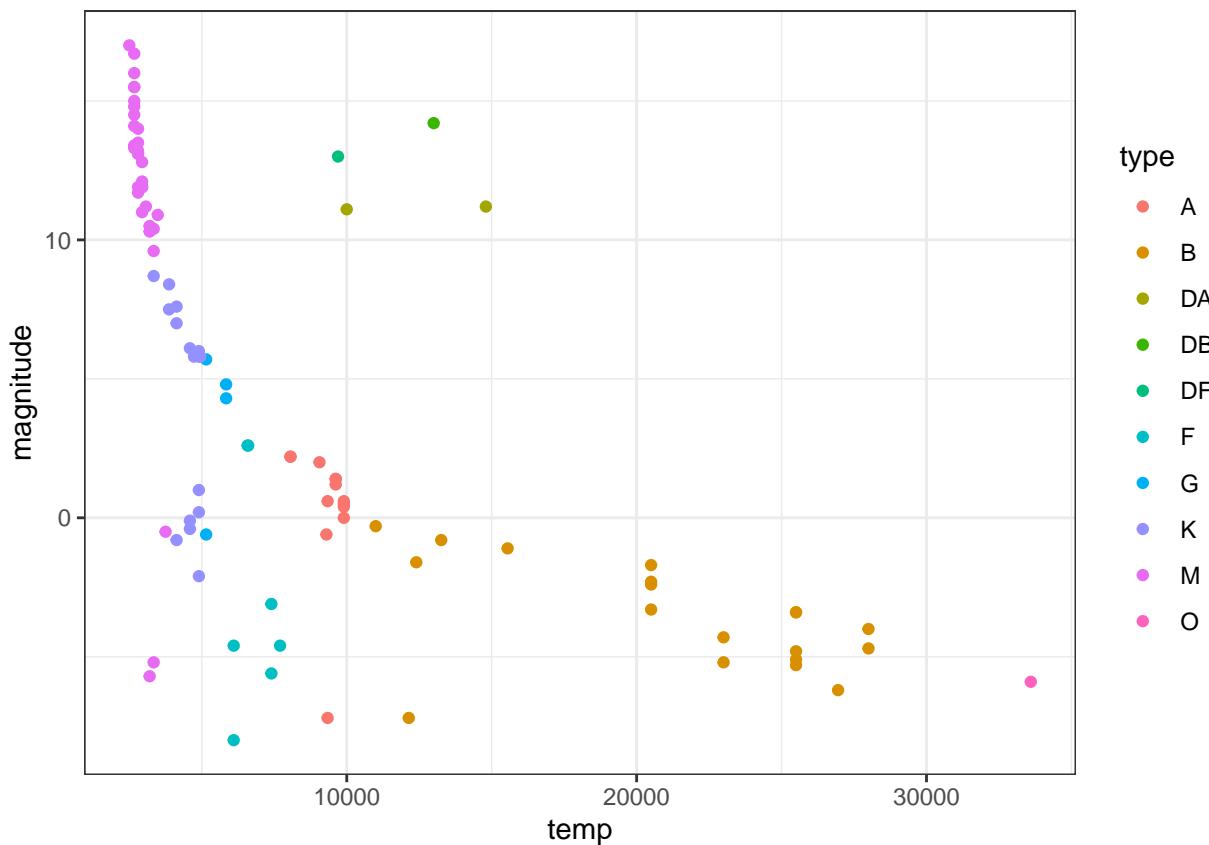
```
stars %>%
  ggplot(aes(temp)) +
  geom_histogram()

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

**Pregunta 4.**

Make a scatter plot of the data with temperature on the x-axis and magnitude on the y-axis and examine the relationship between the variables. Recall that lower magnitude means a more luminous (brighter) star.

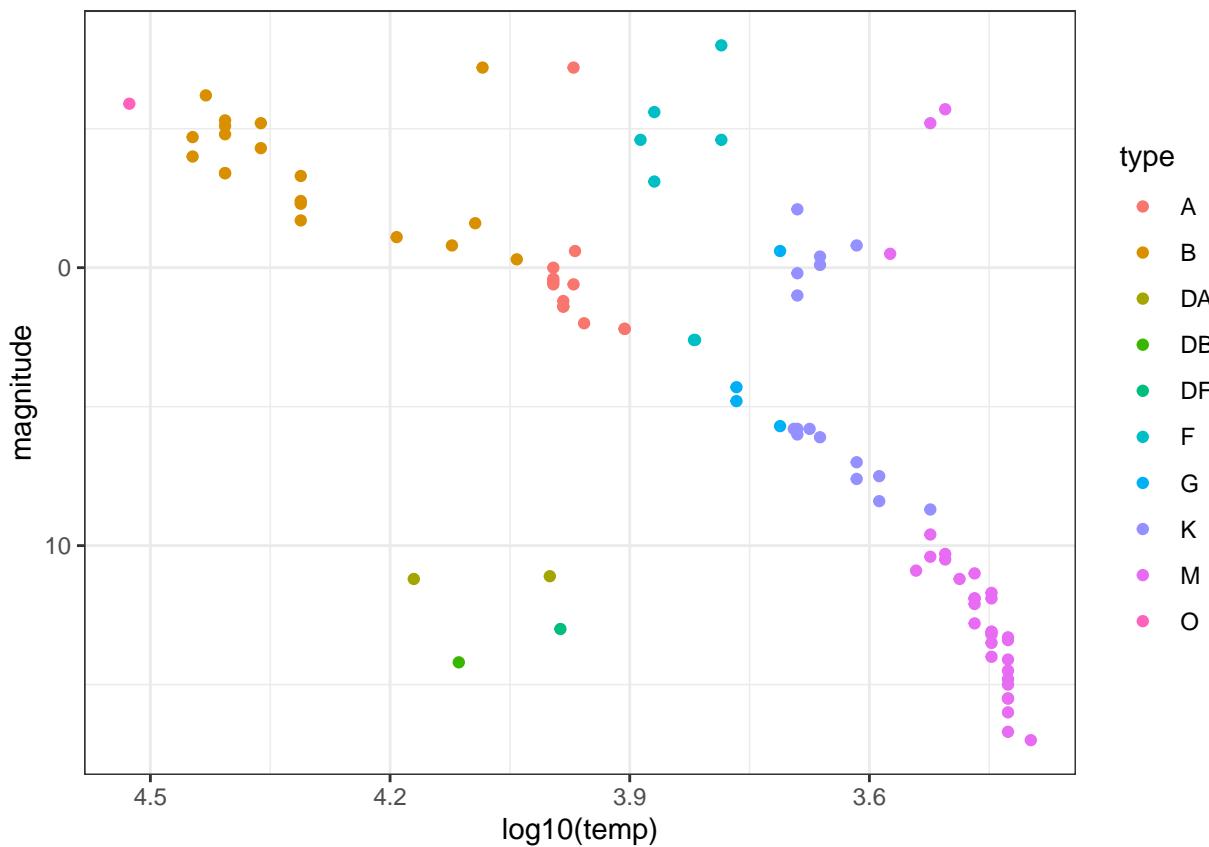
```
stars %>%
  ggplot(aes(temp, magnitude, color = type)) +
  geom_point()
```



### Pregunta 5.

For various reasons, scientists do not always follow straight conventions when making plots, and astronomers usually transform values of star luminosity and temperature before plotting. Flip the y-axis so that lower values of magnitude are at the top of the axis (recall that more luminous stars have lower magnitude) using `scale_y_reverse()`. Take the log base 10 of temperature and then also flip the x-axis.

```
stars %>%
  ggplot(aes(log10(temp), magnitude, color = type)) +
  geom_point() +
  scale_y_reverse() +
  scale_x_reverse()
```



#### Pregunta 6.

The trends you see allow scientists to learn about the evolution and lifetime of stars. The primary group of stars to which most stars belong we will call the main sequence stars (discussed in question 4). Most stars belong to this main sequence, however some of the more rare stars are classified as “old” and “evolved” stars. These stars tend to be hotter stars, but also have low luminosity, and are known as white dwarfs.

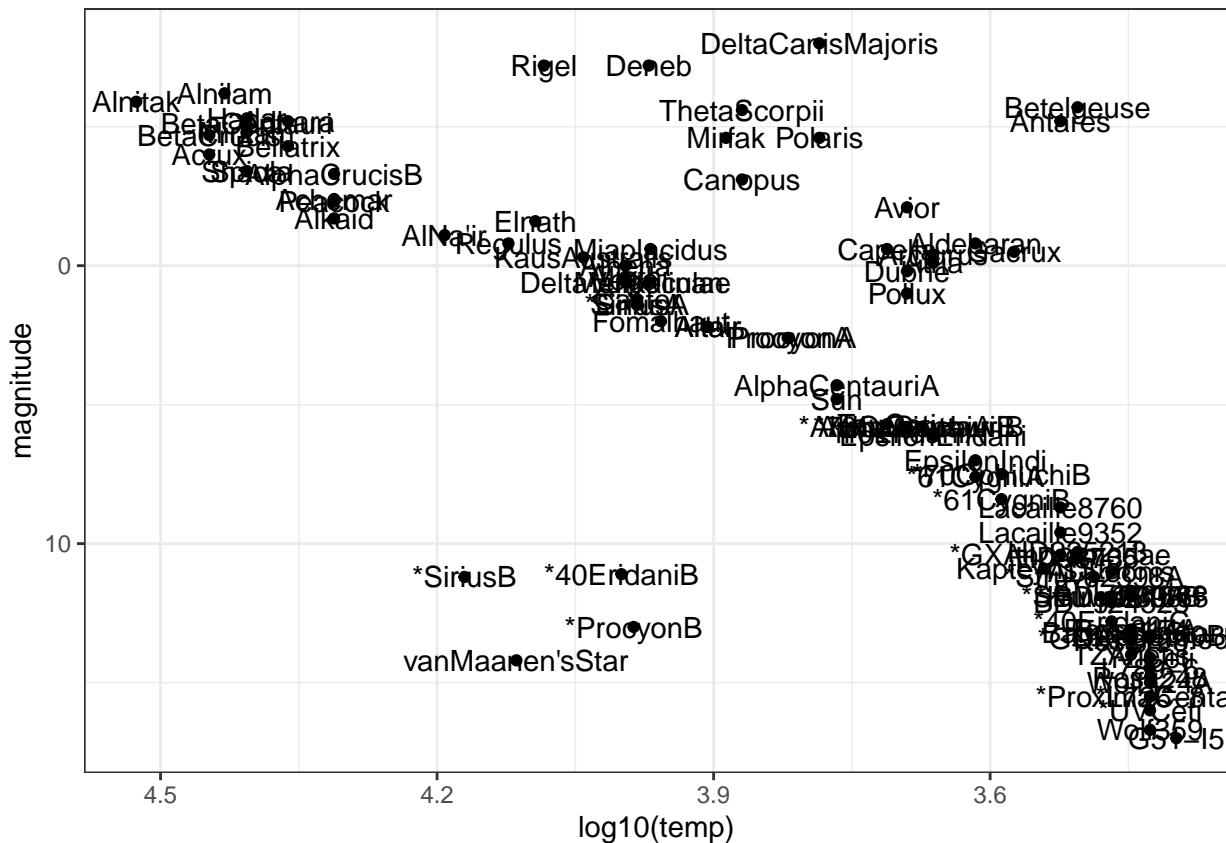
#### Pregunta 7.

Consider stars which are not part of the Main Group but are not old/evolved (white dwarf) stars. These stars must also be unique in certain ways and are known as giants. Use the plot from Question 5 to estimate the average temperature of a giant.

#### Pregunta 8.

We can now identify whether specific stars are main sequence stars, red giants or white dwarfs. Add text labels to the plot to answer these questions. You may wish to plot only a selection of the labels, repel the labels, or zoom in on the plot in RStudio so you can locate specific stars.

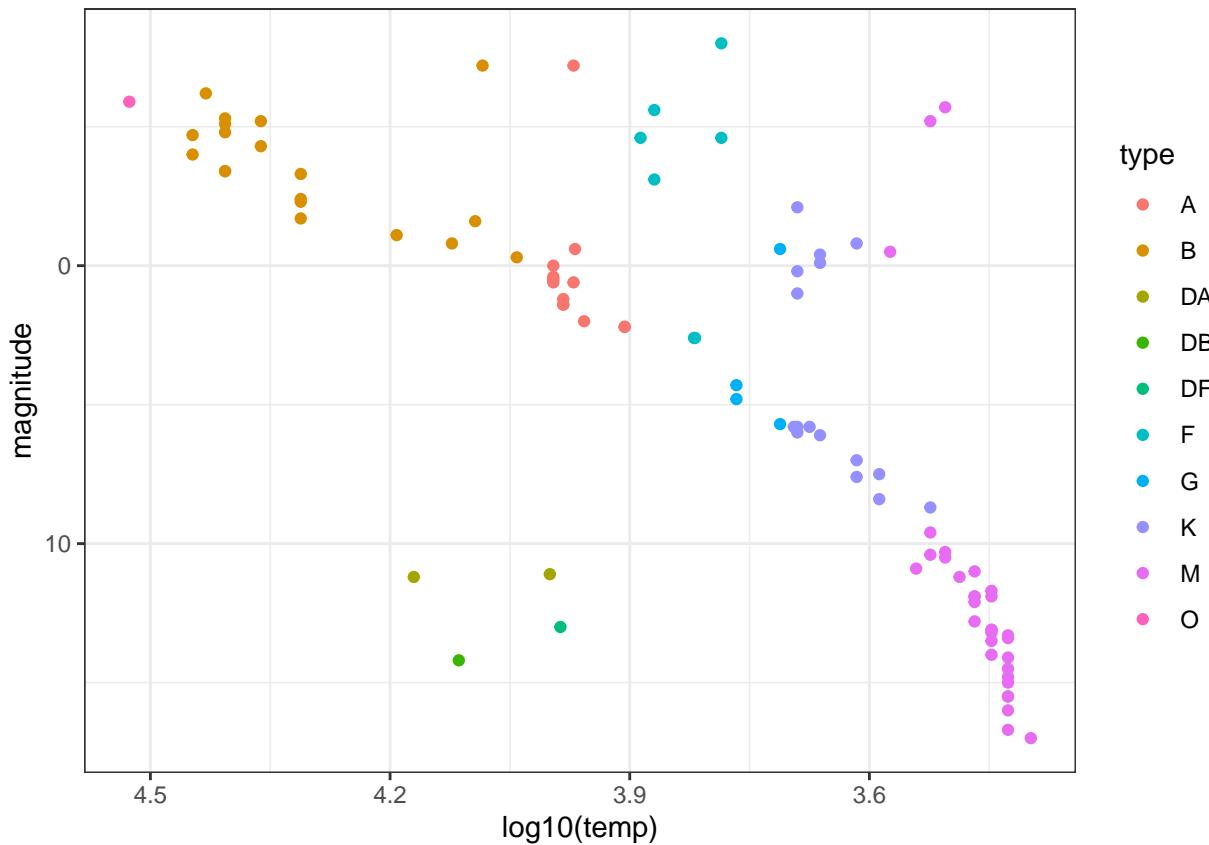
```
stars %>%
  ggplot(aes(log10(temp), magnitude)) +
  geom_point() +
  geom_text(aes(label = star)) +
  scale_x_reverse() +
  scale_y_reverse()
```



### Pregunta 9.

Remove the text labels and color the points by star type. This classification describes the properties of the star's spectrum, the amount of light produced at various wavelengths.

```
stars %>%
  ggplot(aes(log10(temp), magnitude, color = type)) +
  geom_point() +
  scale_x_reverse() +
  scale_y_reverse()
```

**Pregunta 10.**

Load the temp\_carbon dataset from dslabs, which contains annual global temperature anomalies (difference from 20th century mean temperature in degrees Celsius), temperature anomalies over the land and ocean, and global carbon emissions (in metric tons). Note that the date ranges differ for temperature and carbon emissions. Which of these code blocks return the latest year for which carbon emissions are reported?

```
temp_carbon %>%
  filter(!is.na(carbon_emissions)) %>%
  pull(year) %>%
  max()
```

```
## [1] 2014
```

```
temp_carbon %>%
  filter(!is.na(carbon_emissions)) %>%
  .$year %>%
  max()
```

```
## [1] 2014
```

```
temp_carbon %>%
  filter(!is.na(carbon_emissions)) %>%
  select(year) %>%
  max()
```

```
## [1] 2014
```

**Pregunta 11.**

Inspect the difference in carbon emissions in temp\_carbon from the first available year to the last available year.

```
temp_carbon %>%
  filter(!is.na(carbon_emissions)) %>%
  select(year) %>%
  min()

## [1] 1751

temp_carbon %>%
  filter(!is.na(carbon_emissions)) %>%
  select(year) %>%
  max()

## [1] 2014

temp_carbon %>%
  filter(!is.na(carbon_emissions)) %>%
  summarize(ratio = max(carbon_emissions)/min(carbon_emissions))

##    ratio
## 1 3285
```

### Pregunta 12.

Inspect the difference in temperature in temp\_carbon from the first available year to the last available year.

```
temp_carbon %>%
  filter(!is.na(temp_anomaly)) %>%
  select(year) %>%
  min()

## [1] 1880

temp_carbon %>%
  filter(!is.na(temp_anomaly)) %>%
  select(year) %>%
  max()

## [1] 2018

min <- temp_carbon$temp_anomaly[temp_carbon$year==1880]
max <- temp_carbon$temp_anomaly[temp_carbon$year==2018]

max - min

## [1] 0.93
```

### Pregunta 13.

Create a time series line plot of the temperature anomaly. Only include years where temperatures are reported. Save this plot to the object p.

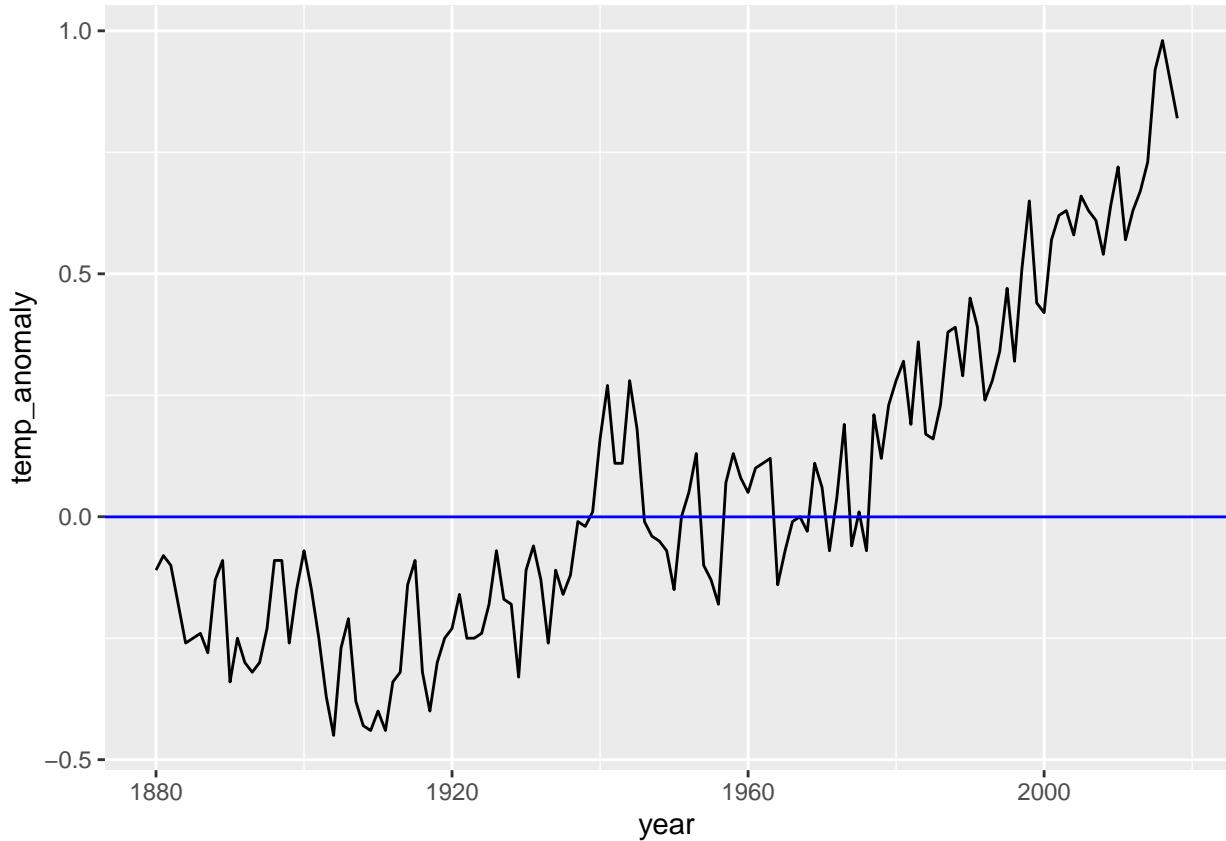
```
p <- temp_carbon %>%
  filter(!is.na(temp_anomaly)) %>%
  ggplot(aes(year, temp_anomaly)) +
  geom_line() +
  theme_gray()
```

```
media <- temp_carbon %>%
  filter(!is.na(temp_anomaly) & year >= 1900 & year <= 2000)

mean(media$temp_anomaly)

## [1] -0.000297

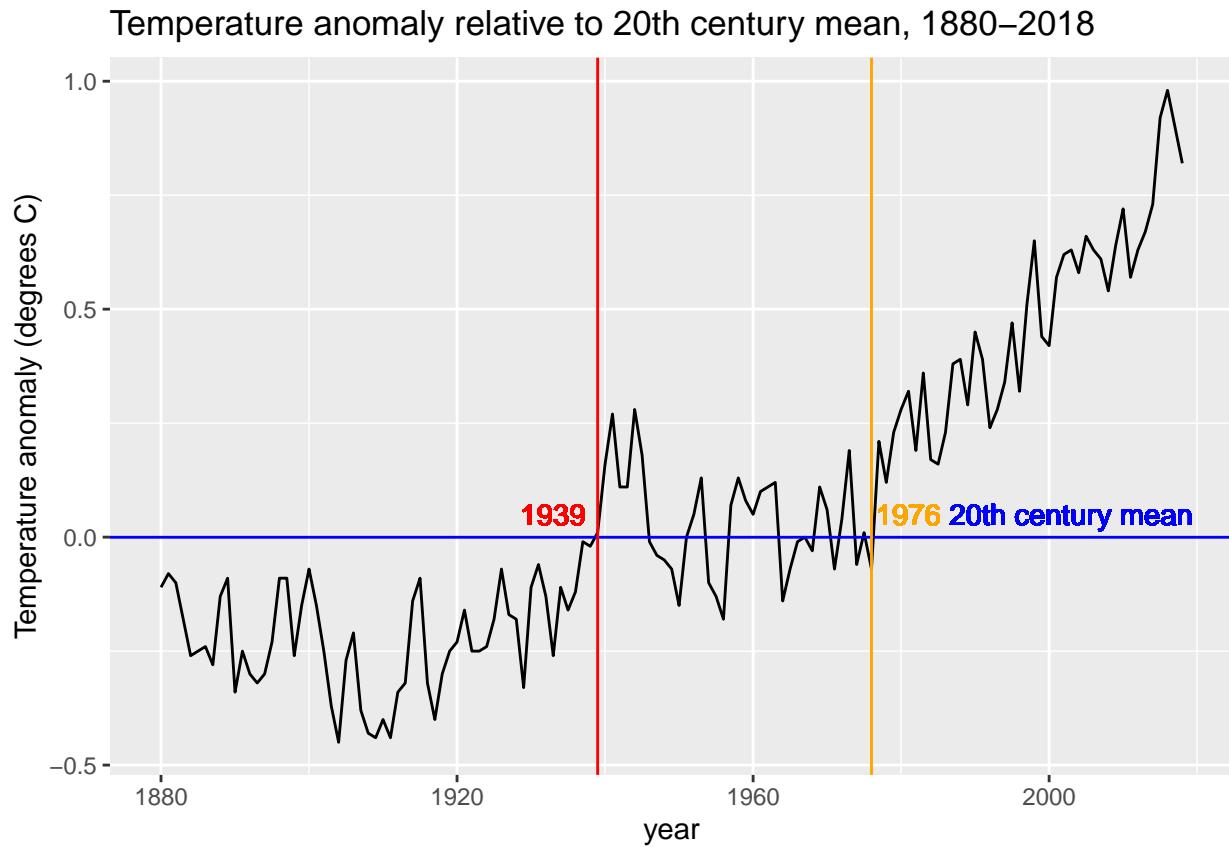
p + geom_hline(aes(yintercept = mean(media$temp_anomaly)), col = "blue")
```



#### Pregunta 14.

Continue working with p, the plot created in the previous question. Change the y-axis label to be “Temperature anomaly (degrees C)”. Add a title, “Temperature anomaly relative to 20th century mean, 1880-2018”. Also add a text layer to the plot: the x-coordinate should be 2000, the y-coordinate should be 0.05, the text should be “20th century mean”, and the text color should be blue.

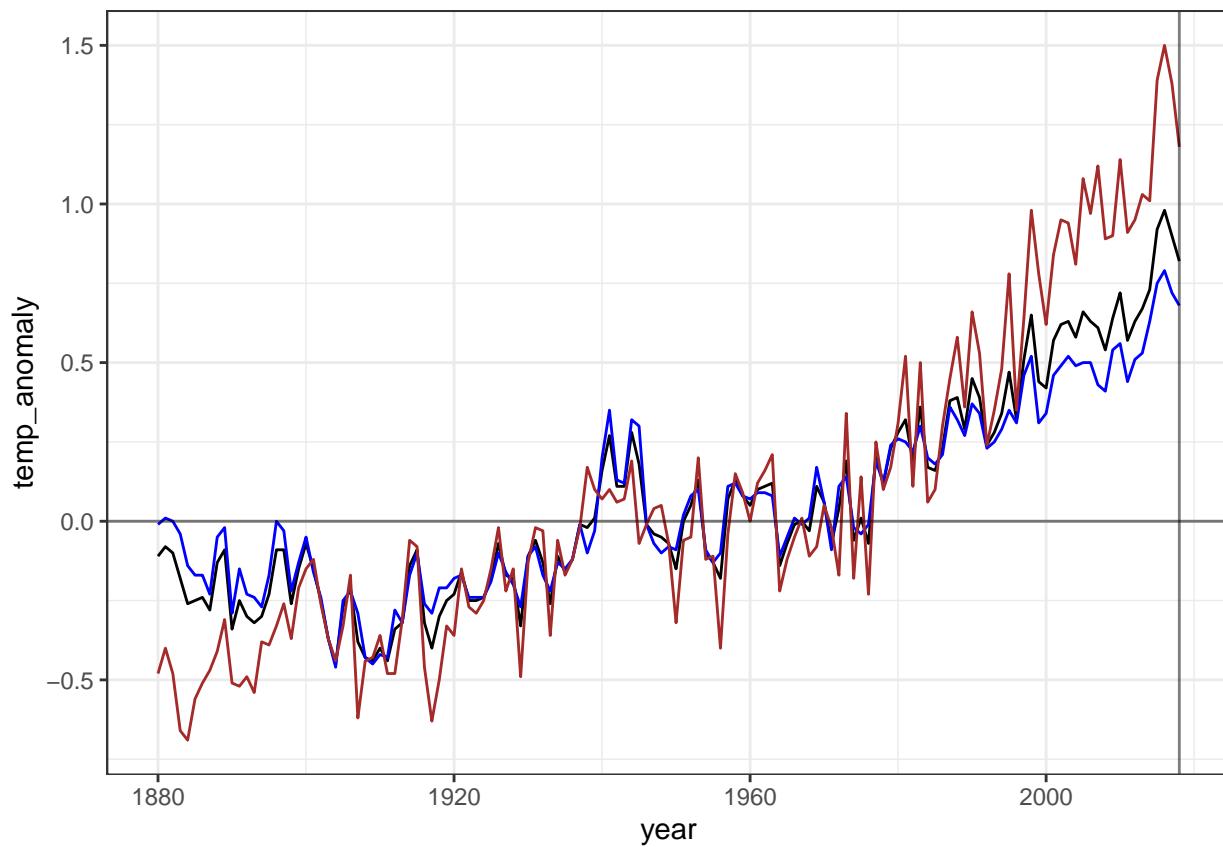
```
p + ylab("Temperature anomaly (degrees C)") +
  ggtitle("Temperature anomaly relative to 20th century mean, 1880-2018") +
  geom_text(aes(x = 2003, y = 0.05, label = "20th century mean"), col = "blue") +
  geom_hline(aes(yintercept = mean(media$temp_anomaly)), col = "blue") +
  geom_vline(aes(xintercept=1939),col="red") +
  geom_vline(aes(xintercept=1976),col="orange") +
  geom_text(aes(x=1933,y=0.05,label="1939"),col="red") +
  geom_text(aes(x=1981,y=0.05,label="1976"),col="orange")
```



### Pregunta 15.

Add layers to the previous plot to include line graphs of the temperature anomaly in the ocean (ocean\_anomaly) and on land (land\_anomaly). Assign different colors to the lines. Compare the global temperature anomaly to the land temperature anomaly and ocean temperature anomaly.

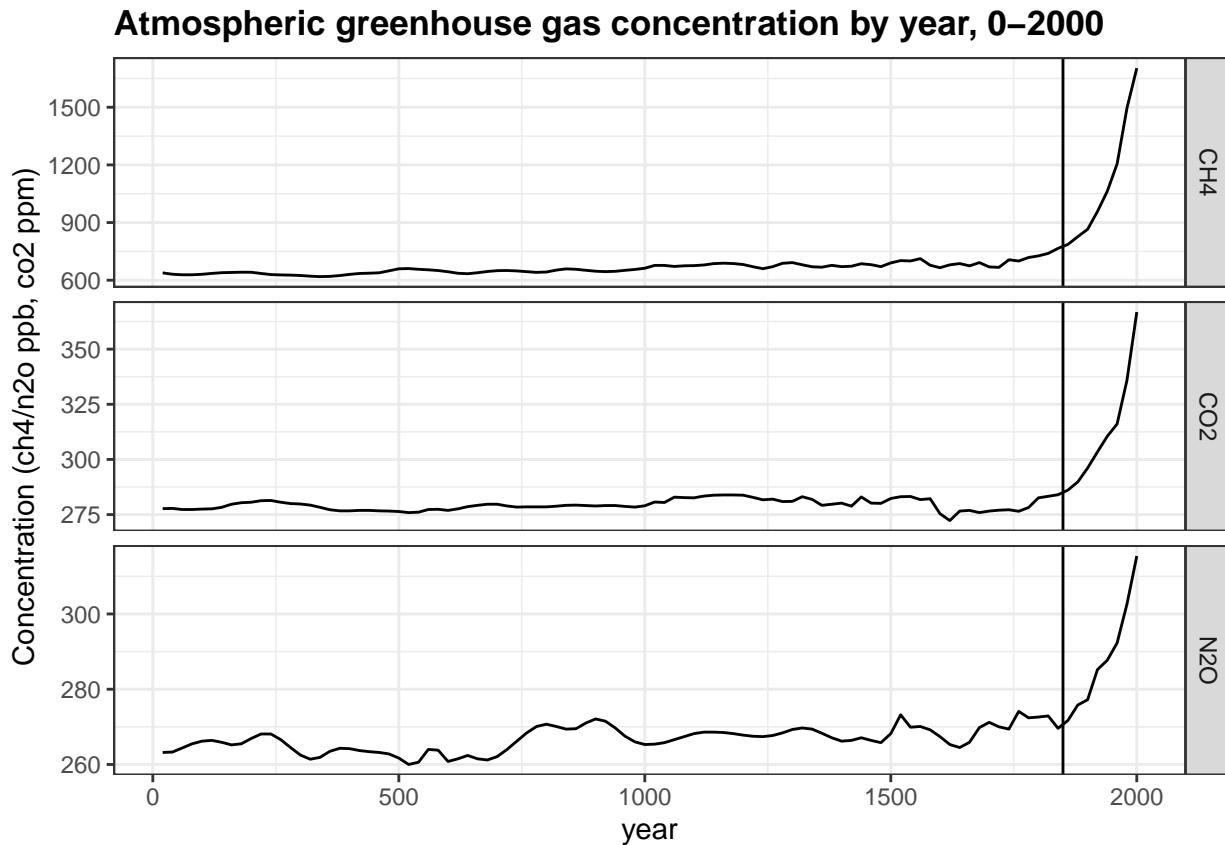
```
temp_carbon %>%
  filter(year %in% c(1880:2020)) %>%
  ggplot() +
  geom_line(aes(year,temp_anomaly)) +
  geom_line(aes(year,ocean_anomaly),col="blue") +
  geom_line(aes(year,land_anomaly), col="brown") +
  geom_hline(yintercept = 0, col="black",alpha=0.5) +
  geom_vline(xintercept = 2018, col="black",alpha=0.5)
```



#### Pregunta 16.

Complete the code outline below to make a line plot of concentration on the y-axis by year on the x-axis. Facet by gas, aligning the plots vertically so as to ease comparisons along the year axis. Add a vertical line with an x-intercept at the year 1850, noting the unofficial start of the industrial revolution and widespread fossil fuel consumption. Note that the units for ch4 and n2o are ppb while the units for co2 are ppm.

```
greenhouse_gases %>%
  ggplot(aes(year, concentration)) +
  geom_line() +
  facet_grid(gas ~ ., scales = "free") +
  geom_vline(xintercept = 1850) +
  ylab("Concentration (ch4/n2o ppb, co2 ppm)") +
  ggtitle("Atmospheric greenhouse gas concentration by year, 0-2000")
```

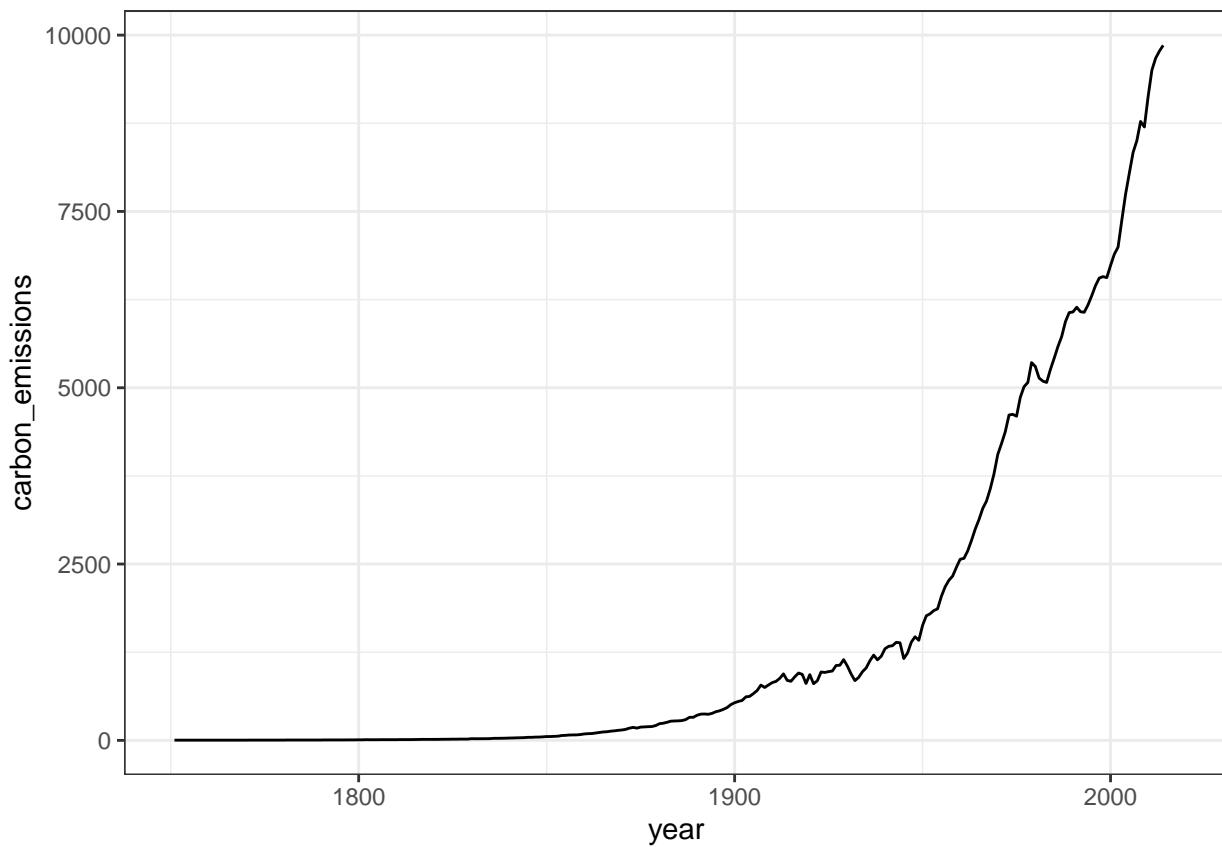


Pregunta 17.

While many aspects of climate are independent of human influence, and co2 levels can change without human intervention, climate models cannot reconstruct current conditions without incorporating the effect of manmade carbon emissions. These emissions consist of greenhouse gases and are mainly the result of burning fossil fuels such as oil, coal and natural gas. Make a time series line plot of carbon emissions (carbon\_emissions) from the temp\_carbon dataset. The y-axis is metric tons of carbon emitted per year.

```
temp_carbon %>%
  ggplot(aes(year, carbon_emissions)) +
  geom_line()

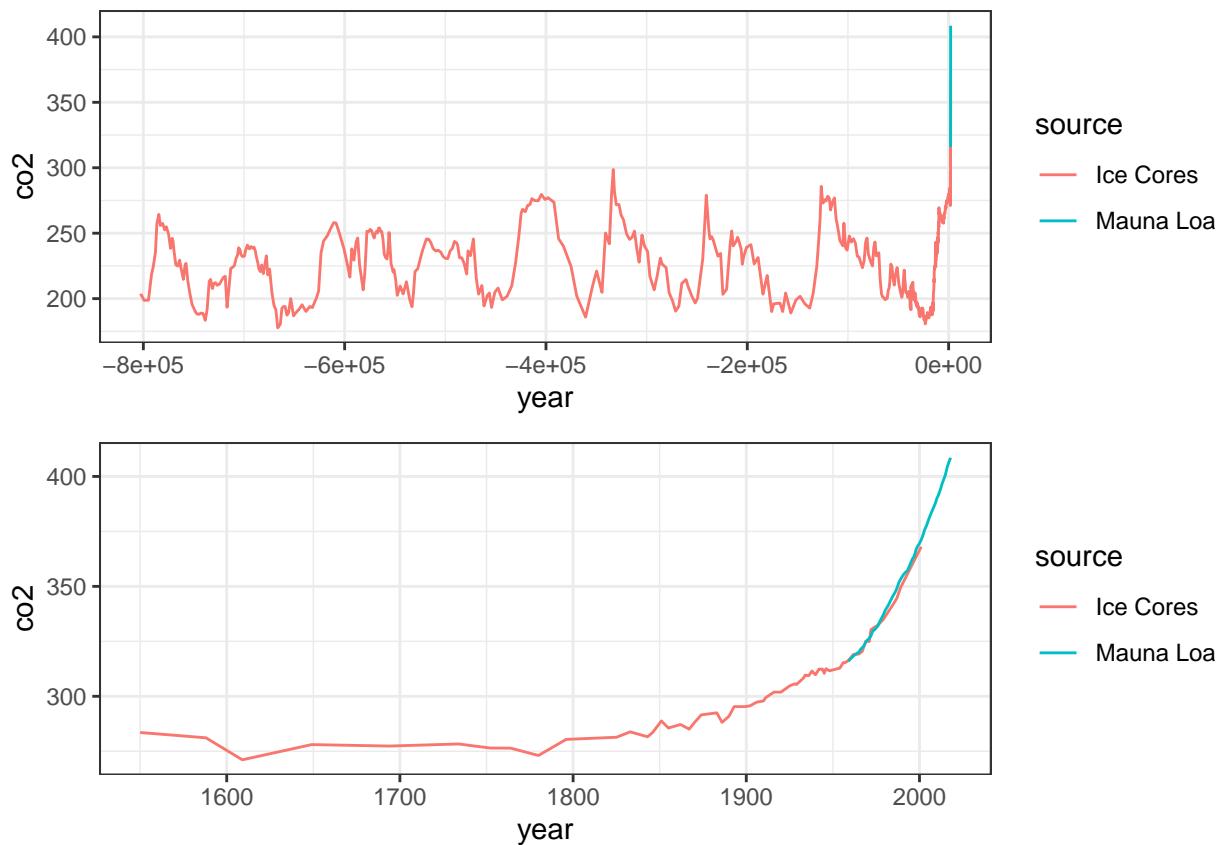
## Warning: Removed 4 row(s) containing missing values (geom_path).
```

**Pregunta 18.**

```
co2_time <- historic_co2 %>%
  ggplot(aes(year,co2,color=source)) +
  geom_line()

co2_time_recent <- historic_co2 %>%
  filter(year>1500) %>%
  ggplot(aes(year,co2,color=source)) +
  geom_line()

grid.arrange(co2_time,co2_time_recent)
```

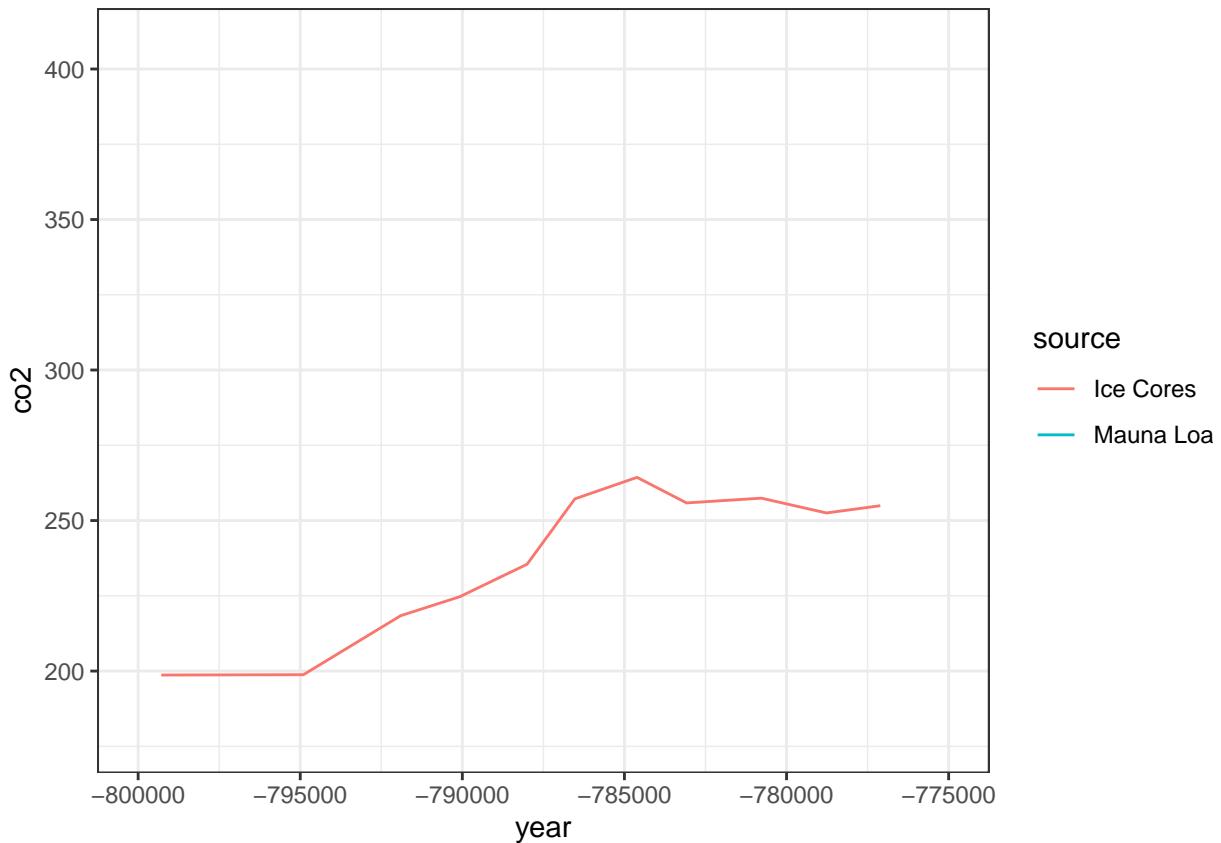


### Pregunta 19.

One way to differentiate natural co2 oscillations from today's manmade co2 spike is by examining the rate of change of co2. The planet is affected not only by the absolute concentration of co2 but also by its rate of change. When the rate of change is slow, living and nonliving systems have time to adapt to new temperature and gas levels, but when the rate of change is fast, abrupt differences can overwhelm natural systems. How does the pace of natural co2 change differ from the current rate of change? Use the co2\_time plot saved above. Change the limits as directed to investigate the rate of change in co2 over various periods with spikes in co2 concentration.

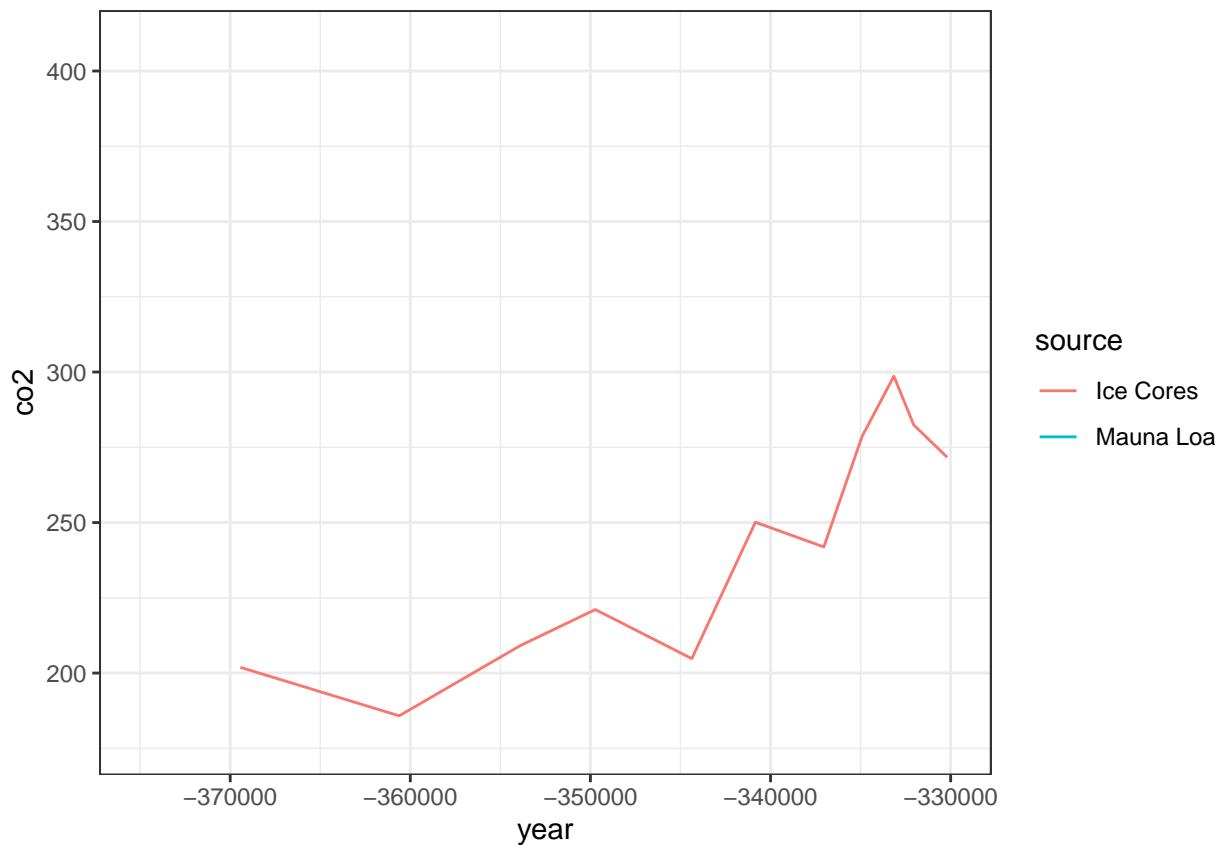
```
historic_co2 %>%
  ggplot(aes(year,co2,color=source)) +
  geom_line() +
  xlim(-800000,-775000)

## Warning: Removed 683 row(s) containing missing values (geom_path).
```



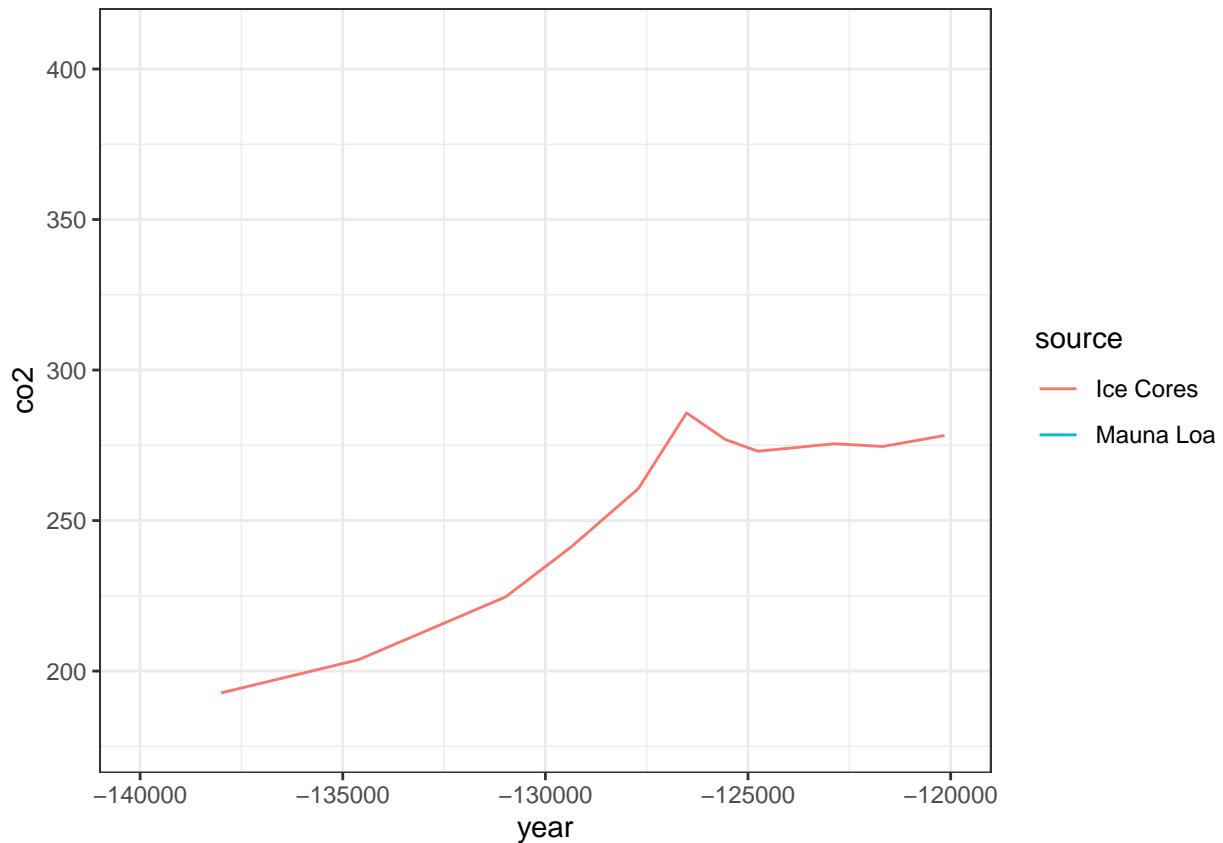
```
historic_co2 %>%
  ggplot(aes(year, co2, color=source)) +
  geom_line() +
  xlim(-375000, -330000)
```

```
## Warning: Removed 683 row(s) containing missing values (geom_path).
```



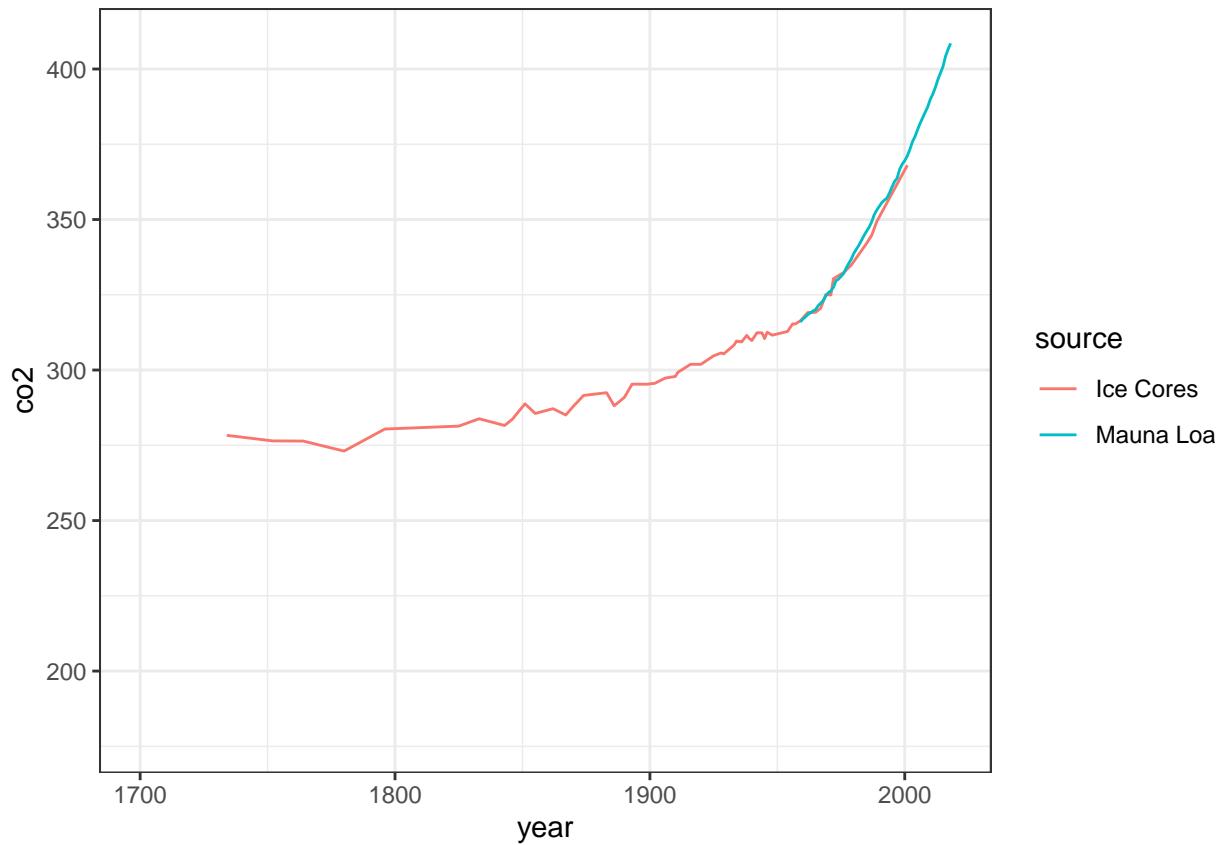
```
historic_co2 %>%
  ggplot(aes(year, co2, color=source)) +
  geom_line() +
  xlim(-140000, -120000)

## Warning: Removed 683 row(s) containing missing values (geom_path).
```



```
historic_co2 %>%
  ggplot(aes(year, co2, color=source)) +
  geom_line() +
  xlim(1700, 2018)
```

```
## Warning: Removed 573 row(s) containing missing values (geom_path).
```



### 3. Probabilidad

#### 3.1. Probabilidad discreta

##### 3.1.1. Introducción

###### 3.1.1.1. Notación

- Probabilidad de ocurrencia de un evento A:

$$Pr(A)$$

- Un evento es la ocurrencia de algo por probabilidad.

**3.1.1.2. Simulaciones Monte Carlo** Para simular un experimento probabilístico, R utiliza la función “sample()”, la cual genera un evento de probabilidad aleatorio a partir de una muestra aleatoria.

```
beads <- rep(c("red", "blue"), times = c(2,3))

beads

## [1] "red"   "red"   "blue"  "blue"  "blue"

sample(beads, 1)

## [1] "red"

sample(beads, 2)

## [1] "blue"  "blue"
```

Dado que no se puede repetir un experimento infinitamente, se hará un número lo más grande que se pueda: esto es una **simulación Monte Carlo**. Para esto, la función “replicate()” permite repetir una tarea el número de veces que necesitemos:

```
B <- 10000

events <- replicate(B, sample(beads, 1))

Así, podemos analizar si la simulación se acerca a las probabilidades teóricas que deberían ocurrir:
```

```
tab <- table(events)
```

```
tab
```

```
## events
## blue red
## 6039 3961

prop.table(tab)
```

```
## events
## blue red
## 0.604 0.396
```

Nótese que la selección ocurre sin reemplazo; sin embargo, puede indicársele a la función que lo haga:

```
events <- sample(beads, B, replace = TRUE)

tab <- table(events)

prop.table(tab)
```

```

## events
## blue red
## 0.599 0.401

3.1.1.3. Función mean() Recordemos que aplicar la función “mean()” a un factor lógico arroja la proporción de elementos que son TRUE. Así, resulta útil utilizar esta función para calcular probabilidades:
```

```

beads <- rep(c("red", "blue"), times = c(2,3))

beads

## [1] "red"   "red"   "blue"  "blue"  "blue"

mean(beads == "blue")

## [1] 0.6

```

**3.1.1.4. Distribuciones de probabilidad** En el caso de variables categóricas, se le asigna una probabilidad a cada categoría, por lo que la proporción de la ocurrencia de estas define su distribución.

**3.1.1.5. Independencia** Se dice que dos eventos son independiente si el resultado de un no afecta el resultado del otro.

```

x <- sample(beads, 5)

x[2:5]

## [1] "blue" "blue" "red"  "red"

```

A partir de esto, es pertinente definir la probabilidad condicional de un evento. Por ejemplo, la probabilidad de obtener un Rey de una baraja dado que ya se obtuvo uno es:

$$Pr(\text{Carta 2 es un Rey} | \text{Carta 1 fue un Rey}) = \frac{3}{51}$$

Por tanto, si los eventos A y B son independientes:

$$Pr(A|B) = Pr(A)$$

Por otro lado, para conocer la probabilidad de que dos eventos ocurran, sean A y B, se utiliza la regla de la multiplicación:

$$Pr(A \cup B) = Pr(A)Pr(B|A)$$

Supóngase un juego de *Blackjack*; las probabilidades de ganar con dos cartas son:

$$Pr(A's) * Pr(10, J, Q, K|A) = \left(\frac{1}{13}\right) \left(\frac{16}{51}\right) \approx 0,02 = 2\%$$

### 3.1.2. Combinaciones y permutaciones

¿Cuál es la probabilidad de obtener un *flush*? Primero, hay que construir una baraja, para lo cual podemos utilizar la función “expand.grid()” y “paste()”, donde la primera arroja todas las combinaciones posibles de dos listas de elementos, y la segunda las pega:

```
expand.grid(pants = c("blue", "black"), shirt = c("white", "grey", "plaid"))

##   pants shirt
## 1 blue white
## 2 black white
## 3 blue grey
## 4 black grey
## 5 blue plaid
## 6 black plaid

suits <- c("Diamonds", "Clubs", "Hearts", "Spades")

numbers <- c("Ace", "Deuce", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "King")

deck <- expand.grid(number = numbers, suit = suits)

deck <- paste(deck$number, deck$suit)
```

Corrobaremos que la probabilidad de obtener un Rey en la primera carta es 1/13:

```
kings <- paste("King", suits)

mean(deck %in% kings)

## [1] 0.0769
```

¿Y la probabilidad de que la segunda carta sea un Rey, dado que la primera también lo fue? Para esto, se utilizarán las funciones “combinations()”, y “permutations()” del paquete gtools.

```
library(gtools)

## Warning: package 'gtools' was built under R version 4.0.5

■ permutations() computa, para cualquier lista de tamaño N, todas las formas diferentes en que se pueden seleccionar r elementos:

permutations(5, 2)

##      [,1] [,2]
## [1,]    1    2
## [2,]    1    3
## [3,]    1    4
## [4,]    1    5
## [5,]    2    1
## [6,]    2    3
## [7,]    2    4
## [8,]    2    5
## [9,]    3    1
## [10,]   3    2
## [11,]   3    4
## [12,]   3    5
## [13,]   4    1
## [14,]   4    2
## [15,]   4    3
## [16,]   4    5
## [17,]   5    1
## [18,]   5    2
```

```
## [19,]    5    3
## [20,]    5    4
```

Por ejemplo, para generar 5 números posibles de 7 dígitos:

```
all_phone_numbers <- permutations(10, 7, v=0:9)
```

```
n <- nrow(all_phone_numbers)
```

```
index <- sample(n, 5)
```

```
all_phone_numbers[index, ]
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
## [1,]    0    3    7    8    6    2    1
## [2,]    5    1    4    0    8    3    9
## [3,]    4    5    1    7    8    9    0
## [4,]    5    1    8    4    2    0    3
## [5,]    5    4    7    3    1    2    8
```

Por tanto, para computar la probabilidad de obtener 2 cartas cuando el orden importa:

$$\Pr(B|A) = \frac{\Pr(A \cap B)}{\Pr(A)}$$

```
hands <- permutations(52, 2, v=deck)
```

```
first_card <- hands[,1]
```

```
second_hand <- hands[,2]
```

```
sum(first_card %in% kings)
```

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

```
mean(first_card %in% kings & second_hand %in% kings) / mean(first_card %in% kings)
```

```
## [1] 0.0588
```

Para calcular la probabilidad de un 21 natural en *Blackjack*:

```
aces <- paste("Ace", suits)
```

```
facecard <- c("King", "Queen", "Jack", "Ten")
```

```
facecard <- expand.grid(number = facecard, suit = suits)
```

```
facecard <- paste(facecard$number, facecard$suit)
```

```
hands <- combinations(52, 2, v = deck)
```

```
mean(hands[,1] %in% aces & hands[,2] %in% facecard)
```

```
## [1] 0.0483
```

Esta misma probabilidad se puede obtener mediante una simulación Monte Carlo:

```
hand <- sample(deck, 2)
```

```
B <- 10000
```

```

results <- replicate(B, {
  hand <- sample(deck, 2)
  (hand[1] %in% aces & hand[2] %in% facecard) |
  (hand[2] %in% aces & hand[1] %in% facecard)
})

mean(results)

## [1] 0.0469

```

**3.1.2.1. El problema del cumpleaños** Supóngase un salón con 50 personas, ¿cuál es la probabilidad de que, al menos, 2 personas tengan el mismo cumpleaños? Aquí será pertinente la función “duplicated()” que arroja TRUE si algún elemento de un vector es igual a otro.

```

n <- 50

bdays <- sample(1:365, n, replace=TRUE)

any(duplicated(bdays))

## [1] TRUE

Repitamos el experimento 10 000 veces:
B <- 10000

results <- replicate(B, {
  bdays <- sample(1:365, n, replace = TRUE)
  any(duplicated(bdays))
})

mean(results)

## [1] 0.97

```

**3.1.2.2. sapply** El mismo experimento puede replicarse para diferentes grupos con distintos tamaños:

```

compute_prob <- function(n, B=10000){
  same_day <- replicate(B, {
    bdays <- sample(1:365, n, replace = TRUE)
    any(duplicated(bdays))
  })

  mean(same_day)
}

```

Veremos cómo se comporta esta simulación para  $n = 1,..,60$ ; para ello, si bien se puede utilizar un bucle, se prefiere realizar la operación con vectores completos. Así, la función “sapply()” será útil, la cual permite aplicar operaciones element-wise a cualquier función:

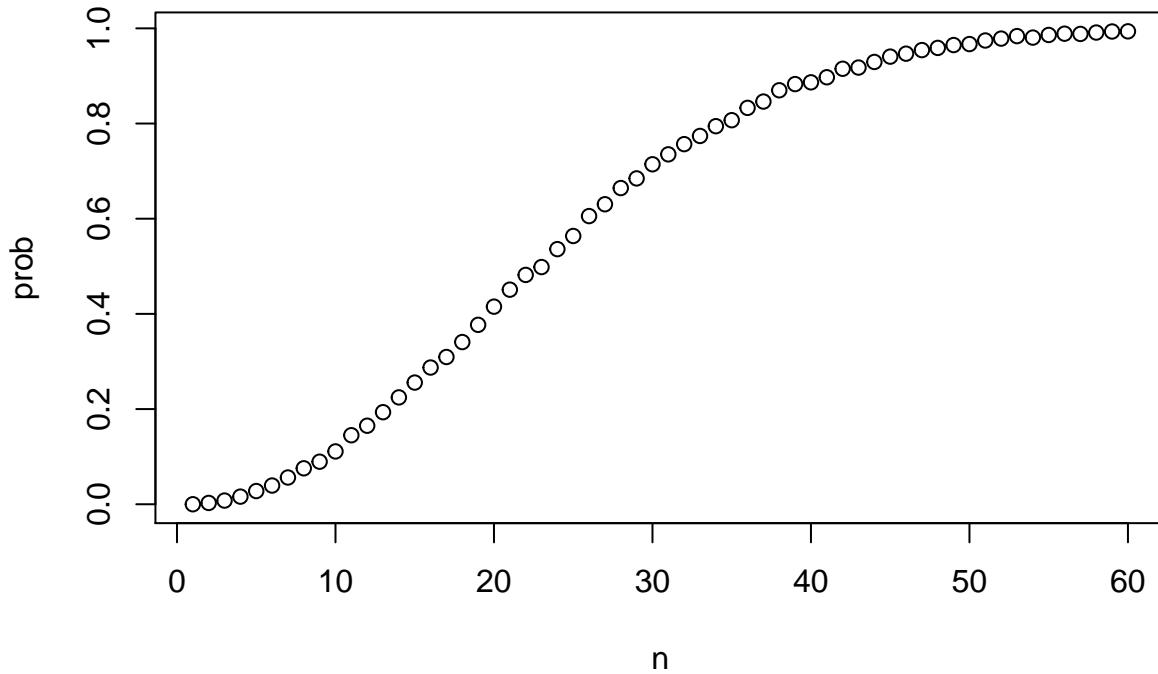
```

n <- seq(1, 60)

prob <- sapply(n, compute_prob)

plot(n, prob)

```



Ahora, computaremos las probabilidades exactas, en vez de simulaciones Monte Carlo. Computaremos la probabilidad de que no ocurra el evento:

Para la primera persona, la probabilidad de que tenga un cumpleaños único es 1:

$$Pr(A_1) = 1$$

La probabilidad de que la segunda persona tenga un cumpleaños único dado que la primera tuvo un cumpleaños único es:

$$Pr(A_2|A_1) = \frac{364}{365}$$

Para la persona 3:

$$Pr(A_3|A_1, A_2) = \frac{363}{365}$$

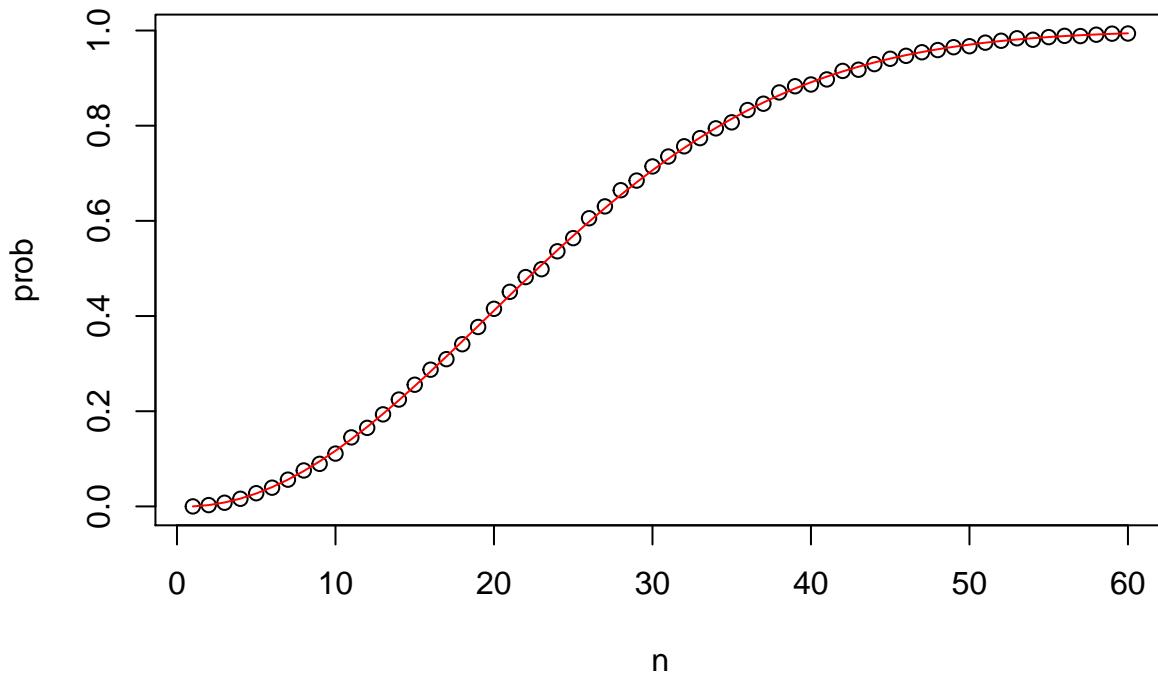
Así, sucesivamente:

$$(1) \left( \frac{364}{365} \right) \left( \frac{363}{365} \right) \dots \left( \frac{365-n-1}{365} \right)$$

```
exact_prob <- function(n){
  prob_unique <- seq(365, 365 - n + 1)/365
  1 - prod(prob_unique)
}
```

```
eprob <- sapply(n, exact_prob)

plot(n, prob)
lines(n, eprob, col="red")
```



**3.1.2.3. ¿Cuántos experimentos Monte Carlo son suficientes?** En los experimentos realizados, se han utilizado 10 000 repeticiones; sin embargo, en algunos cálculos, este número no es ni cercanamente suficiente, o bien, no es matemáticamente verosímil. En general:

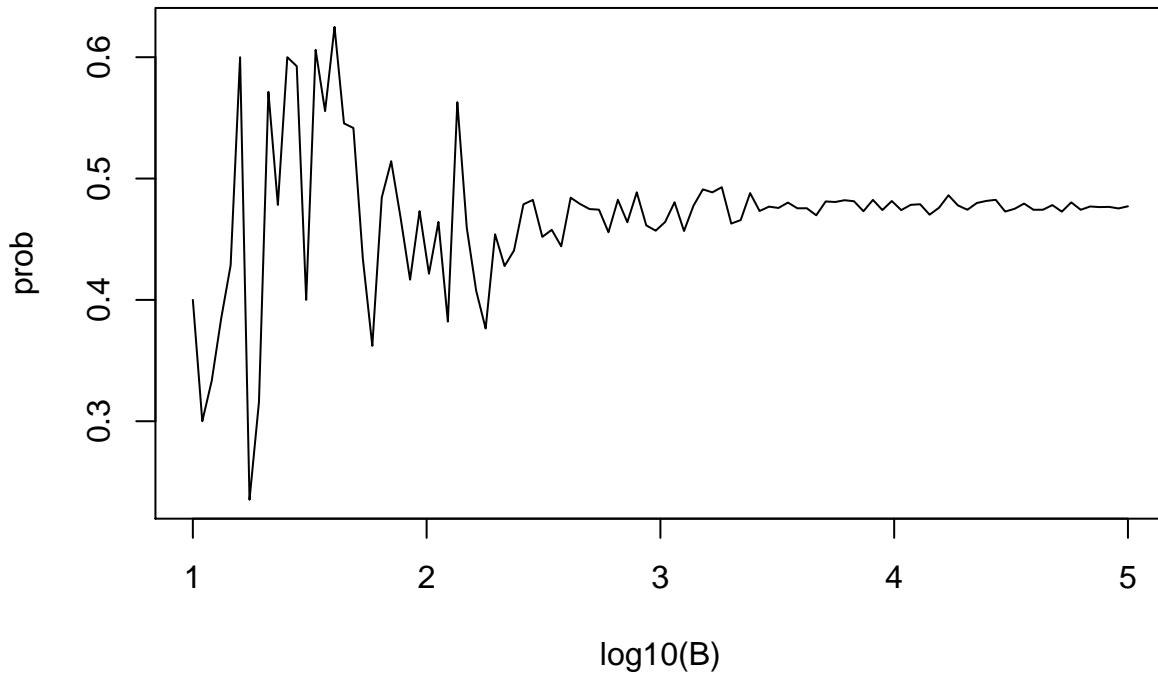
- Mientras mayor el número que Monte Carlo replica, B, mayor la exactitud del estimado.
- La determinación del tamaño apropiado de B puede requerir estadística avanzada.
- Un enfoque práctico es probar varios tamaños de B e identificar aquellos que proveen estimados estables.

```
B <- 10^seq(1, 5, len = 100)

compute_prob <- function(B, n = 22){
  same_day <- replicate(B, {
    bdays <- sample(1:365, n, replace = TRUE)
    any(duplicated(bdays))
  })
  mean(same_day)
}

prob <- sapply(B, compute_prob)
```

```
plot(log10(B), prob, type="l")
```



Donde puede observarse que, alrededor de  $B = 1\,000$ , el estimado comienza a estabilizarse.

### 3.1.3. Regla de adición

Con la regla de la adición, también se puede calcular la probabilidad de ganar al *Blackjack* en la primera mano. Es evidente que lo que queremos obtener es la probabilidad de que ocurra el evento A o el B, donde A es sacar un A's y luego una *facecard* (10, J, Q, K) y B, viceversa. Así:

$$Pr(A \text{ o } B) = Pr(A) + Pr(B) - Pr(A \text{ y } B)$$

De manera equivalente e intuitiva de acuerdo a la teoría de conjuntos:

$$\boxed{Pr(A \cup B) = Pr(A) + Pr(B) - Pr(A \cap B)}$$

Sabemos que la probabilidad de obtener un A's seguido de una *facecard* es  $(\frac{1}{13})(\frac{16}{51})$ . La probabilidad de una *facecard* seguida de un A's es  $(\frac{16}{52})(\frac{4}{51})$  (nótese que ambas son iguales). Por lo tanto:

$$Pr(A \cup B) = \left(\frac{1}{13}\right)\left(\frac{16}{51}\right) + \left(\frac{16}{52}\right)\left(\frac{4}{51}\right) - 0 = 0,0483$$

### 3.1.4. Monty Hall

Supóngase un concurso de televisión donde eliges entre lo que se encuentra detrás de tres puertas. Detrás de una de ellas hay un premio y detrás de las otras dos, una cabra. El juego funciona de la siguiente manera:

- Si el concursante no escoge el premio en su primera elección, Monty Hall descubre una de las dos restantes con una cabra.
- El concursante tiene la oportunidad de cambiar de elección o quedarse con la que eligió desde el principio.

Así, puede usarse probabilidad para mostrar que el quedarse con la misma puerta implica una probabilidad de ganar de  $1/3$ , mientras que el cambiar implica una probabilidad de ganar de  $2/3$ . Para probarlo, utilizaremos una simulación Monte Carlo:

```
# Quedarse con la misma puerta:
```

```
B <- 10000

stick <- replicate(B, {
  doors <- as.character(1:3)
  prize <- sample(c("Car", "Goat", "Goat"))
  prize_door <- doors[prize == "Car"]
  my_pick <- sample(doors, 1)
  show <- sample(doors[!doors %in% c(my_pick, prize_door)], 1)
  stick <- my_pick
  stick == prize_door
})

mean(stick)

## [1] 0.324
```

Puede verse que en un tercio de las ocasiones, la elección de la puerta es la ganadora. Repítase el ejercicio si se cambia de puerta:

```
# Cambiar de puerta:
```

```
B <- 10000

switch <- replicate(B, {
  doors <- as.character(1:3)
  prize <- sample(c("Car", "Goat", "Goat"))
  prize_door <- doors[prize == "Car"]
  my_pick <- sample(doors, 1)
  show <- sample(doors[!doors %in% c(my_pick, prize_door)], 1)
  stick <- my_pick
  switch <- doors[!doors %in% c(my_pick, show)]
  switch == prize_door
})

mean(switch)

## [1] 0.667
```

Así, se demuestra que, en  $2/3$  de las ocasiones, cambiar de puerta resulta una mejor estrategia.

### 3.1.5. Assessment 5

```
library(gtools)
library(tidyverse)
```

In the 200m dash finals in the Olympics, 8 runners compete for 3 medals (order matters). In

the 2012 Olympics, 3 of the 8 runners were from Jamaica and the other 5 were from different countries. The three medals were all won by Jamaica (Usain Bolt, Yohan Blake, and Warren Weir).

### Pregunta 1.

How many different ways can the 3 medals be distributed across 8 runners?

```
str(permuations(8,3))
```

```
##  int [1:336, 1:3] 1 1 1 1 1 1 1 1 1 1 ...
```

How many different ways can the three medals be distributed among the 3 runners from Jamaica?

```
str(permuations(3,3))
```

```
##  int [1:6, 1:3] 1 1 2 2 3 3 2 3 1 3 ...
```

What is the probability that all 3 medals are won by Jamaica?

```
6/336
```

```
## [1] 0.0179
```

Run a Monte Carlo simulation on this vector representing the countries of the 8 runners in this race:

```
runners <- c("Jamaica", "Jamaica", "Jamaica", "USA", "Ecuador", "Netherlands", "France", "South Africa")
```

For each iteration of the Monte Carlo simulation, within a replicate() loop, select 3 runners representing the 3 medalists and check whether they are all from Jamaica. Repeat this simulation 10,000 times. Set the seed to 1 before running the loop. Calculate the probability that all the runners are from Jamaica.

```
set.seed(1)
```

```
B <- 10000
```

```
results <- replicate(B, {
  winners <- sample(runners, 3)
  (winners[1] %in% "Jamaica" & winners[2] %in% "Jamaica" & winners[3] %in% "Jamaica")
})
```

```
mean(results)
```

```
## [1] 0.0174
```

### Pregunta 2.

A restaurant manager wants to advertise that his lunch special offers enough choices to eat different meals every day of the year. He doesn't think his current special actually allows that number of choices, but wants to change his special if needed to allow at least 365 choices. A meal at the restaurant includes 1 entree, 2 sides, and 1 drink. He currently offers a choice of 1 entree from a list of 6 options, a choice of 2 different sides from a list of 6 options, and a choice of 1 drink from a list of 2 options.

How many meal combinations are possible with the current menu?

```
str(combinations(6,2))
```

```
##  int [1:15, 1:2] 1 1 1 1 1 2 2 2 2 3 ...
```

**6\*15\*2**

```
## [1] 180
```

The manager has one additional drink he could add to the special. How many combinations are possible if he expands his original special to 3 drink options?

**6\*15\*3**

```
## [1] 270
```

The manager decides to add the third drink but needs to expand the number of options. The manager would prefer not to change his menu further and wants to know if he can meet his goal by letting customers choose more sides. How many meal combinations are there if customers can choose from 6 entrees, 3 drinks, and select 3 sides from the current 6 options?

```
str(combinations(6, 3))
```

```
## int [1:20, 1:3] 1 1 1 1 1 1 1 1 1 1 ...
```

**6\*3\*20**

```
## [1] 360
```

The manager is concerned that customers may not want 3 sides with their meal. He is willing to increase the number of entree choices instead, but if he adds too many expensive options it could eat into profits. He wants to know how many entree choices he would have to offer in order to meet his goal.

Write a function that takes a number of entree choices and returns the number of meal combinations possible given that number of entree options, 3 drink choices, and a selection of 2 sides from 6 options.

Use `sapply()` to apply the function to entree option counts ranging from 1 to 12.

What is the minimum number of entree options required in order to generate more than 365 combinations?

```
f <- function(entree){  
  print(3*15*entree)  
}  
  
options <- seq(1:12)  
  
sapply(options, f)
```

```
## [1] 45
```

```
## [1] 90
```

```
## [1] 135
```

```
## [1] 180
```

```
## [1] 225
```

```
## [1] 270
```

```
## [1] 315
```

```
## [1] 360
```

```
## [1] 405
```

```
## [1] 450
```

```
## [1] 495
```

```
## [1] 540
```

```
## [1] 45 90 135 180 225 270 315 360 405 450 495 540
```

The manager isn't sure he can afford to put that many entree choices on the lunch menu and thinks it would be cheaper for him to expand the number of sides. He wants to know how many sides he would have to offer to meet his goal of at least 365 combinations.

Write a function that takes a number of side choices and returns the number of meal combinations possible given 6 entree choices, 3 drink choices, and a selection of 2 sides from the specified number of side choices.

Use `sapply()` to apply the function to side counts ranging from 2 to 12.

What is the minimum number of side options required in order to generate more than 365 combinations?

```
f <- function(sides){
  3*6*nrow(combinations(sides, 2))
}

options <- 2:12

sapply(options, f)

## [1] 18 54 108 180 270 378 504 648 810 990 1188
```

### Preguntas 3 y 4

Case-control studies help determine whether certain exposures are associated with outcomes such as developing cancer. The built-in dataset `esoph` contains data from a case-control study in France comparing people with esophageal cancer (cases, counted in `ncases`) to people without esophageal cancer (controls, counted in `ncontrols`) that are carefully matched on a variety of demographic and medical characteristics. The study compares alcohol intake in grams per day (`alcgp`) and tobacco intake in grams per day (`tobgp`) across cases and controls grouped by age range (`agegp`).

```
data(esoph)
head(esoph)

##   agegp     alcgp     tobgp ncases ncontrols
## 1 25-34 0-39g/day 0-9g/day      0       40
## 2 25-34 0-39g/day 10-19       0       10
## 3 25-34 0-39g/day 20-29       0        6
## 4 25-34 0-39g/day 30+         0        5
## 5 25-34    40-79 0-9g/day      0       27
## 6 25-34    40-79 10-19       0        7
```

How many groups are in the study?

```
nrow(esoph)
```

```
## [1] 88
```

How many cases are there?

```
all_cases <- sum(esoph$ncases)

all_cases
```

```
## [1] 200
```

How many controls are there?

```
all_controls <- sum(esoph$ncontrols)

all_controls
```

```
## [1] 975
```

What is the probability that a subject in the highest alcohol consumption group is a cancer case?

```
esoph %>%
  filter(alcgp == "120+") %>%
  summarize(sum_cases = sum(ncases), tot=sum(ncontrols) + sum(ncases), probability = sum_cases/tot)

##   sum_cases tot probability
## 1      45 112      0.402
```

What is the probability that a subject in the lowest alcohol consumption group is a cancer case?

```
esoph %>%
  filter(alcgp == "0-39g/day") %>%
  summarize(sum_cases = sum(ncases), tot=sum(ncontrols) + sum(ncases), probability = sum_cases/tot)

##   sum_cases tot probability
## 1      29 444      0.0653
```

Given that a person is a case, what is the probability that they smoke 10g or more a day?

```
esoph %>%
  summarize(tot_cases = sum(ncases))

##   tot_cases
## 1      200

esoph %>% filter(tobgp != "0-9g/day") %>%
  summarize(smoking10_cases = sum(ncases))

##   smoking10_cases
## 1          122
```

```
122/200
```

```
## [1] 0.61
```

Given that a person is a control, what is the probability that they smoke 10g or more a day?

```
esoph %>%
  summarize(tot_cases = sum(ncontrols))

##   tot_cases
## 1      975

esoph %>%
  filter(tobgp != "0-9g/day") %>%
  summarize(smoking20_cases = sum(ncontrols))

##   smoking20_cases
## 1          450
```

```
450/975
```

```
## [1] 0.462
```

**Preguntas 5 y 6.**

For cases, what is the probability of being in the highest alcohol group?

```
esoph %>%
  filter(alcgp == "120+") %>%
  summarize(sum_cases = sum(ncases))

##   sum_cases
## 1      45
45/all_cases

## [1] 0.225
```

For cases, what is the probability of being in the highest tobacco group?

```
esoph %>%
  filter(tobgp == "30+") %>%
  summarize(sum_cases = sum(ncases))

##   sum_cases
## 1      31
31/all_cases

## [1] 0.155
```

For cases, what is the probability of being in the highest alcohol group and the highest tobacco group?

```
esoph %>%
  filter(tobgp == "30+" & alcgp == "120+") %>%
  summarize(sum_cases = sum(ncases))

##   sum_cases
## 1      10
10/all_cases

## [1] 0.05
```

For cases, what is the probability of being in the highest alcohol group or the highest tobacco group?

```
esoph %>%
  filter(alcgp == "120+" | tobgp == "30+") %>%
  summarize(sum_cases = sum(ncases))

##   sum_cases
## 1      66
66/all_cases

## [1] 0.33
```

For controls, what is the probability of being in the highest alcohol group?

```
esoph %>%
  filter(alcgp == "120+") %>%
  summarize(contr_sum = sum(ncontrols), probability = contr_sum/all_controls)

##   contr_sum probability
```

```
## 1      67     0.0687
```

How many times more likely are cases than controls to be in the highest alcohol group?

```
esoph %>%
  filter(alcgp == "120+") %>%
  summarize(contr_sum = sum(ncontrols), case_sum = sum(ncases),
            co_prob = contr_sum/all_controls, ca_prob = case_sum/all_cases,
            ratio = ca_prob/co_prob)
```

```
##   contr_sum case_sum co_prob ca_prob ratio
## 1       45     45  0.225  0.0687  3.27
```

For controls, what is the probability of being in the highest tobacco group?

```
esoph %>%
  filter(tobgp == "30+") %>%
  summarize(contr_sum = sum(ncontrols), probability = contr_sum/all_controls)
```

```
##   contr_sum probability
## 1       82      0.0841
```

For controls, what is the probability of being in the highest alcohol group and the highest tobacco group?

```
esoph %>%
  filter(tobgp == "30+" & alcgp == "120+") %>%
  summarize(contr_sum = sum(ncontrols), probability=contr_sum/all_controls)
```

```
##   contr_sum probability
## 1       13      0.0133
```

For controls, what is the probability of being in the highest alcohol group or the highest tobacco group?

```
esoph %>%
  filter(tobgp == "30+" | alcgp == "120+") %>%
  summarize(contr_sum = sum(ncontrols), probability=contr_sum/all_controls)
```

```
##   contr_sum probability
## 1       136     0.139
```

How many times more likely are cases than controls to be in the highest alcohol group or the highest tobacco group?

```
esoph %>%
  filter(alcgp == "120+" | tobgp == "30+") %>%
  summarize(contr_sum = sum(ncontrols), case_sum = sum(ncases),
            co_prob = contr_sum/all_controls, ca_prob = case_sum/all_cases,
            ratio = ca_prob/co_prob)
```

```
##   contr_sum case_sum co_prob ca_prob ratio
## 1       136      66  0.139    0.33  2.37
```

### 3.2. Probabilidad continua

En vez de definir las probabilidades para valores específicos, es más útil definir las probabilidades que toman intervalos de valores. La forma estándar de hacer esto es mediante la Función de Probabilidad Acumulada (CDF).

```
library(tidyverse)
library(dslabs)

data("heights")
```

Defínase un vector con las alturas masculinas:

```
x <- heights %>%
  filter(sex == "Male") %>%
  .\$height
```

Así, puede definirse una función de distribución acumulada empírica: para un valor de  $a$ , arrojará la proporción de valores del vector menores o iguales a  $a$

```
F <- function(a) mean(x <= a)
```

Si se escoge a un individuo al azar, ¿cuál es la probabilidad de que sea más alto que 70.5 pulgadas? Dado que cada individuo tiene la misma probabilidad de ser escogido, entonces queremos encontrar la proporción de estudiantes más altos de 70.5:

$$Pr(x > 70.5) = 1 - Pr(x \leq 70.5) = 1 - \Phi(70.5) = 0.3633$$

```
1-F(70.5)
```

```
## [1] 0.363
```

De esta forma, puede encontrarse la probabilidad para cualquier intervalo; por ejemplo, la probabilidad de que un individuo sea más alto que un valor  $a$  y más bajo que un valor  $b$ :

$$\boxed{F(b) - F(a)}$$

#### 3.2.1. Distribución teórica

Recordemos que la distribución acumulada de una distribución normal está definida por una fórmula matemática, la cual puede ser obtenida con la función “pnorm()” en R.

Se dice que una cantidad aleatoria se distribuye normalmente con media  $\mu$  y desviación estándar  $\sigma$  si su distribución de probabilidad está definida como:

$$F(a) = pnorm(a, avg, s)$$

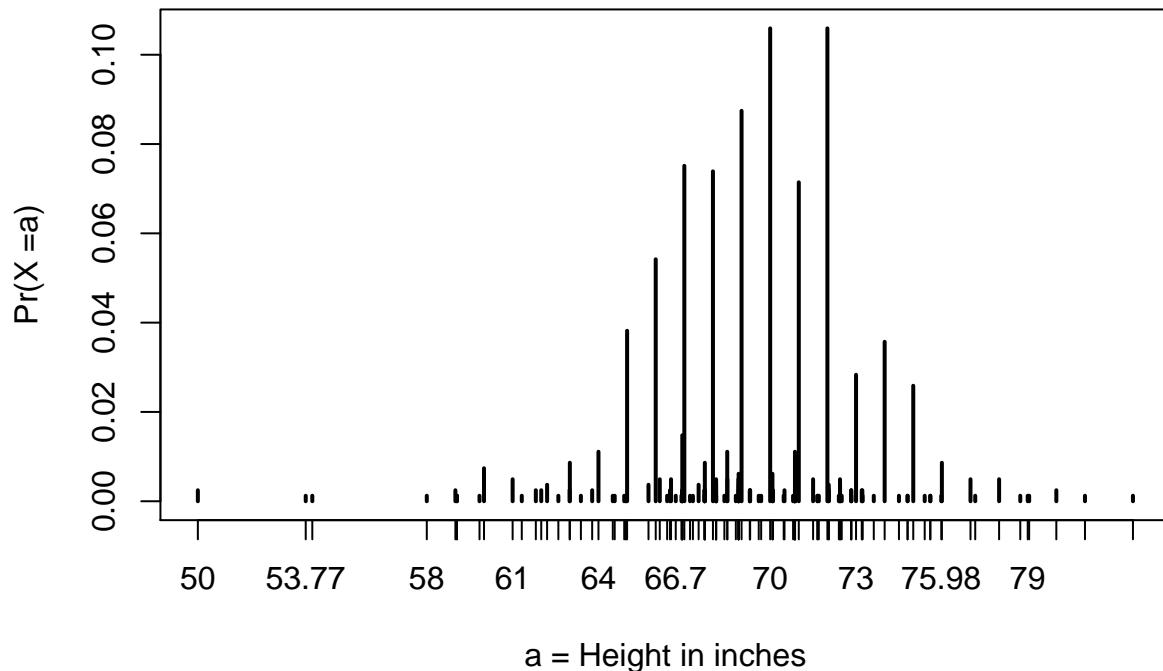
Retomemos el ejemplo que habíamos revisado: puede verse que, efectivamente,  $x \sim N(\mu, \sigma)$

```
1 - pnorm(70.5, mean(x), sd(x))
```

```
## [1] 0.371
```

A partir de la base de datos, piénsese en cada altura específica como una categoría única. Así, la distribución de probabilidad estará definida como la proporción de estudiantes que reportan cada una de estas alturas únicas. A continuación se muestra el gráfico de esto:

```
plot(prop.table(table(x)), xlab = "a = Height in inches", ylab = "Pr(X =a)")
```



Dado que la altura es una variable evidentemente continua, las probabilidades deben definirse a partir de intervalos.

Por ejemplo, repórtese la proporción de individuos con altura de 67.5 - 68.5, de 68.5 - 69.5 y de 69.5 - 70.5:

```
mean(x <= 68.5) - mean(x <= 67.5)
```

```
## [1] 0.115
```

```
mean(x <= 69.5) - mean(x <= 68.5)
```

```
## [1] 0.119
```

```
mean(x <= 70.5) - mean(x <= 69.5)
```

```
## [1] 0.122
```

Es importante recalcar que estos valores surgen de los datos empíricos y no de una aproximación. Para obtener estos últimos:

```
pnorm(68.5, mean(x), sd(x)) - pnorm(67.5, mean(x), sd(x))
```

```
## [1] 0.103
```

```
pnorm(69.5, mean(x), sd(x)) - pnorm(68.5, mean(x), sd(x))
```

```
## [1] 0.11
```

```
pnorm(70.5, mean(x), sd(x)) - pnorm(69.5, mean(x), sd(x))
```

```
## [1] 0.108
```

Es claro que los valores son sustantivamente similares, por lo que se concluye que, para los intervalos dados, la aproximación normal es adecuada. Por otro lado, existen intervalos para los cuales una aproximación normal no es adecuada, específicamente, para aquellos que no incluyen un número entero:

```
mean(x <= 70.9) - mean(x <= 70.1)

## [1] 0.0222

pnorm(70.9, mean(x), sd(x)) - pnorm(70.1, mean(x), sd(x))

## [1] 0.0836
```

Esta situación es conocida como **discretización**: a pesar de que la distribución real de alturas es continua, los valores reportados tienden a ser más comunes en valores discretos (en este ejemplo particular, debido a que los individuos suelen redondear sus alturas).

### 3.2.2. Densidad de probabilidad

Supóngase el lanzamiento de un dado justo. La probabilidad de que  $x$  sea menor o igual a 4 está definida como:

$$F(4) = Pr(X \leq 4) = Pr(X = 4) + Pr(X = 3) + Pr(X = 2) + Pr(X = 1) = \frac{2}{3}$$

En contraste, para distribuciones continuas, la probabilidad de un valor particular no está definido. No obstante, existe una definición que tiene una interpretación similar.

La **densidad de probabilidad** de  $x$  está definida como:

$$F(a) = \Pr(X \leq a) = \int_{-\infty}^a f(x)dx$$

Así, utilizando la aproximación normal para estimar la probabilidad de que un individuo sea más alto que 76 pulgadas:

```
avg <- mean(x)
s <- sd(x)
1 - pnorm(76, avg, s)

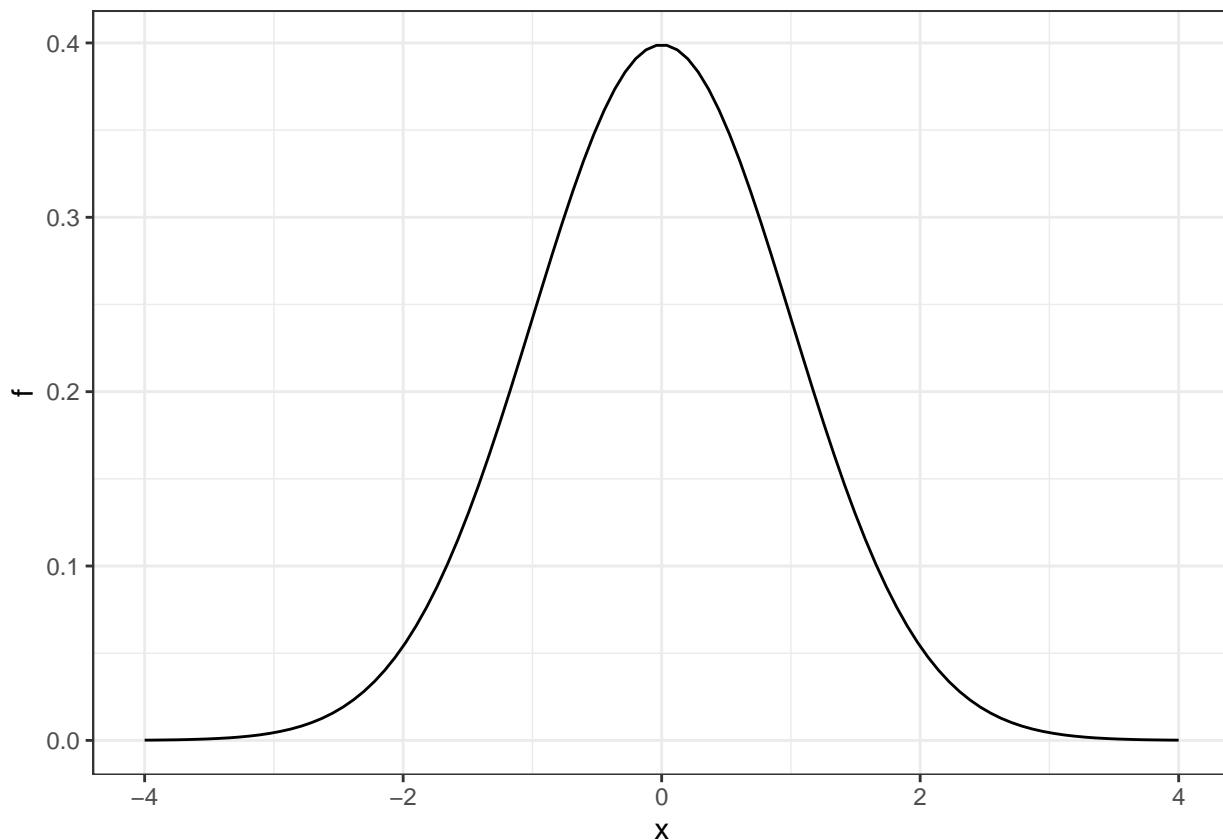
## [1] 0.0321
```

En R, puede obtenerse la función de densidad de probabilidad para una distribución normal con la función “dnorm()”:

- Primero, generamos una serie de valores  $z$  que incluyan el rango típico de una distribución normal. Dado que sabemos que el 99.7 % de las observaciones estarán en el rango  $-3 \leq z \leq 3$ , puede usarse un valor apenas mayor a 3 para cubrir casi todos los valores.
- Luego, se calcula la función  $f(z)$ .
- Finalmente, se grafica  $z$  contra  $f(z)$ .

```
x <- seq(-4, 4, length = 100)

data.frame(x, f = dnorm(x)) %>%
  ggplot(aes(x, f)) +
  geom_line()
```



Nota: la función “dnorm()” arroja la distribución normal estándar por default. Sin embargo, pueden modificarse sus parámetros  $\mu$  y  $\sigma$ : “dnorm(z, mu, sigma)”

### 3.2.3. Simulaciones Monte Carlo

R provee de una función que genera resultados normalmente distribuidos: “rnorm()”, cuyos argumentos son “tamaño”, “promedio” y “desviación estándar”.

A continuación, un ejemplo para generar datos que sean similares a las alturas reportadas:

```
x <- heights %>%
  filter(sex == "Male") %>%
  .$height

n <- length(x)

avg <- mean(x)

s <- sd(x)

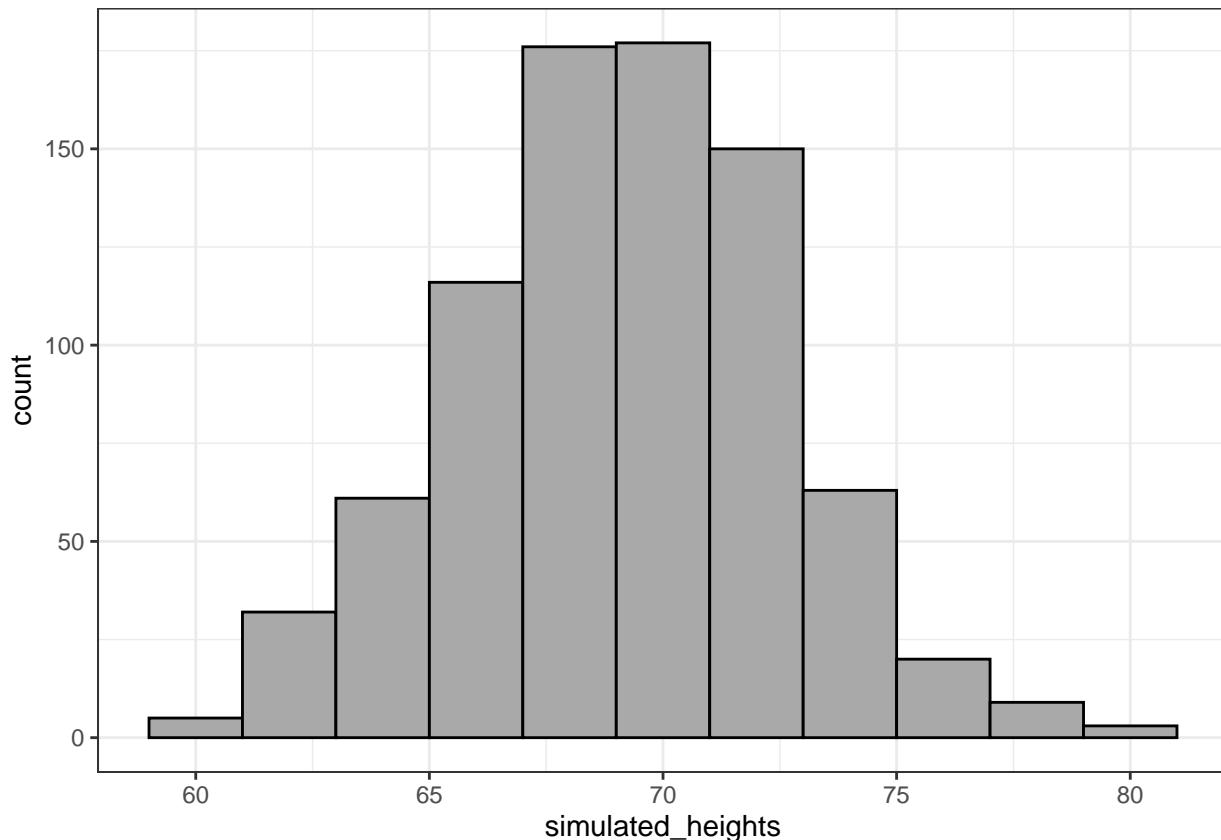
simulated_heights <- rnorm(n, avg, s)
```

No es sorprendente que la distribución de estos datos artificiales luzca normal:

```
ds_theme_set()

data.frame(simulated_heights = simulated_heights) %>%
  ggplot(aes(simulated_heights)) +
```

```
geom_histogram(color = "black", binwidth = 2, fill = "darkgray")
```



Por ejemplo, si se escogen 800 hombres al azar, ¿cuál será la distribución del más alto? ¿Qué proporción será más alto que 7 pies?

```
B <- 10000

tallest <- replicate(B, {
  simulated_heights <- rnorm(800, avg, s)
  max(simulated_heights)
})

mean(tallest >= 7*12)

## [1] 0.0209
```

### 3.2.4. Otras distribuciones continuas

Existen otro tipo de distribuciones:

- T-student
- Chi cuadrada
- Exponencial
- Gamma
- Beta

R tiene funciones para computar la densidad, los cuantiles y las CDF de estas, así como para generar simulaciones Monte Carlo. En general, el nombre de estas funciones sigue la siguiente lógica:

d: densidad  
q: cuantiles  
p: función de densidad de probabilidad  
r: random

### 3.2.5. Assessment 6

The ACT is a standardized college admissions test used in the United States. The four multi-part questions in this assessment all involve simulating some ACT test scores and answering probability questions about them.

For the three year period 2016-2018, ACT standardized test scores were approximately normally distributed with a mean of 20.9 and standard deviation of 5.7. (Real ACT scores are integers between 1 and 36, but we will ignore this detail and use continuous values instead.)

First we'll simulate an ACT test score dataset and answer some questions about it.

Set the seed to 16, then use `rnorm()` to generate a normal distribution of 10000 tests with a mean of 20.9 and standard deviation of 5.7. Save these values as `act_scores`. You'll be using this dataset throughout these four multi-part questions.

```
set.seed(16, sample.kind = "Rounding")  
  
## Warning in set.seed(16, sample.kind = "Rounding"): non-uniform 'Rounding'  
## sampler used  
act_scores <- rnorm(10000, 20.9, 5.7)
```

Pregunta 1.

What is the standard deviation of `act_scores`?

```
mean(act_scores)
```

```
## [1] 20.8
```

What is the standard deviation of `act_scores`?

```
sd(act_scores)
```

```
## [1] 5.68
```

A perfect score is 36 or greater (the maximum reported score is 36). In `act_scores`, how many perfect scores are there out of 10,000 simulated tests?

```
sum(act_scores >= 36)
```

```
## [1] 41
```

In `act_scores`, what is the probability of an ACT score greater than 30?

```
mean(act_scores > 30)
```

```
## [1] 0.0527
```

\*In `act_scores`, what is the probability of an ACT score less than or equal to 10?\*\*

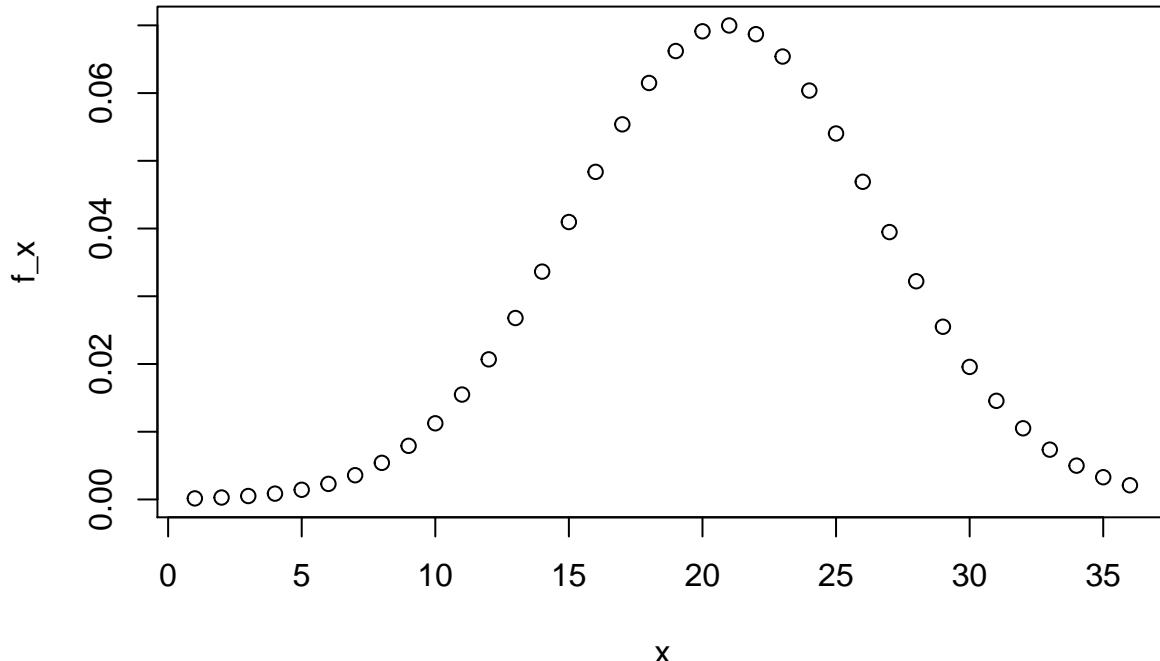
```
mean(act_scores <= 10)
```

```
## [1] 0.0282
```

Pregunta 2.

Set `x` equal to the sequence of integers 1 to 36. Use `dnorm` to determine the value of the probability density function over `x` given a mean of 20.9 and standard deviation of 5.7; save the result as `f_x`. Plot `x` against `f_x`.

```
x <- 1:36
f_x <- dnorm(x, 20.9, 5.7)
plot(x, f_x)
```



Convert `act_scores` to Z-scores. Recall from Data Visualization (the second course in this series) that to standardize values (convert values into Z-scores, that is, values distributed with a mean of 0 and standard deviation of 1), you must subtract the mean and then divide by the standard deviation. Use the mean and standard deviation of `act_scores`, not the original values used to generate random test scores.

```
zscores <- (act_scores - mean(act_scores)) / sd(act_scores)
```

### Pregunta 3.

What is the probability of a Z-score greater than 2 (2 standard deviations above the mean)?

```
mean(zscores > 2)
```

```
## [1] 0.0233
```

What ACT score value corresponds to 2 standard deviations above the mean ( $Z = 2$ )?

```
2*sd(act_scores) + mean(act_scores)
```

```
## [1] 32.2
```

A Z-score of 2 corresponds roughly to the 97.5th percentile. Use `qnorm()` to determine the 97.5th percentile of normally distributed data with the mean and standard deviation observed in `act_scores`. What is the 97.5th percentile of `act_scores`?

```
qnorm(0.975, mean(act_scores), sd(act_scores))
```

```
## [1] 32
```

\*Pregunta 4.\*

Write a function that takes a value and produces the probability of an ACT score less than or equal to that value (the CDF). Apply this function to the range 1 to 36.

```
cdf <- sapply(1:36, function(x){
  mean(act_scores <= x)
})
```

What is the minimum integer score such that the probability of that score or lower is at least .95?

```
min(which(cdf >= 0.95))
```

```
## [1] 31
```

Use `qnorm()` to determine the expected 95th percentile, the value for which the probability of receiving that score or lower is 0.95, given a mean score of 20.9 and standard deviation of 5.7. What is the expected 95th percentile of ACT scores?

```
qnorm(0.95, 20.9, 5.7)
```

```
## [1] 30.3
```

Make a vector containing the quantiles for `p <- seq(0.01, 0.99, 0.01)`, the 1st through 99th percentiles of the `act_scores` data. Save these as `sample_quantiles`. In what percentile is a score of 26?

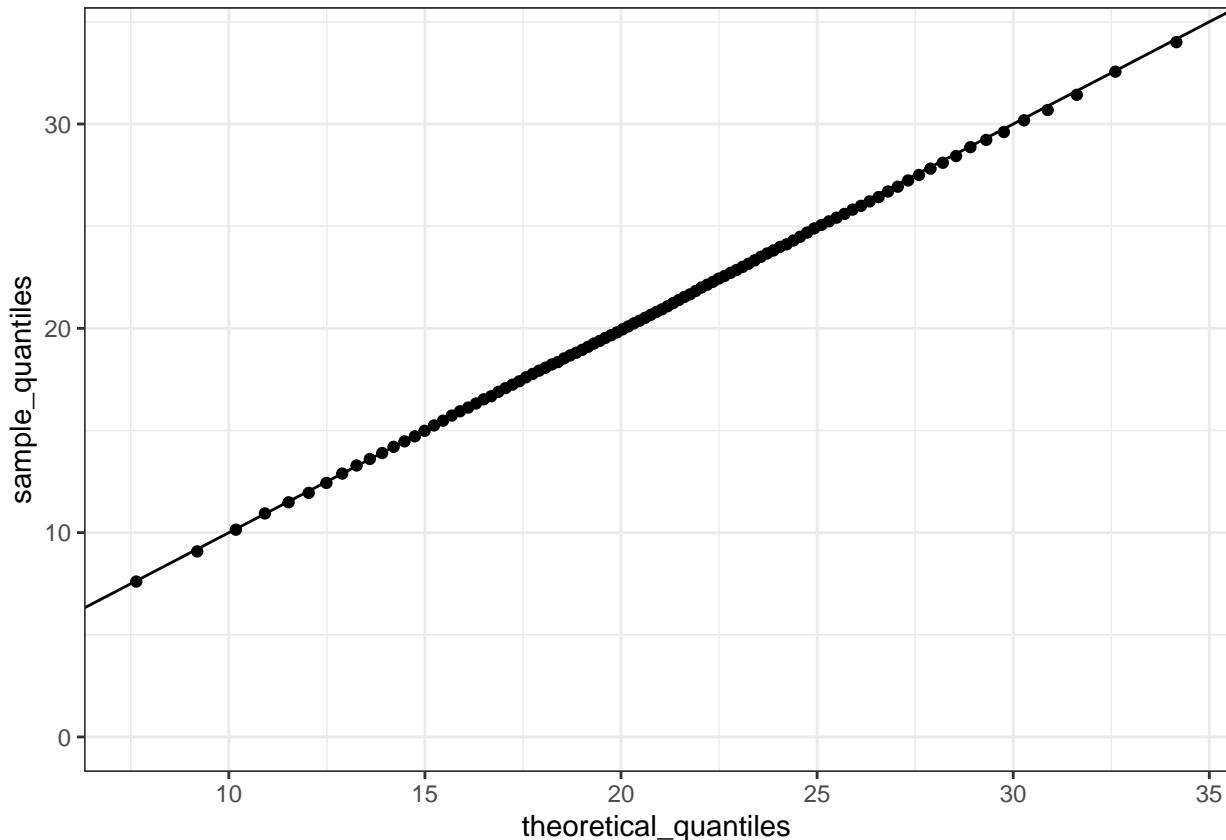
```
sample_quantiles <- seq(0.01, 0.99, 0.01)
quantile(act_scores, sample_quantiles)
```

```
##    1%    2%    3%    4%    5%    6%    7%    8%    9%   10%   11%   12%   13%
##  7.61  9.08 10.14 10.94 11.48 11.95 12.43 12.89 13.29 13.60 13.90 14.20 14.47
##   14%   15%   16%   17%   18%   19%   20%   21%   22%   23%   24%   25%   26%
## 14.71 14.98 15.25 15.48 15.73 15.94 16.12 16.32 16.52 16.68 16.89 17.08 17.24
##   27%   28%   29%   30%   31%   32%   33%   34%   35%   36%   37%   38%   39%
## 17.41 17.60 17.77 17.93 18.07 18.23 18.35 18.53 18.68 18.80 18.95 19.10 19.25
##   40%   41%   42%   43%   44%   45%   46%   47%   48%   49%   50%   51%   52%
## 19.38 19.53 19.66 19.80 19.95 20.09 20.24 20.37 20.51 20.66 20.80 20.93 21.08
##   53%   54%   55%   56%   57%   58%   59%   60%   61%   62%   63%   64%   65%
## 21.23 21.38 21.54 21.66 21.83 22.00 22.13 22.28 22.43 22.56 22.71 22.85 23.00
##   66%   67%   68%   69%   70%   71%   72%   73%   74%   75%   76%   77%   78%
## 23.16 23.33 23.50 23.66 23.81 23.98 24.11 24.30 24.48 24.68 24.89 25.06 25.24
##   79%   80%   81%   82%   83%   84%   85%   86%   87%   88%   89%   90%   91%
## 25.42 25.60 25.81 25.99 26.21 26.42 26.70 26.93 27.24 27.50 27.81 28.10 28.43
##   92%   93%   94%   95%   96%   97%   98%   99%
## 28.87 29.22 29.61 30.18 30.68 31.43 32.56 34.01
```

Make a corresponding set of theoretical quantiles using `qnorm()` over the interval `p <- seq(0.01, 0.99, 0.01)` with mean 20.9 and standard deviation 5.7. Save these as `theoretical_quantiles`.

Make a QQ-plot graphing sample\_quantiles on the y-axis versus theoretical\_quantiles on the x-axis.

```
p <- seq(0.01, 0.99, 0.01)
sample_quantiles <- quantile(act_scores, p)
theoretical_quantiles <- qnorm(p, 20.9, 5.7)
qplot(theoretical_quantiles, sample_quantiles) +
  geom_abline()
```



### 3.3. Variable aleatorias, muestreo y Teorema del Límite Central (CLT)

#### 3.3.1. Variables aleatorias

Las variables aleatorias son resultados numéricos producto de un proceso aleatorio. Recuérdese el ejemplo de una urna con pelotas rojas y azules; defínase la variable aleatoria  $X$ , que toma el valor 1 si la pelota es azul y 0 si es roja:

```
library(tidyverse)

beads <- rep(c("red", "blue"), times = c(2,3))

X <- ifelse(sample(beads, 1) == "blue", 1, 0)
```

#### 3.3.2. Muestras aleatorias

La **distribución de probabilidad de una variable aleatoria** es la probabilidad de que el valor observado esté dentro de un intervalo dado:

$$F(a) = \Pr(S \leq a)$$

Un muestreo modela el comportamiento aleatorio como si se sacaran objetos de una urna.

Supóngase que un pequeño casino te contrata para decidir si instalar o no un juego de ruleta. Se sabe que 1 000 personas jugarán, donde las únicas elecciones son rojo o negro. Se quiere predecir cuánto dinero se ganará o perderá; específicamente, cuál es la probabilidad de perder dinero. Sea  $S$  las ganancias del casino; si ocurre Rojo, el apostador gana y el casino pierde \$1; de otro modo, el casino gana y el apostador pierde \$1. Definan 10 resultados de este experimento:

```
color <- rep(c("Black", "Red", "Green"), c(18, 18, 2))

n <- 1000

X <- sample(ifelse(color == "Red", -1, 1), n, replace = TRUE)

X[1:10]

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

Dado que se conoce la proporción de pelotas que hacen ganar o perder al casino, lo anterior puede computarse de manera más compacta como sigue:

```
X <- sample(c(-1, 1), n, replace = TRUE, prob = c(18/38, 20/38))
```

Así, las ganancias totales,  $S$ , es la suma de las 1 000 repeticiones:

```
S <- sum(X)

S

## [1] 74
```

Si se quiere conocer la probabilidad de perder dinero, esto es equivalente a preguntarse cuál es la probabilidad de que  $S \in S < 0$ . Para esto, es necesario realizar una simulación Monte Carlo, donde 1 000 personas jueguen a la ruleta una y otra vez, 10 000 veces:

```
n <- 1000

B <- 10000

S <- replicate(B, {
```

```
X <- sample(c(-1,1), n, replace =TRUE, prob = c(18/38, 20/38))
sum(X)
}

mean(S < 0)
```

```
## [1] 0.0451
```

¿Estos resultados se distribuyen normalmente? Para ello, pueden compararse las distribuciones de la muestra con su respectiva aproximación normal:

```
mean(S)
```

```
## [1] 52.7
```

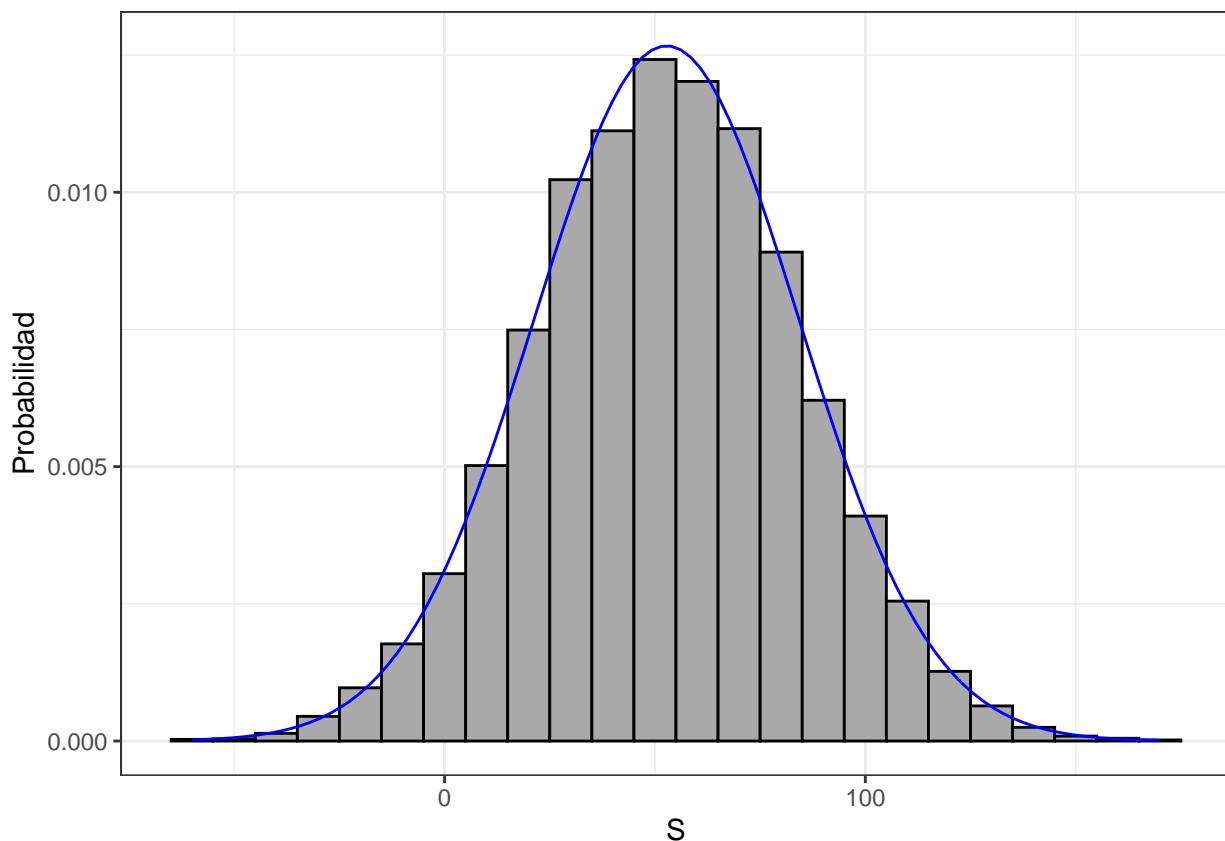
```
sd(S)
```

```
## [1] 31.5
```

```
s <- seq(min(S), max(S), length = 100)

normal_density <- data.frame(s = s, f = dnorm(s, mean(S), sd(S)))

data.frame(S = S) %>%
  ggplot(aes(S, ..density..)) +
  geom_histogram(color = "black", fill = "darkgray", binwidth = 10) +
  ylab("Probabilidad") +
  geom_line(data = normal_density, mapping = aes(s, f), color = "blue")
```



En este caso, la media y desviación estándar son conocidas como **valor esperado** y **error estándar** de la variable S, respectivamente.

### 3.3.3. Distribuciones vs. Distribuciones de probabilidad

Mientras que una distribución indica qué proporción de una lista de valores es menor o igual a un valor dado, una variable aleatoria X tiene una función de distribución (es un concepto teórico, no una lista de números).

### 3.3.4. Notación de variables aleatorias

Nótese que las letras mayúsculas son usadas para denotar variables aleatorias, mientras que las letras minúsculas representan valores observados, por ejemplo:

$$Pr(X \leq x)$$

$$Pr(X = x)$$

### Teorema del Límite Central (CLT)

El CLT dice que cuando el número de repeticiones independientes (tamaño de la muestra) es grande, la suma de las repeticiones independientes es aproximadamente normal.

Así, si una variable aleatoria tiene una distribución de probabilidad que se aproxima a una normal, entonces, para describirla, solo es necesaria la media (valor esperado) y la desviación estándar (error estándar).

El **valor esperado** de una variable aleatoria X es  $\mu$  e, intuitivamente, es el promedio de los valores que puede tomar la variable aleatoria.

$$E[X] = \mu$$

Así, en el ejemplo de la urna, el valor esperado de la variable aleatoria X es:

$$E[X] = (1) \left( \frac{20}{38} \right) + (-1) \left( \frac{18}{38} \right)$$

`20/38-18/38`

```
## [1] 0.0526
```

Intuitivamente, si el juego se repite una y otra vez, el casino gana, en promedio, \$0.05 por juego. Una simulación Monte Carlo confirma esto:

```
B <- 10^6
```

```
X <- sample(c(-1, 1), B, replace = TRUE, prob=c(18/38, 20/38))
```

```
mean(X)
```

```
## [1] 0.0523
```

De manera general, el valor esperado de X, cuando esta solo puede tomar los valores a y b, se define:

$$E[X] = a(p) + b(1 - p)$$

En el ejemplo del casino, si se espera que la ruleta se juegue 1 000 veces, entonces el valor esperado de las ganancias totales S serán  $1000 * 0,05 = \$50$ .

Por su parte, el **error estándar** arroja el tamaño de la variación alrededor del valor esperado. Si los eventos son independientes, está dado por

$$SE[X] = \sqrt{n} * SD$$

Por tanto, si la variable aleatoria toma solo dos valores, su desviación estándar es:

$$\boxed{|\mathbf{b} - \mathbf{a}| \sqrt{\mathbf{p}(1 - \mathbf{p})}}$$

En el ejemplo del casino:

$$|1 - (-1)| \sqrt{18/38 * 20/38} = 0,9986$$

```
2*sqrt(90)/19
```

```
## [1] 0.999
```

Si 1 000 personas juegan a la ruleta, el casino tendrá una ganancia esperada de \$50 con un error estándar de ±\$31,5789

```
n <- 1000
```

```
sqrt(n)*2*sqrt(90)/19
```

```
## [1] 31.6
```

Finalmente, la pregunta ¿qué tan probable es que el casino pierda dinero? se responde:

```
mu <- n*(20-18)/38
```

```
se <- sqrt(n)*2*sqrt(90)/19
```

```
pnorm(0, mu, se)
```

```
## [1] 0.0478
```

Que es muy similar a lo que se obtuvo con la simulación Monte Carlo

### 3.3.5. Promedios y proporciones

Propiedades:

1. El valor esperado de la suma de variables aleatorias es la suma de los valores esperados de variables aleatorias individuales:

$$E[X_1 + X_2 + \dots + X_n] = E[X_1] + E[X_2] + \dots + E[X_n]$$

- I. Si las variables aleatorias tienen la misma distribución, entonces:

$$\boxed{E[\mathbf{X}_1 + \mathbf{X}_2 + \dots + \mathbf{X}_n] = \mathbf{n}\mu}$$

2. El valor esperado de una variable aleatoria multiplicado por una constante no aleatoria es el valor esperado multiplicado por esa constante:

$$\mathbf{E}[aX] = a\mathbf{E}[X]$$

Lo anterior implica que el valor esperado del promedio de variables aleatorias idénticamente distribuidas es igual al valor esperado de la muestra,  $\mu$ :

$$\mathbf{E}[(X_1 + X_2 + \dots + X_n)/n] = \mathbf{E}[X_1 + X_2 + \dots + X_n]/n = n\mu/n = \mu$$

3. El cuadrado del error estándar de la suma de variables aleatorias independientes es la suma del cuadrado del error estándar de cada variable aleatoria:

$$\mathbf{SE}[X_1 + X_2 + \dots + X_n] = \sqrt{\mathbf{SE}[X_1]^2 + \mathbf{SE}[X_2]^2 + \dots + \mathbf{SE}[X_n]^2}$$

4. El error estándar de una variable aleatoria multiplicado por una constante no aleatoria es el error estándar multiplicado por esta constante:

$$\mathbf{SE}[aX] = a\mathbf{SE}[X]$$

Como consecuencia de 3 y 4 es que el error estándar del promedio de valores de una muestra aleatoria idénticamente distribuida es la desviación estándar de la muestra,  $\sigma$ , dividido por la raíz cuadrada de  $n$ :

$$(\mathbf{SE}[X_1 + X_2 + \dots + X_n]/n) = \sigma/\sqrt{n}$$

5. Si  $X$  es una variable aleatoria que se distribuye normalmente, entonces, si  $a$  y  $b$  son constantes no aleatorias,  $aX + b$  también es una variable aleatoria distribuida normalmente.

### 3.3.6. Ley de los grandes números

Una propiedad importante de lo hasta ahora descrito es que el error estándar del promedio de los valores que toma la variable aleatoria se vuelve cada vez más pequeño a medida que el tamaño de la muestra aumenta.

Cuando  $n$  es muy grande, el valor esperado de  $X$  converge a la media muestral.

### 3.3.7. ¿Qué tan grande tiene que ser $n$ ?

En muchas circunstancias,  $n = 30$  es suficiente para que el CLT aplique. Sin embargo, esto no es una regla general. Por ejemplo, en casos donde la probabilidad de éxito es muy pequeña, seguramente se necesite una  $n$  mayor (lotería). Alternativamente, la distribución Poisson es más apropiada para estos casos.

### 3.3.8. Assessment 7

An old version of the SAT college entrance exam had a -0.25 point penalty for every incorrect answer and awarded 1 point for a correct answer. The quantitative test consisted of 44 multiple-choice questions each with 5 answer choices. Suppose a student chooses answers by guessing for all questions on the test.

```
n_q <- 44
```

```
award <- 1
```

```
penalty <- -0.25
```

#### Pregunta 1

What is the expected value of points for guessing on one question?

```
p <- 1/5

e_points <- award*p+penalty*(1-p)

e_points
```

```
## [1] 0
```

What is the expected score of guessing on all 44 questions?

```
m <- n_q*e_points
```

```
m
```

```
## [1] 0
```

What is the standard error of guessing on all 44 questions?

```
se <- sqrt(n_q)*abs(penalty - award)*sqrt(p*(1-p))
```

```
se
```

```
## [1] 3.32
```

Use the Central Limit Theorem to determine the probability that a guessing student scores 8 points or higher on the test.

```
1-pnorm(8, m, se)
```

```
## [1] 0.00793
```

Set the seed to 21, then run a Monte Carlo simulation of 10,000 students guessing on the test. What is the probability that a guessing student scores 8 points or higher?

```
set.seed(21, sample.kind = "Rounding")

B <- 10000

S <- replicate(B, {
  results <- sample(c(award, penalty), n_q, replace = TRUE, prob=c(p, 1-p))
  sum(results)
})

mean(S > 8)
```

```
## [1] 0.008
```

## Pregunta 2

Suppose that the number of multiple choice options is 4 and that there is no penalty for guessing - that is, an incorrect question gives a score of 0. What is the expected value of the score when guessing on this new test?

```
p <- 1/4
```

```
penalty <- 0
```

```
reward <- 1
```

```
e_points <- award*p + penalty*(1-p)
```

```
e_points
```

```
## [1] 0.25
```

```
m <- n_q*e_points
```

```
m
```

```
## [1] 11
```

Consider a range of correct answer probabilities  $p \leftarrow \text{seq}(0.25, 0.95, 0.05)$  representing a range of student skills. What is the lowest  $p$  such that the probability of scoring over 35 exceeds 80%?

```
p <- seq(0.25, 0.95, 0.05)
```

```
score <- sapply(p, function(v){
  e_points <- (award*v) + (penalty*(1-v))
  m <- n_q*e_points
  se <- sqrt(n_q)*abs(penalty - award)*sqrt(v*(1-v))
  1-pnorm(35, m, se)
})
```

```
min(p[which(score > 0.8)])
```

```
## [1] 0.85
```

### Pregunta 3

A casino offers a House Special bet on roulette, which is a bet on five pockets (00, 0, 1, 2, 3) out of 38 total pockets. The bet pays out 6 to 1. In other words, a losing bet yields -\$1 and a successful bet yields \$6. A gambler wants to know the chance of losing money if he places 500 bets on the roulette House Special.

```
p <- 5/38
```

```
win <- 6
```

```
lose <- -1
```

```
n <- 500
```

What is the expected value of the payout for one bet?

```
mu <- win*p + lose*(1-p)
```

```
mu
```

```
## [1] -0.0789
```

What is the standard error of the payout for one bet?

```
sigma <- abs(lose - win)*sqrt(p*(1-p))
```

```
sigma
```

```
## [1] 2.37
```

What is the expected value of the average payout over 500 bets?

```
mu
```

```
## [1] -0.0789
```

What is the standard error of the average payout over 500 bets?

```
sigma/sqrt(n)
```

```
## [1] 0.106
```

What is the expected value of the sum of 500 bets?

```
mu*n
```

```
## [1] -39.5
```

What is the standard error of the sum of 500 bets?

```
sigma*sqrt(n)
```

```
## [1] 52.9
```

Use pnorm() with the expected value of the sum and standard error of the sum to calculate the probability of losing money over 500 bets,  $Pr(X \leq 0)$

```
pnorm(0, mu*n, sigma*sqrt(n))
```

```
## [1] 0.772
```

### 3.4. The Big Short

#### 3.4.1. Tasas de interés

```
library(ggplot2)
library(tidyverse)
```

Supóngase un pequeño banco que es bueno identificando a individuos con altas probabilidades de pagar sus créditos. En promedio, cada año el 2% de los clientes incurren en *defaults*. Así, al cobrar un pequeño extra, el banco compensa por ese porcentaje de impago. A partir de esta información y con la ayuda de la probabilidad, podemos encontrar el nivel del tipo de interés que el banco debe ofertar.

Supóngase que el banco otorga 1 000 préstamos con valor de \$180000. Además, el banco tiene pérdidas por \$200000 (incluyendo costo de operación):

```
n <- 1000

loss_per_foreclosure <- -200000

p <- 0.02

defaults <- sample(c(0,1), n, prob=c(1-p, p), replace = TRUE)

sum(defaults*loss_per_foreclosure)

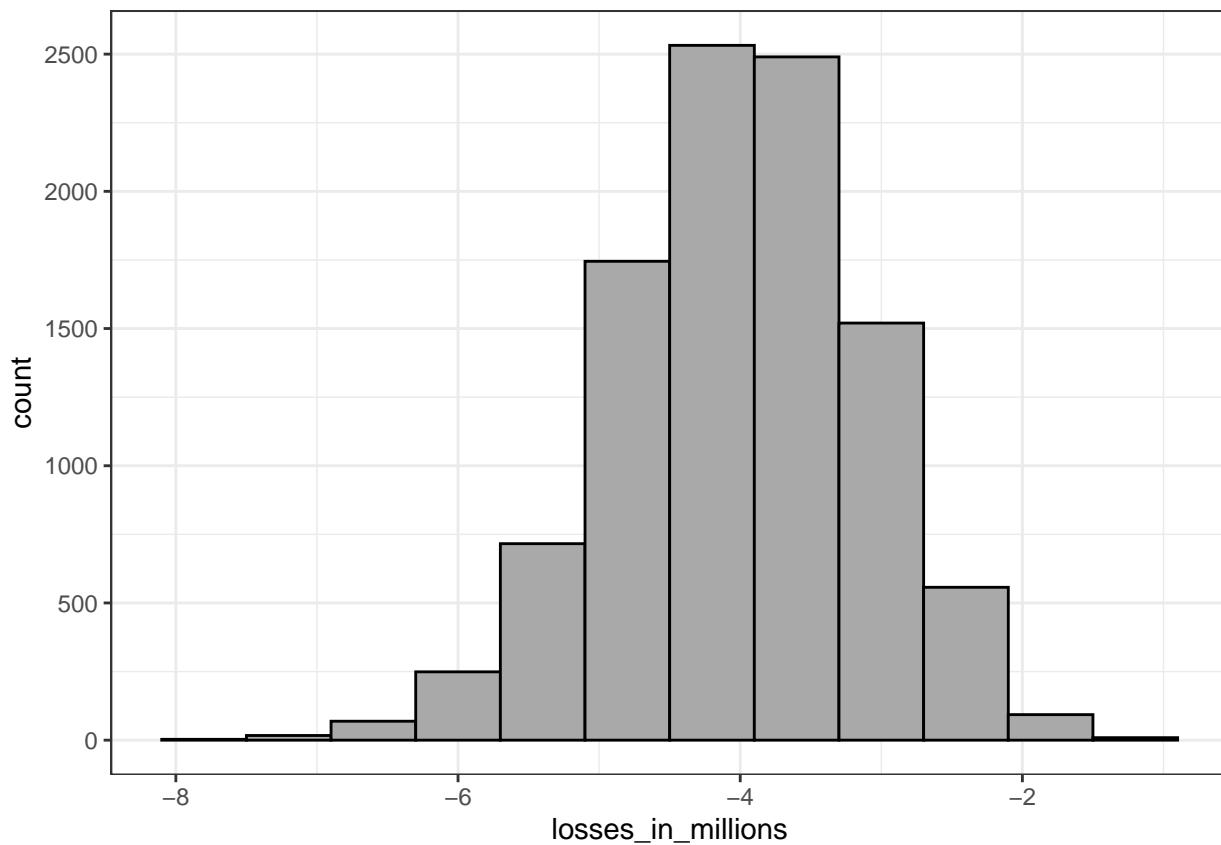
## [1] -6400000
```

Así, constrúyase una simulación Monte Carlo:

```
B <- 10000

losses <- replicate(B, {
  defaults <- sample(c(0,1), n, prob=c(1-p, p), replace = TRUE)
  sum(defaults * loss_per_foreclosure)
})

data.frame(losses_in_millions = losses/10^6) %>%
  ggplot(aes(losses_in_millions)) +
  geom_histogram(binwidth = 0.6, col = "black", fill = "darkgray")
```



Incluso sin una simulación Monte Carlo, y gracias al CLT, sabemos que, dado que las pérdidas son independientes entre sí, su distribución será aproximadamente normal con un valor esperado y desviación estándar:

```
n*(p*loss_per_foreclosure + (1-p)*0)
```

```
## [1] -4e+06
sqrt(n)*abs(loss_per_foreclosure)*sqrt(p*(1-p))
```

```
## [1] 885438
```

Puede fijarse una tasa de interés para garantizar que, en promedio, el banco quede tablas. Para ello, hay que agregar una cantidad  $x$  a cada préstamo, tal que su valor esperado sea 0. Sea  $l$  las pérdidas:

$$lp + x(1 - p) = 0 \leftrightarrow x = -\frac{lp}{1 - p}$$

```
x <- -loss_per_foreclosure*p/(1-p)
x
```

```
## [1] 4082
```

En un préstamo de \$180000, esto representa una tasa de interés de  $\sim 2\%$

```
x/180000
```

```
## [1] 0.0227
```

Sin embargo, con esta tasa de interés aún hay un 50 % de probabilidades de perder dinero, por lo que hay que minimizar este riesgo. Supóngase que se desea que el riesgo de perder dinero sea de 1 %,  $Pr(S < 0) = 0,01$ .

Sabemos que el valor esperado de S es:

$$[lp + x(1 - p)]n$$

Y su error estándar:

$$|x - l| \sqrt{np(1 - p)}$$

Normalizando  $Pr(S < 0) = 0,01$ :

$$Pr\left(\frac{S - E[S]}{SE[S]} < \frac{-E[S]}{SE[S]}\right) = Pr\left(Z < \frac{-E[S]}{SE[S]}\right)$$

Por tanto:

$$Pr\left(Z < \frac{-[lp + x(1 - p)]n}{(x - l)\sqrt{np(1 - p)}}\right) = 0,01$$

Dado que  $Z \sim N(0, 1)$ , entonces el lado derecho de la desigualdad es:

`qnorm(0.01)`

`## [1] -2.33`

Por lo tanto:

$$z = \frac{-[lp + x(1 - p)]n}{(x - l)\sqrt{np(1 - p)}}$$

Resolviendo para x:

$$x = -l \frac{np - z\sqrt{np(1 - p)}}{n(1 - p) + z\sqrt{np(1 - p)}}$$

```
1 <- loss_per_foreclosure
z <- qnorm(0.01)
x <- -l*(n*p-z*sqrt(n*p*(1-p)))/(n*(1-p)+z*sqrt(n*p*(1-p)))
x
## [1] 6249
```

Que representa un tipo de interés de  $\sim 3,4\%$

`x/180000`

`## [1] 0.0347`

Nótese que, con este tipo de interés, esperamos una ganancia por préstamo y ganancias totales de:

`loss_per_foreclosure*p+x*(1-p)`

`## [1] 2124`

```
(loss_per_foreclosure*p+x*(1-p))*n
```

```
## [1] 2124198
```

Finalmente, puede hacerse una simulación Monte Carlo para comprobar la aproximación teórica:

```
B <- 10000
```

```
profit <- replicate(B, {
  draws <- sample(c(x, loss_per_foreclosure), n, prob=c(1-p, p), replace = TRUE)
  sum(draws)
})
```

```
mean(profit)
```

```
## [1] 2136119
```

```
mean(profit<0)
```

```
## [1] 0.0098
```

### 3.4.2. The Big Short

¿Qué ocurre si se fija el tipo de interés en 5 % y el porcentaje de *default* es 4 %? Se ganarán \$640 por préstamo:

```
p <- 0.04
```

```
loss_per_foreclosure <- -200000
```

```
r <- 0.05
```

```
x <- r*180000
```

```
loss_per_foreclosure*p+x*(1-p)
```

```
## [1] 640
```

Recordemos que la probabilidad de perder dinero está definida como:

$$Pr(S < 0) = Pr\left(Z < \frac{-E[S]}{SE[S]}\right)$$

Si se asume que  $\mu$  es el valor esperado y  $\sigma$  la desviación estándar, entonces:

$$E[S] = n\mu \quad SE[S] = \sqrt{n}\sigma$$

Esto implica que si se cumple lo siguiente, la probabilidad de perder dinero será 0.01

$$n \geq z^2\sigma^2/\mu^2$$

La pregunta es: con  $x$  fija, ¿qué  $n$  se necesita para que la probabilidad de perder dinero sea 0.01?

```
z <- qnorm(0.01)
```

```
n <- ceiling((z^2*(x-1)^2*p*(1-p))/(1*p+x*(1-p))^2)
```

```
n
```

```
## [1] 22163
```

Esta  $n$  implica una ganancia esperada total de:

```
n*(loss_per_foreclosure*p + x*(1-p))
```

```
## [1] 14184320
```

Esto puede confirmarse mediante una simulación Monte Carlo:

```
p <- 0.04
```

```
x <- 0.05*180000
```

```
profit <- replicate(B, {
  draws <- sample(c(x, loss_per_foreclosure), n, prob=c(1-p, p), replace = TRUE)
  sum(draws)
})
```

```
mean(profit < 0)
```

```
## [1] 0.0095
```

La siguiente simulación Monte Carlo estima las ganancias esperadas dada una probabilidad de *default* desconocida  $0.03 \leq p \leq 0.05$ , y modelando una situación en la que un evento cambia la probabilidad de *default* de todos los individuos simultáneamente:

```
p <- 0.04
```

```
x <- 0.05*180000
```

```
profit <- replicate(B, {
  new_p <- 0.04 + sample(seq(-0.01, 0.01, length=100), 1)
  draws <- sample(c(x, loss_per_foreclosure), n, prob=c(1-new_p, new_p), replace = TRUE)
  sum(draws)
})
```

```
mean(profit < 0)
```

```
## [1] 0.351
```

Puede verse que la probabilidad de pérdida alcanza niveles de 34.93 %.

### 3.4.3. Assessment 8

```
library(dslabs)
library(tidyverse)
```

```
data("death_prob")
head(death_prob)
```

```
##   age   sex      prob
## 1   0 Male 0.006383
## 2   1 Male 0.000453
## 3   2 Male 0.000282
## 4   3 Male 0.000230
## 5   4 Male 0.000169
## 6   5 Male 0.000155
```

### Preguntas 1 y 2.

An insurance company offers a one-year term life insurance policy that pays \$150,000 in the event of death within one year. The premium (annual cost) for this policy for a 50 year old female is \$1,150. Suppose that in the event of a claim, the company forfeits the premium and loses a total of \$150,000, and if there is no claim the company gains the premium amount of \$1,150. The company plans to sell 1,000 policies to this demographic.

The `death_prob` data frame from the `dslabs` package contains information about the estimated probability of death within 1 year (`prob`) for different ages and sexes. Use `death_prob` to determine the death probability of a 50 year old female,  $p$ .

```
p <- death_prob %>%
  filter(age == 50 & sex == "Female") %>%
  pull(prob)
```

$p$

```
## [1] 0.00319
```

What is the expected value of the company's net profit on one policy for a 50 year old female?

```
-150000*p + 1150*(1-p)
```

```
## [1] 667
```

Calculate the standard error of the profit on one policy for a 50 year old female.

```
abs(-150000-1150)*sqrt(p*(1-p))
```

```
## [1] 8527
```

What is the expected value of the company's profit over all 1,000 policies for 50 year old females?

```
n <- 1000
```

```
n*(-150000*p + 1150*(1-p))
```

```
## [1] 667378
```

What is the standard error of the sum of the expected value over all 1,000 policies for 50 year old females?

```
sqrt(n)*(abs(-150000-1150)*sqrt(p*(1-p)))
```

```
## [1] 269658
```

Use the Central Limit Theorem to calculate the probability that the insurance company loses money on this set of 1,000 policies.

```
pnorm(0, n*(-150000*p + 1150*(1-p)), sqrt(n)*(abs(-150000-1150)*sqrt(p*(1-p))))
```

```
## [1] 0.00666
```

50 year old males have a different probability of death than 50 year old females. We will calculate a profitable premium for 50 year old males in the following four-part question.

Use `death_prob` to determine the probability of death within one year for a 50 year old male.

```
p <- death_prob %>%
  filter(age == 50 & sex == "Male") %>%
  pull(prob)
```

p

```
## [1] 0.00501
```

Suppose the company wants its expected profits from 1,000 50 year old males with \$150,000 life insurance policies to be \$700,000. Use the formula for expected value of the sum of draws with the following values and solve for the premium :

$$E[S] = \mu_S = 700000$$

$$n = 1000$$

p = probabilidad de muerte de hombre de 50 años

$$a = 150000$$

$$b = \text{premium}$$

**What premium should be charged?**

Dado que  $E[S] = n(ap + b(1 - p))$ , entonces  $b = (E[S]/n - ap)/(1 - p)$ :

```
b <- ((700000/1000) - 150000*p)/(1-p)
```

b

```
## [1] 1459
```

Using the new 50 year old male premium rate, calculate the standard error of the sum of 1,000 premiums.

```
SE <- sqrt(1000)*abs(b - 150000)*sqrt(p*(1-p))
```

SE

```
## [1] 338262
```

**What is the probability of losing money on a series of 1,000 policies to 50 year old males?**

```
pnorm(0, 1000*(-150000*p + b*(1-p)), SE)
```

```
## [1] 0.0193
```

Preguntas 3 y 4

Life insurance rates are calculated using mortality statistics from the recent past. They are priced such that companies are almost assured to profit as long as the probability of death remains similar. If an event occurs that changes the probability of death in a given age group, the company risks significant losses.

In this 6-part question, we'll look at a scenario in which a lethal pandemic disease increases the probability of death within 1 year for a 50 year old to .015. Unable to predict the outbreak, the company has sold 1,000 \$150,000 life insurance policies for \$1,150.

**What is the expected value of the company's profits over 1,000 policies?**

```

p <- 0.015

n <- 1000

mu <- n*(p*(-150000) + (1-p)*1150)

mu

## [1] -1117250

What is the standard error of the expected value of the company's profits over 1,000 policies?
SE <- sqrt(n)*abs(-150000-1150)*sqrt(p*(1-p))

SE

## [1] 580994

What is the probability of the company losing money?
pnorm(0, mu, SE)

## [1] 0.973

Suppose the company can afford to sustain one-time losses of $1 million, but larger losses will
force it to go out of business. What is the probability of losing more than $1 million?
pnorm(-1000000, mu, SE)

## [1] 0.58

Investigate death probabilities p <- seq(.01, .03, .001). What is the lowest death probability
for which the chance of losing money exceeds 90%?
p <- seq(.01, .03, .001)

f <- function(p){
  mu <- 1000*(-150000*p + 1150*(1-p))
  SE <- sqrt(n)*abs(-150000-1150)*sqrt(p*(1-p))
  pnorm(0, mu, SE)
}

sapply(p, FUN = f)

## [1] 0.776 0.848 0.899 0.934 0.957 0.973 0.983 0.989 0.993 0.996 0.997 0.998
## [13] 0.999 0.999 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000

Investigate death probabilities p <- seq(.01, .03, .0025). What is the lowest death probability
for which the chance of losing over $1 million exceeds 90%?
p <- seq(.01, .03, .0025)

f <- function(p){
  mu <- 1000*(-150000*p + 1150*(1-p))
  SE <- sqrt(n)*abs(-150000-1150)*sqrt(p*(1-p))
  pnorm(-1000000, mu, SE)
}

sapply(p, FUN = f)

```

```
## [1] 0.0897 0.3118 0.5800 0.7852 0.9040 0.9612 0.9855 0.9949 0.9983
```

Define a sampling model for simulating the total profit over 1,000 loans with probability of claim  $p\_loss = .015$ , loss of  $-\$150,000$  on a claim, and profit of  $\$1,150$  when there is no claim. Set the seed to 25, then run the model once. What is the reported profit (or loss) in millions (that is, divided by  $10^6$ )?

```
set.seed(25, sample.kind = "Rounding")

## Warning in set.seed(25, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used

n <- 1000

p_loss <- 0.015

X <- sample(c(0,1), n, replace = TRUE, prob = c(1-p_loss, p_loss))

loss <- -150000*sum(X==1)/10^6

profit <- 1150*sum(X==0)/10^6

loss + profit

## [1] -1.42
```

Set the seed to 27, then run a Monte Carlo simulation of your sampling model with 10,000 replicates to simulate the range of profits/losses over 1,000 loans. What is the observed probability of losing \$1 million or more?

```
set.seed(27, sample.kind = "Rounding")

## Warning in set.seed(27, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used

S <- replicate(10000, {
  X <- sample(c(0,1), n, replace = TRUE, prob = c(1-p_loss, p_loss))
  loss <- -150000*sum(X==1)/10^6
  profit <- 1150*sum(X==0)/10^6
  loss + profit
})

sum(S<=-1)/10000

## [1] 0.539
```

#### Preguntas 5 y 6.

Suppose that there is a massive demand for life insurance due to the pandemic, and the company wants to find a premium cost for which the probability of losing money is under 5 %, assuming the death rate stays stable at  $p = 0.015$ .

Calculate the premium required for a 5 % chance of losing money given  $n=1000$  loans, probability of death  $p=0.015$ , and loss per claim  $l = -150000$ . Save this premium as  $x$  for use in further questions.

```
p <- .015
n <- 1000
l <- -150000
```

```

z <- qnorm(.05)
x <- -1*(n*p - z*sqrt(n*p*(1-p)))/ (n*(1-p) + z*sqrt(n*p*(1-p)))
x

```

```
## [1] 3268
```

What is the expected profit per policy at this rate?

```
l*p + x*(1-p)
```

```
## [1] 969
```

What is the expected profit over 1,000 policies?

```
n*(l*p + x*(1-p))
```

```
## [1] 969042
```

Run a Monte Carlo simulation with  $B=10000$  to determine the probability of losing money on 1,000 policies given the new premium  $x$ , loss on a claim of \$150,000, and probability of claim  $p$ . Set the seed to 28 before running your simulation. What is the probability of losing money here?

```

set.seed(28)

S <- replicate(10000, {
  X <- sample(c(0,1), n, replace = TRUE, prob=c((1-p), p))
  loss <- l*sum(X==1)/10^6 # in millions
  profit <- x*sum(X==0)/10^6
  loss+profit
})
sum(S<0)/10000

```

```
## [1] 0.0554
```

The company cannot predict whether the pandemic death rate will stay stable. Set the seed to 29, then write a Monte Carlo simulation that for each of iterations:

- randomly changes by adding a value between -0.01 and 0.01 with  $\text{sample}(\text{seq}(-0.01, 0.01, \text{length} = 100), 1)$
- uses the new random to generate a sample of policies with premium  $x$  and loss per claim
- returns the profit over policies (sum of random variable)

The outcome should be a vector of total profits. Use the results of the Monte Carlo simulation to answer the following three questions.

```

set.seed(29)

n <- 1000
B <- 10000
l <- -150000
p <- 0.015
x <- 3268
X <- replicate(B, {
  new_p <- p + sample(seq(-0.01, 0.01, length=100), 1)
  Y <- sample(c(x, 1), n, replace=TRUE, prob=c(1-new_p, new_p))
  sum(Y)
})

```

What is the expected value over 1,000 policies?

```
mean(X)
## [1] 968244
What is the probability of losing money?
sum(X<0)/B
## [1] 0.191
What is the probability of losing more than $1 million?
sum(X < -1000000)/B
## [1] 0.0424
```

## 4. Inferencia y modelaje

### 4.1. Parámetros y estimadores

#### 4.1.1. Parámetros de un muestreo y estimadores

Para los siguientes temas, se usará el ejemplo de una elección; para ello, resultará más conveniente pensar en una urna de votos en vez de votantes. Por otro lado, y dado que las casas encuestadoras compiten entre sí, supóngase un premio monetario de \$25 por “atinarle” al resultado.

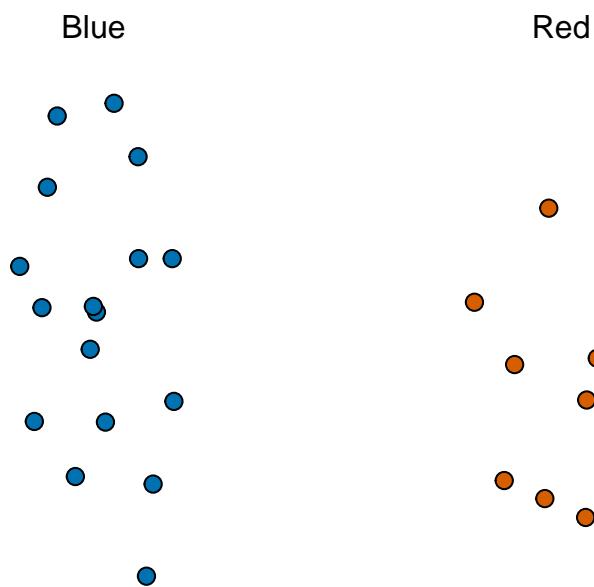
Se buscará definir la difusión de las proporciones de pelotas rojas o azules en la urna.

Además, se asumirá que se toma una muestra del total de pelotas en la urna (equivalente a una encuesta electoral) y, dado que estas son costosas en la vida real, el muestreo costará \$0.10 por pelota.

Así, cada encuestadora propone un intervalo de las proporciones de cada color de pelota: si este contiene a la cantidad verdadera, se les regresa la mitad de lo que pagaron y pasan a la siguiente etapa de la competición. En la segunda etapa, el intervalo más pequeño que contenga la cantidad verdadera es el ganador.

Así, el paquete dslabs contiene una función que muestra una elección aleatoria de la urna:

```
library(tidyverse)  
  
library(dslabs)  
  
ds_theme_set()  
  
take_poll(25)
```



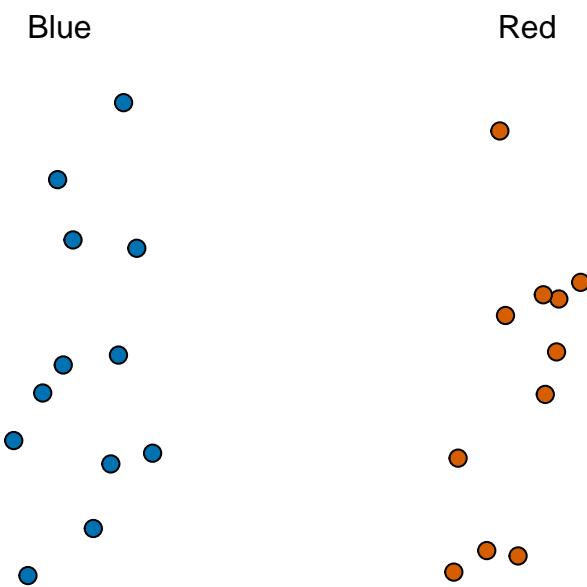
Recuérdese que cada pelota representa un individuo, donde el color rojo son votos Republicanos y los azules

votos Demócratas.

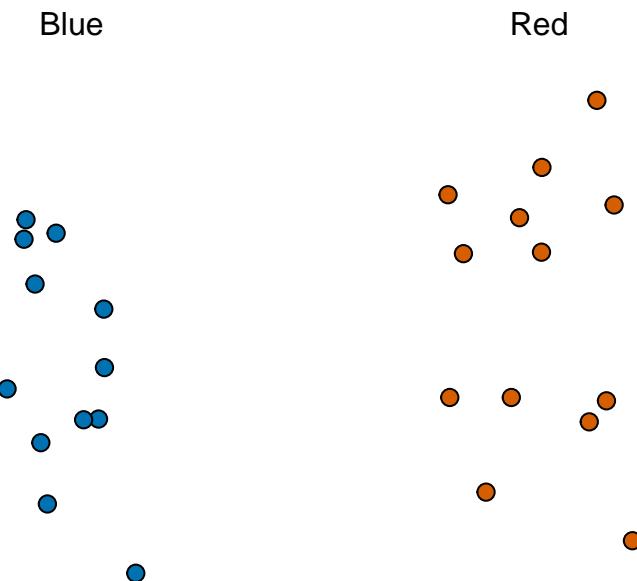
Sea la proporción de pelotas azules  $p$ , lo que implica que la proporción de pelotas rojas es  $(1 - p)$ . Así, la urna representa la **población**,  $p$  es llamado un **parámetro** de la población, las 25 pelotas escogidas al azar de la urna son una **muestra**. El objetivo es predecir el parámetro  $p$  a partir de los datos de la muestra.

Un **estimador** es un resumen de los datos observados que se piensa que son informativos acerca del parámetro de interés. Así, el estimador que se obtenga de la muestra, intuitivamente, debería estar relacionado con la proporción real de pelotas rojas y blancas en la población total. Sin embargo, si se realiza el muestreo varias veces, se observa que las proporciones cambian, esto es debido a que **los parámetros de la muestra son variables aleatorias**:

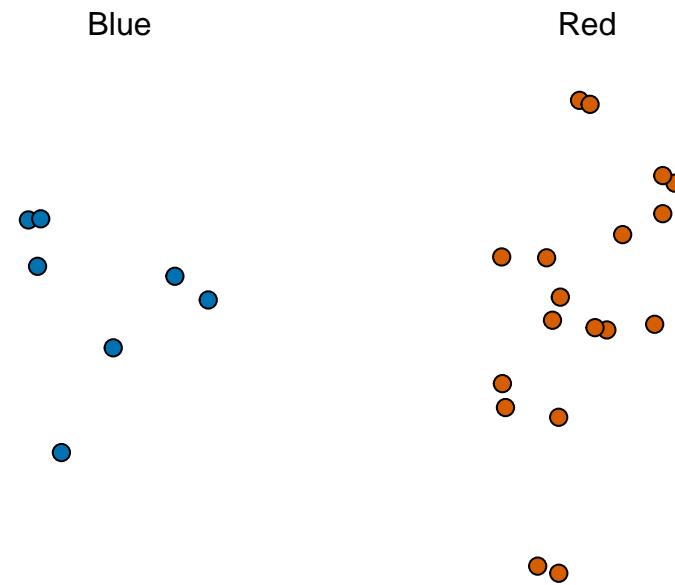
```
take_poll(25)
```



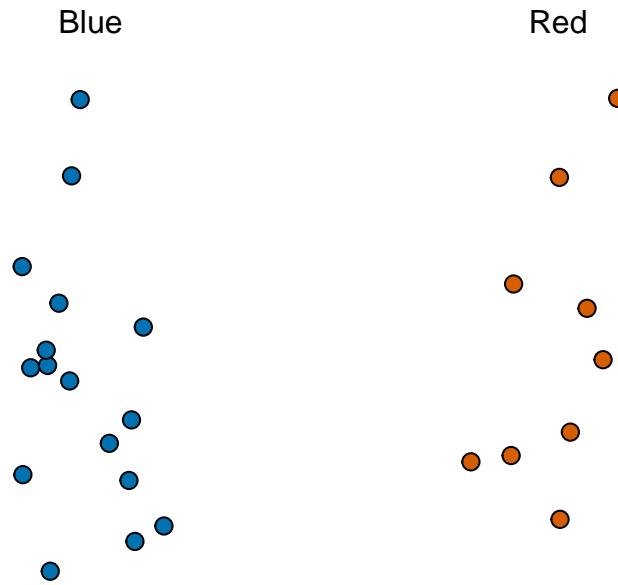
```
take_poll(25)
```



```
take_poll(25)
```



```
take_poll(25)
```



#### 4.1.2. Promedio muestral

Defínase una variable aleatoria  $X$ :

$$X = \begin{cases} 0 & \text{si roja} \\ 1 & \text{si azul} \end{cases}$$

Si se toma una muestra de  $N$  pelotas, entonces el promedio de la muestra es equivalente a la proporción de pelotas azules en esta; formalmente:

$$\bar{X} = \frac{X_1 + X_2 + \dots + X_n}{n}$$

Que implica que la distribución de  $\bar{X}$  puede ser descrita por:

$$n\bar{X} = X_1 + X_2 + \dots + X_n$$

#### 4.1.3. Sondeo vs. pronóstico

Si una encuesta se realiza 4 meses antes de la elección, esta está estimando el parámetro  $p$  en ese momento, no para el día de la elección. Nótese que la verdadera proporción  $p$  puede variar el día de la elección por un sinfín de factores.

Por tal motivo, los pronosticadores tratan de construir herramientas que modelen cómo las opiniones varían a través del tiempo y tratan de predecir el resultado del día de la elección, considerando el hecho de que las opiniones fluctúan.

#### 4.1.4. Propiedades de los estimadores

Es pertinente conocer las propiedades del estimador de la media poblacional  $\bar{X} = \frac{X_1+X_2+\dots+X_n}{n}$ . Nótese que  $n\bar{X} = (X_1 + X_2 + \dots + X_n)$  es la suma de repeticiones independientes.

1.  $E[n\bar{X}] = np \leftrightarrow E[\bar{X}] = p$
2.  $SE[n\bar{X}] = \sqrt{np(1-p)} \leftrightarrow SE[\bar{X}] = \sqrt{p(1-p)/n} = \sqrt{p(1-p)/n}$

Que implica que el dato de interés, la proporción  $p$  poblacional, es igual al valor esperado del promedio muestral y que podemos disminuir el error estándar del estimador al incrementar  $n$ .

## 4.2. EL TLC en práctica

Recordemos que el TLC que la función de distribución de una suma de variables aleatorias es aproximadamente normal:

$$(X_1 + X_2 + \dots + X_n) \sim N(\mu, \sigma)$$

También que, dada  $X \sim N(\mu, \sigma)$  y  $a \in \mathbb{R}$ , entonces:

$$\frac{X}{a} \sim N\left(\frac{\mu}{a}, \frac{\sigma}{a}\right)$$

Que implica que la distribución de  $\bar{X}$  es aproximadamente normal:

$$\bar{X} \sim N(p, \sqrt{p(1-p)/n})$$

Supóngase que se quiere conocer la probabilidad de que el estimado sobre  $p$  está dentro de un intervalo de 1% del valor real. Formalmente, queremos saber  $Pr(|\bar{X} - p| \leq 0,01) \leftrightarrow Pr(\bar{X} \leq p + 0,01) - Pr(\bar{X} \leq p - 0,01)$ .

Normalizando:

$$\begin{aligned} Pr\left(\frac{\bar{X} - E[\bar{X}]}{SE[\bar{X}]} \leq \frac{(p + 0,01) - E[\bar{X}]}{SE[\bar{X}]}\right) - Pr\left(\frac{\bar{X} - E[\bar{X}]}{SE[\bar{X}]} \leq \frac{(p - 0,01) - E[\bar{X}]}{SE[\bar{X}]}\right) \\ Pr\left(Z \leq \frac{(p + 0,01) - E[\bar{X}]}{SE[\bar{X}]}\right) - Pr\left(Z \leq \frac{(p - 0,01) - E[\bar{X}]}{SE[\bar{X}]}\right) \end{aligned}$$

Que es lo mismo que:

$$Pr\left(Z \leq 0,01/\sqrt{p(1-p)/n}\right) - Pr\left(Z \leq -0,01/\sqrt{p(1-p)/n}\right)$$

Sin embargo, esta probabilidad no se puede calcular porque no conocemos  $p$ . No obstante, podemos definir estimadores para el error estándar de  $\bar{X}$ , el cual puede ser calculado a partir de la muestra observada:

$$\hat{SE}[\bar{X}] = \sqrt{\bar{X}(1 - \bar{X})/n}$$

Considérese una muestra de 12 pelotas azules y 13 rojas, que implica que  $\bar{X} = 0,48$ , entonces el error estándar es:

```
X_hat <- 0.48
```

```
se <- sqrt(X_hat*(1-X_hat)/25)
```

```
se
```

```
## [1] 0.0999
```

Ahora sí puede calcularse la probabilidad:

```
pnorm(0.01/se) - pnorm(-0.01/se)
```

```
## [1] 0.0797
```

$$\boxed{\Pr(|\bar{X} - p| \leq 0,01) = 0,0797}$$

#### 4.2.1. Márgenes de error

El margen de error se define como 2 veces el error estándar del estimador  $\bar{X}$ :

```
2*se
```

```
## [1] 0.2
```

Idealmente, deberíamos hallar que  $PR(|\bar{X} - p| \leq 2SE[\bar{X}]) = \dots = Pr(Z \leq 2) - Pr(Z \leq -2)$ ; esto es: hay un 95 % de probabilidades de que  $\bar{X}$  esté dentro de dos errores estándar del parámetro poblacional real  $p$ .

Por tanto, una muestra de 25 pelotas no es muy útil, dado que el margen de error es de casi 20 %.

#### 4.2.2. Una simulación Monte Carlo para el TLC

```
library(gridExtra)
library(ggplot2)
library(tidyverse)
```

Utilizaremos una simulación Monte Carlo para corroborar que las herramientas que hemos usado para calcular los estimadores y los márgenes de error son probabilísticamente adecuadas. En principio, no conocemos  $p$  *a priori*; por tanto, puede escogerse un valor arbitrario de  $p$  (o varios) y correr varias simulaciones Monte Carlo. Así, simúlese una muestra de 1 000 personas:

```
p <- 0.45

N <- 1000

X <- sample(c(0,1), N, replace = TRUE, prob=c(1-p, p))

X_hat <- mean(X)
```

Y así, replicar lo anterior 10 000 veces en una simulación Monte Carlo:

```
B <- 10000

X_hat <- replicate(B, {
  X <- sample(c(0,1), N, replace = TRUE, prob=c(1-p, p))
  mean(X)
})
```

Dado que, según la teoría,  $\bar{X} \sim N(0.45, 0.015)$ , podemos comprobar que efectivamente ocurre:

```
mean(X_hat)
```

```
## [1] 0.45
```

```
sd(X_hat)
```

```
## [1] 0.0156
```

Finalmente, lo anterior se puede observar en un histograma y en un gráfico Q-Q:

```
p1 <- data.frame(X_hat = X_hat) %>%
  ggplot(aes(X_hat)) +
  geom_histogram(binwidth = 0.005, color = "black", fill = "darkgray")

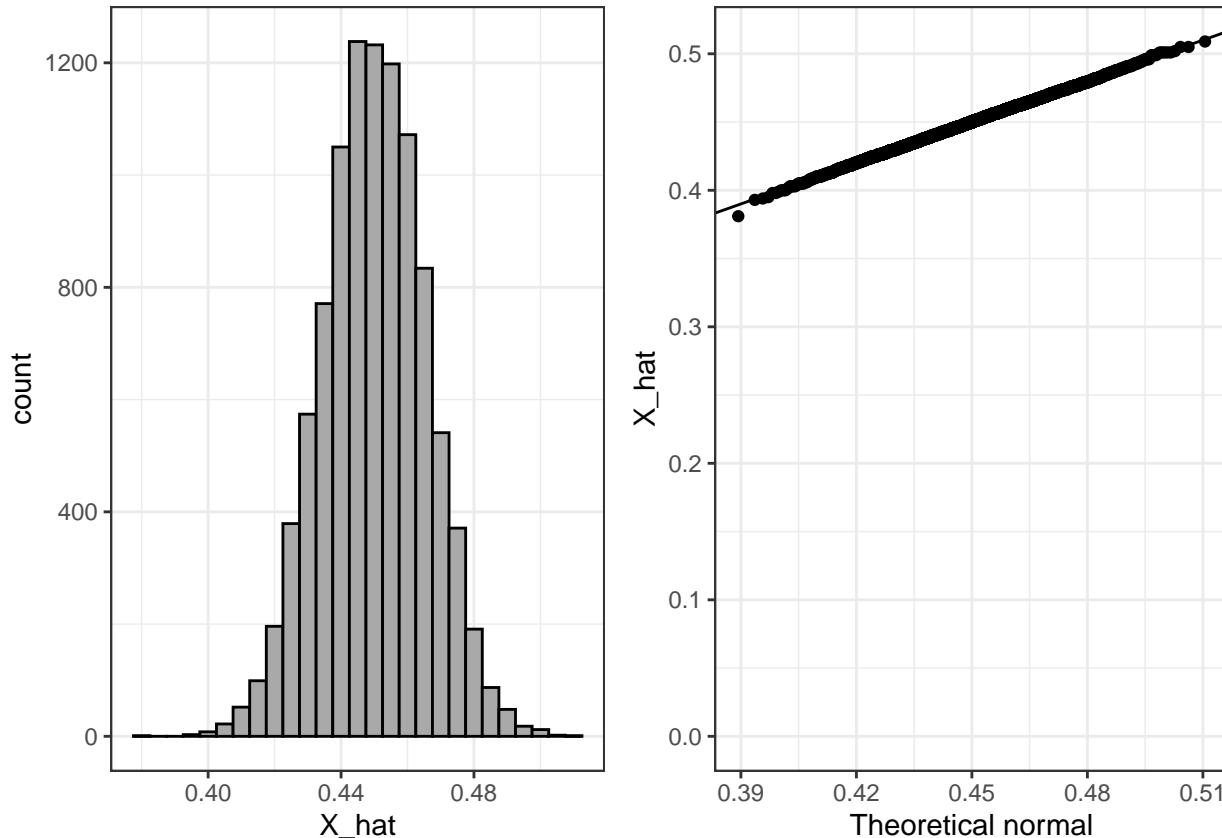
p2 <- data.frame(X_hat = X_hat) %>%
  ggplot(aes(sample=X_hat)) +
  stat_qq(dparams = list(mean = mean(X_hat), sd = sd(X_hat))) +
  geom_abline()
```

```

ylab("X_hat") +
xlab("Theoretical normal")

grid.arrange(p1, p2, nrow = 1)

```



#### 4.2.3. Dispersión

Recordemos que la competición es para predecir la dispersión, no la proporción  $p$ . Además, y dado que solo hay dos partidos, la dispersión es  $p - (1 - p) = 2p - 1$ . Así, puede hacerse un estimado mediante  $2\bar{X} - 1$  (que también implica que el nuevo error estándar se multiplica por 2).

Si  $p = 0,48$  con un margen de error de 0,2, entonces el estimador de la dispersión es 0,04 con un error estándar de 0,4.

#### 4.2.4. Tamaño de las encuestas

Supóngase que se realiza una encuesta de 100 000 personas, entonces, el margen de error será muy pequeño, sea cual sea el valor real de  $p$ :

```

N <- 100000

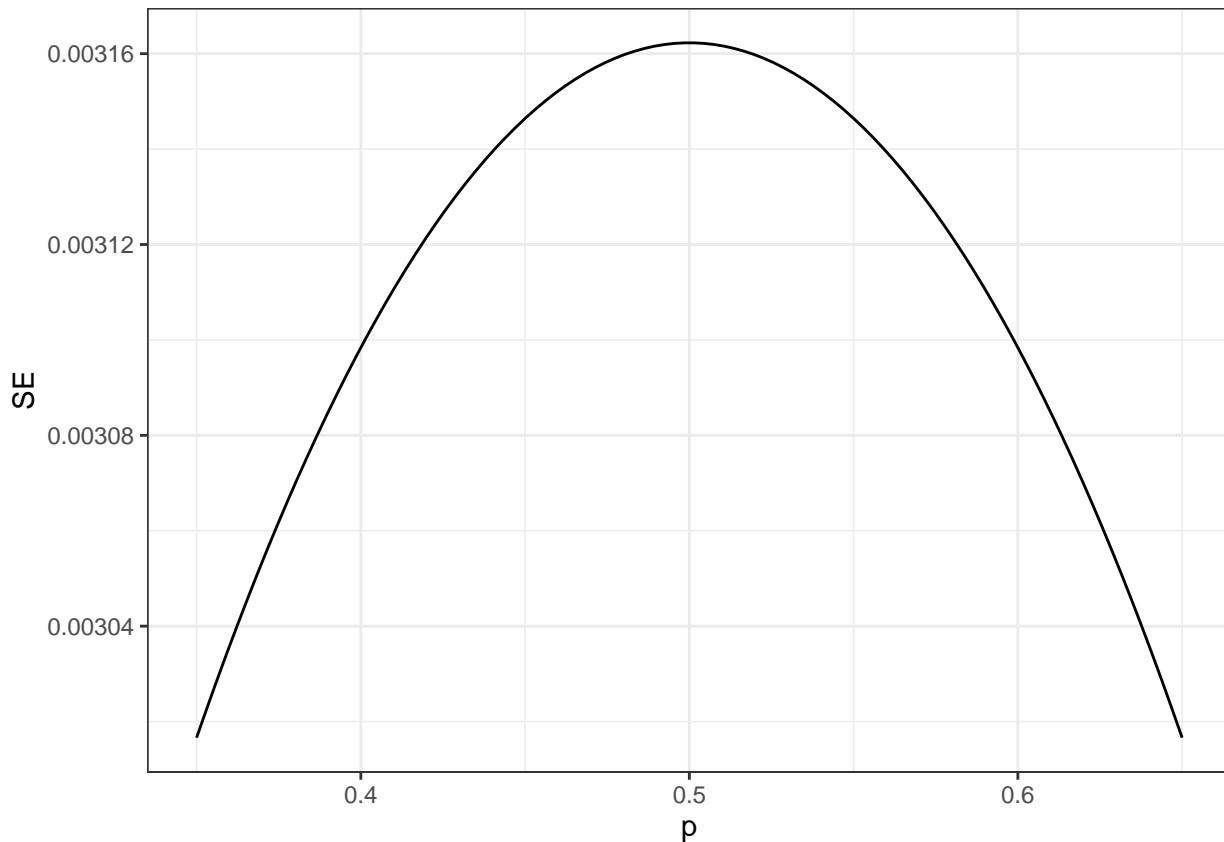
p <- seq(0.35, 0.65, length = 100)

SE <- sapply(p, function(x) 2*sqrt(x*(1-x)/N))

data.frame(p = p, SE = SE) %>%
  ggplot(aes(p, SE)) +

```

```
geom_line()
```



Sin embargo,  $N = 100000$  no es realista, dados los costos que implicaría, las heterogeneidad de las respuestas, la representatividad de la encuesta. Si se encuentra alguno de estos fenómenos, u otros muchos, se dice que las estimaciones pueden estar sesgadas, por lo que el estimador de  $p$  que encontramos puede estar muy alejado del valor real de la población.

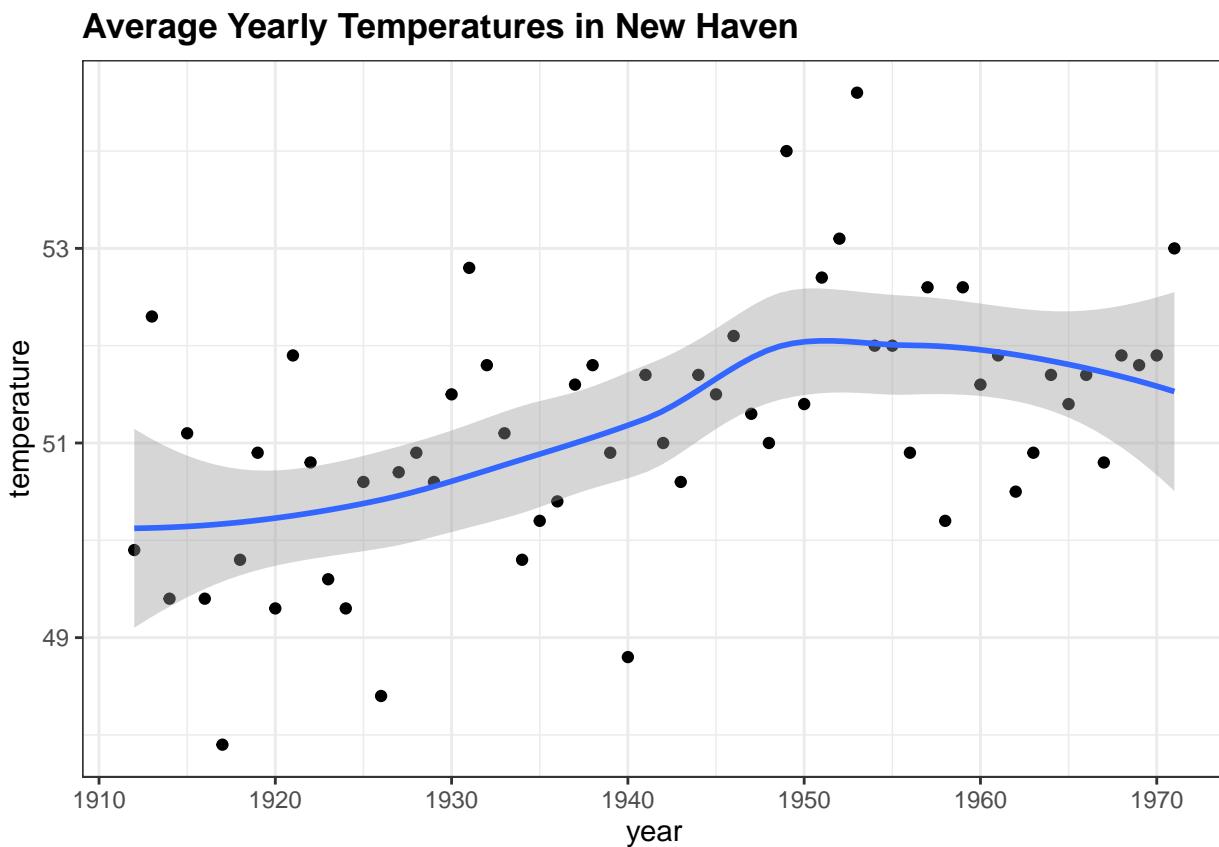
### 4.3. Intervalos de confianza y valores-p

Una manera popular de visualizar los intervalos de confianza es mediante la función de ggplot “geom\_smooth()”:

```
library(tidyverse)

data("nhtemp")

data.frame(year = as.numeric(time(nhtemp)), temperature = as.numeric(nhtemp)) %>%
  ggplot(aes(year, temperature)) +
  geom_point() +
  geom_smooth() +
  ggtitle("Average Yearly Temperatures in New Haven")
```



Donde el área sombreada representa a los intervalos de confianza.

En el ejemplo que se ha utilizado, queremos obtener el intervalo de confianza para el valor del parámetro poblacional:

$$[\bar{X} - 2\hat{S}\bar{E}[\bar{X}], \bar{X} + 2\hat{S}\bar{E}[\bar{X}]]$$

Nótese que los límites de este intervalo son variables aleatorias, dado que cada vez que se toma una muestra, cambian. Para ilustrar esto, realícese una simulación Monte Carlo con los siguientes parámetros:

```
p <- 0.45
```

```
N <- 1000
```

```
X <- sample(c(0,1), N, replace = TRUE, prob = c(1-p, p))

X_hat <- mean(X)

SE_hat <- sqrt(X_hat*(1-X_hat)/N)

c(X_hat - 2*SE_hat, X_hat + 2*SE_hat)

## [1] 0.426 0.490
```

Así, para determinar la probabilidad de que este intervalo contenga a  $p$ , hay que calcular la siguiente probabilidad:

$$Pr(\bar{X} - 2\hat{SE}[\bar{X}] \leq p \leq \bar{X} + 2\hat{SE}[\bar{X}])$$

Que es equivalente a:

$$Pr\left(-2 \leq \frac{\bar{X} - p}{\hat{SE}[\bar{X}]} \leq 2\right) \leftrightarrow Pr(-2 \leq Z \leq 2) \approx 95\%$$

### Simulación Monte Carlo para intervalos de confianza

Puede hacerse una simulación Monte Carlo para confirmar que, efectivamente, un intervalo de confianza de 95 % incluye a  $p$  95 % del tiempo:

```
B <- 10000

inside <- replicate(B, {
  X <- sample(c(0,1), N, replace = TRUE, prob = c(1-p,p))
  X_hat <- mean(X)
  SE_hat <- sqrt(X_hat*(1-X_hat)/N)
  between(p, X_hat-2*SE_hat, X_hat + 2*SE_hat)
})

mean(inside)

## [1] 0.955
```

**Nota:** es importante recalcar que las variables aleatorias son los intervalos, no  $p$ . Así el porcentaje de confianza se refiere a la probabilidad de que el intervalo aleatorio contenga al parámetro de interés. Decir que  $p$  tiene un X % de probabilidades de estar dentro del intervalo es técnicamente incorrecto.

#### 4.3.1. Potencia

Recordemos que al muestrear 25 observaciones, el intervalo de confianza de la dispersión fue:

```
N <- 25

X_hat <- 0.48

(2*X_hat-1) + c(-2,2)*2*sqrt(X_hat*(1-X_hat)/sqrt(N))

## [1] -0.934 0.854
```

Nótese que el intervalo  $(\tau_1, \tau_2) = (-0.9337, 0.8537)$  incluye al 0, y el 0 implica un empate en la elección. El hecho de que el intervalo calculado incluya al 0 no implica una elección increíblemente cerrada, sino más bien que el tamaño de la muestra es insuficiente.

En la teoría estadística, esto se conoce como **falta de potencia**. Formalmente, **la potencia es la probabilidad de detectar dispersiones diferentes de 0**. Al incrementar el tamaño de muestra, se reduce el error estándar y, por tanto, hay más probabilidades de detectar la dirección de la dispersión.

#### 4.3.2. Valores-p

Los valores-p o *p-values* están relacionados con los intervalos de confianza. Considérese el ejemplo de las pelotas rojas y azules de nuevo y que ahora, en vez de estar interesados en la proporción de azules, ahora interesa ¿hay más pelotas azules que rojas? (¿es la dispersión mayor a 0?  $2p - 1 > 0$ ).

Si se toma una muestra de 100 pelotas y se observan 52 azules, implica una dispersión de 4 %. Si bien esto apuntaría a que existen más azules que blancas ( $52\% > 48\%$ ), podría ocurrir que la dispersión fuera 0 aun con la muestra que obtuvimos.

Así, pueden definirse hipótesis al respecto, la nula, donde no hay dispersión, y la alternativa, donde sí hay dispersión:

$$H_0 : p = 0,5 \quad vs. \quad H_1 : p \neq 0,5$$

Así, el p valor es la probabilidad de detectar un efecto de cierto tamaño cuando la hipótesis nula es verdadera:

$$Pr(|\bar{X} - 0,5|) > 0,02)$$

$$\begin{aligned} Pr \left( \sqrt{n} \frac{|\bar{X} - 0,5|}{\sqrt{0,5(1 - 0,5)}} > \sqrt{n} \frac{0,02}{\sqrt{0,5(1 - 0,5)}} \right) \\ Pr \left( \sqrt{n} \frac{|\bar{X} - 0,5|}{0,5} > Z \right) \end{aligned}$$

```
N <- 100
z <- sqrt(N)*0.02/0.5
1-(pnorm(z)- pnorm(-z))
## [1] 0.689
```

Es decir, existe una probabilidad de 68.91 % de, en una muestra, observar 52 pelotas azules o más, incluso si el parámetro poblacional real es  $p = 0,5$ .

**Nota:** Si un intervalo de confianza de 95 % de la dispersión no incluye al 0, entonces el valor-p debe ser menor a 0.05.

## 4.4. Modelos estadísticos

```
library(tidyverse)
library(dslabs)
```

### 4.4.1. Agregados de encuestas

Generemos resultados para 12 encuestas tomadas una semana antes de la elección presidencial de EE.UU. de 2012; repórtense intervalos de confianza al 95 % utilizando la dispersión real 0.039, los tamaños de muestra que se observaron ( $N_s$ ) y la proporción de votos Demócratas ( $p$ ):

```
d <- 0.039

Ns <- c(1298, 533, 1342, 897, 774, 254, 812, 324, 1291, 1056, 2172, 516)

p <- (d+1)/2

confidence_intervals <- sapply(Ns, function(N){
  X <- sample(c(0,1), N, replace = TRUE, prob=c(1-p, p))
  X_hat <- mean(X)
  SE_hat <- sqrt(X_hat*(1-X_hat)/N)
  2*c(X_hat, X_hat - 2*SE_hat, X_hat + 2*SE_hat)-1
})
```

Ahora, se generará un data frame con todos los resultados de las 12 encuestas simuladas mediante simulaciones Monte Carlo. Recuérdese que se está estimando la dispersión (es decir, una dispersión mayor a 0 implica una ventaja de Obama y una dispersión menor a 0 implica una ventaja a Romney):

```
polls <- data.frame(poll = 1:ncol(confidence_intervals),
                     t(confidence_intervals),
                     sample_size = Ns)

names(polls) <- c("poll", "estimate", "low", "high", "sample_size")

polls

##      poll estimate      low    high sample_size
## 1       1   0.04314 -0.01232 0.0986      1298
## 2       2   0.07317 -0.01323 0.1596      533
## 3       3   0.02086 -0.03372 0.0754     1342
## 4       4   0.09030  0.02380 0.1568      897
## 5       5  -0.01034 -0.08222 0.0615      774
## 6       6   0.04724 -0.07811 0.1726      254
## 7       7   0.00985 -0.06033 0.0800      812
## 8       8   0.11728  0.00694 0.2276      324
## 9       9   0.03486 -0.02077 0.0905     1291
## 10     10   0.04167 -0.01983 0.1032      1056
## 11     11   0.03407 -0.00882 0.0770     2172
## 12     12   0.07364 -0.01416 0.1614      516
```

Si se observan los valores mínimos y máximos en cada encuesta, podría pensarse que la elección se encuentra muy cerrada y el resultado será un volado. Sin embargo, este análisis ignora algo esencial: al agregar los resultados de varias encuestas, se puede aumentar significativamente la precisión de las estimaciones. Así, puede construirse una *poll of polls* con una muestra sustantivamente grande y reduciendo el intervalo de confianza de 95 % bastante.

En principio, puede no tenerse acceso a los datos originales de cada encuesta; no obstante, matemáticamente

puede construirse el resultado aproximado de lo que se hubiera obtenido de haber llevado a cabo una encuesta gigante con el siguiente número de encuestados:

```
sum(polls$sample_size)
```

```
## [1] 11269
```

Constrúyase un estimador de la difusión  $d$ :

```
d_hat <- polls %>%
  summarize(avg = sum(estimate*sample_size)/sum(sample_size)) %>%
  .$avg
```

```
d_hat
```

```
## [1] 0.0404
```

Con este estimador, puede hacerse una estimación de la proporción de votos para Obama y el margen de error de la encuesta grande:

```
p_hat <- (1-d_hat)/2
```

```
moe <- 2*1.96*sqrt(p_hat*(1-p_hat)/sum(polls$sample_size))
```

```
moe
```

```
## [1] 0.0184
```

Finalmente, con el promedio ponderado, puede predecirse que la difusión será de 3.6 % con un margen de error de 1.8 %.

```
round(d_hat*100, 1)
```

```
## [1] 4
```

```
round(moe*100, 1)
```

```
## [1] 1.8
```

#### 4.4.2. Modelos multínivel y sesgos

Los agregadores de encuestas generan diferentes modelos de resultados electorales a partir de los mismos datos de encuestas. Esto se debe a que utilizan modelos estadísticos.

Aplicaremos este análisis para predecir los resultados del voto popular en EE.UU. en 2016. Para ello se utilizarán los datos de FiveThirtyEight, que incluye los resultados de encuestas nacionales y estatales realizadas el año de la elección:

```
library(dslabs)
```

```
library(tidyverse)
```

```
library(ggplot2)
```

```
data("polls_us_election_2016")
```

```
names(polls_us_election_2016)
```

```
## [1] "state"           "startdate"        "enddate"          "pollster"
## [5] "grade"           "samplesize"       "population"      "rawpoll_clinton"
## [9] "rawpoll_trump"   "rawpoll_johnson"  "rawpoll_mcmullin" "adjpoll_clinton"
```

```
## [13] "adjpoll_trump"    "adjpoll_johnson"   "adjpoll_mcmullin"
```

Filtraremos a las encuestas que se tomaron en la semana previa a la elección y aquellas que se consideraron fiables (con una calificación mayor o igual a B):

```
polls <- polls_us_election_2016 %>%
  filter(state == "U.S." & enddate >= "2016-10-31" &
         (grade %in% c("A+", "A", "A-", "B+") | is.na(grade)))
```

Constrúyase también un estimador de la difusión:

```
polls <- polls %>%
  mutate(spread = rawpoll_clinton/100 - rawpoll_trump/100)
```

Se asumirán solamente dos partidos y  $p$  la proporción del voto para Clinton, lo que implica que  $1 - p$  es la proporción votante por Trump. Así, la difusión es  $d = 2p - 1$ . Por otro lado, sabemos que los resultados de las encuestas son una variable aleatoria normal con un valor esperado de la difusión  $d$  y un error estándar  $2\sqrt{p(1-p)/n}$ .

Asumiendo que el modelo descrito es adecuado, puede utilizarse esta información para construir un intervalo de confianza con base en los datos agregados. De tal forma, los estimador de la difusión y del error estándar son:

```
d_hat <- polls %>%
  summarize(d_hat = sum(spread*samplesize) / sum(samplesize)) %>%
  .$d_hat

p_hat <- (d_hat+1)/2

moe <- 1.96*2*sqrt(p_hat*(1-p_hat)/sum(polls$samplesize))

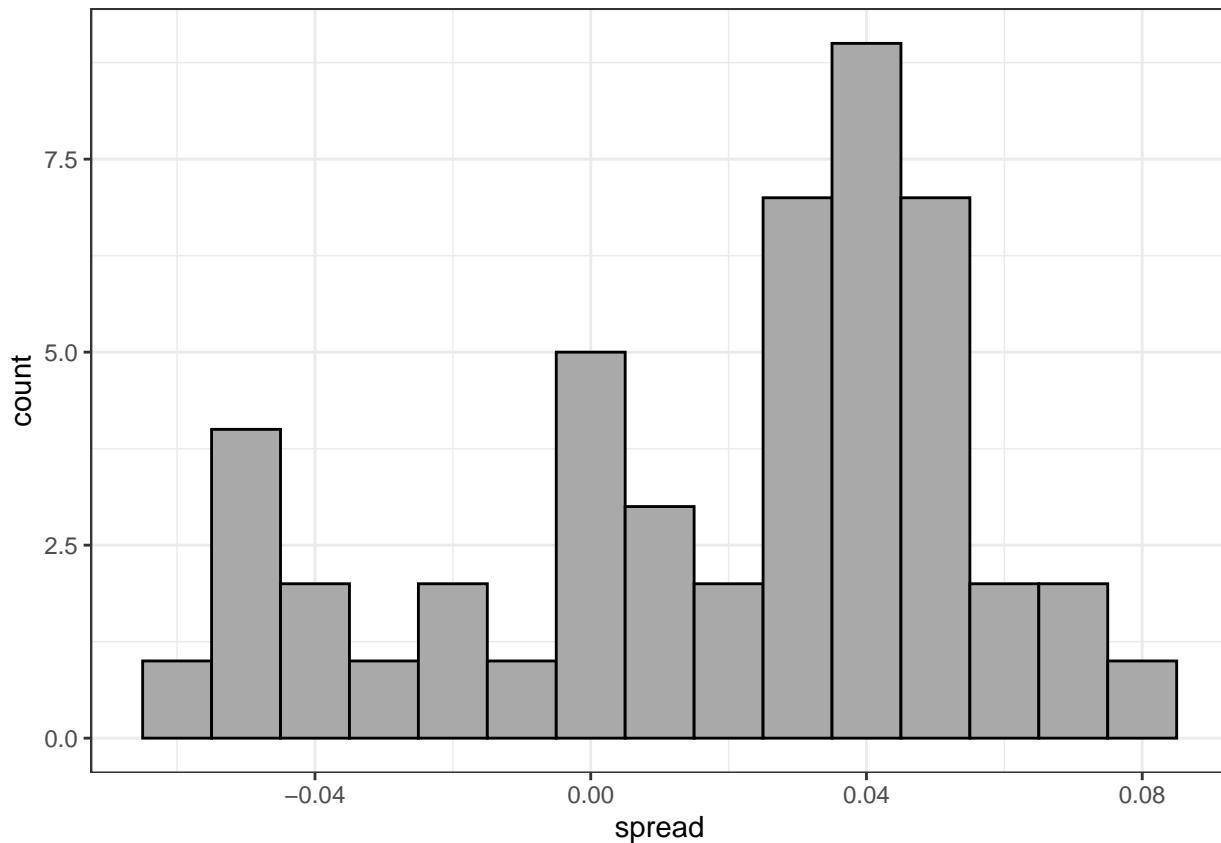
moe

## [1] 0.00662
d_hat

## [1] 0.0143
```

Obsérvese que, de acuerdo a estos datos, la difusión debería ser 1.42% en favor de Clinton con un margen de error de 0.66%. No obstante, en la realidad se observó una difusión de 2.1%, lo cual está fuera del intervalo de confianza al 95%. ¿Qué ocurrió? Puede hacerse un histograma para visualizar las difusiones

```
polls %>%
  ggplot(aes(spread)) +
  geom_histogram(binwidth = 0.01, color = "black", fill = "darkgray")
```



Es evidente que los datos no están normalmente distribuidos y que el error estándar parece ser mayor a 0.66 %. Notese que varias encuestadoras tomaron varias encuestas esa semana:

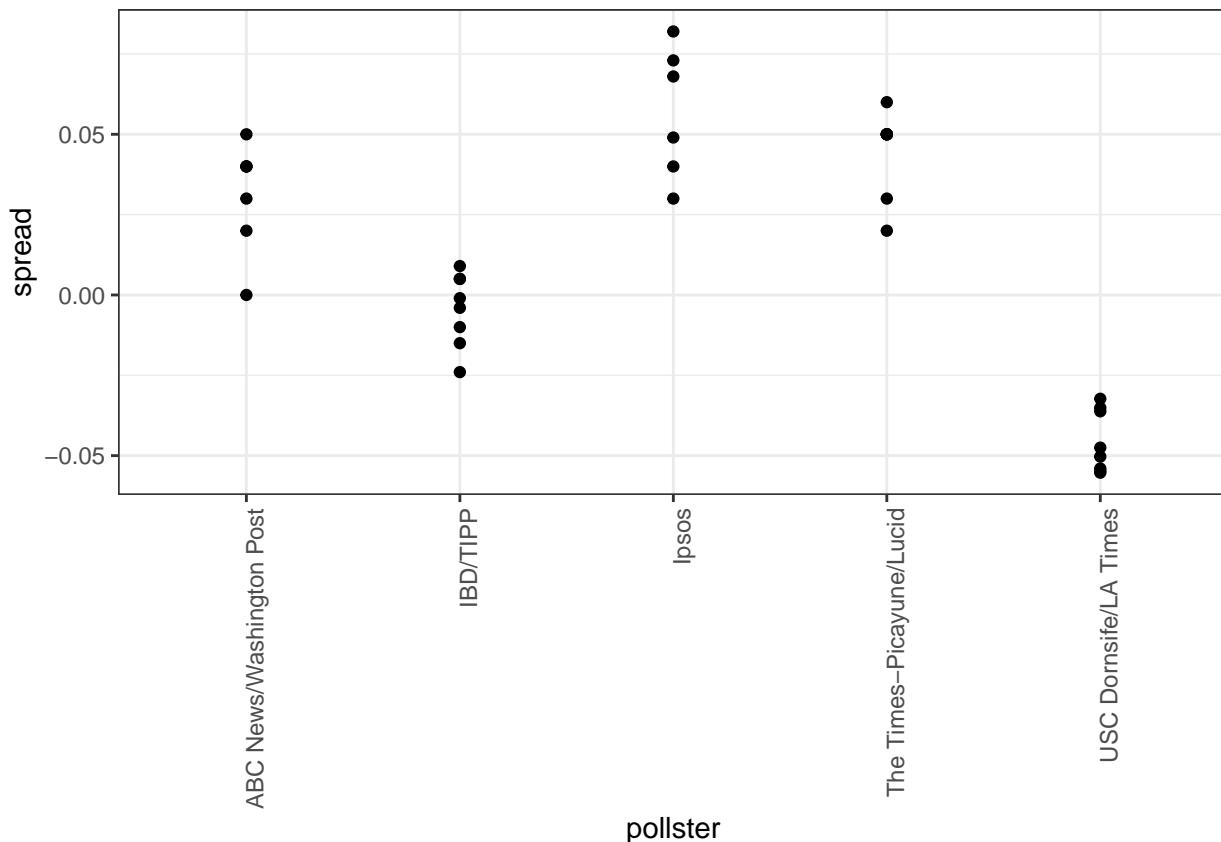
```
polls %>%
  group_by(pollster) %>%
  summarize(n())

## # A tibble: 15 x 2
##   pollster          `n()`
##   <fct>            <int>
## 1 ABC News/Washington Post      7
## 2 Angus Reid Global           1
## 3 CBS News/New York Times     2
## 4 Fox News/Anderson Robbins Research/Shaw & Company Research 2
## 5 IBD/TIPP                   8
## 6 Insights West                1
## 7 Ipsos                      6
## 8 Marist College               1
## 9 Monmouth University          1
## 10 Morning Consult              1
## 11 NBC News/Wall Street Journal 1
## 12 RKM Research and Communications, Inc. 1
## 13 Selzer & Company             1
## 14 The Times-Picayune/Lucid     8
## 15 USC Dornsife/LA Times       8
```

Analicemos los datos para las encuestadoras que realizaron 6 encuestas o más en la última semana antes de

la elección:

```
polls %>%
  group_by(pollster) %>%
  filter(n() >= 6) %>%
  ggplot(aes(pollster, spread)) +
  geom_point() +
  theme(axis.text.x = element_text(angle=90, hjust =1))
```



Esto es un resultado interesante: reportemos los errores estándar teóricos de cada encuestadora, los cuales se encuentran entre 0.018 y 0.033

```
polls %>%
  group_by(pollster) %>%
  summarize(se = 2*sqrt(p_hat*(1-p_hat)/median(samplesize)))
```

```
## # A tibble: 15 x 2
##   pollster          se
##   <fct>        <dbl>
## 1 ABC News/Washington Post 0.0265
## 2 Angus Reid Global 0.0295
## 3 CBS News/New York Times 0.0292
## 4 Fox News/Anderson Robbins Research/Shaw & Company Research 0.0289
## 5 IBD/TIPP 0.0333
## 6 Insights West 0.0326
## 7 Ipsos 0.0225
## 8 Marist College 0.0326
## 9 Monmouth University 0.0366
```

## 10 Morning Consult	0.0260
## 11 NBC News/Wall Street Journal	0.0279
## 12 RKM Research and Communications, Inc.	0.0315
## 13 Selzer & Company	0.0354
## 14 The Times-Picayune/Lucid	0.0196
## 15 USC Dornsife/LA Times	0.0183

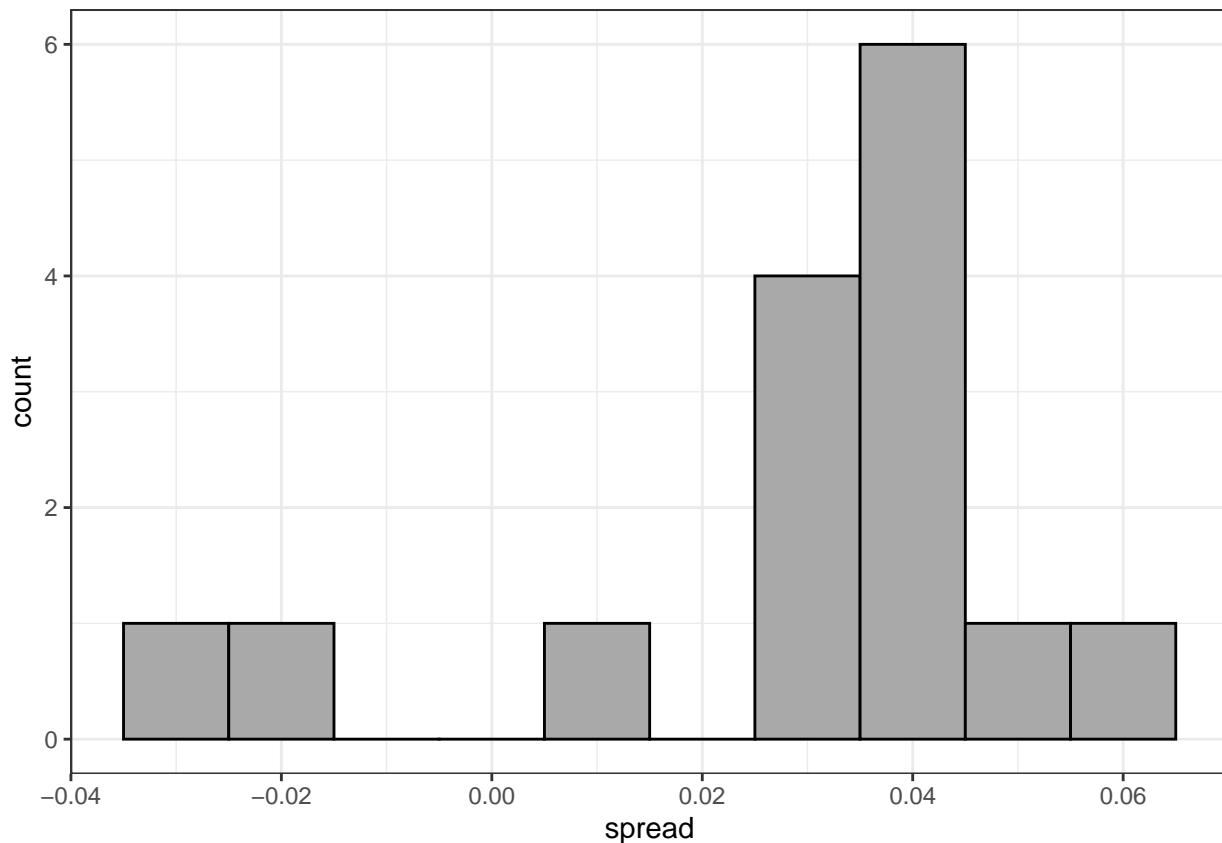
Así, la gráfica anterior demuestra que las encuestadoras reportaron diferentes resultados, algunos en los que ganaba Trump y otros en los que ganaba Clinton. El **sesgo del encuestador** refleja el hecho de que encuestas repetidas por una misma encuestadora tienen un valor esperado diferente a la difusión actual y diferente de otras encuestadoras. Así, cada encuestadora tiene un sesgo único.

#### 4.4.3. Modelos basados en datos

Para cada encuestadora, recolectemos su último resultado reportado antes de la elección:

```
one_poll_per_pollster <- polls %>%
  group_by(pollster) %>%
  filter(enddate==max(enddate)) %>%
  ungroup()

one_poll_per_pollster %>%
  ggplot(aes(spread)) +
  geom_histogram(binwidth = 0.01, color = "black", fill = "darkgray")
```



Se asume que el valor esperado de estas encuestas es igual a la difusión real  $d$ . Por tal motivo, el error estándar ahora incluye a la variabilidad entre encuestadoras. Además, ahora hay dos parámetros desconocidos,  $d$  y la desviación estándar  $\sigma$ . Este último se puede estimar mediante la desviación estándar de la muestra de 15

encuestadoras a través de la fórmula:

$$s = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})^2$$

```
sd(one_poll_per_pollster$spread)

## [1] 0.0242

results <- one_poll_per_pollster %>%
  summarize(avg = mean(spread), se = sd(spread)/sqrt(length(spread))) %>%
  mutate(start = avg - 1.96*se, end = avg +1.96*se)

round(results*100, 1)

##   avg   se start end
## 1 2.9  0.6   1.7  4.1
```

## 4.5. Estadística Bayesiana

En el modelo de la urna, no tiene sentido hablar de la probabilidad de que  $p$  sea mayor a cierto valor porque  $p$  es un valor fijo. Con la estadística Bayesiana, se asumirá que  $p$  es de hecho aleatoria, lo que permite calcular las probabilidades relacionadas con ella. Los modelos **jerárquicos** describen la probabilidad en diferentes niveles y los incorporan en un modelo para estimar  $p$ .

### 4.5.1. Teorema de Bayes

El Teorema de Bayes indica que la probabilidad de que ocurra un evento A, dado que ocurrió un evento B, es igual a la probabilidad de que ambos ocurran entre la probabilidad del evento B:

$$Pr(A|B) = \frac{Pr(A \cap B)}{Pr(B)} = \frac{Pr(B|A)Pr(A)}{Pr(B)}$$

Como ejemplo, supóngase una prueba para detectar una enfermedad con una precisión de 99 %. Se usará la siguiente notación: la probabilidad de que el test sea positivo dado que el paciente efectivamente está enfermo, es 99 %; similarmente, la probabilidad de que el test sea negativo dado que el paciente no tiene la enfermedad es 99 %:

$$Prob(+|D = 1) = 0,99 \quad Prob(-|D = 0) = 0,99$$

Si se selecciona una persona al azar y el testo arroja un resultado positivo, ¿cuál es la probabilidad de que esté enferma? Esto es, queremos saber  $Prob(D = 1|+)$ . Supóngase que la enfermedad tiene una incidencia de 1 en 3 900 personas, que implica que  $Pr(D = 1) = 0,00025$ . Mediante el Teorema de Bayes:

$$\begin{aligned} Pr(D = 1|+) &= \frac{Pr(+|D = 1)Pr(D = 1)}{Pr(+)} = \frac{Pr(+|D = 1)Pr(D = 1)}{Pr(+|D = 1)Pr(D = 1) + Pr(+|D = 0)Pr(D = 0)} \\ &\leftrightarrow \\ Pr(D = 1|+) &= \frac{(0,99)(0,00025)}{(0,99)(0,00025) + (0,01)(0,99975)} = \boxed{0,02} \end{aligned}$$

Lo cual es un resultado contraintuitivo: a pesar de que la prueba tiene una precisión de 99 %, la probabilidad de estar enfermo dado un resultado de la prueba positivo es solo 2 %.

Para ilustrar lo anterior, realícese una simulación Monte Carlo donde se seleccionan aleatoriamente 100 000 personas donde la enfermedad en cuestión tiene una incidencia de 1 en 3 900:

```
prev <- 0.00025
```

```
N <- 100000
```

```
outcome <- sample(c("Disease", "Healthy"), N, replace = TRUE, prob = c(prev, 1-prev))
```

Nótese que como la prevalencia es sumamente baja, cuando se toma una muestra el número de enfermos es también bajo, mientras que las personas sanas son muchas:

```
N_D <- sum(outcome == "Disease")
```

```
N_D
```

```
## [1] 30
```

```
N_H <- sum(outcome == "Healthy")
```

```
N_H
```

```
## [1] 99970
```

Esto implica que la posibilidad de encontrar falsos negativos es bastante alta:

```
accuracy <- 0.99
```

```
test <- vector("character", N)
```

```
test[outcome == "Disease"] <- sample(c("+", "-"), N_D, replace = TRUE, prob = c(accuracy, 1 - accuracy))
test[outcome == "Healthy"] <- sample(c("-", "+"), N_H, replace = TRUE, prob = c(accuracy, 1 - accuracy))
```

Puede hacerse una tabla que muestre el número de personas en cada una de las 4 combinaciones de resultados:

```
table(outcome, test)
```

```
##          test
## outcome      -      +
##   Disease     0     30
##   Healthy 98970  1000
```

#### 4.5.2. Bayes en la práctica

Para ilustrar la utilidad de los modelos jerárquicos y Bayesianos en la práctica, piénsese en un ejemplo de béisbol. Supóngase un jugador que, con 20 turnos al bate, conectó 9 hits, lo que implica un avg de 0.450. Dado que, en 80 años, ningún jugador ha terminado una temporada con un promedio de bateo de más de 0.400, trataremos de predecir el promedio de este jugador al final de su temporada.

En una temporada habitual, un jugador tiene alrededor de 500 turnos al bate. Así, pueden pensarse en los resultados de manera binomial con una tasa de éxito de  $p$ ; el error estándar de 20 turnos al bateo es entonces:

$$\sqrt{\frac{0,450(1 - 0,450)}{20}} = 0,111$$

Con esto puede construirse un intervalo de confianza al 95 %:  $(\tau_1, \tau_2) = (0,228, 0,672)$ . Sin embargo, esta predicción tiene dos problemas: el primero es que es demasiado amplio, por lo que no es muy útil; segundo, que tiene su centro en 0.450, lo que implica que nuestra predicción es que el jugador terminará la temporada con un promedio de 0.450.

#### 4.5.3. El modelo jerárquico

El modelo jerárquico provee una descripción matemática de por qué se observa un valor de 0.450.

Primero, se escoge un jugador al azar con talento intrínseco  $p$ , la proporción de veces que conectarán hits. Así, se observan 20 resultados aleatorios con probabilidad de éxito  $p$ , por lo que se usa un modelo para representar los dos niveles de variabilidad en los datos. En primer lugar, cada jugador tiene asignada una habilidad natural para batear, representada por  $p \sim N(0,270, 0,027)$ ; en segundo lugar, la variabilidad también se determina por la suerte al batear.

Por el TLC, sabemos que el promedio observado es  $Y|p \sim N(p, \sigma)$ . Debido a la existencia de estos dos niveles es que se llaman **modelos jerárquicos**, formados por distribución *a priori* y la distribución muestral (variabilidad jugador a jugador y variabilidad por suerte al batear).

Así, para el caso del jugador particular, el modelo jerárquico está definido mediante:

$$p \sim N(0,275, 0,027) \quad Y|p \sim N(p, 0,111)$$

A partir de lo anterior se computará una distribución posterior con el objetivo de estimar el valor de  $p$  mediante una función de probabilidad posterior de  $p$ :

$$\mathbf{E}[\mathbf{p}|\mathbf{y}] = \mathbf{B}\mu + (\mathbf{1} - \mathbf{B})\mathbf{Y} = \mu + (\mathbf{1} - \mathbf{B})(\mathbf{Y} - \mu)$$

$$\mathbf{B} = \frac{\sigma^2}{\sigma^2 + \tau^2}$$

Sustituyendo los valores que conocemos:

$$E[p|Y = 0,450] = 0,275B + 0,450(1 - B) = 0,275(1 - B)(0,450 - 0,275)$$

$$B = \frac{0,111^2}{0,111^2 + 0,027^2} = 0,944$$

$$E[p|Y = 0,450] \approx \boxed{0,285}$$

Con error estándar de:

$$SE[p|y]^2 = \frac{1}{1/\sigma^2 + 1/\tau^2} = \frac{1}{1/0,111^2 + 1/0,027^2} = 0,00069$$

## 4.6. Pronósticos electorales

Recapitulemos el enfoque Bayesiano de predicción:

- $d \sim N(\mu, \tau)$  describe la mejor conjetura que podemos hacer sin haber observado los datos de ninguna encuesta.
- $\bar{X}|d \sim N(d, \sigma)$  describe la aleatoriedad derivada del muestreo y del sesgo de cada encuestadora.

Si definimos  $\mu = 0$ , significa que partimos de un modelo que no provee ninguna información sobre el posible ganador de la elección; para el caso de la desviación estándar, podemos recurrir a los datos históricos observados, los cuales indican que el ganador del voto popular tiene una dispersión promedio de alrededor de 3.5 %,  $\tau = 0,035$ .

Así, mediante las fórmulas de la distribución posterior que revisamos, puede obtenerse la probabilidad de que  $d > 0$ . Con el código siguiente, se computan la media posterior y el error estándar posterior:

```
mu <- 0

tau <- 0.035

sigma <- results$se

Y <- results$avg

B <- sigma^2/(sigma^2+tau^2)

posterior_mean <- B*mu+(1-B)*Y

posterior_se <- sqrt(1/(1/sigma^2+1/tau^2))
```

posterior\_mean

```
## [1] 0.0281
posterior_se
```

```
## [1] 0.00615
```

Para realizar una afirmación probabilística, tómese en cuenta que la distribución posterior también es normal. Así, se reportará lo que se conoce como **intervalo creíble** al 95 %:

```
posterior_mean +c(-1.96, 1.96)*posterior_se
```

```
## [1] 0.0160 0.0401
```

También se puede reportar la probabilidad de que  $d > 0$  mediante la función “pnorm()”:

```
1-pnorm(0, posterior_mean, posterior_se)
```

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

Lo cual es una estimación sumamente optimista de Clinton ganando el voto popular y es bastante diferente a lo que reportó en su momento, por ejemplo, FiveThirtyEight, 81 %. ¿A qué se debe esta diferencia?

Una observación importante es que es común observar un sesgo general que afecta a casi todas las encuestadoras de la misma manera. Si bien no existe una explicación precisa sobre por qué ocurre esto, se ha observado de manera histórica: en una elección puede existir un sesgo hacia los Demócratas, otra hacia los Republicanos, otra hacia ninguno, etc. Así, se ha concluido que, en la elección de 2016, existió un sesgo hacia el Partido Demócrata de 1-2 %. Nótese que este sesgo solo puede ser determinado *a posteriori* y no antes de la elección.

#### 4.6.1. Representación matemática de modelos

Supóngase que se está recolectando datos de una encuestadora particular y que no existe ningún sesgo general y en donde se recolectan diferentes encuestas de tamaño  $n$ , de tal manera que existen diferentes medidas de la dispersión, sean  $X_1, \dots, X_J$ .

La teoría que hemos revisado indica que estas variables aleatorias tienen un valor esperado  $d$  y un error estándar  $2\sqrt{p(1-p)/n}$ . Consecuentemente, podemos representar esta información mediante un modelo como el que se enuncia a continuación:

$$X_j = d + \varepsilon_j$$

Donde  $\varepsilon$  es un término de error y una variable aleatoria que explica la variabilidad entre encuestas y el subíndice  $j$  representa las diferentes encuestas levantadas. Además, se asume que  $\varepsilon$  tiene un valor esperado de 0 y un error estándar de  $2\sqrt{p(1-p)/n}$ .

Supóngase que  $d = 2.1$  y que  $n = 2000$ , se pueden simular los puntos para 6 encuestadoras:

```
J <- 6
N <- 2000
d <- 0.021
p <- (d+1)/2
X <- d + rnorm(J, 0, 2*sqrt(p*(1-p)/N))
```

Ahora supóngase que se tienen 6 datos de 5 encuestadoras diferentes, lo que implica la necesidad de dos subíndices, uno para cada encuestadora y otro para cada encuesta,  $X_{ij}$ . Así, el modelo evoluciona a:

$$X_{ij} = d + \varepsilon_{ij}$$

Para simular los datos del modelo anterior, es necesario realizar un bucle mediante la función “sapply()”:

```
I <- 5
J <- 6
N <- 2000
X <- sapply(1:I, function(i){
  d + rnorm(J, 0, 2*sqrt(p*(1-p)/N))
})
X
##      [,1]     [,2]     [,3]     [,4]     [,5]
## [1,] -0.007375 -0.00740 -0.01858  0.0198 -0.03338
## [2,] -0.000998  0.02219  0.01597  0.0347 -0.01289
## [3,]  0.039448  0.01195  0.07503  0.0489  0.00795
## [4,] -0.010437  0.00461  0.03296  0.0105  0.00871
## [5,]  0.024032  0.02071  0.01547  0.0588  0.03876
## [6,] -0.035377  0.04400  0.00362  0.0710  0.00304
```

Sin embargo, este modelo no toma en cuenta la variabilidad entre encuestadoras, por lo que se agregará un nuevo término  $h_i$ , que representa el efecto o sesgo interno de cada casa encuestadora:

$$X_{ij} = d + h_i + \varepsilon_{ij}$$

Para simular los datos de una encuestadora específica, es necesario tomar una  $h_i$  de cada una y después sumar los  $\varepsilon$  (asumiéndose que la variabilidad entre encuestadoras es 2.5 %):

```
I <- 5
J <- 6
N <- 2000
d <- 0.021
p <- (d+1)/2
h <- rnorm(I, 0, 0.025)
X <- sapply(1:I, function(i){
  d + h[i] + rnorm(J, 0, 2*sqrt(p*(1-p)/N))
})
X
```

##	[,1]	[,2]	[,3]	[,4]	[,5]	
##	[1,]	0.01141	0.030766	0.030708	-0.0138	0.04055
##	[2,]	0.00432	0.026113	0.011656	0.0384	0.01079
##	[3,]	0.05344	0.046321	0.000611	-0.0376	0.05293
##	[4,]	0.02640	0.000426	0.035204	0.0251	-0.00724
##	[5,]	0.00814	0.024701	0.011410	0.0089	0.03074
##	[6,]	0.03661	0.014075	-0.011908	-0.0139	0.02268

Hasta ahora hemos asumido que el sesgo general es igual a 0; sin embargo, y como se mencionó, históricamente se ha observado un sesgo general inclinado hacia algún partido en la mayoría de las elecciones. Por tanto, se agrega un término por este sesgo general,  $b$ , la cual tiene una media de 0 y un error estándar de 2.5 %:

$$X_{ij} = d + b + h_i + \varepsilon_{ij}$$

```
mu <- 0
tau <- 0.035
sigma <- sqrt(results$se^2 + 0.025^2)
Y <- results$avg
B <- sigma^2/(sigma^2+tau^2)
posterior_mean <- B*mu+(1-B)*Y
posterior_se <- sqrt(1/(1/sigma^2+1/tau^2))
1- pnorm(0, posterior_mean, posterior_se)
## [1] 0.817
```

La cual es un resultado mucho más cercano a lo reportado por FiveThirtyEight.

#### 4.6.2. Prediciendo el Colegio Electoral

Sabemos que en EE.UU. las elecciones no se deciden mediante el voto popular, sino a través de los votos ganados en el Colegio Electoral. A continuación se presentan los cinco estados con mayor número de votos electorales:

```
library(tidyverse)
library(dslabs)
data("polls_us_election_2016")

results_us_election_2016 %>% arrange(desc(electoral_votes)) %>% top_n(5, electoral_votes)

## # A tibble: 5 x 5
##   state electoral_votes clinton trump others
##   <chr>        <dbl>    <dbl>   <dbl>   <dbl>
## 1 California      55     61.7    31.6    6.7
## 2 Texas          38     43.2    52.2    4.5
## 3 Florida         29     47.8    49.0    3.2
## 4 New York        29     59.0    36.5    4.5
## 5 Illinois        20     55.8    38.8    5.4
## 6 Pennsylvania    20     47.9    48.6    3.6
```

Para predecir los resultados del Colegio Electoral en 2016, comenzaremos por agregar los resultados de las encuestas tomadas durante la última semana antes de la elección, computar la difusión y el promedio y desviación estándar de cada estado:

```
results <- polls_us_election_2016 %>%
  filter(state != "U.S." &
         !grepl("CD", state) &
         enddate >= "2016-10-31" &
         (grade %in% c("A+", "A", "A-", "B+") | is.na(grade))) %>%
  mutate(spread = rawpoll_clinton/100 - rawpoll_trump/100) %>%
  group_by(state) %>%
  summarize(avg = mean(spread), sd = sd(spread), n = n()) %>%
  mutate(state = as.character(state))
```

A continuación se presentan las elecciones más cerradas de acuerdo a la manipulación realizada:

```
results %>%
  arrange(abs(avg))

## # A tibble: 47 x 4
##   state           avg      sd      n
##   <chr>        <dbl>  <dbl>  <int>
## 1 Florida       0.00356 0.0163     7
## 2 North Carolina -0.0073  0.0306     9
## 3 Ohio          -0.0104  0.0252     6
## 4 Nevada        0.0169  0.0441     7
## 5 Iowa          -0.0197  0.0437     3
## 6 Michigan       0.0209  0.0203     6
## 7 Arizona        -0.0326  0.0270     9
## 8 Pennsylvania    0.0353  0.0116     9
## 9 New Mexico     0.0389  0.0226     6
## 10 Georgia       -0.0448  0.0238     4
## # ... with 37 more rows
```

Ahora, usaremos una función llamada “left\_join()”, la cual permite agregar el número de votos electorales de cada estado:

```
results <- left_join(results, results_us_election_2016, by="state")
```

También, nótese que no se realizaron encuestas en DC, Rhode Island, Alaska y Wyoming, dado que los resultados eran completamente predecibles:

```
results_us_election_2016 %>%
  filter(!state %in% results$state)
```

```
##           state electoral_votes clinton trump others
## 1      Rhode Island            4   54.4  38.9   6.7
## 2          Alaska             3   36.6  51.3  12.2
## 3        Wyoming             3   21.9  68.2  10.0
## 4 District of Columbia       3   90.9   4.1   5.0
```

El siguiente código asigna una desviación estándar a los estados con solo una encuesta sustituyendo el valor faltante por la mediana de las desviaciones estándar de los otros estados.

```
results <- results %>%
  mutate(sd = ifelse(is.na(sd), median(results$sd, na.rm = TRUE), sd))
```

Ahora, se hará una simulación Monte Carlo para generar resultados de elecciones simuladas; posteriormente, se utilizará esto para hacer afirmaciones probabilísticas. Para ello, se asumirá  $\mu = 0$  y  $\tau = 0,02$

```
mu <- 0

tau <- 0.02

results %>% mutate(sigma = sd/sqrt(n),
  B = sigma^2 / (sigma^2 + tau^2),
  posterior_mean = B*mu + (1-B)*avg,
  posterior_se = sqrt( 1 / (1/sigma^2 + 1/tau^2))) %>%
  arrange(abs(posterior_mean))

## # A tibble: 47 x 12
##   state      avg     sd     n electoral_votes clinton trump others    sigma
##   <chr>     <dbl>   <dbl> <int>        <dbl>    <dbl> <dbl> <dbl>    <dbl>
## 1 Florida    0.00356 0.0163     7         29    47.8  49    3.2  0.00618
## 2 North Car~ -0.0073  0.0306     9         15    46.2  49.8   4  0.0102
## 3 Iowa      -0.0197  0.0437     3          6    41.7  51.1   7.1  0.0252
## 4 Ohio       -0.0104  0.0252     6         18    43.5  51.7   4.8  0.0103
## 5 Nevada     0.0169  0.0441     7          6    47.9  45.5   6.6  0.0167
## 6 Michigan    0.0209  0.0203     6         16    47.3  47.5   5.2  0.00827
## 7 Arizona    -0.0326  0.0270     9         11    45.1  48.7   6.2  0.00898
## 8 New Mexico  0.0389  0.0226     6          5    48.3  40    11.7  0.00921
## 9 Georgia    -0.0448  0.0238     4         16    45.9  51    3.1  0.0119
## 10 Pennsylva~  0.0353  0.0116    9         20    47.9  48.6   3.6  0.00387
## # ... with 37 more rows, and 3 more variables: B <dbl>, posterior_mean <dbl>,
## #   posterior_se <dbl>
```

Repítase lo anterior 10 000 veces:

```
mu <- 0

tau <- 0.02

clinton_EV <- replicate(1000, {
  results %>% mutate(sigma = sd/sqrt(n),
    B = sigma^2 / (sigma^2 + tau^2),
```

```

posterior_mean = B*mu + (1-B)*avg,
posterior_se = sqrt( 1 / (1/sigma^2 + 1/tau^2)),
simulated_result = rnorm(length(posterior_mean), posterior_mean, posterior_se),
clinton = ifelse(simulated_result > 0, electoral_votes, 0)) %>%
summarize(clinton = sum(clinton)) %>% # Votos totales para Clinton
.$clinton + 7      # 7 votos por Rhode Island y DC
}

mean(clinton_EV > 269)

## [1] 0.993

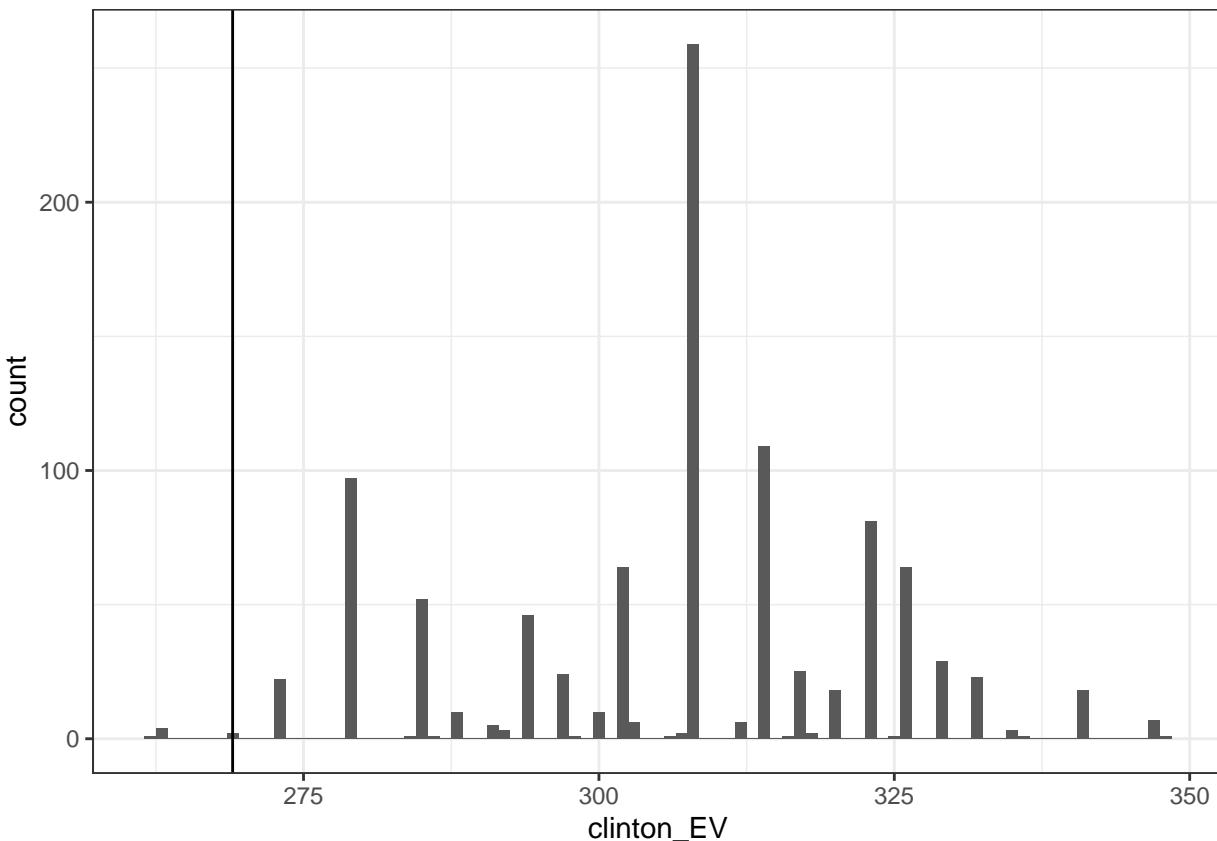
```

A continuación se presenta el histograma de los resultados:

```

data.frame(clinton_EV) %>%
  ggplot(aes(clinton_EV)) +
  geom_histogram(binwidth = 1) +
  geom_vline(xintercept = 269)

```



Los resultados anteriores parecen ser abrumadoramente favorables a Clinton. Sin embargo, estos ignoran el sesgo general. Ahora simularemos los resultados tomando en cuenta este sesgo, sea  $\sigma_b = 0,03$

```

mu <- 0

tau <- 0.02

bias_sd <- 0.03

```

```

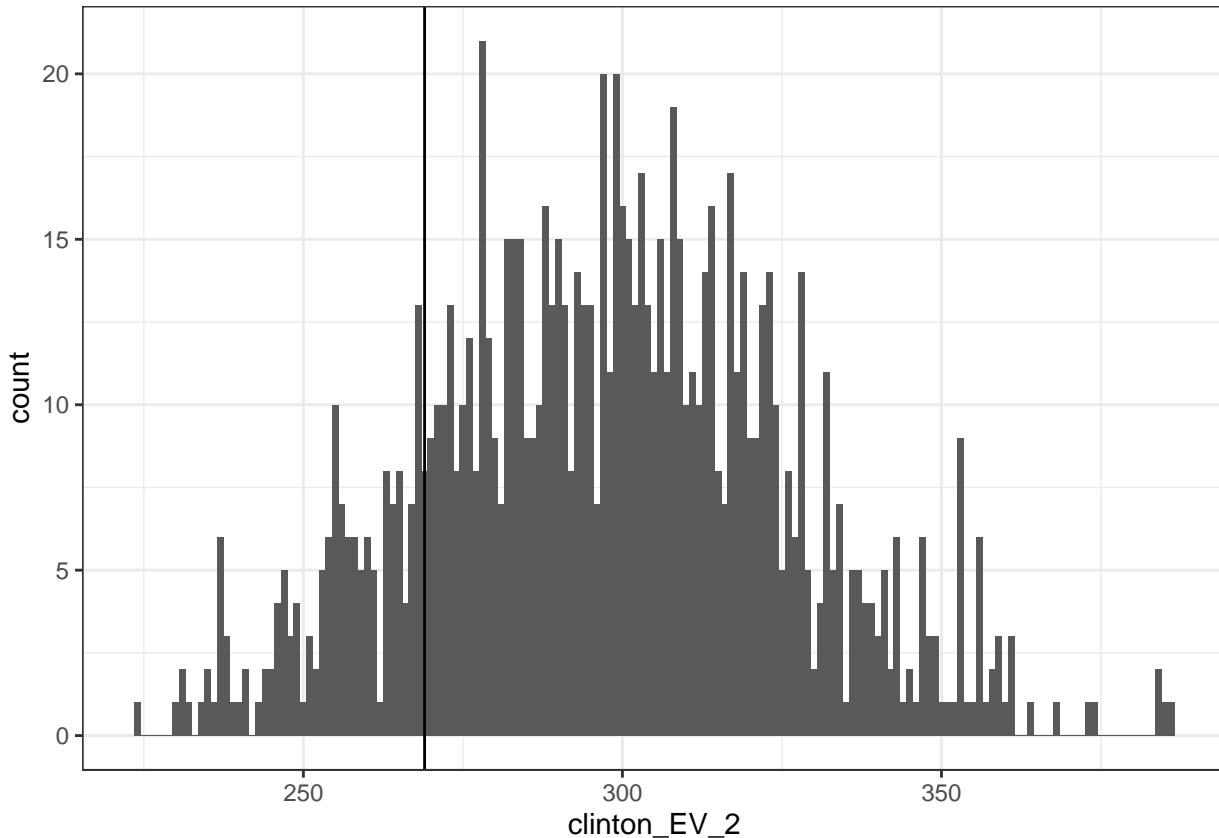
clinton_EV_2 <- replicate(1000, {
  results %>% mutate(sigma = sqrt(sd^2/(n) + bias_sd^2),      # Término agregado del sesgo
                        B = sigma^2/ (sigma^2 + tau^2),
                        posterior_mean = B*mu + (1-B)*avg,
                        posterior_se = sqrt( 1 / (1/sigma^2 + 1/tau^2)),
                        simulated_result = rnorm(length(posterior_mean), posterior_mean, posterior_se),
                        clinton = ifelse(simulated_result > 0, electoral_votes, 0)) %>%
  summarize(clinton = sum(clinton)) %>%
  .$clinton + 7
})

mean(clinton_EV_2 > 269)

## [1] 0.839

data.frame(clinton_EV_2) %>%
  ggplot(aes(clinton_EV_2)) +
  geom_histogram(binwidth = 1) +
  geom_vline(xintercept = 269)

```



#### 4.6.3. Pronósticos

¿Qué tan informativas son las encuestas tomadas semanas antes de la elección? Analizaremos la variabilidad de los resultados de las encuestas a través del tiempo.

```

one_pollster <- polls_us_election_2016 %>%
  filter(pollster == "Ipsos" & state == "U.S.") %>%

```

```
mutate(spread = rawpoll_clinton/100 - rawpoll_trump/100)
```

Dado que se asume que no existe un sesgo de la encuestadora, podría suponerse que el error estándar teórico será similar a la desviación estándar de la muestra:

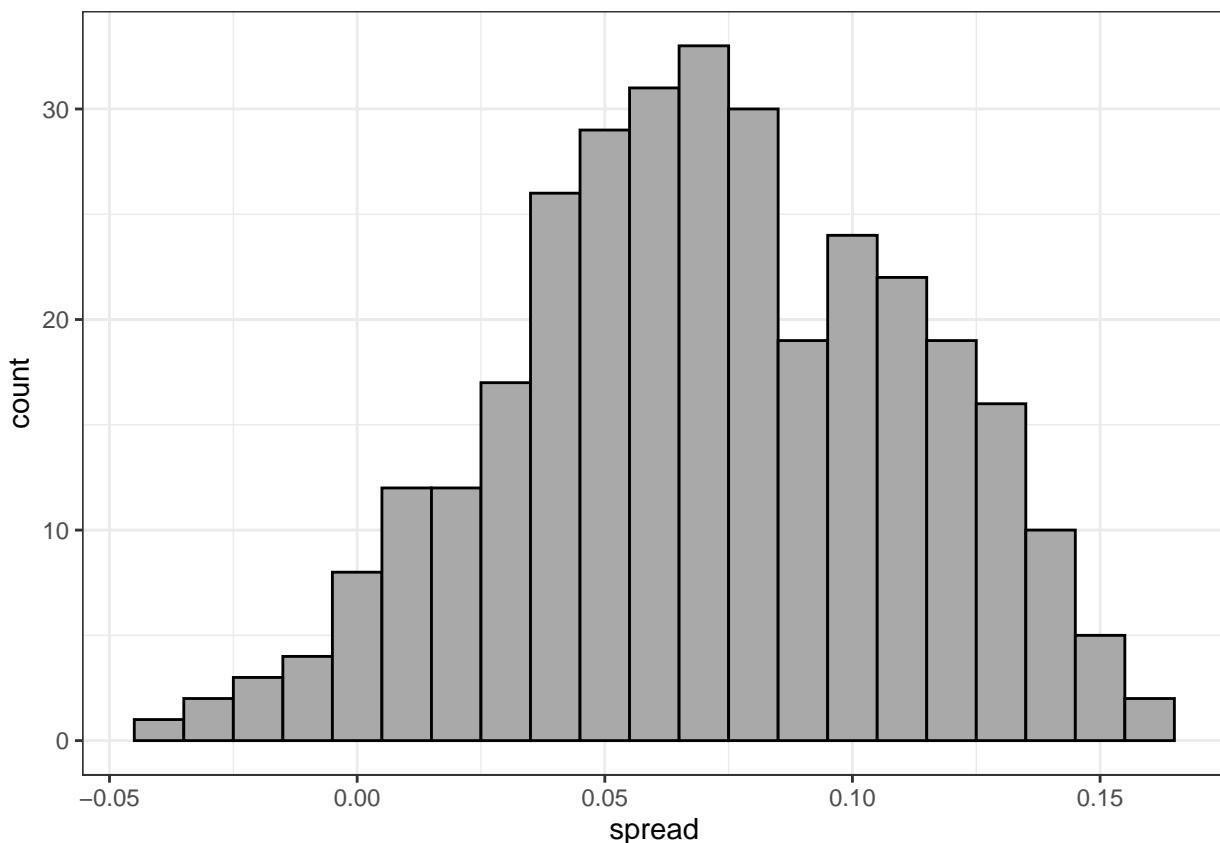
```
se <- one_pollster %>%
  summarize(empirical = sd(spread),
           theoretical = 2*sqrt(mean(spread)*(1-mean(spread))/min(samplesize)))

se

##   empirical theoretical
## 1      0.0403       0.0326
```

Puede verse que el error estándar empírico es ligeramente superior al teórico. Adicionalmente, la distribución de los datos no parece normal como la teoría predaría:

```
one_pollster %>%
  ggplot(aes(spread)) +
  geom_histogram(binwidth = 0.01, color = "black", fill = "darkgray")
```



Con el siguiente código, se muestra la tendencia a través del tiempo para varias encuestadoras

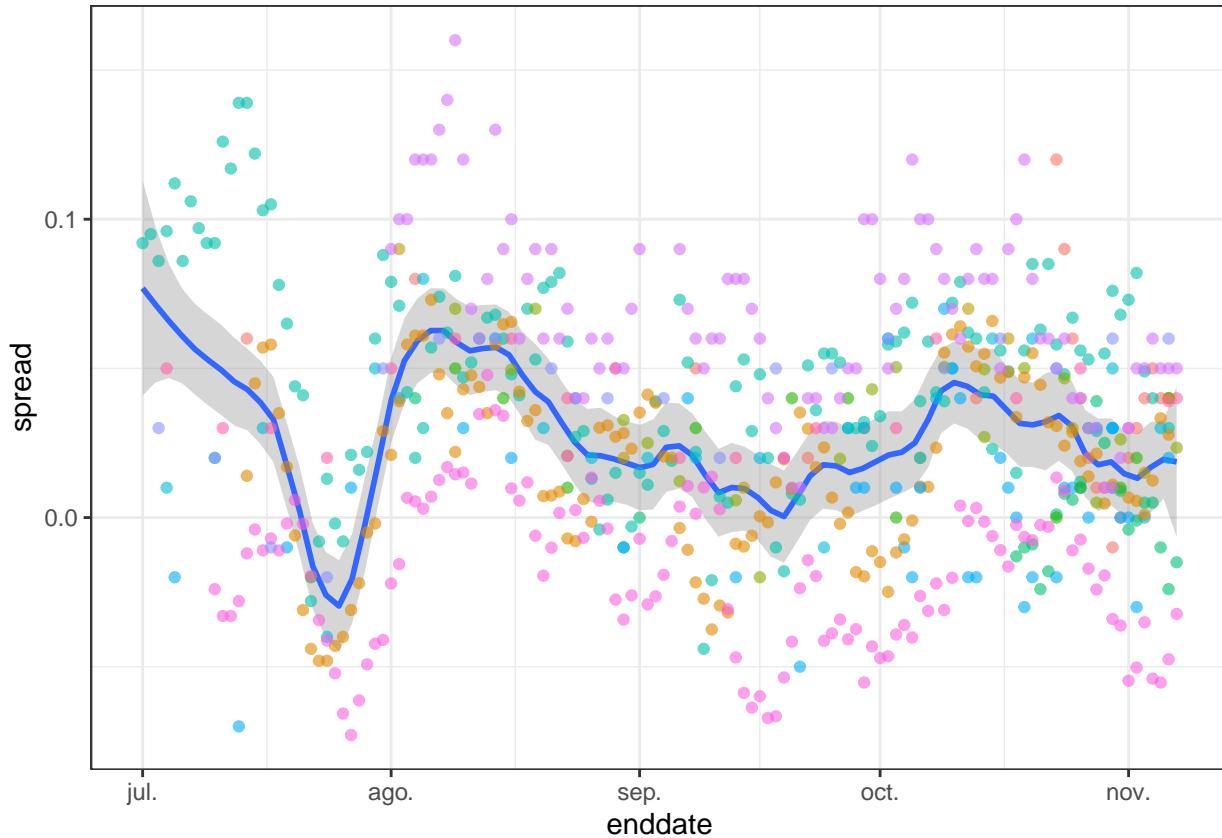
```
polls_us_selection_2016 %>%
  filter(state == "U.S." & enddate >= "2016-07-01") %>%
  group_by(pollster) %>%
  filter(n() >= 10) %>%
  ungroup() %>%
```

```

mutate(spread=rawpoll_clinton/100 - rawpoll_trump/100) %>%
ggplot(aes(enddate, spread)) +
geom_smooth(method = "loess", span = 0.1) +
geom_point(aes(color = pollster), show.legend = FALSE, alpha = 0.6)

## `geom_smooth()` using formula 'y ~ x'

```



Es claro que existen varios picos y valles, los cuales seguramente están asociados a convenciones partidistas o eventos específicos que fortalecieron o dañaron la imagen de Clinton. Esto implica que, si se quiere hacer un pronóstico, el modelo debe incluir un término que tome en cuenta el efecto del tiempo,  $b_t$ :

$$Y_{ijt} = d + b + h_j + b_t + \varepsilon_{ijt}$$

Donde la desviación estándar de  $b_t$  dependerá del tiempo, dado que , mientras más cercano estemos al día de la elección, más baja tendría que ser esta variabilidad.

Adicionalmente, los pronosticadores suelen tratar de estimar tendencias mediante una función  $f(t)$  y a partir de los datos observados. En la gráfica anterior, la línea azul representa estas estimaciones; así, el modelo final se define:

$$Y_{ijt} = d + b + h_j + b_t + f(t) + \varepsilon_{ijt}$$

```

polls_us_election_2016 %>%
  filter(state == "U.S." & enddate >= "2016-07-01") %>%
  select(enddate, pollster, rawpoll_clinton, rawpoll_trump) %>%
  rename(Clinton = rawpoll_clinton, Trump = rawpoll_trump) %>%

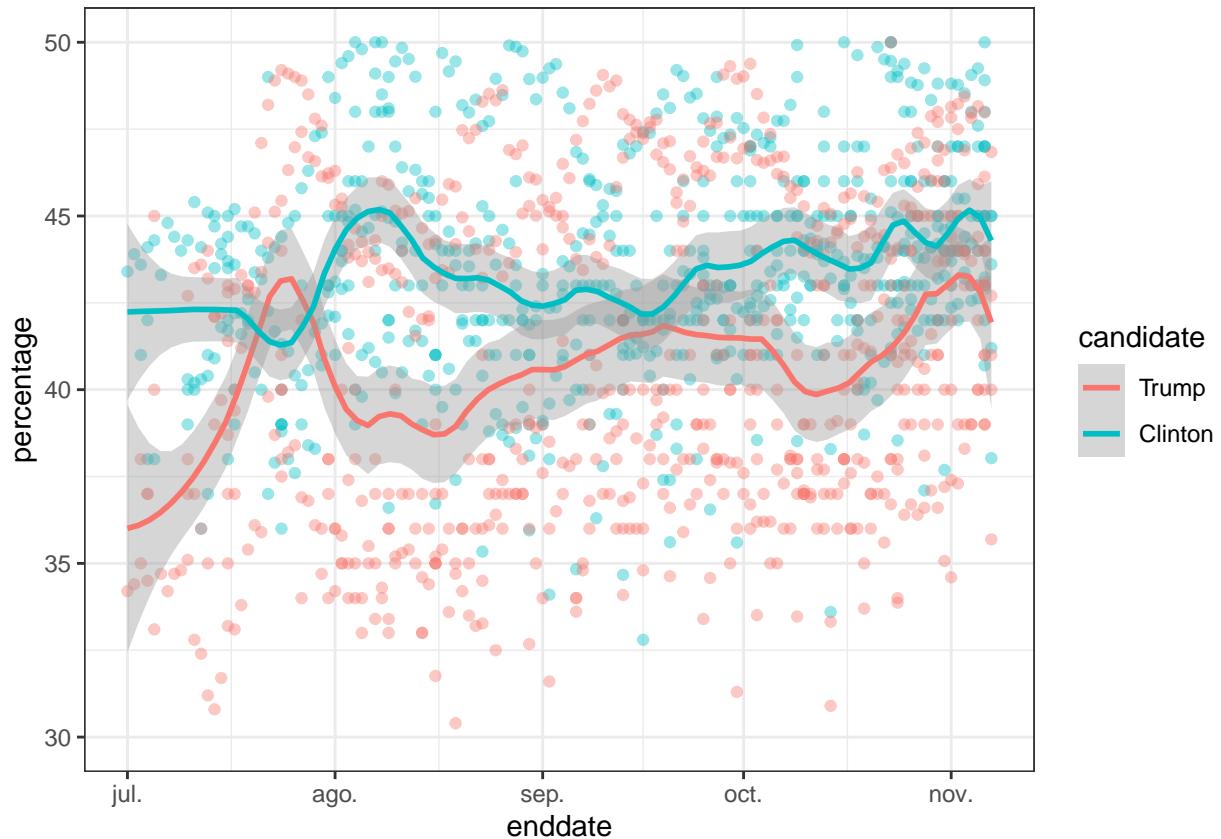
```

```

gather(candidate, percentage, -enddate, -pollster) %>%
  mutate(candidate = factor(candidate, levels = c("Trump", "Clinton"))) %>%
  group_by(pollster) %>%
  filter(n() >= 10) %>%
  ungroup() %>%
  ggplot(aes(enddate, percentage, color = candidate)) +
  geom_point(show.legend = FALSE, alpha = 0.4) +
  geom_smooth(method = "loess", span = 0.15) +
  scale_y_continuous(limits = c(30, 50))

## `geom_smooth()` using formula 'y ~ x'
## Warning: Removed 22 rows containing non-finite values (stat_smooth).
## Warning: Removed 22 rows containing missing values (geom_point).

```



#### 4.6.4. La distribución t

En los modelos donde debemos estimar dos parámetros,  $p$  y  $\sigma$ , el TLC puede resultar demasiado optimista al arrojar intervalos de confianza para muestras menores a 30. Si los datos poblacionales siguen una distribución normal, la teoría nos dice qué tan grande deben ser los intervalos de confianza para la estimación de  $\sigma$ .

Dada  $s$  como estimador de  $\sigma$ , entonces  $Z = \frac{\bar{X} - d}{s/\sqrt{n}} \sim t_{n-1}$ . Aquí, los grados de libertad determinan el peso de las colas en la distribución; valores pequeños indican una mayor probabilidad de valores extremos. Pueden determinarse los intervalos de confianza utilizando la distribución t en lugar de una normal al calcular el cuantil deseado mediante la función “qt()”:

```
z <- qt(0.975, nrow(one_poll_per_pollster) - 1)
one_poll_per_pollster %>%
  summarize(avg = mean(spread), moe = z*sd(spread)/sqrt(length(spread))) %>%
  mutate(start = avg - moe, end = avg + moe)

## # A tibble: 1 x 4
##       avg     moe   start   end
##   <dbl>  <dbl>  <dbl>  <dbl>
## 1 0.0290 0.0134 0.0156 0.0424
qt(0.975, 14)

## [1] 2.14
qnorm(0.975)

## [1] 1.96
```

## 4.7. Pruebas de asociación

Las pruebas estadísticas que se han revisado hasta ahora han dejado afuera a una gran parte de tipos de datos; específicamente, no se han cubierto a los datos binarios, categóricos ni ordinales.

Considérese el siguiente caso de estudio: un estudio de 2014 analiza las tasas de éxito de agencias de financiación en Países Bajos donde se concluye que existe un sesgo de género que favorece a los aplicantes masculinos sobre los femeninos:

```
library(dslabs)
```

```
library(tidyverse)
```

```
data("research_funding_rates")
```

```
research_funding_rates
```

```
##          discipline applications_total applications_men applications_women
## 1   Chemical sciences            122           83             39
## 2   Physical sciences           174          135             39
## 3       Physics                76           67              9
## 4      Humanities              396          230            166
## 5  Technical sciences           251          189             62
## 6 Interdisciplinary            183          105             78
## 7 Earth/life sciences          282          156            126
## 8   Social sciences            834          425            409
## 9  Medical sciences            505          245             260
##    awards_total awards_men awards_women success_rates_total success_rates_men
## 1         32      22        10        26.2          26.5
## 2         35      26        9        20.1          19.3
## 3         20      18        2        26.3          26.9
## 4         65      33        32       16.4          14.3
## 5         43      30        13       17.1          15.9
## 6         29      12        17       15.8          11.4
## 7         56      38        18       19.9          24.4
## 8        112      65        47       13.4          15.3
## 9         75      46        29       14.9          18.8
##    success_rates_women
## 1           25.6
## 2           23.1
## 3           22.2
## 4           19.3
## 5           21.0
## 6           21.8
## 7           14.3
## 8           11.5
## 9           11.2
```

Pueden calcularse los porcentajes de éxito para hombres y mujeres: primeramente, calcúlese el total de éxitos y de fracasos mediante el siguiente código:

```
totals <- research_funding_rates %>%
  select(-discipline) %>%
  summarize_all(funs(sum)) %>%
  summarize(yes_men = awards_men,
            no_men = applications_men - awards_men,
```

```

yes_women = awards_women,
no_women = applications_women - awards_women)

percentage <- totals %>%
  summarize(percent_men = yes_men/(yes_men+no_men),
            percent_women = yes_women/(yes_women+no_women))

percentage

##   percent_men percent_women
## 1      0.177      0.149

```

Donde se observa una tasa de aceptación mayor para los hombres. ¿Puede esto deberse a una variabilidad aleatoria? A continuación se realizará inferencia estadística para este tipo de datos.

El **Test Exacto de Fisher** determina el p-value como la probabilidad de observar un resultado extremo o más extremo que el resultado observado bajo la hipótesis nula. Los datos de un experimento binario son usualmente resumidos en tablas 2x2. Así, el p-value puede ser calculado a través de esta tabla utilizando el test exacto de Fisher con la función “fisher.test()”.

#### 4.7.1. Test Chi-Cuadrada

Dado que con el ejemplo de la agencia de financiación no es posible utilizar la tabla 2x2 del test exacto de Fisher. Sin embargo, existe otro enfoque muy similar, el test Chi-Cuadrada.

Supóngase que tenemos 2 823 individuos donde algunos son hombres, algunas mujeres, algunos sí consiguen financiamiento y otros no: existen dos variables binarias. Ya calculamos la tasa de éxito por género, para calcular la general:

```

funding_rate <- totals %>%
  summarize(percent_total =
            (yes_men+yes_women)/(yes_men+no_men+yes_women+no_women)) %>%
  .$percent_total

funding_rate

## [1] 0.165

```

¿Se encontrará una diferencia entre hombres y mujeres igual a la observada si se asignan los financiamientos al azar, utilizando esta tasa de éxito general? El test Chi-Cuadrada responde a esta pregunta. El primer paso es crear una tabla 2x2 de la siguiente manera:

```

two_by_two <- tibble(awarded = c("no", "yes"),
                      men = c(totals$no_men, totals$yes_men),
                      women = c(totals$no_women, totals$yes_women))

two_by_two

## # A tibble: 2 x 3
##   awarded   men women
##   <chr>     <dbl> <dbl>
## 1 no        1345  1011
## 2 yes       290   177

```

La idea de este test es comparar la tabla observada que se definió arriba con lo que se esperaría ver a partir de la tasa de financiamiento general, la cual se obtiene mediante el siguiente código:

```

tibble(awarded = c("no", "yes"),
       men = (totals$no_men+totals$yes_men)*

```

```

c(1-funding_rate, funding_rate),
women = (totals$no_women+totals$yes_women)*
c(1-funding_rate, funding_rate)

## # A tibble: 2 x 3
##   awarded men women
##   <chr>    <dbl> <dbl>
## 1 no        1365.  991.
## 2 yes       270.   197.

```

Así, el test nos indicará qué tan probable es observar una desviación así de grande a través de la función “chisq.test()”:

```

two_by_two %>%
  select(-awarded) %>%
  chisq.test()

##
## Pearson's Chi-squared test with Yates' continuity correction
##
## data: .
## X-squared = 4, df = 1, p-value = 0.05

```

Obsérvese que el p-value es 0.051, que indica que la probabilidad de observar los resultados que efectivamente ocurrieron es de alrededor de 5 %.

Por otro lado, una estadística informativa pertinente para este ejemplo es el **ratio de posibilidades**. Defínase mujer  $X = 0$  y hombre  $X = 1$ ; con financiamiento  $Y = 1$  y sin financiamiento  $Y = 0$ .

Las posibilidades de ser financiado siendo hombre están definidas como  $\frac{Pr(Y=1|X=1)}{Pr(Y=0|X=1)}$ :

```

odds_men <- (two_by_two$men[2]/sum(two_by_two$men))/
(two_by_two$men[1]/sum(two_by_two$men))

```

```
odds_men
```

```
## [1] 0.216
```

Mientras que las posibilidades de ser financiada siendo mujer son  $\frac{Pr(Y=1|X=0)}{Pr(Y=0|X=0)}$ :

```

odds_women <- (two_by_two$women[2] / sum(two_by_two$women)) /
(two_by_two$women[1] / sum(two_by_two$women))

```

```
odds_women
```

```
## [1] 0.175
```

El ratio de posibilidades es el ratio entre los dos valores calculados: ¿qué tan mayores son las posibilidades de los hombres de obtener financiamiento comparadas con las de las mujeres?

```
odds_men/odds_women
```

```
## [1] 1.23
```

Nótese que si multiplicamos las observaciones por 10, el p-value disminuye bastante, mientras que el ratio de posibilidades continúa siendo el mismo:

```

two_by_two %>%
  select(-awarded) %>%
  mutate(men = men*10, women = women*10) %>%
  chisq.test()

```

```
##  
## Pearson's Chi-squared test with Yates' continuity correction  
##  
## data: .  
## X-squared = 40, df = 1, p-value = 3e-10
```

## 4.8. Comprehensive Assessment

In June 2016, the United Kingdom (UK) held a referendum to determine whether the country would “Remain” in the European Union (EU) or “Leave” the EU. This referendum is commonly known as Brexit. Although the media and others interpreted poll results as forecasting “Remain” ( $p > 0,5$ ), the actual proportion that voted “Remain” was only 48.1% ( $p = 0,481$ ) and the UK thus voted to leave the EU. Pollsters in the UK were criticized for overestimating support for “Remain”.

```
library(dslabs)
library(tidyverse)
library(ggplot2)
options(digits = 3)
data("brexit_polls")
```

Define  $p = 0,481$  as the actual percent voting “Remain” on the Brexit referendum and  $d = 2p - 1 = -0,038$  as the actual spread of the Brexit referendum with “Remain” defined as the positive outcome:

```
p <- 0.481
d <- 2*p-1
```

### Pregunta 1.

The final proportion of voters choosing “Remain” was  $p = 0,481$ . Consider a poll with a sample of  $N = 1500$  voters. What is the expected total number of voters in the sample choosing “Remain”?

```
N <- 1500
expected <- N*p
expected
```

```
## [1] 722
```

What is the standard error of the total number of voters in the sample choosing “Remain”?

```
se_remain <- sqrt(N*p*(1-p))
```

```
se_remain
```

```
## [1] 19.4
```

What is the expected value of  $\hat{X}$ , the proportion of “Remain” voters?

```
x_hat <- p
x_hat
```

```
## [1] 0.481
```

What is the standard error of  $\hat{X}$ , the proportion of “Remain” voters?

```
se_x_hat <- sqrt(x_hat*(1-x_hat)/N)
se_x_hat
```

```
## [1] 0.0129
```

What is the expected value of  $d$ , the spread between the proportion of “Remain” voters and “Leave” voters?

```
d <- 2*p-1
```

```
d
```

```
## [1] -0.038
```

What is the standard error of  $d$ , the spread between the proportion of “Remain” voters and “Leave” voters?

```
se_d <- 2*se_x_hat
```

```
se_d
```

```
## [1] 0.0258
```

Pregunta 2.

Load and inspect the brexit\_polls dataset from dslabs, which contains actual polling data for the 6 months before the Brexit vote. Raw proportions of voters preferring “Remain”, “Leave”, and “Undecided” are available (remain, leave, undecided). The spread is also available (spread), which is the difference in the raw proportion of voters choosing “Remain” and the raw proportion choosing “Leave”.

Calculate  $x_{\text{hat}}$  for each poll, the estimate of the proportion of voters choosing “Remain” on the referendum day ( $p = 0.481$ ), given the observed spread and the relationship  $\hat{d} = 2\hat{X} - 1$ . Use mutate() to add a variable  $x_{\text{hat}}$  to the brexit\_polls object by filling in the skeleton code below:

```
brexit_polls <- brexit_polls %>%
  mutate(x_hat = (spread+1)/2)
```

What is the average of the observed spreads (spread)?

```
mean(brexit_polls$spread)
```

```
## [1] 0.0201
```

What is the standard deviation of the observed spreads?

```
sd(brexit_polls$spread)
```

```
## [1] 0.0588
```

What is the average of  $x_{\text{hat}}$ , the estimates of the parameter ?

```
mean(brexit_polls$x_hat)
```

```
## [1] 0.51
```

What is the standard deviation of  $x_{\text{hat}}$ ?

```
sd(brexit_polls$x_hat)
```

```
## [1] 0.0294
```

Pregunta 3.

Consider the first poll in brexit\_polls, a YouGov poll run on the same day as the Brexit referendum:

```
yougov <- brexit_polls[1,]
```

Use `qnorm()` to compute the 95 % confidence interval for  $\hat{X}$ . What are the lower and upper bounds of the 95 % confidence interval?

```
x_hat <- 0.52
N <- 4722
se <- sqrt(x_hat*(1-x_hat)/N)
ci <- c(x_hat - qnorm(0.975)*se, x_hat+qnorm(0.975)*se)
ci
## [1] 0.506 0.534
```

Pregunta 4.

Create the data frame `june_polls` containing only Brexit polls ending in June 2016 (enddate of “2016-06-01” and later). We will calculate confidence intervals for all polls and determine how many cover the true value of  $d$ .

First, use `mutate()` to calculate a plug-in estimate `se_x_hat` for the standard error of the estimate  $\hat{SE}[X]$  for each poll given its sample size and value of  $\hat{X}$  (`x_hat`). Second, use `mutate()` to calculate an estimate for the standard error of the spread for each poll given the value of `se_x_hat`. Then, use `mutate()` to calculate upper and lower bounds for 95 % confidence intervals of the spread. Last, add a column `hit` that indicates whether the confidence interval for each poll covers the correct spread  $d = -0.038$ .

How many polls are in `june_polls`?

```
june_polls <- brexit_polls %>%
  filter(enddate >= "2016-06-1")
nrow(june_polls)
```

```
## [1] 32
```

What proportion of polls have a confidence interval that covers the value 0?

```
d <- -0.038
june_polls <- june_polls %>%
  mutate(se_x_hat = sqrt(x_hat*(1-x_hat)/samplesize), se_spread = 2*se_x_hat,
        lower = spread - qnorm(0.975)*se_spread,
        upper = spread + qnorm(0.975)*se_spread,
        hit = (lower < 2*p-1 & upper > 2*p-1))
mean(june_polls$lower < 0 & june_polls$upper > 0)
```

```
## [1] 0.625
```

What proportion of polls predict “Remain” (confidence interval entirely above 0)?

```
mean(june_polls$lower > 0 & june_polls$upper > 0)
```

```
## [1] 0.125
```

What proportion of polls have a confidence interval covering the true value of ?

```
mean(june_polls$hit)
```

```
## [1] 0.562
```

**Pregunta 5.**

Group and summarize the june\_polls object by pollster to find the proportion of hits for each pollster and the number of polls per pollster. Use arrange() to sort by hit rate.

```
june_polls_grouped <- june_polls %>%
  group_by(pollster) %>%
  summarize(hits=n()) %>%
  arrange(desc(hits))
```

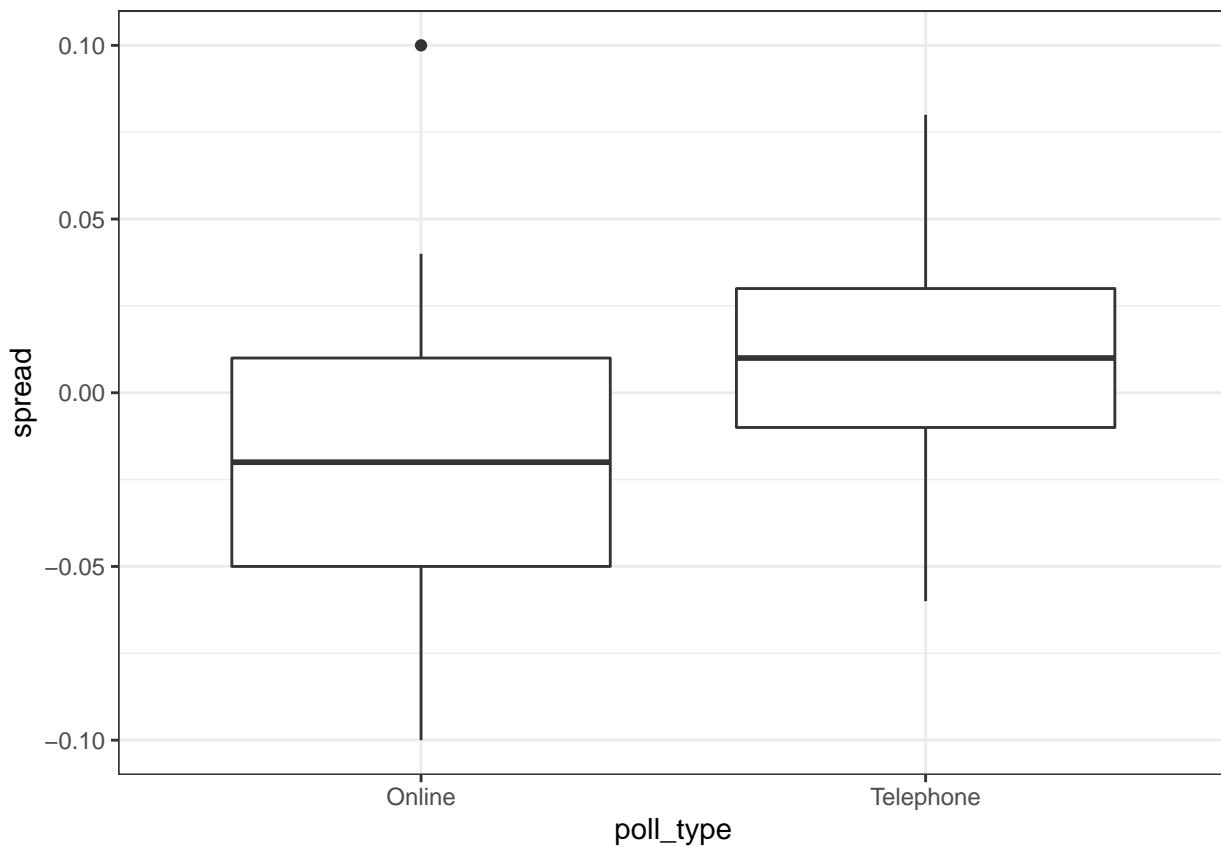
```
head(june_polls_grouped)
```

```
## # A tibble: 6 x 2
##   pollster     hits
##   <fct>       <int>
## 1 YouGov         9
## 2 ComRes         3
## 3 ICM            3
## 4 Opinium        3
## 5 ORB            3
## 6 BMG Research   2
```

**Pregunta 6.**

Make a boxplot of the spread in june\_polls by poll type.

```
june_polls %>%
  ggplot(aes(poll_type, spread)) +
  geom_boxplot()
```

**Pregunta 7.**

Calculate the confidence intervals of the spread combined across all polls in june\_polls, grouping by poll type. Recall that to determine the standard error of the spread, you will need to double the standard error of the estimate.

Use this code (which determines the total sample size per poll type, gives each spread estimate a weight based on the poll's sample size, and adds an estimate of p from the combined spread) to begin your analysis:

```
combined_by_type <- june_polls %>%
  group_by(poll_type) %>%
  summarize(N = sum(samplesize),
            spread = sum(spread*samplesize)/N,
            p_hat = (spread + 1)/2,
            se_spread = 2*sqrt(p_hat*(1-p_hat)/N),
            lower = spread - qnorm(0.975) * se_spread,
            upper = spread + qnorm(0.975)*se_spread)

combined_by_type
```

poll_type	N	spread	p_hat	se_spread	lower	upper
Online	46711	-0.00741	0.496	0.00463	-0.0165	0.00165
Telephone	13490	0.0127	0.506	0.00861	-0.00414	0.0296

**Pregunta 8.**

Define `brexit_hit`, with the following code, which computes the confidence intervals for all Brexit polls in 2016 and then calculates whether the confidence interval covers the actual value of the spread  $d = -0.038$ :

```
brexit_hit <- brexit_polls %>%
  mutate(p_hat = (spread + 1)/2,
        se_spread = 2*sqrt(p_hat*(1-p_hat)/samplesize),
        spread_lower = spread - qnorm(.975)*se_spread,
        spread_upper = spread + qnorm(.975)*se_spread,
        hit = spread_lower < -0.038 & spread_upper > -0.038) %>%
  select(poll_type, hit)
```

se `brexit_hit` to make a two-by-two table of poll type and hit status. Then use the `chisq.test()` function to perform a chi-squared test to determine whether the difference in hit rate is significant. What is the p-value of the chi-squared test comparing the hit rate of online and telephone polls?

```
totals <- brexit_hit %>%
  summarize(online_yes = sum(hit == "TRUE" & poll_type == "Online"),
            online_no = sum(hit == "FALSE" & poll_type == "Online"),
            telephone_yes = sum(hit == "TRUE" & poll_type == "Telephone"),
            telephone_no = sum(hit == "FALSE" & poll_type == "Telephone"))
totals

##   online_yes online_no telephone_yes telephone_no
## 1       48      37       10      32
two_by_two <- tibble(hit = c('yes', 'no'),
                      online = c(totals$online_yes, totals$online_no),
                      telephone = c(totals$telephone_yes, totals$telephone_no))
two_by_two
```

```
## # A tibble: 2 x 3
##   hit   online telephone
##   <chr> <int>     <int>
## 1 yes      48       10
## 2 no       37       32
chisq_test <- two_by_two %>%
  select(-hit) %>%
  chisq.test()
```

`chisq_test`

```
##
## Pearson's Chi-squared test with Yates' continuity correction
##
## data: .
## X-squared = 11, df = 1, p-value = 0.001
```

#### Pregunta 9.

Use the two-by-two table constructed in the previous exercise to calculate the odds ratio between the hit rate of online and telephone polls to determine the magnitude of the difference in performance between the poll types. Calculate the odds that an online poll generates a confidence interval that covers the actual value of the spread.

```
online_yes_odds <- (two_by_two$online[1] / sum(two_by_two$online)) / (two_by_two$online[2] / sum(two_by_two$online))
```

```
## [1] 1.3
```

Calculate the odds that a telephone poll generates a confidence interval that covers the actual value of the spread.

```
telephone_yes_odds <- (two_by_two$telephone[1] / sum(two_by_two$telephone)) / (two_by_two$telephone[2] / sum(two_by_two$telephone))
```

```
## [1] 0.312
```

Calculate the odds ratio to determine how many times larger the odds are for online polls to hit versus telephone polls.

```
online_yes_odds / telephone_yes_odds
```

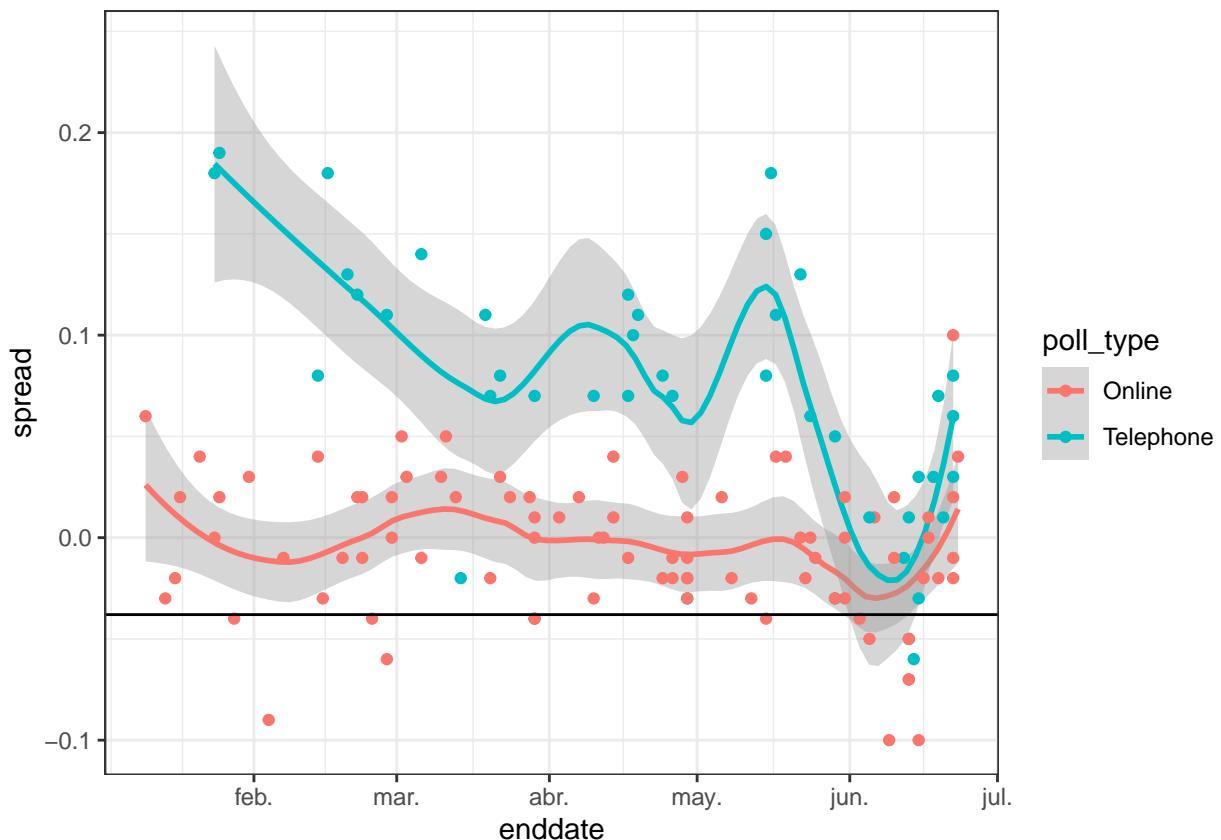
```
## [1] 4.15
```

Pregunta 10.

Use brexit\_polls to make a plot of the spread (spread) over time (enddate) colored by poll type (poll\_type). Use geom\_smooth() with method = “loess” to plot smooth curves with a span of 0.4. Include the individual data points colored by poll type. Add a horizontal line indicating the final value of  $d = -0.038$ .

```
brexit_polls %>%
  ggplot(aes(enddate, spread, color = poll_type)) +
  geom_smooth(method = "loess", span = 0.4) +
  geom_point() +
  geom_hline(yintercept = -0.038)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



### Pregunta 11.

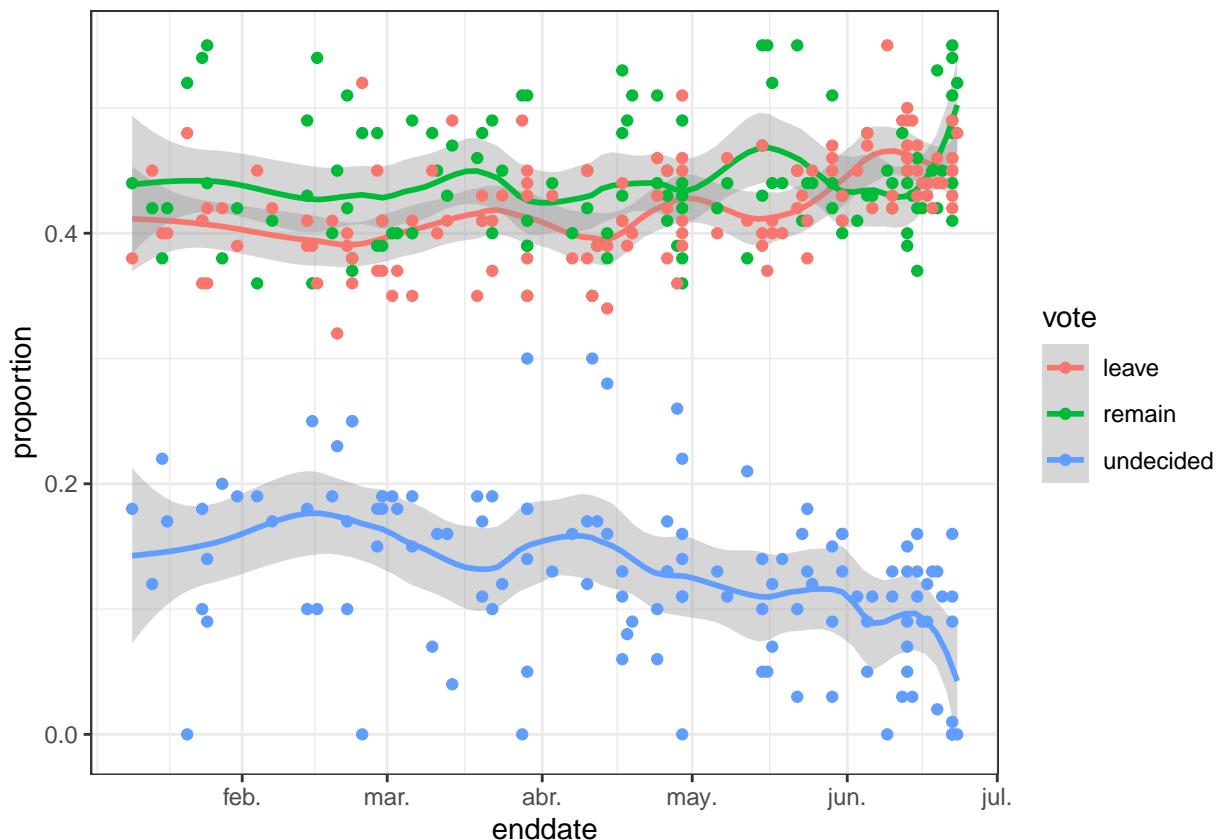
Use the following code to create the object brexit\_long, which has a column vote containing the three possible votes on a Brexit poll (“remain”, “leave”, “undecided”) and a column proportion containing the raw proportion choosing that vote option on the given poll:

```
brexit_long <- brexit_polls %>%
  gather(vote, proportion, "remain":"undecided") %>%
  mutate(vote = factor(vote))
```

Make a graph of proportion over time colored by vote. Add a smooth trendline with geom\_smooth() and method = “loess” with a span of 0.3.

```
brexit_long %>%
  ggplot(aes(enddate, proportion, color = vote)) +
  geom_smooth(method = "loess", span = 0.3) +
  geom_point()

## `geom_smooth()` using formula 'y ~ x'
```



## 5. Herramientas de productividad

Existen tres principios guías para la productividad en la ciencia de datos:

1. Sistematización al organizar un sistema de archivos;
  - Se busca minimizar el tiempo que se busca algo
2. Automatización cuando sea posible; y
  - Si se repite una tarea una y otra vez, probablemente exista una manera de automatizarla.
3. Minimización del uso del mouse.
  - Cada vez que la mano se aleja del teclado, se pierde productividad.

### 5.1. Instalando el software

#### 5.1.1. Introducción a R Studio

RStudio ofrece una manera de mantener todos los componentes de un proyecto de análisis de datos organizados en una sola carpeta.

El nombre del proyecto aparecerá en la esquina superior izquierda. Si se inicia una sesión de RStudio sin proyecto, se desplegará “Project: (None)”.

#### 5.1.2. Introducción a Git y GitHub

Una ventaja de los proyectos de RStudio es que pueden compartirse con colaboradores o al público mediante el software Git, que ayuda a llevar un control de los cambios en el código y ayuda a la coordinación de su edición. Por su parte, GitHub es un sistema hospedador de código, a veces utilizado para mostrar públicamente los proyectos personales.

La idea de crear repositorios en GitHub es tener una copia del documento en la computadora y otra en la nube, mediante GitHub. Al agregar colaboradores, cada quien tendrá una copia del proyecto en su computadora. Así, la versión depositada en GitHub es considerada la **copia maestra** a partir de la cual todos se sincronizan.

## 5.2. Unix Básico

### 5.2.1. Introducción a Unix

Unix es el sistema operativo por excelencia en la ciencia de datos. A través de Unix se aprenderá una serie de comandos y un modo de pensar que facilitan la organización de archivos

La Terminal es la ventana para acceder a Unix. Su sintaxis es similar a la de la consola de R, con la diferencia que es utilizada para organizar archivos en nuestro sistema.

Hay que pensar un sistema de archivos como una serie de carpetas anidadas que contienen más carpetas, archivos y ejecutables. En Unix, nos referimos a las carpetas como directorios y subdirectorios.

El *home directory* es el directorio donde se guarda todo. Así, existe una naturaleza jerárquica del sistema de archivos. Por otro lado, el *working directory* es la locación actual.

Cada ventana de terminal tiene un *working directory* asociado a ella. El comando “pwd” desplegará al *working directory*, “/” separa a los directorios. Finalmente, “~” significa el *home directory*.

### 5.2.2. Trabajando con Unix

El comando “ls” permite listar el contenido de un directorio.

El comando “mkdir” permite hacer un nuevo directorio (carpeta) y “rmdir” permite eliminarlos.

Para hacer que un directorio se vuelva nuestro *working directory* utilizamos el comando “cd”. Si queremos regresar a que el *working directory* sea el directorio “de arriba”, utilizamos “cd ..”. Podemos movernos dos niveles “hacia arriba” mediante “cd../..”. Además, “cd ~” nos llevará al *home directory*.

Para ir a un directorio específico, podemos utilizar “cd~/nombre”

Para mover archivos, utilizamos el comando “mv” (existe un *shortcut* para el *working directory*, “.”); para copiarlos, “cp”. Por su parte, “rm” elimina **permanentemente** los directorios o archivos.

Puede verse el contenido de un archivo mediante el comando “less”, que abrirá un visualizador automáticamente, el cual puede cerrarse con la letra q.

### 5.3. Reportes

Usualmente, el producto final de un análisis de ciencia de datos es un reporte. El objetivo de estos es generar reproducibilidad, dado que la característica de RMarkdown es que combina texto y código en el mismo documento, al tiempo que las tablas y figuras se agregan automáticamente al documento.

R Markdown es un formato para documentos que se conoce como programación alfabetizada: esta entrelaza instrucciones, documentación y comentarios detallados con el código ejecutable, produciendo un documento que describe de manera excelente el análisis y resultados.

A diferencia de Word, R Markdown no es un formato *what you see is what you get*, dado que es necesario **compilar** el documento para producir el reporte final. Esto se logra mediante knitr y los archivos tienen la extensión “.rmd”.

Un documento de R Markdown comienza con un YAML header donde se puede personalizar los datos del documento, el índice y otras funciones.

Para conocer más sobre la sintaxis y funcionamiento de R Markdown, visítese este tutorial

Para insertar un *chunk* de código, puede usarse el *shortcut* Ctrl+Alt+I. Así, cada *chunk* puede personalizarse para que el reporte final lo muestre o no y lo corra o no. Finalmente, pueden agregarse etiquetas a los *chunks* con el objetivo de hacer más fácil su búsqueda y mejorar la organización del documento.

# *Chunk de código de muestra*

Como se mencionó, el paquete knitr es el que ayuda a compilar el documento mediante la función “knit”. Existen varios formatos de salida: HTML, PDF, Word y .md (este último permite que GitHub pueda compilar el reporte y mostrarlo correctamente en su página.)

## 5.4. Git y GitHub

Hay tres razones principales para utilizar Git y GitHub:

1. Control de versiones, la cual permite llevar un control de cambios que se han hecho al código, restaurar versiones del archivo pasadas, y probar ideas mediante la utilización de *branches*, las cuales pueden eventualmente combinarse con la original.
2. Colaboración. Una vez creado el repositorio central, pueden haber múltiples personas que realicen cambios y mantengan las versiones sincronizadas. Una *pull request* permite a cualquiera la sugerencia de cambios al código.
3. Compartir. GitHub nos permite compartir individual o públicamente nuestro código.

Para clonar un repositorio que se encuentra en la nube, utilizaremos el comando “clone git” en la terminal. En este contexto, clonar se refiere a copiar la estructura completa del Git hasta nuestra computadora.

Algunos comandos pertinentes para trabajar desde la terminal con Git son:

- `git status`: indica cómo los archivos en el directorio de trabajo están relacionados con los archivos en otras etapas.
- `git log`: realiza un seguimiento de todos los cambios que se han hecho en el repositorio local.
- `git push`: permite pasar del repositorio local al repositorio en la nube, solo si se cuenta con el permiso.
- `git fetch`: actualiza el repositorio local para que sea igual al que está en la nube.
- `git merge`: realiza una sincronización local a partir del directorio de trabajo.

Para crear un repositorio de Git local, se usa el comando “`git init`”, con el cual Git comienza a rastrear todo en el repositorio local. Así, para mover archivos a nuestro repositorio local utilizamos “`git remote add origin`”.

## 5.5. Unix avanzado

### 5.5.1. Argumentos

La mayoría de los comandos de Unix pueden ser corridos mediante argumentos, los cuales están típicamente definidos mediante uno o dos guiones seguidos por una palabra.

Un ejemplo es “rm -r” donde la r quiere decir *recursive*; el resultado del comando es que los archivos y directorios son removidos de manera recursiva (equivalente a enviar una carpeta con carpetas y archivos a la papelera).

Para remover archivos protegidos se utiliza el argumento de *force*, “-f”. Así, para remover un directorio sin importar que tenga archivos protegidos, el comando adecuado es “rm -rf nombredeldirectorio”.

Otro comando que suele ser utilizado con otros argumentos es “ls”:

- ls -a: muestra todos los archivos del directorio, incluyendo archivos ocultos.
- ls -l: muestra más información sobre los archivos del directorio.
- ls -t: muestra los archivos en orden cronológico.
- ls -r: revierte el orden de los archivos mostrados.
- ls -lart: muestra más información sobre todos los archivos en orden cronológico revertido.

### 5.5.2. Ayudas y pipas

Como se habrá notado, Unix utiliza una abreviación extrema, lo cual incrementa la eficiencia pero a costa de la dificultad de recordar todos los comandos. Para ayudar con esto, Unix incluye manuales de apoyo, los cuales son llamados mediante el comando “man nombredelcomando” o “nombredelcomando –help”, dependiendo del sistema que se esté utilizando.

Por otra parte, las pipas “|” encadenan los resultados de un comando con otro escrito después de ella, similarmente a la pipa utilizada en R con dplyr “%>%”.

### 5.5.3. Comodines

Supóngase que se quieren remover todos los archivos temporales HTML producidos durante un diagnóstico de error. Con Unix, podemos utilizar escribir lo siguiente para indicar que queremos remover todos los archivos con esa combinación: rm \*.html”.

Otro comodín es “?”, que significa “cualquier carácter”. Por ejemplo, para borrar todos los archivos en la forma file-001.html con una numeración del 1 al 999, escribimos “rm file-???.html”

### 5.5.4. Variables ambientales y *shells*

En Unix, las variables son distinguidas con otras entidades colocando “\$” antes. Por ejemplo, el *home directory* está guardado en “\$HOME”.

Muchos de los comandos utilizados hasta ahora son parte de lo que se conoce como *Unix shell*; la más común es bash. Para cambiar variables ambientales se usa “export PATH = /usr/bin/”.

### 5.5.5. Ejecutables, permisos y tipos de archivos

En Unix, todos los programas son archivos; aquellos que corren algo son llamados ejecutables (ls, mv, git). Para encontrar dónde se encuentran estos archivos de programa, utilizamos el comando “which”.

El comando “ls -l” mostrará una serie de informaciones con el tipo que es cada archivo, donde además se puede observar si el archivo es leíble, editable y ejecutable.

### 5.5.6. Comandos útiles

- start: trata de encontrar la mejor aplicación para abrir ese archivo y lo abre.
- nano: editor de texto básico.
- ln: crea links simbólicos.
- tar: permite crear o extraer archivos o subdirectorios en un solo archivo.
- ssh: permite conectarse con otras computadoras.
- grep: busca por patrones dentro de un archivo.
- awk/sed: encuentra *strings* específicas en un archivo y las modifica.

Como alternativa, se sugiere el uso de GitHub Desktop para evitar el uso de Unix y los comandos que implica.

## 6. Data Wrangling

Las bases de datos utilizadas hasta ahora han estado disponibles mediante la librería “dslabs” y en forma de *data frames*. Además, esta información se ha presentado de forma ordenada; la librería “tidyverse” asume que nuestros datos se encuentran limpios y ordenados.

Así, el proceso mediante el cual se convierten los datos crudos, sucios y brutos en una base de datos limpia, manejable, relevante y ordenada se conoce como *data wrangling*.

### 6.1. Importación de datos

#### 6.1.1. Hojas de cálculo

Como se sabe, una de las maneras más comunes de almacenar datos es mediante hojas de cálculo. Además, un archivo separado por comas (.csv) almacena la información separando las columnas con comas y los renglones con enteros.

Antes de importar un archivo a R, es importante conocerlo y averiguar si tiene encabezado o no. Por default, R asume que sí lo tiene.

#### 6.1.2. Rutas y directorio de trabajo

Es sumamente importante definir el directorio de trabajo, el cual es el lugar donde R buscará y guardará los archivos por default. Para conocerlo, úsese el siguiente comando:

```
getwd()
```

```
## [1] "C:/Users/marco/OneDrive/Documents/Cursos/Professional Certificate in Data Science"
```

Si se quiere cambiar el directorio de trabajo, utilícese:

```
# setwd(C:/Users/marco/OneDrive/Documents/Cursos/Professional Certificate in Data Science)
```

Una cuestión importante sobre las funciones de lectura de archivos es que estas buscarán el nombre del archivo en el directorio de trabajo que se haya indicado a menos que se especifique la ruta exacta del archivo.

Como ejemplo, se usará una base de datos incluida en la librería dslabs:

```
library(dslabs)
```

```
library(tidyverse)
```

Los archivos se localizarán en el directorio de datos externos, “extdata”, el cual puede ser obtenido mediante:

```
system.file("extdata", package = "dslabs")
```

```
## [1] "C:/Users/marco/OneDrive/Documents/R/win-library/4.0/dslabs/extdata"
```

También pueden verse los archivos incluidos en este directorio con la función “list.files()”:

```
path <- system.file("extdata", package = "dslabs")
```

```
list.files(path)
```

```
## [1] "2010_bigfive_regents.xls"
## [2] "carbon_emissions.csv"
## [3] "fertility-two-countries-example.csv"
## [4] "HRlist2.txt"
## [5] "life-expectancy-and-fertility-two-countries-example.csv"
## [6] "murders.csv"
```

```
## [7] "olive.csv"
## [8] "RD-Mortality-Report_2015-18-180531.pdf"
## [9] "ssa-death-probability.csv"
```

Ahora ya conocemos la ubicación de los archivos, por lo que estamos listos para importarlos a R. Para hacer el código más sencillo, puede moverse el archivo que se trabajará a nuestro directorio de trabajo; esto puede hacerse manualmente mediante el explorador de archivos o con la función “file.copy()”. Para esto, será útil definir una variable con la ruta completa con la función “file.path()”:

```
filename <- "murders.csv"

fullpath <- file.path(path, filename)

fullpath

## [1] "C:/Users/marco/OneDrive/Documents/R/win-library/4.0/dslabs/extdata/murders.csv"
file.copy(fullpath,
          "C:/Users/marco/OneDrive/Documents/Cursos/Professional Certificate in Data Science")

## [1] FALSE
```

Para comprobar que el archivo efectivamente se copió en el directorio de trabajo:

```
file.exists(filename)
```

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

### 6.1.3. readr y readxl

readr es la librería de tidyverse que incluye funciones para leer datos contenidos en hojas de cálculo tipo texto. Las siguientes funciones son pertinentes, donde se diferencian por el tipo de delimitador de columnas en cada tipo de archivo:

- read\_table(): archivos .txt
- read\_csv() y read\_csv2(): archivos .csv separados por comas y punto y coma, respectivamente.
- read\_tsv(): archivos .tsv
- read\_delim(): archivos .txt de texto general (hay que definir delimitador).

Por otro lado, la librería readxl provee de funciones para leer datos en formatos Microsoft Excel:

- read\_excel(): autodetección del formato, archivos .xls y .xlsx
- read\_xls(): formato original .xls
- read\_xlsx(): nuevo formato .xlsx

Nótese que un archivo de Excel permite tener varias hojas de cálculo en un solo archivo; la función “excel\_sheets()” arroja los nombres de las hojas en un archivo de Excel.

Normalmente es útil abrir el archivo que queremos importar para conocerlo y asegurarnos de que es la extensión que estamos considerando. Sin embargo, esto puede hacerse a través de R con la función “read\_lines()”, la cual muestra las primeras líneas de un archivo:

```
read_lines(filename, n_max = 3)

## [1] "state,abb,region,population,total" "Alabama,AL,South,4779736,135"
## [3] "Alaska,AK,West,710231,19"
```

Podemos ver que se trata de un archivo .csv y que sí tiene encabezado. Por tanto, es necesario usar la función “read\_csv()”:

```
dat <- read_csv(filename)

##
## -- Column specification -----
## cols(
##   state = col_character(),
##   abb = col_character(),
##   region = col_character(),
##   population = col_double(),
##   total = col_double()
## )
```

Nótese que se recibe un mensaje que especifica sobre el tipo de datos que se asumió para cada columna.

Finalmente, puede verse el objeto recién creado con la siguiente función:

```
head(dat)
```

```
## # A tibble: 6 x 5
##   state     abb   region population total
##   <chr>    <chr> <chr>      <dbl>   <dbl>
## 1 Alabama   AL    South       4779736   135
## 2 Alaska    AK    West        710231    19
## 3 Arizona   AZ    West        6392017   232
## 4 Arkansas  AR    South       2915918    93
## 5 California CA    West       37253956  1257
## 6 Colorado  CO    West        5029196    65
```

#### 6.1.4. Importación de datos con funciones de R base

Como se mencionó, las funciones que se mencionaron para la importación de datos pertenecen a la librería tidyverse. No obstante, es posible realizar algunas de estas acciones con las funciones predeterminadas de R.

- `read.table()`
- `read.csv()`
- `read.delim()`

Es importante mencionar que existe una diferencia entre ambos tipos de funciones:

```
filename <- "murders.csv"

dat2 <- read.csv(filename)

class(dat2)

## [1] "data.frame"
```

A diferencia de la tabla que se crea con “`read_csv()`”, “`read.csv()`” crea un *data frame*.

#### 6.1.5. Archivos del internet

Existen dos opciones: por un lado, descargar los archivos a nuestra computadora y después leerlos mediante alguna de las funciones ya vistas; por otro, leerlos directamente desde el internet.

Por ejemplo, los datos sobre asesinatos se encuentran en una dirección de GitHub asociada a la librería dlab del curso:

```
url <- "https://raw.githubusercontent.com/rafaelab/dslabs/master/inst/extdata/murders.csv"

dat <- read_csv(url)

## 
## -- Column specification -----
## cols(
##   state = col_character(),
##   abb = col_character(),
##   region = col_character(),
##   population = col_double(),
##   total = col_double()
## )
```

Si se quiere tener una copia local del archivo, puede descargarse mediante la función (el archivo se guardará en el directorio de trabajo):

```
download.file(url, "murders.csv")
```

Dos funciones útiles al descargar archivos desde el internet son “tempdir()” y “tempfile()”. El primero crea un directorio con un nombre que es muy probable que no sea único. Similarmente, la segunda crea una cadena de caracteres, no un archivo, con un nombre probablemente único.

```
tempfile()

## [1] "C:\\\\Users\\\\marco\\\\AppData\\\\Local\\\\Temp\\\\Rtmp0ursnv\\\\file28a44bb16c72"

tempdir()

## [1] "C:\\\\Users\\\\marco\\\\AppData\\\\Local\\\\Temp\\\\Rtmp0ursnv"

Así, y como ejemplo, se descarga un archivo, se le da un nombre temporal, se importa y después se elimina

tmp_filename <- tempfile()

download.file(url, tmp_filename)

dat <- read_csv(tmp_filename)

## 
## -- Column specification -----
## cols(
##   state = col_character(),
##   abb = col_character(),
##   region = col_character(),
##   population = col_double(),
##   total = col_double()
## )
```

```
file.remove(tmp_filename)

## [1] TRUE
```

### 6.1.6. Assessment 9

In this part of the assessment, you will import real datasets and learn more about useful arguments to `readr` functions. You will encounter common issues that arise when importing raw data. This part of the assessment will require you to program in R. Use the `readr` package in the `tidyverse` library:

```
library(tidyverse)
```

**Inspect the file at the following URL:**

<https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wdbc.data>

\*Does this file have a header row? Does the `readr` function you chose need any additional arguments to import the data correctly?\*\*

```
url <- "https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wdbc.data"

data <- read_csv(url, col_names = FALSE)

## 
## -- Column specification -----
## cols(
##   .default = col_double(),
##   X2 = col_character()
## )
## i Use `spec()` for the full column specifications.
```

**How many rows are in the dataset?**

```
nrow(data)
```

```
## [1] 569
```

**How many columns are in the dataset?**

```
ncol(data)
```

```
## [1] 32
```

## 6.2. Ordenación de datos

### 6.2.1. Reorganización de datos

Para dar un ejemplo de la reorganización de datos, retómese el ejemplo de la tasa de fertilidad en Corea del Sur y Alemania:

```
library(tidyverse)
library(dslabs)
data("gapminder")

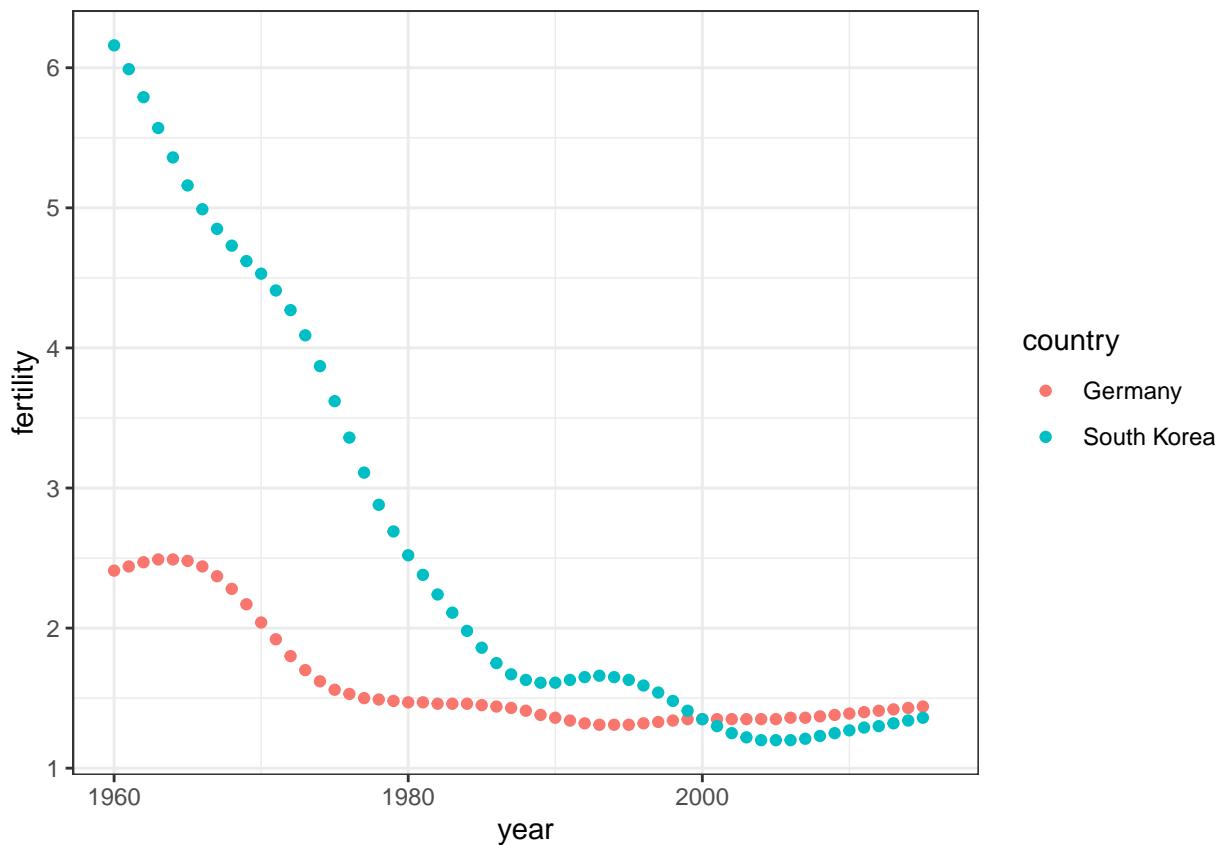
tidy_data <- gapminder %>%
  filter(country %in% c("South Korea", "Germany")) %>%
  select(country, year, fertility)

head(tidy_data)

##          country year fertility
## 1      Germany 1960     2.41
## 2 South Korea 1960     6.16
## 3      Germany 1961     2.44
## 4 South Korea 1961     5.99
## 5      Germany 1962     2.47
## 6 South Korea 1962     5.79

tidy_data %>%
  ggplot(aes(year, fertility, color = country)) +
  geom_point()

## Warning: Removed 2 rows containing missing values (geom_point).
```



Nótese que la utilización de los comandos de dplyr “filter” y “select” permitió conformar un *data frame* adecuado, limpio e intuitivo para la posterior construcción del gráfico (puede verse que cada punto del gráfico está representado en un renglón de “tidy\_data”).

Así, podemos definir a la **tidy data** de la siguiente manera: cada renglón representa una observación y cada columna representa las diferentes variables de las cuales se tienen datos para cada una de esas observaciones.

De hecho, los datos originales de Gapminder no cumplen estos requisitos. Además, el formato de los datos puede ser **amplio**:

```
path <- system.file("extdata", package = "dslabs")
filename <- file.path(path, "fertility-two-countries-example.csv")
wide_data <- read_csv(filename)
```

```
##
## -- Column specification -----
## cols(
##   .default = col_double(),
##   country = col_character()
## )
## i Use `spec()` for the full column specifications.
```

Es evidente que en el formato amplio, cada renglón incluye varias observaciones y una de las variables (en este caso, año) se encuentra almacenada en el encabezado. Así, el código de ggplot que utilizamos apenas no funcionará con un formato de datos amplio.

Hasta ahora, hemos revisado el primer paso del proceso de análisis de datos, **la importación de datos**; ahora, comentaremos el segundo paso habitual, **remodelación de datos** en una forma que facilite el resto del análisis.

Una de las funciones más utilizadas en este contexto es “gather()”, el cual convierte *wide data* en *tidy data*. El primer argumento de esta función define el nombre de la columna que contendrá a la variable que está actualmente almacenada en el nombre de columna de los datos amplios (en nuestro ejemplo, será año); el segundo argumento define el nombre de la columna que contendrá a los valores en las celdas de columna (en nuestro ejemplo, será fertilidad); finalmente, el tercer argumento especifica las columnas en las que la información se reunirá.

```
head(wide_data)

## # A tibble: 2 x 57
##   country `1960` `1961` `1962` `1963` `1964` `1965` `1966` `1967` `1968` `1969` ...
##   <chr>     <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl> ...
## 1 Germany    2.41    2.44    2.47    2.49    2.49    2.48    2.44    2.37    2.28    2.17
## 2 South K~   6.16    5.99    5.79    5.57    5.36    5.16    4.99    4.85    4.73    4.62
## # ... with 46 more variables: 1970 <dbl>, 1971 <dbl>, 1972 <dbl>, 1973 <dbl>,
## # 1974 <dbl>, 1975 <dbl>, 1976 <dbl>, 1977 <dbl>, 1978 <dbl>, 1979 <dbl>,
## # 1980 <dbl>, 1981 <dbl>, 1982 <dbl>, 1983 <dbl>, 1984 <dbl>, 1985 <dbl>,
## # 1986 <dbl>, 1987 <dbl>, 1988 <dbl>, 1989 <dbl>, 1990 <dbl>, 1991 <dbl>,
## # 1992 <dbl>, 1993 <dbl>, 1994 <dbl>, 1995 <dbl>, 1996 <dbl>, 1997 <dbl>,
## # 1998 <dbl>, 1999 <dbl>, 2000 <dbl>, 2001 <dbl>, 2002 <dbl>, 2003 <dbl>,
## # 2004 <dbl>, 2005 <dbl>, 2006 <dbl>, 2007 <dbl>, 2008 <dbl>, 2009 <dbl>,
## # 2010 <dbl>, 2011 <dbl>, 2012 <dbl>, 2013 <dbl>, 2014 <dbl>, 2015 <dbl>
new_tidy_data <- wide_data %>%
  gather(year, fertility, `1960`:`2015`)

head(new_tidy_data)

## # A tibble: 6 x 3
##   country     year   fertility
##   <chr>       <chr>     <dbl>
## 1 Germany     1960      2.41
## 2 South Korea 1960      6.16
## 3 Germany     1961      2.44
## 4 South Korea 1961      5.99
## 5 Germany     1962      2.47
## 6 South Korea 1962      5.79
```

Nótese que la única columna que no fue “reunida” es *country*; así, existe una manera más rápida de escribir este código al especificar cuáles columnas no reunir:

```
new_tidy_data <- wide_data %>%
  gather(year, fertility, -country)
```

Es importante mencionar que existe una ligera diferencia entre las bases de datos creadas “tidy\_data” y “new\_tidy\_data”:

```
class(tidy_data$year)

## [1] "integer"

class(new_tidy_data$year)

## [1] "character"
```

Esto se debe a que la función “gather()” asume que los nombres de columnas son caracteres (incluso en nuestro ejemplo, donde eran años/números). Por tanto, es necesario un último ajuste antes de poder realizar el gráfico, hay que convertir la columna de caracteres a números. Para esto puede usarse la función base de R “as.numeric()”, sin embargo, “gather()” tiene un argumento para esto:

```

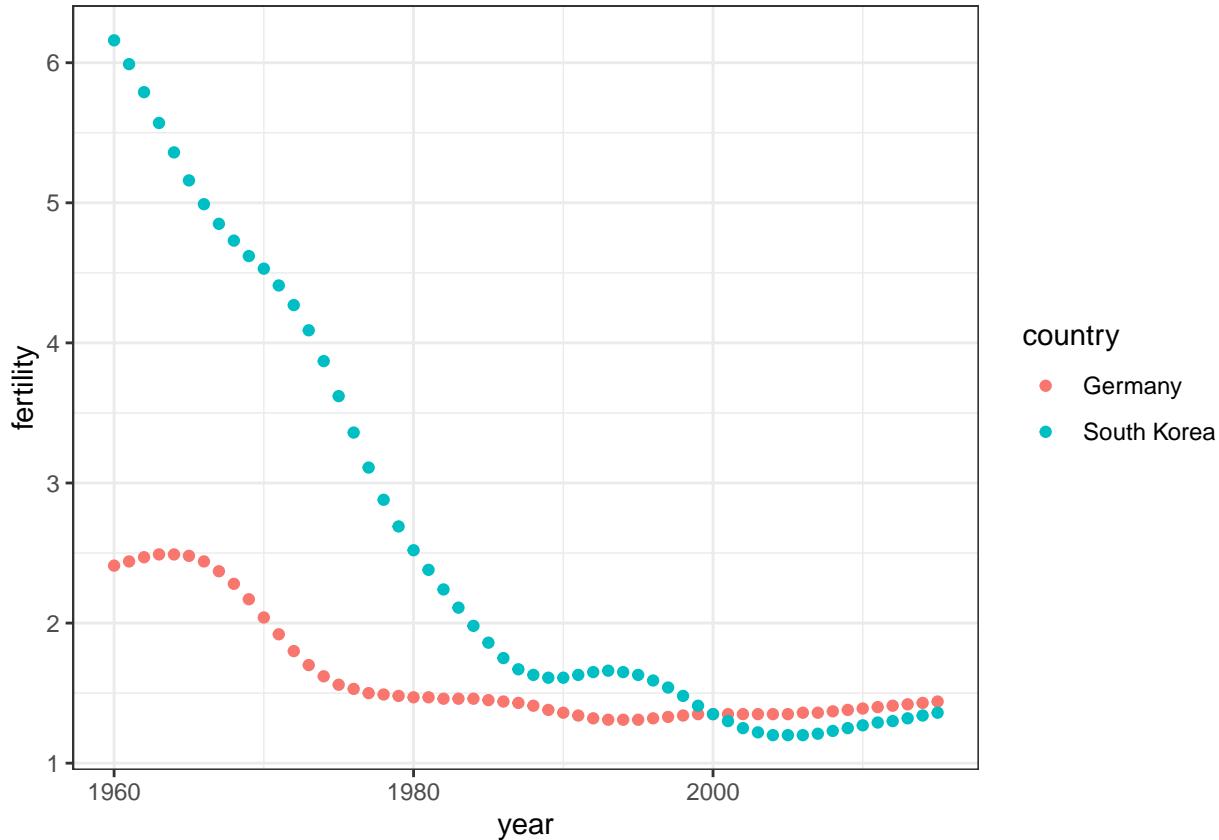
new_tidy_data <- wide_data %>%
  gather(year, fertility, -country, convert = TRUE)

class(new_tidy_data$year)

## [1] "integer"

new_tidy_data %>%
  ggplot(aes(year, fertility, color = country)) +
  geom_point()

```



Por otro lado, en algunos casos puede ser conveniente convertir *tidy data* en *wide data*. Para ello, la función “spread()” es básicamente lo opuesto a “gather()”: su primer argumento define qué variable será usada como nombre de columna; el segundo especifica qué variable se usará para llenar las celdas:

```

new_wide_data <- new_tidy_data %>%
  spread(year, fertility)

select(new_wide_data, country, `1960`:`1967`)

## # A tibble: 2 x 9
##   country   `1960` `1961` `1962` `1963` `1964` `1965` `1966` `1967`
##   <chr>     <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
## 1 Germany    2.41    2.44    2.47    2.49    2.49    2.48    2.44    2.37
## 2 South Korea 6.16    5.99    5.79    5.57    5.36    5.16    4.99    4.85

```

Los ejemplos revisados son relativamente simples, por lo que se hará uso de una base de datos un poco más complicada y realista. Este archivo incluye dos variables: expectativa de vida y fertilidad; sin embargo, la

información no se encuentra ordenada.

```
path <- system.file("extdata", package = "dslabs")

filename <- file.path(path, "life-expectancy-and-fertility-two-countries-example.csv")

raw_data <- read_csv(filename)

## 
## -- Column specification -----
## cols(
##   .default = col_double(),
##   country = col_character()
## )
## i Use `spec()` for the full column specifications.

select(raw_data, 1:5)

## # A tibble: 2 x 5
##   country `1960_fertility` `1960_life_expec` `1961_fertility` `1961_life_expec`
##   <chr>        <dbl>            <dbl>        <dbl>            <dbl>
## 1 Germany      2.41           69.3       2.44           69.8
## 2 South K~     6.16           53.0       5.99           53.8
```

Puede verse que los datos se encuentran en formato amplio; además, existen valores para dos variables en los nombres de columna. Para comenzar la limpieza, úsese la función “gather()”, aunque ahora no hay que usar el nombre Año para las nuevas columnas, dado que también contiene al tipo de variable:

```
data <- raw_data %>%
  gather(key, value, -country)

head(data)

## # A tibble: 6 x 3
##   country   key          value
##   <chr>     <chr>        <dbl>
## 1 Germany   1960_fertility 2.41
## 2 South Korea 1960_fertility 6.16
## 3 Germany   1960_life_expectancy 69.3
## 4 South Korea 1960_life_expectancy 53.0
## 5 Germany   1961_fertility 2.44
## 6 South Korea 1961_fertility 5.99
```

Nótese que estos datos todavía no son *tidy*, dado que cada observación está asociada con dos renglones y no uno, como se esperaría (queremos que los valores de fertility y life expectancy se muestren en dos columnas diferentes para cada año).

El siguiente paso es separar los valores de “key” en año y el tipo de variable (fertility/life\_expectancy). Esto puede lograrse con la función “separate()”, la cual toma cuatro argumento: los datos, en primer lugar; segundo, el nombre de la columna que será separada; tercero, los nombres de las nuevas columnas; y finalmente, el carácter que separa las variables:

```
data %>%
  separate(key, c("year", "variable_name"), "_")

## Warning: Expected 2 pieces. Additional pieces discarded in 112 rows [3, 4, 7, 8,
## 11, 12, 15, 16, 19, 20, 23, 24, 27, 28, 31, 32, 35, 36, 39, 40, ...].

## # A tibble: 224 x 4
```

```
##   country   year variable_name value
##   <chr>     <chr> <chr>        <dbl>
## 1 Germany   1960 fertility      2.41
## 2 South Korea 1960 fertility     6.16
## 3 Germany   1960 life          69.3
## 4 South Korea 1960 life         53.0
## 5 Germany   1961 fertility      2.44
## 6 South Korea 1961 fertility     5.99
## 7 Germany   1961 life          69.8
## 8 South Korea 1961 life         53.8
## 9 Germany   1962 fertility      2.47
## 10 South Korea 1962 fertility     5.79
## # ... with 214 more rows
```

Nótese que recibimos una advertencia debido a que la variable “life\_expectancy” se trunca a solo “life” debido a que está separada por un “\_”. Para solucionar esto se puede agregar una tercera columna que atrape lo anterior y dejar que la función “separate()” decida cuál columna hay que llenar con los valores faltantes.

```
data %>%
  separate(key, c("year", "first_variable_name", "second_variable_name"), fill = "right")
```

```
## # A tibble: 224 x 5
##   country   year first_variable_name second_variable_name value
##   <chr>     <chr> <chr>           <chr>             <dbl>
## 1 Germany   1960 fertility          <NA>              2.41
## 2 South Korea 1960 fertility          <NA>              6.16
## 3 Germany   1960 life               expectancy        69.3
## 4 South Korea 1960 life               expectancy        53.0
## 5 Germany   1961 fertility          <NA>              2.44
## 6 South Korea 1961 fertility          <NA>              5.99
## 7 Germany   1961 life               expectancy        69.8
## 8 South Korea 1961 life               expectancy        53.8
## 9 Germany   1962 fertility          <NA>              2.47
## 10 South Korea 1962 fertility         <NA>              5.79
## # ... with 214 more rows
```

No obstante, al leer la documentación de “separate()”, encontramos que existe una manera más directa de lograr lo que queremos, al juntar las dos últimas variables cuando existe una separación extra:

```
data %>%
  separate(key, c("year", "variable_name"), sep = "_", extra = "merge")
```

```
## # A tibble: 224 x 4
##   country   year variable_name  value
##   <chr>     <chr> <chr>        <dbl>
## 1 Germany   1960 fertility      2.41
## 2 South Korea 1960 fertility     6.16
## 3 Germany   1960 life_expectancy 69.3
## 4 South Korea 1960 life_expectancy 53.0
## 5 Germany   1961 fertility      2.44
## 6 South Korea 1961 fertility     5.99
## 7 Germany   1961 life_expectancy 69.8
## 8 South Korea 1961 life_expectancy 53.8
## 9 Germany   1962 fertility      2.47
## 10 South Korea 1962 fertility     5.79
## # ... with 214 more rows
```

Ahora solo resta crear una columna para cada variable: para ello, recordemos que podemos utilizar la función “spread()”:

```
data %>%
  separate(key, c("year", "variable_name"), sep = "_", extra = "merge") %>%
  spread(variable_name, value)

## # A tibble: 112 x 4
##   country year  fertility life_expectancy
##   <chr>    <chr>     <dbl>           <dbl>
## 1 Germany  1960      2.41            69.3
## 2 Germany  1961      2.44            69.8
## 3 Germany  1962      2.47            70.0
## 4 Germany  1963      2.49            70.1
## 5 Germany  1964      2.49            70.7
## 6 Germany  1965      2.48            70.6
## 7 Germany  1966      2.44            70.8
## 8 Germany  1967      2.37            71.0
## 9 Germany  1968      2.28            70.6
## 10 Germany 1969     2.17            70.5
## # ... with 102 more rows
```

Un procedimiento que arroja el mismo resultado, aunque un poco más largo, y que ejemplifica el uso de la función “unite()”, es:

```
data %>%
  separate(key, c("year", "first_variable_name", "second_variable_name"), fill = "right") %>%
  unite(variable_name, first_variable_name, second_variable_name, sep="_") %>%
  spread(variable_name, value) %>%
  rename(fertility = fertility_NA)

## # A tibble: 112 x 4
##   country year  fertility life_expectancy
##   <chr>    <chr>     <dbl>           <dbl>
## 1 Germany  1960      2.41            69.3
## 2 Germany  1961      2.44            69.8
## 3 Germany  1962      2.47            70.0
## 4 Germany  1963      2.49            70.1
## 5 Germany  1964      2.49            70.7
## 6 Germany  1965      2.48            70.6
## 7 Germany  1966      2.44            70.8
## 8 Germany  1967      2.37            71.0
## 9 Germany  1968      2.28            70.6
## 10 Germany 1969     2.17            70.5
## # ... with 102 more rows
```

### 6.2.2. Assessment 10

Use the following libraries for these questions:

```
library(tidyverse)
library(dslabs)
```

Examine the built-in dataset `co2`. This dataset comes with base R, not dslabs - just type `co2` to access the dataset. Is `co2` tidy? Why or why not?

```
data(co2)
head(co2)

## [1] 315 316 316 318 318 318
```

Run the following command to define the co2\_wide object:

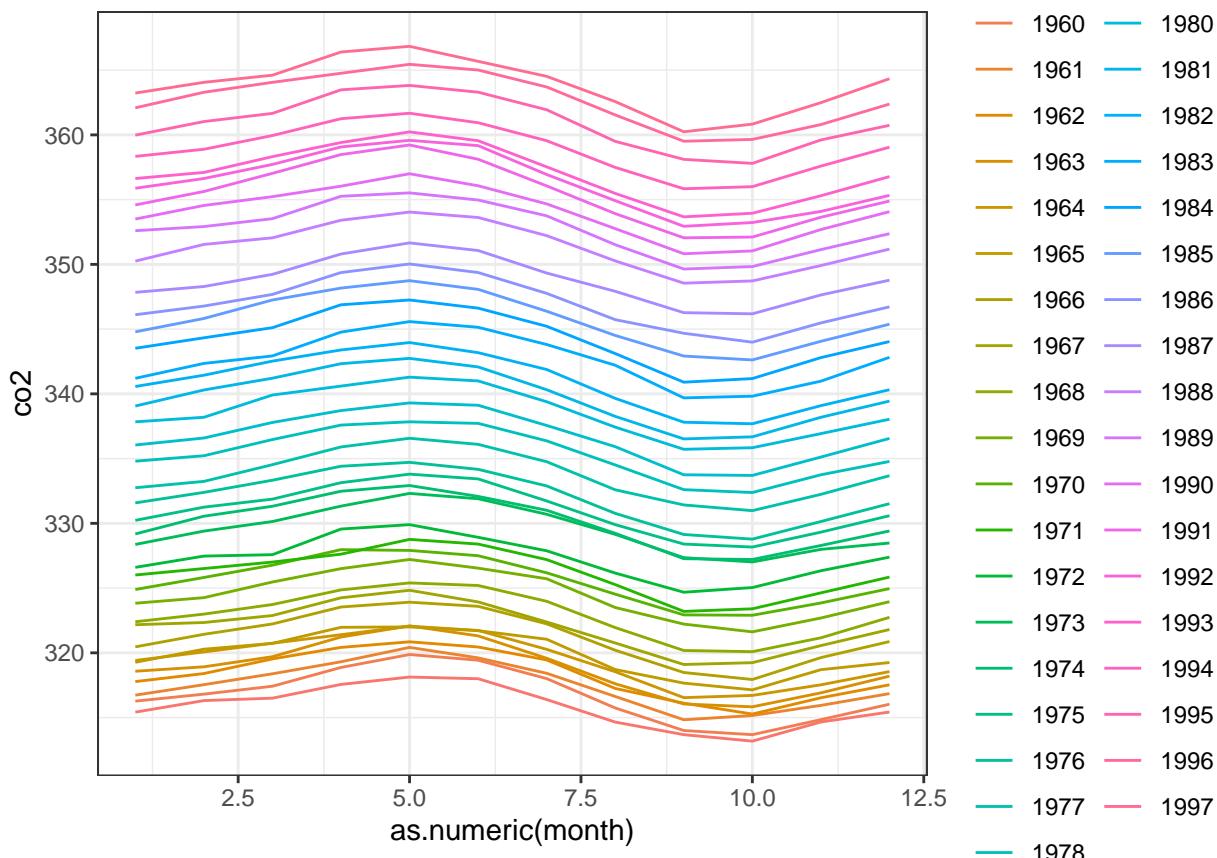
```
co2_wide <- data.frame(matrix(co2, ncol = 12, byrow = TRUE)) %>%
  setNames(1:12) %>%
  mutate(year = as.character(1959:1997))
```

Use the gather() function to make this dataset tidy. Call the column with the CO2 measurements co2 and call the month column month. Name the resulting object co2\_tidy.

```
co2_tidy <- gather(co2_wide, month, co2, -year)
```

Use co2\_tidy to plot CO2 versus month with a different curve for each year:

```
co2_tidy %>%
  ggplot(aes(as.numeric(month), co2, color = year)) +
  geom_line()
```



Load the admissions dataset from dslabs, which contains college admission information for men and women across six majors, and remove the applicants percentage column:

```
library(dslabs)
data(admissions)
dat <- admissions %>% select(-applicants)
```

Your goal is to get the data in the shape that has one row for each major

```
dat_tidy <- spread(dat, gender, admitted)
```

Now use the admissions dataset to create the object tmp, which has columns major, gender, key and value:

```
tmp <- gather(admissions, key, value, admitted:applicants)
tmp
```

```
##   major gender      key value
## 1     A    men admitted    62
## 2     B    men admitted    63
## 3     C    men admitted    37
## 4     D    men admitted    33
## 5     E    men admitted    28
## 6     F    men admitted     6
## 7     A  women admitted   82
## 8     B  women admitted   68
## 9     C  women admitted   34
## 10    D  women admitted   35
## 11    E  women admitted   24
## 12    F  women admitted    7
## 13    A    men applicants 825
## 14    B    men applicants 560
## 15    C    men applicants 325
## 16    D    men applicants 417
## 17    E    men applicants 191
## 18    F    men applicants 373
## 19    A  women applicants 108
## 20    B  women applicants  25
## 21    C  women applicants 593
## 22    D  women applicants 375
## 23    E  women applicants 393
## 24    F  women applicants 341
```

Combine the key and gender and create a new column called column\_name to get a variable with the following values: admitted\_men, admitted\_women, applicants\_men and applicants\_women. Save the new data as tmp2.

```
tmp2 <- unite(tmp, column_name, c(key, gender))
```

Which function can reshape tmp2 to a table with six rows and five columns named major, admitted\_men, admitted\_women, applicants\_men and applicants\_women?

```
tmp2 %>%
  spread(column_name, value)
```

```
##   major admitted_men admitted_women applicants_men applicants_women
## 1     A          62           82        825          108
## 2     B          63           68        560           25
## 3     C          37           34        325          593
## 4     D          33           35        417          375
## 5     E          28           24        191          393
## 6     F           6            7        373          341
```

### 6.2.3. Combinación de tablas

A veces, la información que necesitamos se encuentra en dos tablas o bases de datos.

Como ejemplo, supóngase que se quiere explorar la relación entre el tamaño de la población en cada estado de EE.UU. y los votos electorales.

```
library(tidyverse)
library(ggrepel)
library(dslabs)

ds_theme_set()

data("murders")
head(murders)

##           state abb region population total
## 1      Alabama  AL   South    4779736   135
## 2       Alaska  AK    West     710231    19
## 3    Arizona  AZ    West    6392017   232
## 4   Arkansas  AR   South    2915918    93
## 5 California  CA    West   37253956  1257
## 6 Colorado  CO    West    5029196    65

data("polls_us_election_2016")
head(results_us_election_2016)

##           state electoral_votes clinton trump others
## 1  California                 55   61.7  31.6   6.7
## 2      Texas                   38   43.2  52.2   4.5
## 3     Florida                  29   47.8  49.0   3.2
## 4 New York                   29   59.0  36.5   4.5
## 5 Illinois                  20   55.8  38.8   5.4
## 6 Pennsylvania                20   47.9  48.6   3.6
```

Nótese que no se pueden unir ambas tablas arbitrariamente dado que el ordenamiento de los estados difiere entre ambas:

```
identical(results_us_election_2016$state, murders$state)
```

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

Así, la función “join()” del paquete dplyr, se asegura de que las tablas se combinen de tal forma que los renglones coincidentes estén juntos. La idea general es identificar una o más columnas que contengan la información requerida para emparejar ambas tablas. Nótese qué ocurre si combinamos las tablas anteriores, por estado, utilizando “left\_join()”:

```
tab <- left_join(murders, results_us_election_2016, by = "state")
```

```
head(tab)
```

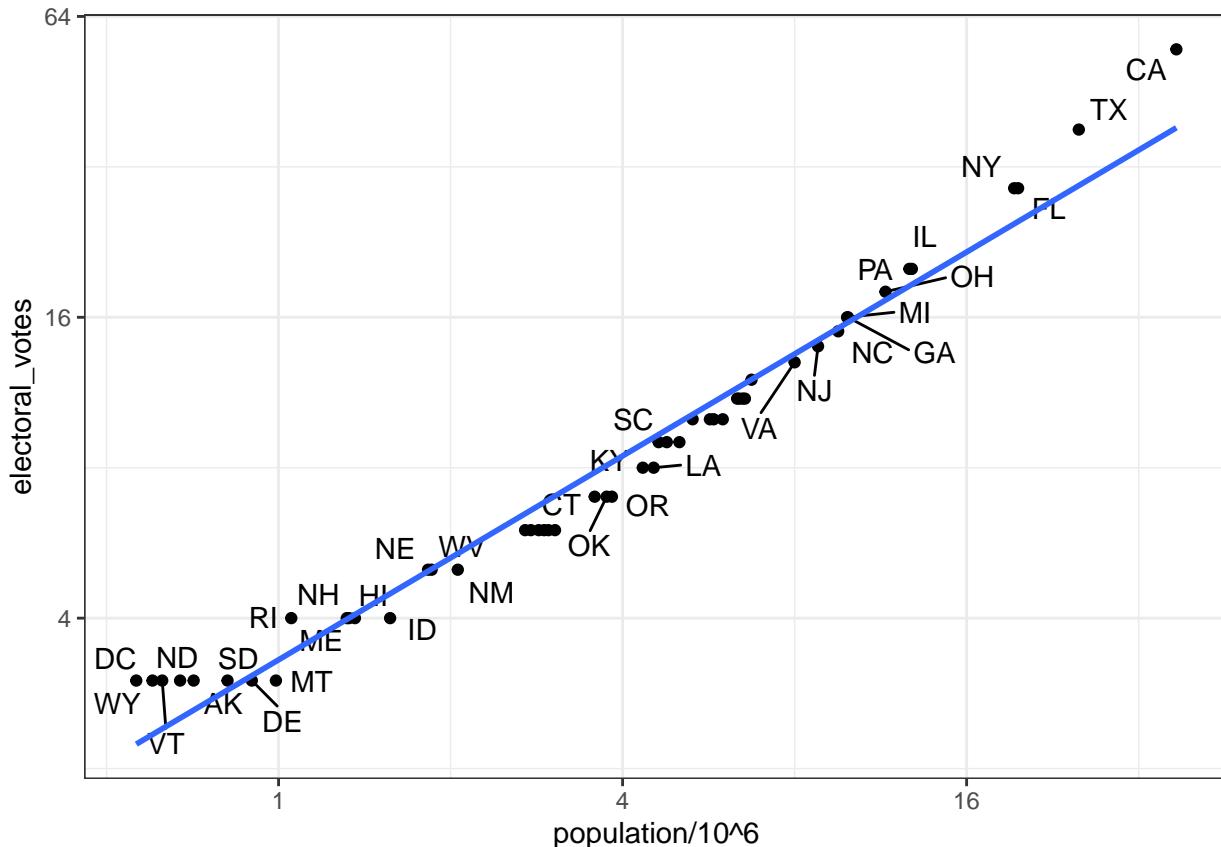
```
##           state abb region population total electoral_votes clinton trump others
## 1      Alabama  AL   South    4779736   135                  9   34.4  62.1   3.6
## 2       Alaska  AK    West     710231    19                  3   36.6  51.3  12.2
## 3    Arizona  AZ    West    6392017   232                 11   45.1  48.7   6.2
## 4   Arkansas  AR   South    2915918    93                  6   33.7  60.6   5.8
## 5 California  CA    West   37253956  1257                 55   61.7  31.6   6.7
## 6 Colorado  CO    West    5029196    65                  9   48.2  43.3   8.6
```

Puede verse que la unión fue exitosa y ahora podemos realizar un gráfico para analizar la relación que nos interesa:

```
tab %>%
  ggplot(aes(population/10^6, electoral_votes, label = abb)) +
  geom_point() +
  geom_text_repel() +
  scale_x_continuous(trans = "log2") +
  scale_y_continuous(trans = "log2") +
  geom_smooth(method = "lm", se = FALSE)

## `geom_smooth()` using formula 'y ~ x'

## Warning: ggrepel: 17 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```



Sin embargo, en la práctica, no siempre ocurre que cada renglón tenga un renglón emparejable en la otra. Para ilustrar este caso, se tomarán dos subconjuntos de las tablas que se han utilizado:

```
tab1 <- slice(murders, 1:6) %>%
  select(state, population)

tab1

##      state population
## 1  Alabama     4779736
## 2   Alaska      710231
## 3  Arizona     6392017
## 4 Arkansas    2915918
```

```

## 5 California 37253956
## 6 Colorado 5029196
tab2 <- slice(results_us_election_2016, c(1:25)) %>%
  select(state, electoral_votes)

tab2

##           state electoral_votes
## 1      California        55
## 2          Texas         38
## 3       Florida         29
## 4    New York         29
## 5     Illinois        20
## 6 Pennsylvania        20
## 7          Ohio         18
## 8       Georgia         16
## 9      Michigan         16
## 10 North Carolina       15
## 11 New Jersey         14
## 12      Virginia        13
## 13 Washington         12
## 14      Arizona         11
## 15      Indiana         11
## 16 Massachusetts        11
## 17 Tennessee          11
## 18      Maryland         10
## 19 Minnesota          10
## 20      Missouri         10
## 21 Wisconsin          10
## 22      Alabama          9
## 23      Colorado          9
## 24 South Carolina        9
## 25      Kentucky          8

```

Se observa que los estados en cada tabla difieren; a continuación se presentan diferentes funciones join con diferentes resultados:

1. `leftjoin()`: Si queremos que queremos una tabla como tab1 pero agregando los votos electorales a aquellos estados que estén disponibles en ella:

```

left_join(tab1, tab2)

## Joining, by = "state"

##           state population electoral_votes
## 1      Alabama   4779736             9
## 2      Alaska    710231            NA
## 3      Arizona   6392017            11
## 4      Arkansas  2915918            NA
## 5 California  37253956            55
## 6      Colorado  5029196            9

```

2. `rightjoin()`: Si queremos un tabla como tab2 pero agregando la población a aquellos estados disponibles en ella:

```
right_join(tab1, tab2)
```

```
## Joining, by = "state"

##           state population electoral_votes
## 1      Alabama     4779736             9
## 2      Arizona     6392017            11
## 3    California    37253956            55
## 4     Colorado     5029196             9
## 5       Texas        NA            38
## 6     Florida        NA            29
## 7   New York        NA            29
## 8    Illinois        NA            20
## 9 Pennsylvania        NA            20
## 10      Ohio        NA            18
## 11    Georgia        NA            16
## 12   Michigan        NA            16
## 13 North Carolina        NA            15
## 14  New Jersey        NA            14
## 15    Virginia        NA            13
## 16  Washington        NA            12
## 17    Indiana        NA            11
## 18 Massachusetts        NA            11
## 19  Tennessee        NA            11
## 20    Maryland        NA            10
## 21   Minnesota        NA            10
## 22    Missouri        NA            10
## 23   Wisconsin        NA            10
## 24 South Carolina        NA            9
## 25    Kentucky        NA            8
```

3. `inner_join()`: Si queremos conservar solo los renglones que tengan información en ambas tablas (puede pensarse como una intersección):

```
inner_join(tab1, tab2)
```

```
## Joining, by = "state"

##           state population electoral_votes
## 1      Alabama     4779736             9
## 2      Arizona     6392017            11
## 3    California    37253956            55
## 4     Colorado     5029196             9
```

4. `full_join()`: Si queremos conservar todas los renglones de ambas tablas, al tiempo de dejar como NAs los valores faltantes (puede pensarse como una unión):

```
full_join(tab1, tab2)
```

```
## Joining, by = "state"

##           state population electoral_votes
## 1      Alabama     4779736             9
## 2      Alaska      710231            NA
## 3      Arizona     6392017            11
## 4     Arkansas     2915918            NA
## 5    California    37253956            55
## 6     Colorado     5029196             9
## 7       Texas        NA            38
## 8     Florida        NA            29
```

```

## 9      New York      NA      29
## 10     Illinois     NA      20
## 11    Pennsylvania   NA      20
## 12      Ohio        NA      18
## 13      Georgia     NA      16
## 14      Michigan     NA      16
## 15 North Carolina  NA      15
## 16 New Jersey      NA      14
## 17 Virginia       NA      13
## 18 Washington     NA      12
## 19 Indiana        NA      11
## 20 Massachusetts  NA      11
## 21 Tennessee      NA      11
## 22 Maryland        NA      10
## 23 Minnesota      NA      10
## 24 Missouri        NA      10
## 25 Wisconsin      NA      10
## 26 South Carolina NA      9
## 27 Kentucky        NA      8

```

Las siguientes funciones no unen tablas técnicamente, sino que permiten conservar partes de una tabla dependiendo de qué hay en otra.

5. `semi_join()`: Si queremos conservar la parte de la primera tabla de la cual tenemos información en la segunda:

```
semi_join(tab1, tab2)
```

```

## Joining, by = "state"
##           state population
## 1 Alabama     4779736
## 2 Arizona     6392017
## 3 California   37253956
## 4 Colorado     5029196

```

6. `anti_join()`: Si queremos conservar los elementos de la primera tabla para los cuales no hay información en la segunda:

```
anti_join(tab1, tab2)
```

```

## Joining, by = "state"
##           state population
## 1 Alaska      710231
## 2 Arkansas    2915918

```

Por su parte, y a diferencia de las funciones `join`, las funciones de ***binding*** no tratan de emparejar ninguna variable, ya que solo combinan *datasets*. Si estos no son de las dimensiones adecuadas para combinarse, se obtendrá un error.

- `bind_cols()`: une dos objetos al poner las columnas de cada uno en una *tibble*:

```
bind_cols(a = 1:3, b = 4:6)
```

```

## # A tibble: 3 x 2
##       a     b
##   <int> <int>
## 1     1     4
## 2     2     5

```

```
## 3     3     6
tab1 <- tab[, 1:3]
tab2 <- tab[, 4:6]
tab3 <- tab[, 7:9]
new_tab <- bind_cols(tab1, tab2, tab3)
head(new_tab)

##           state abb region population total electoral_votes clinton trump others
## 1    Alabama  AL   South     4779736   135                 9   34.4  62.1   3.6
## 2     Alaska  AK    West      710231    19                 3   36.6  51.3  12.2
## 3   Arizona  AZ    West     6392017   232                11   45.1  48.7   6.2
## 4 Arkansas  AR   South     2915918    93                 6   33.7  60.6   5.8
## 5 California CA    West    37253956  1257                55   61.7  31.6   6.7
## 6 Colorado  CO    West     5029196    65                 9   48.2  43.3   8.6
```

- `bin_rows()`: une dos objetos a través de sus renglones y crea una *tibble*.

```
tab1 <- tab[1:2,]
tab2 <- tab[3:4,]
bind_rows(tab1, tab2)
```

```
##           state abb region population total electoral_votes clinton trump others
## 1    Alabama  AL   South     4779736   135                 9   34.4  62.1   3.6
## 2     Alaska  AK    West      710231    19                 3   36.6  51.3  12.2
## 3   Arizona  AZ    West     6392017   232                11   45.1  48.7   6.2
## 4 Arkansas  AR   South     2915918    93                 6   33.7  60.6   5.8
```

Existen funciones base de R, “`cbind()`” y “`rbind()`” que hacen lo mismo, con la diferencia de que crea objetos y no *tibbles*.

Otro conjunto de comandos útiles para la combinación de datos son los *set operators*. Cuando se aplican a vectores, se comportan como uniones, intersecciones, etc.

Sin embargo, si estamos utilizando `dplyr`, estos comandos pueden ser utilizados en *data frames* y no solo en vectores:

- `intersect()`: puede tomarse la intersección de vectores numéricos o de caracteres; asimismo tomará la intersección de renglones para tablas que tengan los mismos nombres de columnas:

```
intersect(1:10, 6:15)

## [1]  6  7  8  9 10

intersect(c("a", "b", "c"), c("b", "c", "d"))

## [1] "b" "c"

tab1 <- tab[1:5,]
tab2 <- tab[3:7,]

intersect(tab1, tab2)
```

```
##           state abb region population total electoral_votes clinton trump others
## 1   Arizona  AZ    West     6392017   232                11   45.1  48.7   6.2
## 2 Arkansas  AR   South     2915918    93                 6   33.7  60.6   5.8
## 3 California CA    West    37253956  1257                55   61.7  31.6   6.7
```

- `union()`: toma la unión de vectores numéricos o de caracteres, así como de *data frames* si se está utilizando `dplyr`:

```
union(1:10, 6:15)

## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
union(c("a","b","c"), c("b","c","d"))

## [1] "a" "b" "c" "d"

tab1 <- tab[1:5,]
tab2 <- tab[3:7,]
union(tab1, tab2)

##      state abb    region population total electoral_votes clinton trump
## 1    Alabama AL     South   4779736   135                 9    34.4 62.1
## 2     Alaska AK      West   710231    19                 3    36.6 51.3
## 3   Arizona AZ      West   6392017   232                11    45.1 48.7
## 4 Arkansas AR     South  2915918    93                 6    33.7 60.6
## 5 California CA      West  37253956  1257                55    61.7 31.6
## 6 Colorado CO      West  5029196    65                 9    48.2 43.3
## 7 Connecticut CT Northeast 3574097    97                 7    54.6 40.9
##   others
## 1    3.6
## 2   12.2
## 3    6.2
## 4    5.8
## 5    6.7
## 6    8.6
## 7    4.5
```

- `setdiff()`: permite obtener diferencia de conjuntos entre el primer y segundo argumento; nótese que no es una función simétrica como las anteriores:

```
setdiff(1:10, 6:15)

## [1] 1 2 3 4 5
setdiff(6:15, 1:10)

## [1] 11 12 13 14 15
tab1 <- tab[1:5,]
tab2 <- tab[3:7,]
setdiff(tab1, tab2)

##      state abb region population total electoral_votes clinton trump others
## 1 Alabama AL     South   4779736   135                 9    34.4 62.1    3.6
## 2 Alaska AK      West   710231    19                 3    36.6 51.3   12.2
```

- `setequal()`: esta función nos indica si dos conjuntos son iguales independientemente del orden de sus elementos:

```
setequal(1:5, 1:6)

## [1] FALSE
setequal(1:5, 5:1)

## [1] TRUE
setequal(tab1, tab2)
```

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

#### 6.2.4. Assessment 11

Install and load the Lahman library. This library contains a variety of datasets related to US professional baseball. We will use this library for the next few questions and will discuss it more extensively in the Regression course. For now, focus on wrangling the data rather than understanding the statistics.

The Batting data frame contains the offensive statistics for all baseball players over several seasons. Filter this data frame to define top as the top 10 home run (HR) hitters in 2016:

```
library(Lahman)
```

```
## Warning: package 'Lahman' was built under R version 4.0.5
```

```
top <- Batting %>%
  filter(yearID == 2016) %>%
  arrange(desc(HR)) %>%      # arrange by descending HR count
  slice(1:10)      # take entries 1-10
top %>% as_tibble()
```

```
## # A tibble: 10 x 22
##   playerID yearID stint teamID lgID     G    AB     R     H    X2B    X3B    HR
##   <chr>     <int> <int> <fct> <fct> <int> <int> <int> <int> <int> <int> <int>
## 1 trumbma01  2016     1 BAL    AL    159   613   94   157   27    1   47
## 2 cruzne02   2016     1 SEA    AL    155   589   96   169   27    1   43
## 3 daviskh01  2016     1 OAK    AL    150   555   85   137   24    2   42
## 4 doziebr01  2016     1 MIN    AL    155   615   104  165   35    5   42
## 5 encared01  2016     1 TOR    AL    160   601   99   158   34    0   42
## 6 arenano01  2016     1 COL    NL    160   618   116  182   35    6   41
## 7 cartech02  2016     1 MIL    NL    160   549   84   122   27    1   41
## 8 frazito01  2016     1 CHA    AL    158   590   89   133   21    0   40
## 9 bryankr01  2016     1 CHN    NL    155   603   121  176   35    3   39
## 10 canoro01  2016     1 SEA    AL    161   655   107  195   33    2   39
## # ... with 10 more variables: RBI <int>, SB <int>, CS <int>, BB <int>,
## #   SO <int>, IBB <int>, HBP <int>, SH <int>, SF <int>, GIDP <int>
```

Also Inspect the Master data frame, which has demographic information for all players:

```
Master %>% as_tibble()
```

```
## # A tibble: 20,093 x 26
##   playerID birthYear birthMonth birthDay birthCountry birthState birthCity
##   <chr>     <int>     <int>     <int> <chr>       <chr>       <chr>
## 1 aardsda01  1981      12        27 USA         CO          Denver
## 2 aaronha01  1934      2         5 USA         AL          Mobile
## 3 aaronto01  1939      8         5 USA         AL          Mobile
## 4 aasedo01   1954      9         8 USA         CA          Orange
## 5 abadan01  1972      8        25 USA         FL          Palm Beach
## 6 abadife01  1985      12       17 D.R.       La Romana La Romana
## 7 abadijo01  1850      11       4 USA         PA          Philadelphia
## 8 abbated01  1877      4        15 USA         PA          Latrobe
## 9 abbeybe01  1869      11       11 USA         VT          Essex
## 10 abbeych01 1866      10       14 USA         NE          Falls City
## # ... with 20,083 more rows, and 19 more variables: deathYear <int>,
## #   deathMonth <int>, deathDay <int>, deathCountry <chr>, deathState <chr>,
```

```
## #  deathCity <chr>, nameFirst <chr>, nameLast <chr>, nameGiven <chr>,
## #  weight <int>, height <int>, bats <fct>, throws <fct>, debut <chr>,
## #  finalGame <chr>, retroID <chr>, bbrefID <chr>, deathDate <date>,
## #  birthDate <date>
```

Use the correct join or bind function to create a combined table of the names and statistics of the top 10 home run (HR) hitters for 2016. This table should have the player ID, first name, last name, and number of HR for the top 10 players. Name this data frame top\_names.

```
top_names <- top %>%
  left_join(Master) %>%
  select(playerID, nameFirst, nameLast, HR)
```

```
## Joining, by = "playerID"
```

Inspect the Salaries data frame. Filter this data frame to the 2016 salaries, then use the correct bind join function to add a salary column to the top\_names data frame from the previous question. Name the new data frame top\_salary. Use this code framework:

```
top_salary <- Salaries %>%
  filter(yearID == 2016) %>%
  right_join(top_names) %>%
  select(nameFirst, nameLast, teamID, HR, salary)
```

```
## Joining, by = "playerID"
```

Inspect the AwardsPlayers table. Filter awards to include only the year 2016. How many players from the top 10 home run hitters won at least one award in 2016?

```
awards2016 <- AwardsPlayers %>%
  filter(yearID == 2016)
```

```
top_names %>%
  inner_join(awards2016)
```

```
## Joining, by = "playerID"
```

	playerID	nameFirst	nameLast	HR	awardID	yearID	lgID	tie	notes
## 1	trumbma01	Mark	Trumbo	47	Silver Slugger	2016	AL	<NA>	OF
## 2	arenano01	Nolan	Arenado	41	Silver Slugger	2016	NL	<NA>	3B
## 3	arenano01	Nolan	Arenado	41	Gold Glove	2016	NL	<NA>	3B
## 4	bryankr01	Kris	Bryant	39	Most Valuable Player	2016	NL	<NA>	<NA>
## 5	bryankr01	Kris	Bryant	39	Hank Aaron Award	2016	NL	<NA>	3B
## 6	bryankr01	Kris	Bryant	39	TSN All-Star	2016	NL	<NA>	3B

How many players won an award in 2016 but were not one of the top 10 home run hitters in 2016?

```
dat4 <- awards2016 %>%
  anti_join(top_names)
```

```
## Joining, by = "playerID"
```

```
unique(dat4$playerID)
```

```
## [1] "perezsa02" "cabremi01" "altuvjo01" "donaljo02" "bogaexa01" "bettsmo01"
## [7] "troutmi01" "ortizda01" "ramoswi01" "rizzoan01" "murphda08" "seageco01"
## [13] "blackch02" "cespeyo01" "yelicch01" "arrieja01" "morelmi01" "kinslia01"
## [19] "beltrad01" "lindofr01" "gardnbr01" "kiermke01" "keuchda01" "poseybu01"
## [25] "panikjo01" "crawfb01" "martest01" "inciaen01" "heywaja01" "greinza01"
```

```
## [31] "porceri01" "scherma01" "fulmemi01" "grandcu01" "zobribe01" "millean01"
## [37] "baezja01"   "lestejo01"  "rendoan01"  "brittza01"  "janseke01"  "klubeco01"
## [43] "freemfr01" "bradlja02"
```

### 6.2.5. Web scraping

A veces, la información que necesitamos no está disponible tan fácilmente. *Web scraping* se refiere al proceso de extracción de datos de un sitio web.

De manera general, podemos acceder al código fuente de un sitio web, descargarlo, importarlo a R y extraer la información ahí.

De hecho, los datos que se han utilizado varias veces sobre asesinatos en EE.UU. con “`data("murders")`” fueron extraídos originalmente de la página de Wikipedia.

El paquete `rvest` (incluido en `tidyverse`) incluye funciones para extraer nodos (`< >`) de un documento HTML: “`html_nodes()`” extrae todos los nodos de diferentes tipos, “`html_node()`” extrae el primer nodo y “`html_table()`” convierte una tabla HTML en un *data frame*.

Así, en primer lugar, importaremos el código fuente de la página a R:

```
library(rvest)

url <- "https://en.wikipedia.org/wiki/Murder_in_the_United_States_by_state"

h <- read_html(url)
```

```
class(h)
```

```
## [1] "xml_document" "xml_node"
```

¿Cómo extraemos la tabla del objeto?

```
tab <- h %>%
  html_nodes("table")
```

```
tab <- tab[[2]]
```

```
tab
```

```
## {html_node}
```

```
## <table class="wikitable sortable">
```

```
## [1] <tbody>\n<tr>\n<th data-sort-type="text">State\n</th>\n<th data-sort-type ...
```

Ahora, hay que convertir esta tabla a formato *data frame*:

```
tab <- tab %>%
  html_table
```

```
class(tab)
```

```
## [1] "tbl_df"      "tbl"        "data.frame"
```

Finalmente, podemos modificar los nombres de las columnas como queramos:

```
tab <- tab %>%
  setNames(c("state", "population", "total", "murders", "gun_murders", "gun_ownership",
            "total_rate", "murder_rate", "gun_murder_rate"))
```

```
head(tab)
```

```
## # A tibble: 6 x 9
##   state      population total_murders gun_murders gun_ownership total_rate
##   <chr>        <chr>       <chr>     <chr>           <dbl>       <dbl>
## 1 Alabama    4,853,875     348     -[a]          -[a]         48.9        7.2
## 2 Alaska     737,709       59       57            39          61.7        8
## 3 Arizona    6,817,565     306     278           171          32.3        4.5
## 4 Arkansas   2,977,853     181     164           110          57.9        6.1
## 5 California 38,993,940   1,861   1,861          1,275         20.1        4.8
## 6 Colorado   5,448,819     176     176           115          34.3        3.2
## # ... with 2 more variables: murder_rate <chr>, gun_murder_rate <chr>
```

### 6.3. Procesamiento de cadenas

```
library(tidyverse)
```

```
library(rvest)
```

Una de las tareas más importantes del *data wrangling* es el procesamiento de cadenas; específicamente, convertir o extraer datos numéricos contenidos en cadenas de caracteres, con el objetivo de realizar gráficos, resumir estadísticamente o ajustar modelos.

Retómese el ejemplo de extraer información de la página web de asesinatos en EE.UU.:

```
url <- "https://en.wikipedia.org/w/index.php?title=Gun_violence_in_the_United_States_by_state&direction=prev&limit=1000&offset=0&ns0=1"

murders_raw <- read_html(url) %>%
  html_nodes("table") %>%
  html_table()

murders_raw <- murders_raw[[1]] %>%
  setNames(c("state", "population", "total", "murders", "gun_murders", "gun_ownership",
            "total_rate", "murder_rate", "gun_murder_rate"))

head(murders_raw)

## # A tibble: 6 x 4
##   state      population total  murders
##   <chr>        <chr>     <dbl>    <dbl>
## 1 Alabama     4,853,875   348     7.2
## 2 Alaska      737,709    59      8
## 3 Arizona     6,817,565   309     4.5
## 4 Arkansas    2,977,853   181     6.1
## 5 California  38,993,940  1,861    4.8
## 6 Colorado    5,448,819   176     3.2
```

Al inspeccionar los datos, nos damos cuenta de que dos de las columnas que esperábamos que contuvieran números son realmente caracteres:

```
class(murders_raw$population)
```

```
## [1] "character"
```

```
class(murders_raw$total)
```

```
## [1] "character"
```

Esto es muy común al realizar *web scraping* dado que muchas páginas y documentos utilizan comas u otros separadores para almacenar datos, por lo que R los reconoce como caracteres. Dado que se trata de un fenómeno muy observado, existe una función que realiza fácilmente la conversión de números con coma a cadenas numéricas (i.e. 1,000 se vuelve 1000), “parse\_number()”

```
parse_number(murders_raw$population)
```

```
## [1] 4853875  737709  6817565  2977853  38993940  5448819  3584730  944076
## [9] 670377  20244914 10199398  1425157  1652828  12839047  6612768  3121997
## [17] 2906721  4424611  4668960  1329453  5994983  6784240  9917715  5482435
## [25] 2989390  6076204  1032073  1893765  2883758  1330111  8935421  2080328
## [33] 19747183 10035186  756835  11605090  3907414  4024634  12791904  1055607
## [41] 4894834  857919  6595056  27429639  2990632  626088  8367587  7160290
## [49] 1841053  5767891  586107
```

Entre las tareas más comunes del procesamiento de cadenas se encuentran:

1. Remover caracteres no deseados de texto
2. Extraer valores numéricos de texto
3. Encontrar y reemplazar caracteres
4. Extraer partes específicas de una cadena
5. Convertir texto de forma libre a formatos más uniformes
6. Separar cadenas en valores múltiples.

### 6.3.1. Comillas simples y dobles

Para definir cadenas de texto en R, podemos utilizar comillas simples o dobles:

```
s <- "Hola"
s <- 'Hola'
```

Nótese que definir un objeto con tildes arrojará un error:

```
# s <- `Hola`
# s <- `Hola`
```

Así, para incluir una comilla simple dentro de una cadena, hay que usar comillas dobles alrededor; similarmente, para incluir una comilla doble dentro de una cadena, hay que usar comillas simples alrededor:

```
s <- "10"
t <- '10'
```

Con la función “cat()” podemos observar cómo luce la cadena:

```
cat(s)
## 10'
cat(t)
## 10"
```

Si se requiere escribir una cadena de texto que incluya comillas simples y dobles, es necesario utilizar ""para" escapar la comilla doble (de manera similar a como se hace en LaTeX en la escritura matemática):

```
s <- '5'10"'
cat(s)
## 5'10"
s <- "5'10\\\""
cat(s)
## 5'10"
```

### 6.3.2. stringr

En general, el procesamiento de cadenas de texto involucra a una cadena y a un patrón. En R, usualmente guardamos a nuestras cadenas en vectores de caracteres.

Como ejemplo, recuérdese que los datos de “murders” tienen definida a la población y al total como vectores de caracteres, cuando queremos que sean numéricos. Podemos observar las primeras tres observaciones de la población:

```
murders_raw$population[1:3]
```

```
## [1] "4,853,875" "737,709"    "6,817,565"
```

Es importante recalcar que la conversión usual o esperada mediante la función “as.numeric()” no funciona (debido a las comas que contienen los números):

```
as.numeric(murders_raw$population)
```

## Warning: NAs introducidos por coerción

Así, la tarea que nos ocupa es remover las comas de la columna de datos en “population”. Las tareas principales del procesamiento de cadenas son (1) detección, (2) localización, (3) extracción/reemplazo de elementos de la cadena.

En nuestro ejemplo, tenemos que localizar la coma y reemplazarla con un carácter vacío . Así, y a pesar de que R tiene funciones base para lograr esto, utilizamos el paquete stringr, debido a su facilidad y simplicidad.

### 6.3.3. Ejemplo: Asesinatos en EE.UU.

Retómese la tabla de datos de asesinatos en EE.UU., donde notamos que las columnas que necesitan conversión de caracteres a números son “population” y “total”:

head(murders raw)

```
## # A tibble: 6 x 4
##   state      population total    murders
##   <chr>        <chr>     <chr>    <dbl>
## 1 Alabama     4,853,875  348     7.2
## 2 Alaska      737,709    59      8
## 3 Arizona     6,817,565  309     4.5
## 4 Arkansas    2,977,853  181     6.1
## 5 California  38,993,940 1,861    4.8
## 6 Colorado    5,448,819  176     3.2
```

Podemos utilizar la función “str\_detect()” para ver el patrón que contienen las cadenas de la base de datos:

```
commas <- function(x) any(str_detect(x, ","))
```

```
murders_raw %>%  
  summarize_all(funs(commas))
```

```
## # A tibble: 1 x 4
##   state population total murders
##   <lgl>     <lgl>    <lgl> <lgl>
## 1 FALSE      TRUE     TRUE  FALSE
```

Después, con la función “str\_replace\_all()”, podemos removerlas:

```
test_1 <- str_replace_all(murders, raw$population, ".", "")
```

```
class(test 1)
```

```
## [1] "character"
```

Ya se removieron las comas, pero las cadenas siguen siendo de caracteres; no obstante, ahora sí podemos usar la función “`as.numeric()`” para coercionar los valores a numéricos:

```
test_1 <- as.numeric(test_1)
```

```
class(test_1)
```

```
## [1] "numeric"
```

Así, con la función “`mutate_at()`” se realiza la misma función para todas las columnas, donde no se verán afectadas aquellas que no tengan coma. Además, dado que esta operación es sumamente común, la función “`parse_number()`” remueve los signos de puntuación de cadenas y las convierte a numéricas:

```
test_2 <- parse_number(murders_raw$population)
```

```
identical(test_1, test_2)
```

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

Resumiendo, podemos obtener nuestra tabla deseada mediante el siguiente código:

```
murders_new <- murders_raw %>%
  mutate_at(2:3, parse_number)
```

```
murders_new %>%
  head
```

```
## # A tibble: 6 x 4
##   state      population total    murders
##   <chr>       <dbl>     <dbl>     <dbl>
## 1 Alabama     4853875    348     7.2
## 2 Alaska      737709     59      8
## 3 Arizona     6817565    309     4.5
## 4 Arkansas    2977853    181     6.1
## 5 California  38993940   1861    4.8
## 6 Colorado    5448819    176     3.2
```

#### 6.3.4. Ejemplo: Alturas reportadas

En módulos anteriores, utilizamos la base de datos de alturas reportadas incluida en la librería dslabs:

```
library(dslabs)
```

```
data("reported_heights")
```

```
head(reported_heights)
```

```
##           time_stamp    sex height
## 1 2014-09-02 13:40:36 Male    75
## 2 2014-09-02 13:46:59 Male    70
## 3 2014-09-02 13:59:20 Male    68
## 4 2014-09-02 14:51:53 Male    74
## 5 2014-09-02 15:16:15 Male    61
## 6 2014-09-02 15:16:16 Female  65
```

Nótese que los datos de “heights” son sumamente heterogéneos (algunos tienen “cm”, otros utilizan comillas simples o dobles, otros se reportan en pies e incluso algunos están escritos con letras) por lo que R detecta

esta columna como caracteres:

```
class(reported_heights$height)
```

```
## [1] "character"
```

Si intentamos coercionar los datos mediante “as.numeric()”, nos arroja una advertencia de que se introdujeron muchos NAs que no pudieron ser convertidos a formato numérico

```
x <- as.numeric(reported_heights$height)
```

```
## Warning: NAs introducidos por coerción
```

```
sum(is.na(x))
```

```
## [1] 81
```

Podemos obtener solo las entradas que resultaron en NAs mediante el siguiente código:

```
reported_heights %>%
  mutate(new_height = as.numeric(height)) %>%
  filter(is.na(new_height)) %>%
  head(n=20)
```

```
## Warning in mask$eval_all_mutate(quo): NAs introducidos por coerción
```

	time_stamp	sex	height	new_height
## 1	2014-09-02 15:16:28	Male	5' 4"	NA
## 2	2014-09-02 15:16:37	Female	165cm	NA
## 3	2014-09-02 15:16:52	Male	5'7	NA
## 4	2014-09-02 15:16:56	Male	>9000	NA
## 5	2014-09-02 15:16:56	Male	5'7"	NA
## 6	2014-09-02 15:17:09	Female	5'3"	NA
## 7	2014-09-02 15:18:00	Male	5 feet and 8.11 inches	NA
## 8	2014-09-02 15:19:48	Male	5'11	NA
## 9	2014-09-04 00:46:45	Male	5'9''	NA
## 10	2014-09-04 10:29:44	Male	5'10''	NA
## 11	2014-09-09 20:00:38	Male	5,3	NA
## 12	2014-09-11 01:02:37	Male	6'	NA
## 13	2014-09-15 14:38:58	Male	6,8	NA
## 14	2014-09-18 21:40:23	Male	5' 10	NA
## 15	2014-10-08 19:19:33	Female	Five foot eight inches	NA
## 16	2014-10-19 13:08:30	Male	5'5"	NA
## 17	2014-10-24 10:59:48	Female	5'2"	NA
## 18	2014-11-09 15:43:21	Female	5,4	NA
## 19	2014-11-27 12:45:00	Female	5'3	NA
## 20	2014-12-11 14:17:23	Male	5'10''	NA

Puede verse que los datos que se volvieron NAs son especialmente difíciles de coercionar en numéricos. Si bien podría continuarse el análisis descartando estas observaciones, puede hacerse un esfuerzo por convertir el mayor número de entradas posible en numéricas.

Por ejemplo, se observa que muchos estudiantes reportaron su altura con el formato x'y". Esto es el primer paso de nuestra tarea: sondear las entradas problemáticas e identificar patrones que se repitan mucho

Así, escribiremos una función que mantenga las entradas que resultaron NAs al aplicar “as.numeric()” y que además se encuentren dentro de un rango plausible de alturas; además, se hace uso de “suppressWarnings()” para evitar advertencias irrelevantes para nuestro caso:

```
not_inches <- function(x, smallest = 50, tallest = 84){
  inches <- suppressWarnings(as.numeric(x))
  ind <- is.na(inches) | inches < smallest | inches > tallest
  ind
}
```

Al aplicar esta función, encontramos que existen el siguiente número de entradas que no cumplen los requisitos establecidos:

```
problems <- reported_heights %>%
  filter(not_inches(height)) %>%
  .$height

length(problems)
```

```
## [1] 292
```

Obsérvense los primeros 50:

```
problems[1:50]
```

## [1] "6"	"5' 4\""	"5.3"
## [4] "165cm"	"511"	"6"
## [7] "2"	"5'7"	
## [10] "5'7\""	"5'3\""	"5 feet and 8.11 inches"
## [13] "5.25"	"5'11"	"5.5"
## [16] "11111"	"5'9' ''"	"6"
## [19] "6.5"	"150"	"5'10' ''"
## [22] "103.2"	"5.8"	"19"
## [25] "5"	"5.6"	"175"
## [28] "177"	"300"	"5,3"
## [31] "6' "	"6"	"5.9"
## [34] "6,8"	"5' 10"	"5.5"
## [37] "178"	"163"	"6.2"
## [40] "175"	"Five foot eight inches"	"6.2"
## [43] "5.8"	"5.1"	"178"
## [46] "165"	"5.11"	"5'5\""
## [49] "165"	"180"	

Al analizarlos detenidamente, pueden observarse tres patrones:

- x'y ó x'y" ó x'y"

```
pattern <- "^\d\s*\d{1,2}\.*\d*'*$"

str_subset(problems, pattern) %>%
  head(n=10) %>%
  cat

## 5' 4" 5'7 5'7" 5'3" 5'11 5'9' ' 5'10' ' 5' 10 5'5" 5'2"

```

- x.y ó x,y

```
pattern <- "[4-6]\s*[.,]\s*([0-9]|10|11)$"

str_subset(problems, pattern) %>%
  head(n=10) %>%
  cat
```

```
## 5.3 5.5 6.5 5.8 5.6 5,3 5.9 6,8 5.5 6.2
  ■ Entradas en centímetros
ind <- which(between(suppressWarnings(as.numeric(problems))/2.54, 54, 81))

ind <- ind[!is.na(ind)]

problems[ind] %>%
  head(n=10) %>%
  cat

## 150 175 177 178 163 175 178 165 165 180
```

### 6.3.5. Regex

La expresión regex (*regular expressions*) es un modo de describir un patrón específico de caracteres de texto, así como una técnica que permite detectar patrones y extraer las partes que se necesitan. Para hacer esto eficiente y precisamente, se han definido un conjunto de reglas.

El ejemplo más básico es buscar por una coma:

```
patter <- ","

str_detect(murders_raw$total, pattern)

## [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [25] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [37] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [49] FALSE FALSE FALSE
```

Como vimos, algunas entradas se escribieron con “cm”; esto también es un ejemplo de regex. Podemos mostrar todas las entradas que tengan esta expresión con el siguiente código:

```
str_subset(reported_heights$height, "cm")
```

```
## [1] "165cm"  "170 cm"
```

Ahora, preguntémonos cuál de las siguientes cadenas satisfacen el patrón, donde se definirá como “yes” las que sí lo cumplen y “no” las que no, para después crear un vector de cadenas, “s”, que incluya a ambas:

```
yes <- c("180 cm", "70 inches")

no <- c("180", "70'")

s <- c(yes, no)
```

Esto es, estamos preguntando cuáles cadenas incluyen el patrón “cm” o el patrón “inches”. Para ello, usamos la función “str\_detect()” de la siguiente forma:

```
str_detect(s, "cm") | str_detect(s, "inches")
```

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

Sin embargo, lo anterior no es necesario: Regex permite la utilización de caracteres especiales que tienen diferentes significados:

- “|”: o

```
str_detect(s, "cm|inches")
```

```
## [1] TRUE TRUE FALSE FALSE
  ■ “\d”: cualquier dígito (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
yes <- c("5", "6", "5'10", "5 feet", "4'11")

no <- c("", ".", "Five", "six")

s <- c(yes, no)

pattern <- "\\d"

str_detect(s, pattern)
```

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

Es pertinente mencionar la función “str\_view()”, la cual muestra el primer emparejamiento para cada cadena de texto:

```
str_view(s, pattern)
```

```
## PhantomJS not found. You can install it with webshot::install_phantomjs(). If it is installed, please...
```

Por su parte, “str\_view\_all()” muestra todos los emparejamientos en una cadena:

```
str_view_all(s, pattern)
```

### 6.3.6. Clases de caracteres, anclajes y cuantificadores

Las **clases de caracteres** son usadas para definir una serie de caracteres que pueden ser emparejados; estos se definen mediante bracketts, “[ ]”. Así, si queremos que nuestro patrón solo busque por cadenas que contengan un 5 o un 6:

```
str_view(s, "[56]")
```

Por otro lado, si queremos buscar por un rango de dígitos, use utiliza un guión. Por ejemplo, “[0-9]” es equivalente a “\d”:

```
yes <- as.character(4:7)
```

```
no <- as.character(1:3)
```

```
s <- c(yes, no)
```

```
str_detect(s, "[4-7])")
```

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

Nótese que en la notación Regex, todo se toma como carácter. Así, un dígito será un carácter, no un número. Así, “[1-20]” no significa buscar del 1 al 20, significa buscar del 1 al 2 y luego el 0. Similarmente, “[a-z]” y “[A-Z]” buscará por todas las letras minúsculas y mayúsculas, respectivamente; si queremos buscar por todas las letras indistintamente, escribiríamos “[a-zA-Z]”.

En nuestro ejemplo, es útil que nuestro patrón busque cuando exista un solo dígito (dado que los pies nunca podrán ser mayor a un dígito). Para ello, podemos hacer uso de los **anclajes**, que permiten definir patrones que deben terminar o empezar en lugares determinados. Los dos más comunes son “^” y “\$”, que representan el inicio y fin de una cadena, respectivamente.

Así, el patrón “^\\d\$” se lee como: inicio de la cadena seguido por cualquier dígito seguido por el final de la cadena:

```
pattern <- "^\d$"
yes <- c("1", "5", "9")
no <- c("12", "123", " 1", "a4", "b")
s <- c(yes, no)
str_view(s, pattern)
```

Por otro lado, el reporte de alturas en pulgadas se da en uno o dos dígitos; esto se especifica en Regex con **cuantificadores**. Se utilizan las llaves, “{}” y muestran cuántas veces un carácter puede ser repetido en el patrón. Así, el patrón “\d{1,2}” buscará por entradas con uno o dos dígitos:

```
pattern <- "^\d{1,2}$"
yes <- c("1", "5", "9", "12")
no <- c("123", "a4", "b")
s <- c(yes, no)
str_view(s, pattern)
```

En consecuencia, para buscar por el patrón de pie + pulgadas, podemos agregar el símbolo de pies y de pulgadas y construir un patrón para buscar por el formato “x pies y y pulgadas”:

```
pattern <- "^[4-7]'\d{1,2}\"$"
```

Que se lee: comienza la cadena, seguida de un dígito entre 4 y 7, luego el símbolo de pies, seguido por uno o dos dígitos, luego el símbolo de pulgadas y el final de la cadena. Probémoslo:

```
yes <- c("5'7\"", "6'2\"", "5'12\"")
no <- c("6,2\"", "6.2\"", "I am 5'11\"", "3'2\"", "64")
str_detect(yes, pattern)

## [1] TRUE TRUE TRUE
str_detect(no, pattern)
```

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

Retomemos el objeto “problems”, el cual contenía las cadenas que no estaban en pulgadas:

```
pattern <- "^[4-7]'\d{1,2}\"$"
sum(str_detect(problems, pattern))

## [1] 14
```

Sin embargo, todavía existen algunas entradas que nuestro patrón no detecta:

```
problems[c(2,10,11,12,15)] %>%
  str_view(pattern)
```

En primer lugar, observamos que algunos estudiantes escribieron las palabras “pies” y “pulgadas”:

```
str_subset(problems, "inches")

## [1] "5 feet and 8.11 inches" "Five foot eight inches" "5 feet 7inches"
## [4] "5ft 9 inches"           "5 ft 9 inches"        "5 feet 6 inches"
```

También vemos que algunas entradas muestran el uso de dos comillas simples para representar comillas dobles:

```
str_subset(problems, "''")

## [1] "5'9''"    "5'10''"   "5'10''"   "5'3''"    "5'7''"    "5'6''"    "5'7.5''"
## [8] "5'7.5''"  "5'10''"   "5'11''"   "5'10''"   "5'5''"
```

Así, una primera solución es homogenizar la forma de escritura de las pulgadas con un solo símbolo (en nuestro ejemplo, usaremos una comilla simple para los pies y nada para las pulgadas; i.e. 5'9). En consecuencia, ahora nuestro patrón es:

```
pattern <- "^[4-7]'\\d{1,2}$"
```

A continuación se reemplazan “feet”, “ft” y “foot” con el símbolo acordado ‘, e “inches”, “in”, “’” y con un espacio vacío.

```
problems %>%
  str_replace("feet|ft|foot", "") %>%
  str_replace("inches|in|''|\\"", "") %>%
  str_detect(pattern) %>%
  sum
```

```
## [1] 48
```

Otro problema que enfrentamos son los espacios: dado que estos son caracteres, R no los ignora, como se puede ver en el siguiente ejemplo:

```
identical("Hi", "Hi ")
```

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

En Regex, los espacios son representados mediante “\s”, por lo que nuestro nuevo patrón es:

```
pattern_2 <- "^[4-7]'\\s\\d{1,2}\\'$"
```

```
str_subset(problems, pattern_2)
```

```
## [1] "5' 4\"" "5' 11\"" "5' 7\""
```

¿Es necesario un patrón que incluya el espacio y otro que no? No, para esto podemos usar los cuantificadores (queremos un patrón que permita espacios, pero que no los requiera necesariamente). En Regex, el asterisco “\*” significa 0 o más instancias del carácter previo. Nótese el siguiente ejemplo:

```
yes <- c("AB", "A1B", "A11B", "A111B", "A1111B")
```

```
no <- c("A2B", "A21B")
```

```
str_detect(yes, "A1*B")
```

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

```
str_detect(no, "A1*B")
```

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

Existen otros dos cuantificadores relevantes: para “ninguno o una vez” se utiliza “?” y para “uno o más” usamos “+”:

```
data.frame(string = c("AB", "A1B", "A11B", "A111B", "A1111B"),
           none_or_more = str_detect(yes, "A1*B"),
           nore_or_once = str_detect(yes, "A1?B"),
           once_or_more = str_detect(yes, "A1+B"))

##   string none_or_more nore_or_once once_or_more
## 1     AB        TRUE      TRUE      FALSE
## 2    A1B        TRUE      TRUE      TRUE
## 3   A11B        TRUE     FALSE      TRUE
## 4  A111B        TRUE     FALSE      TRUE
## 5 A1111B        TRUE     FALSE      TRUE
```

Por tanto, el nuevo patrón es:

```
pattern <- "[4-7]\\s*\\s*\\d{1,2}$"

problems %>%
  str_replace("feet|ft|foot", "") %>%
  str_replace("inches|in|'", "") %>%
  str_detect(pattern) %>%
  sum
```

```
## [1] 53
```

**Nota:** Podemos estar tentados a utilizar las funciones “str\_replace()” y “str\_replace\_all()” para reemplazar los espacios; sin embargo, es importante considerar si este reemplazo pudiera tener efectos no deseados en nuestras entradas de datos.

### 6.3.7. Grupos con Regex

Como se mencionó, el segundo gran grupo de problemas en las entradas eran de la forma “x.y”, “x,y” y “x y”. Recuérdese que queremos cambiar este formato al que acordamos: “x'y”.

Los **grupos** en Regex permiten extraer valores de cadenas y están definidos con el uso de paréntesis. Si bien estos no alteran la estructura del patrón, permiten identificar partes específicas de este para poder extraerlas.

Por ejemplo, supóngase se quiere modificar 5.6 a 5'6. Nótese que no quisiéramos reemplazar entradas como 70.2 o 180cm, requerimos que el primer dígito esté entre 4 y 7, “4-7” y que el segundo sea ninguno o un solo dígito, “\\d\*”:

```
pattern_without_groups <- "[4-7],\\d*$"
```

Así, lo que queremos es extraer los dígitos para después convertirlos al formato deseado, x'y. Para ello, tenemos que crear dos grupos para cada uno de los dígitos que se extraerán:

```
pattern_with_groups <- "([4-7]),(\\d*)$"
```

```
yes <- c("5,9", "5,11", "6,", "6,1")

no <- c("5'9", "", "2,8", "6.1.1")

s <- c(yes, no)

str_detect(s, pattern_without_groups)
```

```
## [1] TRUE TRUE TRUE TRUE FALSE FALSE FALSE FALSE
str_detect(s, pattern_with_groups)
```

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

Puede verse que el uso de paréntesis (grupos) no afecta la detección del patrón, solo sirve para señalar las partes de este que extraeremos posteriormente. En consecuencia, podemos usar la función “str\_match()” para extraer los valores que estos grupos definen:

```
str_match(s, pattern_with_groups)
```

```
##      [,1]  [,2]  [,3]
## [1,] "5,9"  "5"   "9"
## [2,] "5,11" "5"   "11"
## [3,] "6,"    "6"   ""
## [4,] "6,1"   "6"   "1"
## [5,] NA     NA    NA
## [6,] NA     NA    NA
## [7,] NA     NA    NA
## [8,] NA     NA    NA
```

Donde la primera columna es el patrón original y la segunda y tercera los valores extraídos; en nuestro caso, pies y pulgadas, respectivamente. Ahora es más clara la diferencia entre “str\_extract()” y “str\_match()”: la primera solo extrae las cadenas que cumplen con el patrón, no los valores definidos por los grupos:

```
str_extract(s, pattern_with_groups)
```

```
## [1] "5,9"  "5,11" "6,"   "6,1"  NA     NA     NA
```

Otra ventaja del uso de grupos es que nos podemos referir al valor extraído al momento de buscar y reemplazar: el carácter especial Regex para referirnos al *i*ésimo grupo es “\i” con  $i = 1, \dots, n$ .

Como ejemplo simple, nótese que el siguiente código reemplazará una coma con un punto, pero solo si está entre dos dígitos:

```
pattern_with_groups <- "^(4-7),(\d*)$"
```

```
yes <- c("5,9", "5,11", "6,", "6,1")
```

```
no <- c("5'9", "", "2,8", "6.1.1")
```

```
s <- c(yes, no)
```

```
str_replace(s, pattern_with_groups, "\1.\2")
```

```
## [1] "5'9"    "5'11"   "6'"     "6'1"    "5'9"    ",,"     "2,8"    "6.1.1"
```

Así, nuestro patrón pertinente ahora luce de la siguiente forma:

```
pattern_with_groups <- "^(4-7)\s*,(\.\s+)\s*(\d*)$"
```

Desmenuzándolo:

- ^: inicio de la cadena
- 
- \s\*: ninguno o más espacios en blanco
- 
- \s\*: ninguno o más espacios en blanco

- \d\*: ninguno o más dígitos
- \$: fin de la cadena

```
str_subset(problems, pattern_with_groups) %>%
  str_replace(pattern_with_groups, "\\1\\2")
```

```
## [1] "5'3"   "5'25"  "5'5"   "6'5"   "5'8"   "5'6"   "5'3"   "5'9"   "6'8"   "5'5"
## [11] "6'2"   "6'2"   "5'8"   "5'1"   "5'11"  "5'75"  "5'4"   "5'4"   "6'1"   "5'6"
## [21] "5'6"   "5'4"   "5'9"   "5'6"   "5'6"   "5'5"   "5'2"   "5'5"   "5'5"   "6'5"
## [31] "5'8"   "5'11"  "5'5"   "6'7"   "5'1"   "5'6"   "5'5"   "5'2"   "5'6"   "5'7"
## [41] "5'9"   "6'5"   "5'11"  "5'11"  "5'7"   "5'5"   "5'11"  "5'8"   "5'8"   "5'1"
## [51] "5'11"  "5'7"   "5'9"   "5'2"   "5'5"   "5'51"  "5'8"   "5'7"   "6'1"   "5'69"
## [61] "5'7"   "5'25"  "5'5"   "5'1"   "6'04"  "6'3"   "5'5"   "5'7"   "5'57"  "5'7"
```

En términos del formato, parece que tuvimos éxito. Sin embargo, siguen existiendo algunos valores problemáticos (aquellos donde las pulgadas se reportan como un número mayor a 12).

### 6.3.8. Testeo y mejora

Escribamos una función que capture todas las entradas que no pueden ser convertidas a números, recordando que algunas están en centímetros:

```
not_inches_or_cm <- function(x, smallest = 50, tallest = 84){
  inches <- suppressWarnings(as.numeric(x))
  ind <- !is.na(inches) &
    ((inches >= smallest & inches <= tallest) |
     (inches/2.54 >= smallest & inches/2.54<= tallest))
  !ind
}

problems <- reported_heights %>%
  filter(not_inches_or_cm(height)) %>%
  .$height
length(problems)

## [1] 200

converted <- problems %>%
  str_replace("feet|foot|ft", "") #convertimos las expresiones de pies a '
  str_replace("inches|in|'"|\"", "") #removemos las expresiones de pulgadas
  str_replace("^(4-7)\\s*[,\\.\\s+]\\s*(\\d*)$", "\\1 \\2") #cambiamos formato

pattern <- "[4-7]\\s*[,\\.\\s+]\\s*(\\d{1,2})$"

index <- str_detect(converted, pattern)

mean(index)

## [1] 0.615
```

Vemos que estamos detectando más de la mitad. Examinemos los casos restantes:

```
converted[!index]

## [1] "6"          "165cm"      "511"        "6"
## [5] "2"          ">9000"      "5 ' and 8.11 " "11111"
## [9] "6"          "103.2"      "19"         "5"
## [13] "300"        "6'"          "6"          "Five ' eight "
```

```

## [17] "7"          "214"        "6"          "0.7"
## [21] "6"          "2'33"       "612"        "1,70"
## [25] "87"         "5'7.5"      "5'7.5"      "111"
## [29] "5' 7.78"    "12"         "6"          "yyy"
## [33] "89"         "34"         "25"         "6"
## [37] "6"          "22"         "684"        "6"
## [41] "1"          "1"          "6*12"      "87"
## [45] "6"          "1.6"        "120"        "120"
## [49] "23"         "1.7"        "6"          "5"
## [53] "69"         "5' 9 "     "5 ' 9 "    "6"
## [57] "6"          "86"         "708,661"   "5 ' 6 "
## [61] "6"          "649,606"   "10000"     "1"
## [65] "728,346"   "0"          "6"          "6"
## [69] "6"          "100"        "88"         "6"
## [73] "170 cm"    "7,283,465" "5"          "5"
## [77] "34"

```

Vemos los siguientes problemas: algunos estudiantes registraron su altura exacta sin poner pulgadas (5', 6'); otros, solo escribieron el número entero (5, 6); algunas pulgadas fueron escritas con puntos decimales; otras tienen muchos espacios aleatorios en la entrada; otras más están en centímetros con diferentes notaciones.

### 6.3.9. Separación con Regex

Considérese el ejemplo:

```

s <- c("5'10", "6'1")

tab <- data.frame(x=s)

tab

##      x
## 1 5'10
## 2 6'1

```

Podemos usar la función “separate()” para separar la parte de pies de la parte de pulgadas:

```

tab %>%
  separate(x, c("feet", "inches"), sep = "''")

##   feet inches
## 1    5     10
## 2    6      1

```

Similarmente, la función “extract()” de tidyverse nos permite usar grupos Regex para extraer los valores deseados:

```

tab %>%
  extract(x, c("feet", "inches"), regex = "(\\d)'(\\d{1,2})")

##   feet inches
## 1    5     10
## 2    6      1

```

Si, por ejemplo, definimos un ejemplo como el siguiente, donde solo queremos los números:

```

s <- c("5'10", "6'1\"", "5'8inches")

tab <- data.frame(x=s)

```

Obsérvese que “separate()” no funciona:

```
tab %>%
  separate(x, c("feet", "inches"), sep = "", fill = "right")

##   feet  inches
## 1    5      10
## 2    6      1"
## 3    5 8inches
```

Sin embargo, podemos usar “extract()”:

```
tab %>%
  extract(x, c("feet", "inches"), regex = "(\\d' (\\d{1,2})")"

##   feet inches
## 1    5      10
## 2    6      1
## 3    5       8
```

En resumen, se han identificado cuatro patrones claros presentes en nuestras entradas, así como algunos problemas menores:

1. Muchos estudiantes que miden exactamente 5 o 6 pies no registraron ninguna pulgada. Por ejemplo, 6'.
2. Algunos estudiantes que miden exactamente 5 o 6 pies escribieron solo un número.
3. Algunas de las pulgadas fueron registradas con puntos decimales. Por ejemplo, 5'7.5"
4. Algunas entradas tienen un espacio al final. Por ejemplo, 5 '9
5. Algunas entradas están en metros, donde una parte se escribieron en formato de decimales europeos: 1.6 o 1,7.
6. Dos estudiantes escribieron “cm”.
7. Un estudiante escribió “Five foot eight inches”.

**Caso 1:** Para este caso, si añadimos '0 para convertir, por ejemplo, 6 a 6'0, entonces nuestro patrón lo detectará. Esto puede lograrse de la siguiente manera:

```
yes <- c("5", "6", "5")
no <- c("5'", "5''", "5'4")
s <- c(yes, no)

str_replace(s, "^([4-7])$", "\\\1'0")
```

```
## [1] "5'0" "6'0" "5'0" "5'" "5''" "5'4"
```

Donde el patrón dice: comienzo de la cadena (^), seguido de un dígito entre 4 y 7 (4-7) y luego el final de la cadena (\$). El paréntesis define el grupo que después utilizamos para reemplazar

**Casos 2 y 4:** Podemos adaptar nuestro código ligeramente para manejar el caso de la entrada 5', la cual no es detectada por nuestro patrón de arriba. Así, queremos que nuestro patrón permita que el dígito pueda estar seguido de un símbolo: puede agregarse '{0,1}. También podría utilizarse el carácter ?:

```
str_replace(s, "^(56)??", "\\\1'0")
```

```
## [1] "5'0" "6'0" "5'0" "5'0" "5''" "5'4"
```

Donde solo permitimos 5 y 6 pero no 4 y 7. Esto es debido a que las alturas comunes de exactamente 5 o 6 pies son bastante comunes, al tiempo que alturas de exactamente 4 o 7 pies son extremadamente raras.

**Caso 3:** Podemos usar cuantificadores para este caso, dado que estas entradas no son detectadas porque las pulgadas están con decimales y nuestro patrón no contempla esto. Así, debemos permitir que el segundo grupo de nuestro patrón permita ninguno o un punto, seguido de 0 o más dígitos (usaremos \* y ?):

```
pattern <- "^[4-7] \\s* \\s*(\\d+\\.?\\d*)$"
```

**Caso 5:** En este caso, donde se usan metros y comas, utilizaremos un enfoque similar a la conversión del formato x.y al x'y. Una diferencia es que requerimos que el primer dígito sea 1 o 2:

```
yes <- c("1,7", "1,8", "2, ")
no <- c("5,8", "5,3,2", "1.7")
s <- c(yes, no)
str_replace(s, "^( [12]) \\s*, \\s*(\\d*)$", "\\1\\.\\2")
## [1] "1.7"   "1.8"   "2."    "5,8"   "5,3,2" "1.7"
```

Por otro lado, los espacios al final o principio de una cadena son inservibles. Afortunadamente, existe una función dedicada a removerlos, “str\_trim()”:

```
str_trim("5 ' 9 ")
## [1] "5 ' 9"
```

Así, finalmente podemos definir un procedimiento general para tratar de resolver la mayoría de los problemas encontrados:

```
convert_format <- function(s){
  s %>%
    str_replace("feet|foot|ft", "") %>% #símbolos de pies a '
    str_replace_all("inches|in|'"|"\cm|and", "") %>% #remover símbolos de pulgadas
    str_replace("^( [4-7]) \\s*[,\\.\\s+] \\s*(\\d*)$", "\\1\\.\\2") %>% # cambiar formato de x.y, x,y y x y
    str_replace("^( [56])'?$", "\\1'0") %>% #agregar un 0 cuando sea 5 o 6
    str_replace("^( [12]) \\s*, \\s*(\\d*)$", "\\1\\.\\2") %>% #cambiar decimales europeos
    str_trim() #eliminar espacios
}
```

También defínase una función para convertir palabras a números:

```
words_to_numbers <- function(s){
  str_to_lower(s) %>%
    str_replace_all("zero", "0") %>%
    str_replace_all("one", "1") %>%
    str_replace_all("two", "2") %>%
    str_replace_all("three", "3") %>%
    str_replace_all("four", "4") %>%
    str_replace_all("five", "5") %>%
    str_replace_all("six", "6") %>%
    str_replace_all("seven", "7") %>%
    str_replace_all("eight", "8") %>%
    str_replace_all("nine", "9") %>%
    str_replace_all("ten", "10") %>%
    str_replace_all("eleven", "11")
}
```

Así, restan las siguientes entradas problemáticas:

```
converted <- problems %>% words_to_numbers %>% convert_format

remaining_problems <- converted[not_inches_or_cm(converted)]

pattern <- "^[4-7]\\s*\\s*\\d+\\.?\\d*$"

index <- str_detect(remaining_problems, pattern)

remaining_problems[!index]

## [1] "511"      "2"        ">9000"    "11111"    "103.2"    "19"
## [7] "300"      "7"        "214"      "0.7"      "2'33"    "612"
## [13] "1.70"     "87"      "111"      "12"       "yyy"     "89"
## [19] "34"       "25"      "22"       "684"      "1"       "1"
## [25] "6*12"     "87"      "1.6"      "120"      "120"     "23"
## [31] "1.7"      "86"      "708,661" "649,606" "10000"   "1"
## [37] "728,346" "0"       "100"      "88"       "7,283,465" "34"
```

### 6.3.9.1. Resultado final

Ya estamos listos para utilizar todo lo revisado y limpiar nuestra base de datos.

Primero, comenzaremos limpiando la columna *height* de tal forma que el formato de las entradas sea lo más parecido posible al deseado, x'y. Añadiremos la columna original para poder comparar:

```
pattern <- "^(4-7)\\s*\\s*(\\d+\\.?\\d*)$"

smallest <- 50
tallest <- 84
new_heights <- reported_heights %>%
  mutate(original = height,
         height = words_to_numbers(height) %>% convert_format()) %>%
  extract(height, c("feet", "inches"), regex = pattern, remove = FALSE) %>%
  mutate_at(c("height", "feet", "inches"), as.numeric) %>%
  mutate(guess = 12*feet + inches) %>%
  mutate(height = case_when(
    !is.na(height) & between(height, smallest, tallest) ~ height,
    !is.na(height) & between(height/2.54, smallest, tallest) ~ height/2.54,
    !is.na(height) & between(height*100/2.54, smallest, tallest) ~ height*100/2.54,
    !is.na(guess) & inches < 12 & between(guess, smallest, tallest) ~ guess,
    TRUE ~ as.numeric(NA))) %>%
  select(-guess)

## Warning in mask$eval_all_mutate(quo): NAs introducidos por coerción
```

Las entradas convertidas son:

```
new_heights %>%
  filter(not_inches(original)) %>%
  select(original, height) %>%
  arrange(height) %>%
  head()

##   original height
## 1      150  59.1
## 2      150  59.1
## 3      152  59.8
```

```
## 4      5   60.0
## 5      5   60.0
## 6      5   60.0
```

### 6.3.10. Separación de cadenas

Otra operación común del *data wrangling* es la separación de cadenas.

Supóngase que no podemos usar la función “`read_csv()`”, por lo que debemos recurrir a la función base de R. La función “`readLines()`” lee los datos línea por línea para crear un vector de cadenas:

```
filename <- system.file("extdata/murders.csv", package = "dslabs")

lines <- readLines(filename)

lines %>%
  head()

## [1] "state,abb,region,population,total" "Alabama,AL,South,4779736,135"
## [3] "Alaska,AK,West,710231,19"          "Arizona,AZ,West,6392017,232"
## [5] "Arkansas,AR,South,2915918,93"       "California,CA,West,37253956,1257"
```

Queremos extraer los valores separados por comas; el comando “`str_split()`” hace exactamente esto:

```
x <- str_split(lines, ",")

x %>%
  head()

## [[1]]
## [1] "state"      "abb"        "region"     "population" "total"
##
## [[2]]
## [1] "Alabama"    "AL"         "South"      "4779736"   "135"
##
## [[3]]
## [1] "Alaska"     "AK"         "West"       "710231"   "19"
##
## [[4]]
## [1] "Arizona"    "AZ"         "West"       "6392017"  "232"
##
## [[5]]
## [1] "Arkansas"   "AR"         "South"     "2915918"  "93"
##
## [[6]]
## [1] "California" "CA"         "West"      "37253956" "1257"
```

Nótese que la primera entrada tiene los nombres de columnas, por lo que la podemos separar:

```
col_names <- x[[1]]

x <- x[-1]
```

Para convertir nuestra lista en un *data frame* podemos usar una atajo de la función “`map()`” del paquete `purrr`, la cual aplica la misma función para cada elemento de una lista:

```
library(purrr)
```

```
map(x, function(y) y[1]) %>%  
  head()  
  
## [[1]]  
## [1] "Alabama"  
##  
## [[2]]  
## [1] "Alaska"  
##  
## [[3]]  
## [1] "Arizona"  
##  
## [[4]]  
## [1] "Arkansas"  
##  
## [[5]]  
## [1] "California"  
##  
## [[6]]  
## [1] "Colorado"
```

Sin embargo, como se trata de una tarea muy común, purrr tiene una manera más fácil de hacerlo:

```
map(x, 1) %>%  
  head()  
  
## [[1]]  
## [1] "Alabama"  
##  
## [[2]]  
## [1] "Alaska"  
##  
## [[3]]  
## [1] "Arizona"  
##  
## [[4]]  
## [1] "Arkansas"  
##  
## [[5]]  
## [1] "California"  
##  
## [[6]]  
## [1] "Colorado"
```

Para forzar a la función “map()” de arrojar un vector de caracteres en vez de una lista, podemos utilizar “map\_chr()” (similarmente, “map\_int()” arroja enteros). Así para nuestro *data frame* utilizamos el siguiente código:

```
dat <- data.frame(parse_guess(map_chr(x, 1)),  
                   parse_guess(map_chr(x, 2)),  
                   parse_guess(map_chr(x, 3)),  
                   parse_guess(map_chr(x, 4)),  
                   parse_guess(map_chr(x, 5))) %>%  
  setNames(col_names)  
  
dat %>% head
```

```
##      state abb region population total
## 1    Alabama  AL   South  4779736   135
## 2     Alaska  AK    West   710231    19
## 3   Arizona  AZ    West  6392017   232
## 4  Arkansas  AR   South  2915918    93
## 5 California  CA    West 37253956  1257
## 6 Colorado  CO    West  5029196    65
```

Gracias a purrr, podemos lograr lo anterior con un código más compacto:

```
dat <- x %>%
  transpose() %>%
  map(~ parse_guess(unlist(.))) %>%
  setNames(col_names) %>%
  as.data.frame()

head(dat)

##      state abb region population total
## 1    Alabama  AL   South  4779736   135
## 2     Alaska  AK    West   710231    19
## 3   Arizona  AZ    West  6392017   232
## 4  Arkansas  AR   South  2915918    93
## 5 California  CA    West 37253956  1257
## 6 Colorado  CO    West  5029196    65
```

La función “str\_split()” siempre arroja una lista. Sin embargo, con el argumento simplify = TRUE se creará una matriz, por lo que un procedimiento equivalente es:

```
x <- str_split(lines, ",", simplify = TRUE)
col_names <- x[1,]
x <- x[-1,]
x %>% as_data_frame() %>%
  setNames(col_names) %>%
  mutate_all(parse_guess)

## Warning: `as_data_frame()` was deprecated in tibble 2.0.0.
## Please use `as_tibble()` instead.
## The signature and semantics have changed, see `?as_tibble`.

## Warning: The `x` argument of `as_tibble.matrix()` must have unique column names if ` `.name_repair` is
## Using compatibility ` `.name_repair` .

## # A tibble: 51 x 5
##       state           abb region  population  total
##       <chr>          <chr> <chr>        <dbl> <dbl>
## 1  Alabama          AL   South      4779736   135
## 2  Alaska           AK   West       710231    19
## 3  Arizona          AZ   West      6392017   232
## 4  Arkansas          AR   South     2915918    93
## 5  California        CA   West     37253956  1257
## 6  Colorado          CO   West     5029196    65
## 7  Connecticut       CT   Northeast  3574097   97
## 8  Delaware          DE   South      897934    38
## 9 District of Columbia DC   South      601723    99
## 10 Florida           FL   South     19687653   669
## # ... with 41 more rows
```

### 6.3.11. Recodificar

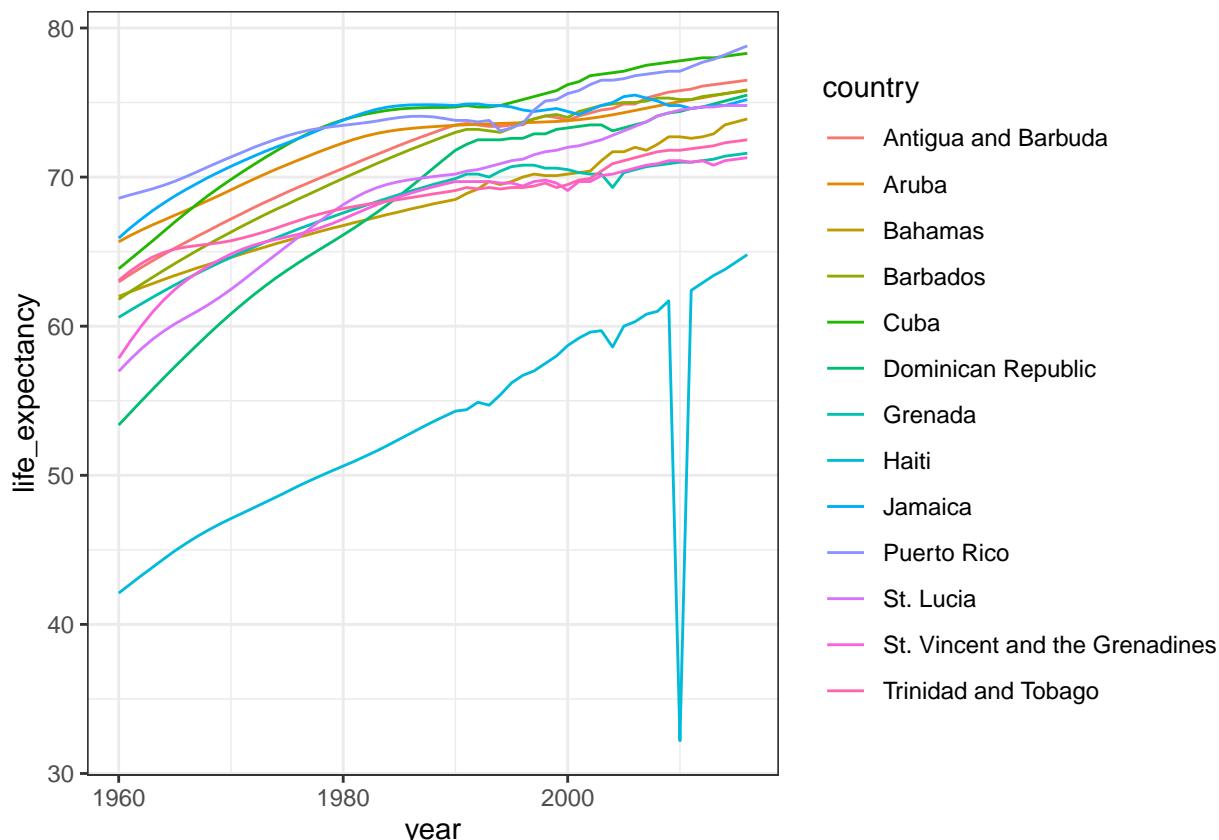
Una última operación común es recodificar los nombres de las variables categóricas. Por ejemplo, si nuestros niveles tienen nombres muy largos y queremos hacer una gráfica que los incluya, tal vez queramos acortarlos. Para ello, usaremos la función “case\_when()” o “recode()” de tidyverse. Considérese un ejemplo con nombres de países:

```
library(dslabs)
```

```
data("gapminder")
```

Supóngase que queremos mostrar una serie de tiempo de la expectativa de vida para países del Caribe:

```
gapminder %>%
  filter(region == "Caribbean") %>%
  ggplot(aes(year, life_expectancy, color = country)) +
  geom_line()
```



Donde se desperdicia una gran cantidad de espacio para tratar de acomodar los nombres de países; de hecho, hay cuatro países con una longitud de caracteres mayor a 12:

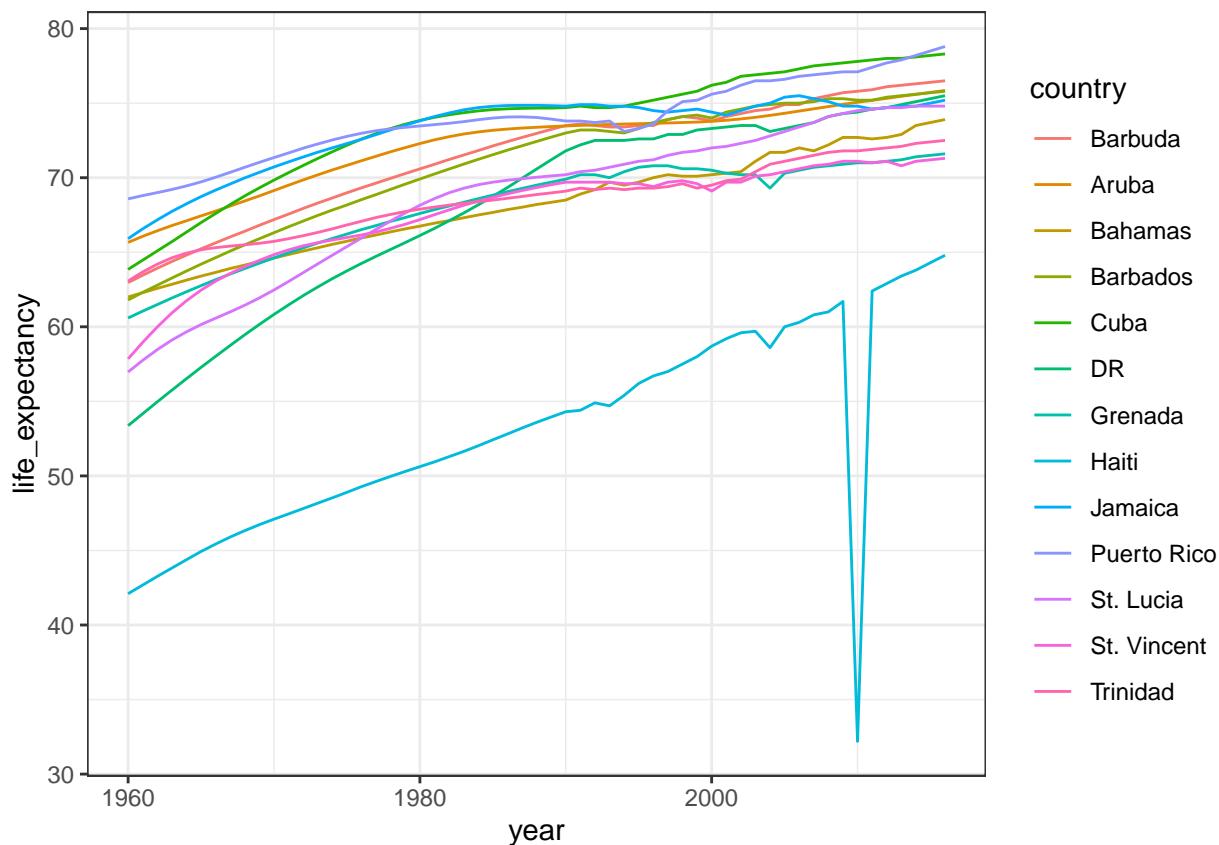
```
gapminder %>%
  filter(region=="Caribbean") %>%
  filter(str_length(country) >= 12) %>%
  distinct(country)
```

```
##                                     country
## 1      Antigua and Barbuda
## 2 Dominican Republic
```

```
## 3 St. Vincent and the Grenadines
## 4 Trinidad and Tobago
```

Así, nuestra tarea con la función “recode()” se escribe como sigue:

```
gapminder %>%
  filter(region == "Caribbean") %>%
  mutate(country = recode(country,
    'Antigua and Barbuda' = "Barbuda",
    'Dominican Republic' = "DR",
    'St. Vincent and the Grenadines' = "St. Vincent",
    'Trinidad and Tobago' = "Trinidad")) %>%
  ggplot(aes(year, life_expectancy, color = country)) +
  geom_line()
```



Otras funciones similares son “recode\_factor()” y “fct\_recode()” en el paqueteforcats incluido en tidyverse.

### 6.3.12. Assessment 12

Using the gapminder data, you want to recode countries longer than 12 letters in the region “Middle Africa” to their abbreviations in a new column, “country\_short”. Which code would accomplish this?

```
dat <- gapminder %>%
  filter(region == "Middle Africa") %>%
  mutate(country_short = recode(country,
    "Central African Republic" = "CAR",
    "Congo, Dem. Rep." = "DRC",
```

```
"Equatorial Guinea" = "Eq. Guinea"))
```

Import raw Brexit referendum polling data from Wikipedia:

```
library(rvest)
library(tidyverse)
library(stringr)

url <- "https://en.wikipedia.org/w/index.php?title=Opinion_polling_for_the_United_Kingdom_European_Union_Brexit_referendum&oldid=910000000"
tab <- read_html(url) %>%
  html_nodes("table")

polls <- tab[[6]] %>%
  html_table(fill = TRUE)
```

Some rows in this table do not contain polls. You can identify these by the lack of the percent sign (%) in the Remain column. Update polls by changing the column names to c("dates", "remain", "leave", "undecided", "lead", "samplesize", "pollster", "poll\_type", "notes") and only keeping rows that have a percent sign (%) in the remain column. How many rows remain in the polls data frame?

```
names(polls) <- c("dates", "remain", "leave", "undecided", "lead", "samplesize", "pollster", "poll_type", "notes")

polls <- polls[str_detect(polls$remain, "%"), -9]

nrow(polls)
```

```
## [1] 129
```

The remain and leave columns are both given in the format “48.1 %”: percentages out of 100 % with a percent symbol. Which of these commands converts the remain vector to a proportion between 0 and 1?

```
as.numeric(str_replace(polls$remain, "%", ""))/100

## [1] 0.481 0.520 0.550 0.510 0.490 0.440 0.540 0.480 0.410 0.450 0.420 0.530
## [13] 0.450 0.440 0.440 0.420 0.420 0.370 0.460 0.430 0.390 0.450 0.440 0.460
## [25] 0.400 0.480 0.530 0.420 0.440 0.450 0.430 0.430 0.480 0.410 0.430 0.400
## [37] 0.410 0.420 0.440 0.510 0.440 0.440 0.410 0.410 0.450 0.550 0.440 0.440
## [49] 0.520 0.550 0.470 0.430 0.550 0.380 0.360 0.380 0.440 0.420 0.440 0.430
## [61] 0.420 0.490 0.390 0.410 0.450 0.430 0.440 0.510 0.510 0.490 0.480 0.430
## [73] 0.530 0.380 0.400 0.390 0.350 0.450 0.420 0.400 0.390 0.440 0.510 0.390
## [85] 0.350 0.410 0.510 0.450 0.490 0.400 0.480 0.410 0.460 0.470 0.430 0.450
## [97] 0.480 0.490 0.400 0.400 0.400 0.390 0.410 0.390 0.480 0.480 0.370 0.380
## [109] 0.420 0.510 0.450 0.400 0.540 0.360 0.430 0.490 0.410 0.360 0.420 0.380
## [121] 0.550 0.440 0.540 0.410 0.520 0.420 0.380 0.420 0.440
```

```
parse_number(polls$remain)/100
```

```
## [1] 0.481 0.520 0.550 0.510 0.490 0.440 0.540 0.480 0.410 0.450 0.420 0.530
## [13] 0.450 0.440 0.440 0.420 0.420 0.370 0.460 0.430 0.390 0.450 0.440 0.460
## [25] 0.400 0.480 0.530 0.420 0.440 0.450 0.430 0.430 0.480 0.410 0.430 0.400
## [37] 0.410 0.420 0.440 0.510 0.440 0.440 0.410 0.410 0.450 0.550 0.440 0.440
## [49] 0.520 0.550 0.470 0.430 0.550 0.380 0.360 0.380 0.440 0.420 0.440 0.430
## [61] 0.420 0.490 0.390 0.410 0.450 0.430 0.440 0.510 0.510 0.490 0.480 0.430
## [73] 0.530 0.380 0.400 0.390 0.350 0.450 0.420 0.400 0.390 0.440 0.510 0.390
## [85] 0.350 0.410 0.510 0.450 0.490 0.400 0.480 0.410 0.460 0.470 0.430 0.450
```

```
## [97] 0.480 0.490 0.400 0.400 0.400 0.390 0.410 0.390 0.480 0.480 0.370 0.380
## [109] 0.420 0.510 0.450 0.400 0.540 0.360 0.430 0.490 0.410 0.360 0.420 0.380
## [121] 0.550 0.440 0.540 0.410 0.520 0.420 0.380 0.420 0.440
```

\*\*The undecided column has some “N/A” values. These “N/A”s are only present when the remain and leave columns total 100 %, so they should actually be zeros. Use a function from stringr to convert “N/A” in the undecided column to 0. The format of your command should be function\_name(polls\$undecided, “arg1”, “arg2”).

```
polls$undecided <- str_replace_all(polls$undecided, "N/A", "0")
```

```
polls$undecided
```

```
## [1] "0"    "0"    "0"    "0"    "1%"   "9%"   "0"    "11%"  "16%"  "11%"  "13%"  "2%"
## [13] "13%"  "9%"   "12%"  "9%"   "13%"  "16%"  "11%"  "3%"   "15%"  "5%"   "7%"   "9%"
## [25] "13%"  "3%"   "0"    "11%"  "13%"  "0"    "11%"  "9%"   "5%"   "11%"  "16%"  "16%"
## [37] "13%"  "15%"  "9%"   "3%"   "12%"  "18%"  "13%"  "16%"  "10%"  "3%"   "14%"  "12%"
## [49] "7%"   "5%"   "14%"  "10%"  "5%"   "21%"  "22%"  "16%"  "11%"  "13%"  "11%"  "11%"
## [61] "14%"  "0"    "26%"  "13%"  "17%"  "13%"  "10%"  "6%"   "9%"   "8%"   "11%"  "13%"
## [73] "6%"   "28%"  "16%"  "17%"  "30%"  "17%"  "12%"  "16%"  "18%"  "13%"  "5%"   "18%"
## [85] "30%"  "14%"  "0"    "12%"  "10%"  "19%"  "11%"  "17%"  "19%"  "4%"   "16%"  "16%"
## [97] "7%"   "15%"  "19%"  "18%"  "19%"  "19%"  "18%"  "18%"  "15%"  "0"    "25%"  "25%"
## [109] "17%"  "10%"  "23%"  "19%"  "10%"  "25%"  "18%"  "10%"  "17%"  "19%"  "19%"  "20%"
## [121] "9%"   "14%"  "10%"  "18%"  "0"    "17%"  "22%"  "12%"  "18%"
```

The dates column contains the range of dates over which the poll was conducted. The format is “8-10 Jan” where the poll had a start date of 2016-01-08 and end date of 2016-01-10. Some polls go across month boundaries (16 May-12 June). The end date of the poll will always be one or two digits, followed by a space, followed by the month as one or more letters (either capital or lowercase). In these data, all month abbreviations or names have 3, 4 or 5 letters. Write a regular expression to extract the end day and month from dates. Insert it into the skeleton code below:

```
temp <- str_extract_all(polls$dates, "\\\d{1,2}\\s[a-zA-Z]+")
end_date <- sapply(temp, function(x) x[length(x)])
```

## 6.4. Fechas y tiempos

A lo largo del curso hemos descrito tres tipos de vectores: numéricos, de caracteres y lógicos. En la ciencia de datos, no obstante, es común encontrar datos en formato de fecha.

Dada la complejidad e incomodidad de manejar tiempos y fechas en formato de texto, R define un tipo de datos solo para ellos. Recuérdese que vimos un ejemplo en los datos de encuestas:

```
library(dslabs)

data("polls_us_election_2016")

polls_us_election_2016$startdate %>%
  head

## [1] "2016-11-03" "2016-11-01" "2016-11-02" "2016-11-04" "2016-11-03"
## [6] "2016-11-03"
```

Las cuales lucen como cadenas, pero no lo son, obsérvese la clase:

```
class(polls_us_election_2016$startdate)

## [1] "Date"
```

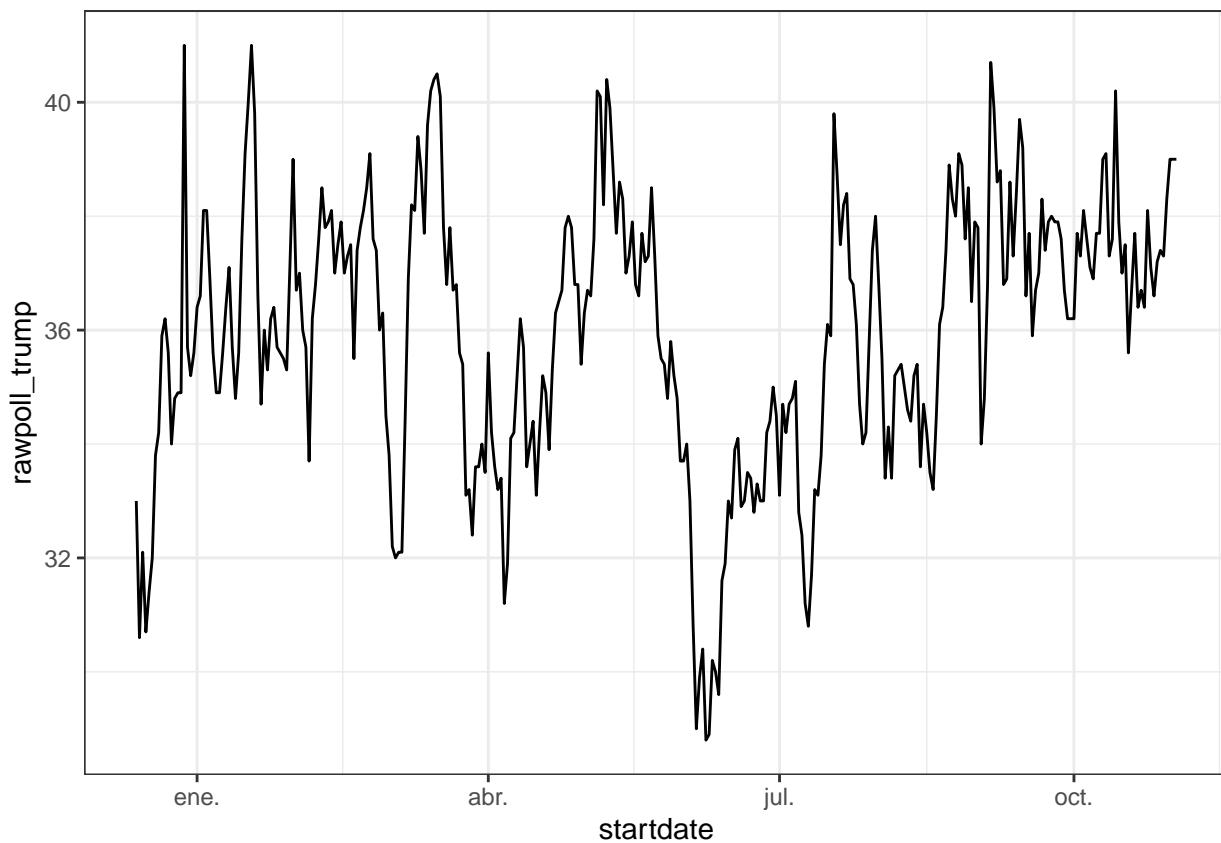
Si las convertimos a números, se vuelven al formato “epoch”, el cual genera un número único para cada día, pero no es intuitivo ni práctico para interpretar:

```
as.numeric(polls_us_election_2016$startdate) %>%
  head

## [1] 17108 17106 17107 17109 17108 17108
```

**Nota:** ggplot está al tanto del formato de fecha:

```
polls_us_election_2016 %>%
  filter(pollster == "Ipsos" & state == "U.S.") %>%
  ggplot(aes(startdate, rawpoll_trump)) +
  geom_line()
```



Dado que las fechas son tan comunes, tidyverse incluye un paquete, lubridate, que permite trabajar y lidiar con fechas.

```
library(lubridate)
```

Tomaremos una muestra de fechas para ejemplificar su utilidad:

```
set.seed(2)

dates <- sample(polls_us_election_2016$startdate, 10) %>%
  sort

dates

## [1] "2015-11-09" "2016-02-15" "2016-07-09" "2016-08-12" "2016-08-17"
## [6] "2016-09-18" "2016-10-04" "2016-10-10" "2016-10-18" "2016-10-25"
```

Podemos extraer los valores del día, mes y año mediante las siguientes funciones:

```
data.frame(date = days(dates),
           month = month(dates),
           day = day(dates),
           year = year(dates))

##          date month day year
## 1 16748d 0H 0M 0S    11    9 2015
## 2 16846d 0H 0M 0S     2   15 2016
## 3 16991d 0H 0M 0S     7    9 2016
## 4 17025d 0H 0M 0S     8   12 2016
```

```
## 5 17030d OH OM OS     8 17 2016
## 6 17062d OH OM OS     9 18 2016
## 7 17078d OH OM OS    10  4 2016
## 8 17084d OH OM OS    10 10 2016
## 9 17092d OH OM OS    10 18 2016
## 10 17099d OH OM OS   10 25 2016
```

También pueden extraerse las etiquetas de los meses:

```
month(dates, label = TRUE)

## [1] nov feb jul ago ago sep oct oct oct
## 12 Levels: ene < feb < mar < abr < may < jun < jul < ago < sep < ... < dic
```

Otra función útil son los *parsers*, con los que podemos convertir una cadena de texto a fecha. A continuación se presentan varios ejemplos:

```
x <- c(20090101, "2009-01-02", "2009 01 03", "2009-1-4",
      "2009-1, 5", "Created on 2009 1 6", "200901 !!! 07")
```

```
ymd(x)

## [1] "2009-01-01" "2009-01-02" "2009-01-03" "2009-01-04" "2009-01-05"
## [6] "2009-01-06" "2009-01-07"
```

Otra complicación con las fechas es que puede ser escrita en diferentes formatos. El estándar y preferido es “YYYY-MM-DD” (ISO 8601). Nótese que la función “ymd()” convierte y ordena las fechas de esta forma.

Los *parsers* más comunes son:

- ymd(x)
- mdy(x)
- ydm(x)
- myd(x)
- dmy(x)
- dym(x)

Lubridate también es útil con tiempos:

```
now()

## [1] "2021-07-18 19:52:06 CDT"

now("GMT")

## [1] "2021-07-19 00:52:06 GMT"
```

También se puede extraer la hora, minutos y segundos:

```
now() %>%
  hour()

## [1] 19

now() %>%
  minute()

## [1] 52
```

```
now() %>%  
  second()
```

```
## [1] 6.19
```

También tiene funciones para convertir cadenas en tiempo:

```
x <- c("12:34:56")
```

```
hms(x)
```

```
## [1] "12H 34M 56S"
```

Finalmente, podemos convertir simultáneamente hora y fecha:

```
x <- "Nov/2/2012 12:34:56"
```

```
mdy_hms(x)
```

```
## [1] "2012-11-02 12:34:56 UTC"
```

## 6.5. Minería de texto

El paquete tidytext nos ayuda a convertir texto en forma libre en una tabla *tidy*. Con “unnest\_tokens()” podemos extraer palabras individuales y otras partes pertinente de un texto.

En los siguientes ejemplos, los datos están compuestos por texto. Nuestra tarea es extraer información pertinente de este

### 6.5.1. Caso de estudio: Tweets de Trump

```
library(tidyverse)
library(ggplot2)
library(lubridate)
library(tidyr)
library(scales)
set.seed(1)

data("trump_tweets")

head(trump_tweets)

##           source      id_str
## 1 Twitter Web Client 6971079756
## 2 Twitter Web Client 6312794445
## 3 Twitter Web Client 6090839867
## 4 Twitter Web Client 5775731054
## 5 Twitter Web Client 5364614040
## 6 Twitter Web Client 5203117820
##
## 1      From Donald Trump: Wishing everyone a wonderful holiday & a happy, healthy, prosperous New Y
## 2 Trump International Tower in Chicago ranked 6th tallest building in world by Council on Tall Buildi
## 3                                         Wishing you and yours a
## 4                               Donald Trump Partners with TV1 on New Reality Series Entitled, Omarosa's Ulti
## 5                               --Work has begun, ahead of schedule, to build the greatest golf course in t
## 6 --From Donald Trump: "Ivanka and Jared's wedding was spectacular, and they make a beau
##   created_at retweet_count in_reply_to_user_id_str favorite_count
## 1 2009-12-23 12:38:18          28             <NA>            12
## 2 2009-12-03 14:39:09          33             <NA>              6
## 3 2009-11-26 14:55:38          13             <NA>            11
## 4 2009-11-16 16:06:10          5              <NA>              3
## 5 2009-11-02 09:57:56          7              <NA>              6
## 6 2009-10-27 10:31:48          4              <NA>              5
##   is_retweet
## 1 FALSE
## 2 FALSE
## 3 FALSE
## 4 FALSE
## 5 FALSE
## 6 FALSE

names(trump_tweets)

## [1] "source"                  "id_str"
## [3] "text"                     "created_at"
## [5] "retweet_count"            "in_reply_to_user_id_str"
## [7] "favorite_count"           "is_retweet"
```

Los tweets son representados por una variable de texto:

```
trump_tweets %>%
  select(text) %>%
  head

##
## 1      From Donald Trump: Wishing everyone a wonderful holiday & a happy, healthy, prosperous New Y
## 2 Trump International Tower in Chicago ranked 6th tallest building in world by Council on Tall Buildi
## 3
## 4      Donald Trump Partners with TV1 on New Reality Series Entitled, Omarosa's Ulti
## 5          --Work has begun, ahead of schedule, to build the greatest golf course in L
## 6      --From Donald Trump: "Ivanka and Jared's wedding was spectacular, and they make a beau
```

Y la variable fuente nos dice el dispositivo en el que se escribió u publicó cada tweet:

```
trump_tweets %>%
  count(source) %>%
  arrange(desc(n))

##           source     n
## 1 Twitter Web Client 10718
## 2 Twitter for Android 4652
## 3 Twitter for iPhone 3962
## 4 TweetDeck 468
## 5 TwitLonger Beta 288
## 6 Instagram 133
## 7 Media Studio 114
## 8 Facebook 104
## 9 Twitter Ads 96
## 10 Twitter for BlackBerry 78
## 11 Mobile Web (M5) 54
## 12 Twitter for iPad 39
## 13 Twitlonger 22
## 14 Twitter QandA 10
## 15 Vine - Make a Scene 10
## 16 Periscope 7
## 17 Neatly For BlackBerry 10 4
## 18 Twitter for Websites 1
## 19 Twitter Mirror for iPad 1
```

Podemos usar “extract()” para remover “Twitter” y filtrar los retweets:

```
trump_tweets %>%
  extract(source, "source", "Twitter for (.*)") %>%
  count(source)

##           source     n
## 1     Android 4652
## 2 BlackBerry 78
## 3      iPad 39
## 4     iPhone 3962
## 5    Websites 1
## 6     <NA> 12029
```

Nos interesa qué ocurrió durante la campaña, por lo que nos enfocaremos en lo que fue tweeteado entre el día que Trump anunció su campaña y el día de la elección:

```

campaign_tweets <- trump_tweets %>%
  extract(source, "source", "Twitter for (.*)") %>%
  filter(source %in% c("Android", "iPhone") &
         created_at >= ymd("2015-06-17") &
         created_at < ymd("2016-11-08")) %>%
  filter(!is_retweet) %>%
  arrange(created_at)

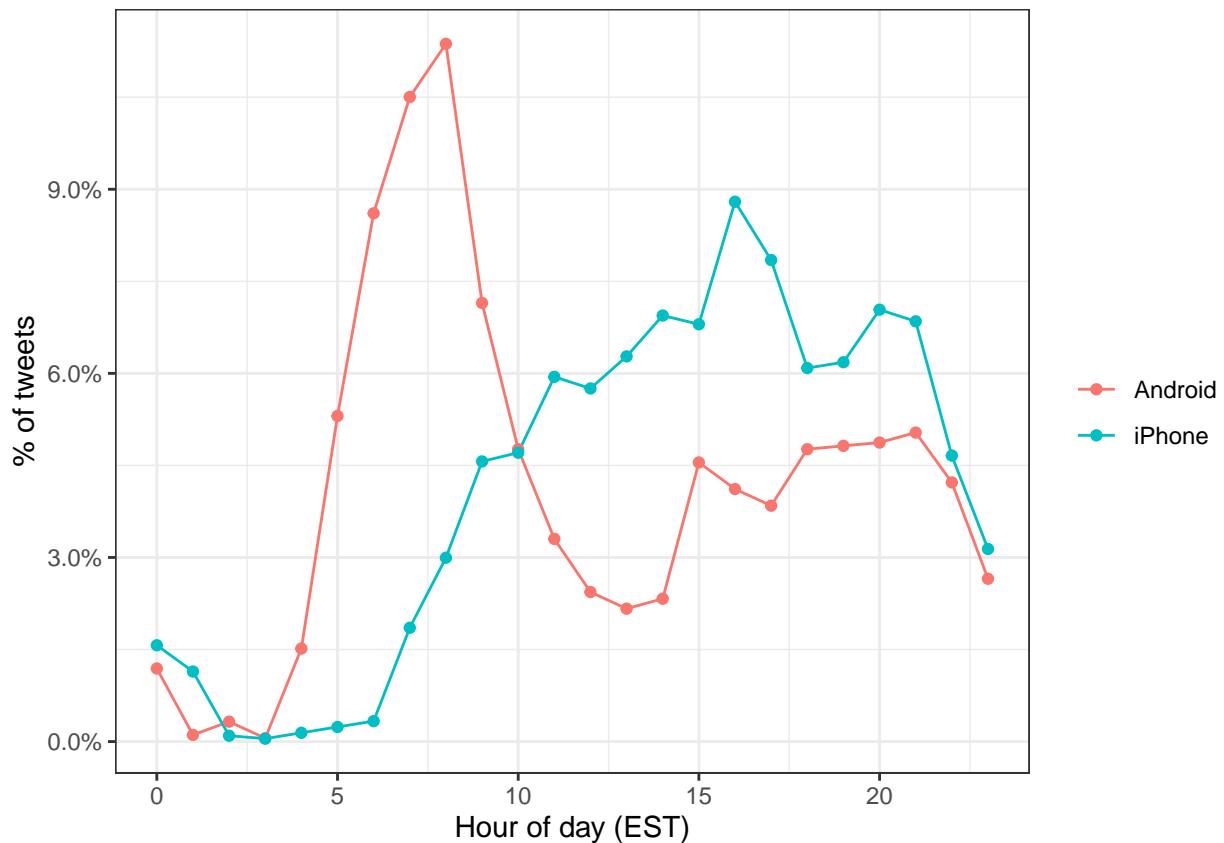
```

Ahora podemos observar la posibilidad de que dos diferentes grupos de tweets hayan sido tweeteados desde diferentes dispositivos:

```

ds_theme_set()
campaign_tweets %>%
  mutate(hour = hour(with_tz(created_at, "EST"))) %>%
  count(source, hour) %>%
  group_by(source) %>%
  mutate(percent = n / sum(n)) %>%
  ungroup %>%
  ggplot(aes(hour, percent, color = source)) +
  geom_line() +
  geom_point() +
  scale_y_continuous(labels = percent_format()) +
  labs(x = "Hour of day (EST)",
       y = "% of tweets",
       color = "")

```



Como mencionamos, el paquete tidytext nos permite convertir texto libre en una tabla *tidy*. La principal

función para ello es “`unnest_tokens()`”. Un *token* se refiere a una unidad que estamos considerando como punto de dato. Los más comunes son palabras, aunque también pueden ser caracteres simples, ngramas, oraciones, líneas o un patrón definido en Regex. Las funciones tomarán un vector de cadenas y extraerán los *tokens* de tal forma que cada uno esté en un renglón en una nueva tabla:

```
library(tidytext)

## Warning: package 'tidytext' was built under R version 4.0.5
example <- tibble(line = c(1, 2, 3, 4),
                  text = c("Roses are red,", "Violets are blue,", "Sugar is sweet,", "And so are you."))
example

## # A tibble: 4 x 2
##   line    text
##   <dbl> <chr>
## 1     1 Roses are red,
## 2     2 Violets are blue,
## 3     3 Sugar is sweet,
## 4     4 And so are you.

example %>% unnest_tokens(word, text)

## # A tibble: 13 x 2
##   line word
##   <dbl> <chr>
## 1     1 roses
## 2     1 are
## 3     1 red
## 4     2 violets
## 5     2 are
## 6     2 blue
## 7     3 sugar
## 8     3 is
## 9     3 sweet
## 10    4 and
## 11    4 so
## 12    4 are
## 13    4 you
```

Considérese un ejemplo con el tweet número 3008:

```
i <- 3008
campaign_tweets$text[i]

## [1] "Great to be back in Iowa! #TBT with @JerryJrFalwell joining me in Davenport- this past winter. #"

campaign_tweets[i,] %>%
  unnest_tokens(word, text) %>%
  select(word)

##          word
## 1         great
## 2           to
## 3           be
## 4          back
## 5           in
```

```

## 6          iowa
## 7          tbt
## 8          with
## 9 @jerryjrfalwell
## 10         joining
## 11         me
## 12         in
## 13         davenport
## 14         this
## 15         past
## 16         winter
## 17         #maga
## 18         https
## 19         t.co
## 20         a5if0qhnic

```

Tómese en cuenta que la función intenta convertir *tokens* en palabras y elimina los caracteres importantes para Twitter, como # y @. Un *token* en Twitter no es lo mismo que en inglés normal. Por esta razón, en lugar de usar el *token* predeterminado, palabras, definimos una expresión regular que captura el carácter de Twitter. El patrón parece complejo, pero todo lo que estamos definiendo es un patrón que comienza con @, # o ninguno y es seguido por cualquier combinación de letras o dígitos:

```
pattern <- "([^\w\#\@\!']|'(?!\w\#\@\!'))"
```

Ahora podemos utilizar “`unnest_tokens()`” con la opción `regex` para extraer apropiadamente los hashtags y menciones:

```

campaign_tweets[i,] %>%
  unnest_tokens(word, text, token = "regex", pattern = pattern) %>%
  select(word)

```

```

##             word
## 1          great
## 2            to
## 3            be
## 4           back
## 5            in
## 6          iowa
## 7          #tbt
## 8          with
## 9 @jerryjrfalwell
## 10         joining
## 11         me
## 12         in
## 13         davenport
## 14         this
## 15         past
## 16         winter
## 17         #maga
## 18         https
## 19           t
## 20           co
## 21         a5if0qhnic

```

Otro ajuste pequeño es remover los links a imágenes:

```

campaign_tweets[i,] %>%
  mutate(text = str_replace_all(text, "https://t.co/[A-Za-z\\d]+|&", ""))
  unnest_tokens(word, text, token = "regex", pattern = pattern) %>%
  select(word)

##          word
## 1        great
## 2         to
## 3         be
## 4       back
## 5         in
## 6       iowa
## 7       #tbt
## 8       with
## 9 @jerryjrfalwell
## 10      joining
## 11        me
## 12        in
## 13    davenport
## 14        this
## 15        past
## 16      winter
## 17      #maga

```

Estamos listos para extraer las palabras de todos los tweets:

```

tweet_words <- campaign_tweets %>%
  mutate(text = str_replace_all(text, "https://t.co/[A-Za-z\\d]+|&", ""))
  unnest_tokens(word, text, token = "regex", pattern = pattern)

```

Y podemos responder, ¿cuáles son las palabras más comúnmente usadas?

```

words <- tweet_words %>%
  count(word) %>%
  arrange(desc(n))

head(words)

##   word     n
## 1 the 2335
## 2 to 1413
## 3 and 1246
## 4 a 1210
## 5 in 1189
## 6 i 1161

```

Vemos que existen, obviamente, muchas palabras muy usadas pero poco informativas. El paquete tidytext tiene una base de datos de palabras comúnmente usadas (stop\_words):

```

tweet_words <- campaign_tweets %>%
  mutate(text = str_replace_all(text, "https://t.co/[A-Za-z\\d]+|&", ""))
  unnest_tokens(word, text, token = "regex", pattern = pattern) %>%
  filter(!word %in% stop_words$word )

tweet_words %>%
  count(word) %>%
  top_n(10, n) %>%

```

```

mutate(word = reorder(word, n)) %>%
arrange(desc(n))

##          word     n
## 1      #trump2016 415
## 2        hillary 407
## 3       people 303
## 4 #makeamericagreatagain 296
## 5       america 254
## 6      clinton 239
## 7        poll 220
## 8      crooked 206
## 9       trump 199
## 10      cruz 162

```

Un poco de exploración de las palabras resultantes revela un par de características no deseadas en nuestros *tokens*. Primero, algunos de nuestros *tokens* son solo números (años, por ejemplo). Queremos eliminarlos y podemos encontrarlos usando la expresión regular `^ d + $`. En segundo lugar, algunos de nuestros *tokens* provienen de una cita y comienzan con `'`. Queremos eliminar el `'`cuando esté al comienzo de una palabra, por lo que usaremos `"str_replace ()"`. Agregamos estas dos líneas al código anterior para generar nuestra tabla final:

```

tweet_words <- campaign_tweets %>%
  mutate(text = str_replace_all(text, "https://t.co/[A-Za-z\\d]+|&", ""))
unnest_tokens(word, text, token = "regex", pattern = pattern) %>%
  filter(!word %in% stop_words$word &
         !str_detect(word, "^\\d+$")) %>%
  mutate(word = str_replace(word, "'", ""))

```

Para cada palabra queremos saber si es más probable que provenga de un tweet de Android o de un tweet de iPhone. Anteriormente introdujimos la razón de probabilidades, una estadística útil para cuantificar estas diferencias. Para cada dispositivo y una palabra dada, llamémoslo `y`, calculemos las probabilidades o la proporción entre la proporción de palabras que son `y` y no `y` y calculamos la proporción de esas probabilidades. Aquí tendremos muchas proporciones que son 0, por lo que usamos la corrección de 0.5:

```

android_iphone_or <- tweet_words %>%
  count(word, source) %>%
  spread(source, n, fill = 0) %>%
  mutate(or = (Android + 0.5) / (sum(Android) - Android + 0.5) /
    (iPhone + 0.5) / (sum(iPhone) - iPhone + 0.5))

head(android_iphone_or %>% arrange(desc(or)))

##          word Android iPhone     or
## 1      mails     22      0 39.3
## 2      poor      13      0 23.6
## 3   poorly      12      0 21.8
## 4 @cbsnews     11      0 20.1
## 5    bosses      11      0 20.1
## 6 turnberry     11      0 20.1

head(android_iphone_or %>% arrange(or))

##          word Android iPhone     or
## 1 #makeamericagreatagain      0 296 0.00144

```

```
## 2      #americafirst      0    71 0.00607
## 3      #draintheswamp     0    63 0.00683
## 4      #trump2016         3   412 0.00718
## 5      #votetrump        0    56 0.00769
## 6      join              1   157 0.00821
```

Dado que varias de estas palabras son palabras de baja frecuencia en general, podemos imponer un filtro basado en la frecuencia total como este:

```
android_iphone_or %>% filter(Android+iPhone > 100) %>%
  arrange(desc(or))
```

	word	Android	iPhone	or
## 1	@cnn	104	18	4.95144
## 2	bad	104	26	3.45466
## 3	crooked	157	49	2.79222
## 4	ted	85	28	2.62463
## 5	interviewed	76	25	2.62370
## 6	media	77	26	2.55767
## 7	cruz	116	46	2.19335
## 8	hillary	290	119	2.14061
## 9	win	74	30	2.13519
## 10	president	84	35	2.08126
## 11	debate	80	35	1.98224
## 12	@foxnews	77	37	1.80598
## 13	people	195	109	1.56373
## 14	country	67	38	1.53101
## 15	night	71	41	1.50456
## 16	time	86	54	1.38603
## 17	amazing	66	42	1.36589
## 18	crowd	61	43	1.23367
## 19	enjoy	69	51	1.17749
## 20	clinton	136	108	1.09782
## 21	trump	112	92	1.06092
## 22	poll	117	103	0.98982
## 23	iowa	62	65	0.83135
## 24	tonight	71	84	0.73662
## 25	america	114	141	0.70343
## 26	vote	46	67	0.59950
## 27	tomorrow	25	101	0.21780
## 28	join	1	157	0.00821
## 29	#trump2016	3	412	0.00718
## 30	#makeamericagreatagain	0	296	0.00144

```
android_iphone_or %>% filter(Android+iPhone > 100) %>%
  arrange(or)
```

	word	Android	iPhone	or
## 1	#makeamericagreatagain	0	296	0.00144
## 2	#trump2016	3	412	0.00718
## 3	join	1	157	0.00821
## 4	tomorrow	25	101	0.21780
## 5	vote	46	67	0.59950
## 6	america	114	141	0.70343
## 7	tonight	71	84	0.73662
## 8	iowa	62	65	0.83135

```

## 9          poll      117     103 0.98982
## 10         trump     112      92 1.06092
## 11        clinton    136     108 1.09782
## 12         enjoy      69      51 1.17749
## 13        crowd       61      43 1.23367
## 14        amazing     66      42 1.36589
## 15         time       86      54 1.38603
## 16        night       71      41 1.50456
## 17        country     67      38 1.53101
## 18        people     195     109 1.56373
## 19        @foxnews    77      37 1.80598
## 20        debate      80      35 1.98224
## 21        president   84      35 2.08126
## 22         win        74      30 2.13519
## 23        hillary     290     119 2.14061
## 24         cruz       116     46 2.19335
## 25         media       77      26 2.55767
## 26        interviewed 76      25 2.62370
## 27         ted         85      28 2.62463
## 28        crooked     157     49 2.79222
## 29         bad        104     26 3.45466
## 30        @cnn        104     18 4.95144

```

En el análisis de sentimientos, asignamos una palabra a uno o más “sentimientos”. Aunque este enfoque pasará por alto los sentimientos dependientes del contexto, como el sarcasmo, cuando se realiza con un gran número de palabras, los resúmenes pueden proporcionar información.

El primer paso en el análisis de sentimientos es asignar un sentimiento a cada palabra. El paquete tidytext incluye varios mapas o léxicos en los sentimientos del objeto:

Hay varios léxicos en el paquete tidytext que dan diferentes sentimientos. Por ejemplo, el léxico de Bing divide las palabras en positivas y negativas. Podemos ver esto usando la función tidytext get\_sentiments ():

```

get_sentiments("bing")

## # A tibble: 6,786 x 2
##   word      sentiment
##   <chr>    <chr>
## 1 2-faces  negative
## 2 abnormal negative
## 3 abolish  negative
## 4 abominable negative
## 5 abominably negative
## 6 abominate negative
## 7 abomination negative
## 8 abort    negative
## 9 aborted   negative
## 10 aborts   negative
## # ... with 6,776 more rows

```

Los léxicos loughran y nrc proporcionan varios sentimientos diferentes:

```

get_sentiments("loughran") %>%
  count(sentiment)

## # A tibble: 6 x 2
##   sentiment     n
##   <chr>    <int>
## 1 positive    100
## 2 negative    99
## 3 neutral     99
## 4 mixed       99
## 5 sarcasm     99
## 6 irony       99

```

```

## <chr>      <int>
## 1 constraining 184
## 2 litigious   904
## 3 negative    2355
## 4 positive     354
## 5 superfluous  56
## 6 uncertainty  297
get_sentiments("nrc") %>%
  count(sentiment)

## # A tibble: 10 x 2
##       sentiment     n
##       <chr>      <int>
## 1 anger          1247
## 2 anticipation   839
## 3 disgust         1058
## 4 fear            1476
## 5 joy              689
## 6 negative        3324
## 7 positive         2312
## 8 sadness          1191
## 9 surprise          534
## 10 trust           1231

```

Para el análisis aquí estamos interesados en explorar los diferentes sentimientos de cada tweet, por lo que usaremos el léxico nrc:

```

nrc <- get_sentiments("nrc") %>%
  select(word, sentiment)

```

Podemos combinar las palabras y los sentimientos usando “inner\_join ()”, que solo mantendrá las palabras asociadas con un sentimiento. Aquí hay 10 palabras al azar extraídas de los tweets:

```

tweet_words %>% inner_join(nrc, by = "word") %>%
  select(source, word, sentiment) %>% sample_n(10)

```

```

##       source     word     sentiment
## 1 iPhone      poll      trust
## 2 iPhone     special      joy
## 3 Android     vote     negative
## 4 iPhone      vote     surprise
## 5 Android    incompetent      sadness
## 6 iPhone      plan    anticipation
## 7 iPhone      join      positive
## 8 iPhone     money     surprise
## 9 iPhone      honor      trust
## 10 Android    vote    anticipation

```

Ahora estamos listos para realizar un análisis cuantitativo comparando Android y iPhone a partir de los sentimientos de los tweets publicados desde cada dispositivo. Aquí podríamos realizar un análisis de tweet por tweet, asignando un sentimiento a cada tweet. Sin embargo, esto es algo complejo ya que cada tweet tendrá varios sentimientos adjuntos, uno por cada palabra que aparece en el léxico. A título ilustrativo, realizaremos un análisis mucho más sencillo: contaremos y compararemos las frecuencias de cada sentimiento que aparece para cada dispositivo.

```

sentiment_counts <- tweet_words %>%
  left_join(nrc, by = "word") %>%

```

```

count(source, sentiment) %>%
  spread(source, n) %>%
  mutate(sentiment = replace_na(sentiment, replace = "none"))
sentiment_counts

##      sentiment Android iPhone
## 1      anger     965   527
## 2 anticipation    915   710
## 3      disgust    641   318
## 4      fear      802   486
## 5      joy       698   540
## 6 negative    1668   935
## 7 positive    1834  1497
## 8 sadness      907   514
## 9 surprise     530   365
## 10 trust     1253  1010
## 11 none     11523 10739

```

Dado que se usaron más palabras en Android que en el iPhone:

```
tweet_words %>% group_by(source) %>% summarize(n = n())
```

```

## # A tibble: 2 x 2
##   source     n
##   <chr>   <int>
## 1 Android 15829
## 2 iPhone  13802

```

Para cada sentimiento podemos calcular las probabilidades de estar en el dispositivo: proporción de palabras con sentimiento versus proporción de palabras sin él y luego calcular la razón de probabilidades comparando los dos dispositivos:

```

sentiment_counts %>%
  mutate(Android = Android / (sum(Android) - Android) ,
         iPhone = iPhone / (sum(iPhone) - iPhone),
         or = Android/iPhone) %>%
  arrange(desc(or))

##      sentiment Android iPhone     or
## 1      disgust  0.0304 0.0184 1.655
## 2      anger    0.0465 0.0308 1.509
## 3 negative   0.0831 0.0560 1.485
## 4      sadness  0.0435 0.0300 1.451
## 5      fear     0.0383 0.0283 1.352
## 6 surprise   0.0250 0.0211 1.183
## 7      joy      0.0332 0.0316 1.051
## 8 anticipation 0.0439 0.0419 1.048
## 9      trust    0.0612 0.0607 1.007
## 10     positive  0.0922 0.0927 0.994
## 11      none    1.1283 1.5559 0.725

```

Pero, ¿son estadísticamente significativos? ¿Cómo se compara esto si solo estamos asignando sentimientos al azar?

Para responder a esa pregunta, podemos calcular, para cada sentimiento, una razón de probabilidades y un intervalo de confianza. Agregaremos los dos valores que necesitamos para formar una tabla de dos por dos y la razón de probabilidades:

```

library(broom)

## Warning: package 'broom' was built under R version 4.0.5
log_or <- sentiment_counts %>%
  mutate( log_or = log( (Android / (sum(Android) - Android)) / (iPhone / (sum(iPhone) - iPhone))),
         se = sqrt( 1/Android + 1/(sum(Android) - Android) + 1/iPhone + 1/(sum(iPhone) - iPhone)),
         conf.low = log_or - qnorm(0.975)*se,
         conf.high = log_or + qnorm(0.975)*se) %>%
  arrange(desc(log_or))

log_or

## #> #>   sentiment  Android  iPhone  log_or      se conf.low conf.high
## #> 1      disgust     641     318  0.50398  0.0694    0.3680    0.6399
## #> 2       anger      965     527  0.41127  0.0551    0.3032    0.5193
## #> 3    negative     1668     935  0.39548  0.0422    0.3128    0.4781
## #> 4     sadness      907     514  0.37223  0.0562    0.2621    0.4823
## #> 5       fear       802     486  0.30182  0.0584    0.1874    0.4163
## #> 6    surprise      530     365  0.16801  0.0688    0.0332    0.3028
## #> 7       joy        698     540  0.04946  0.0582   -0.0647    0.1636
## #> 8 anticipation     915     710  0.04684  0.0511   -0.0533    0.1469
## #> 9       trust     1253    1010  0.00726  0.0436   -0.0781    0.0926
## #> 10      positive    1834    1497 -0.00624  0.0364   -0.0776    0.0651
## #> 11       none     11523   10739 -0.32139  0.0206   -0.3617   -0.2811

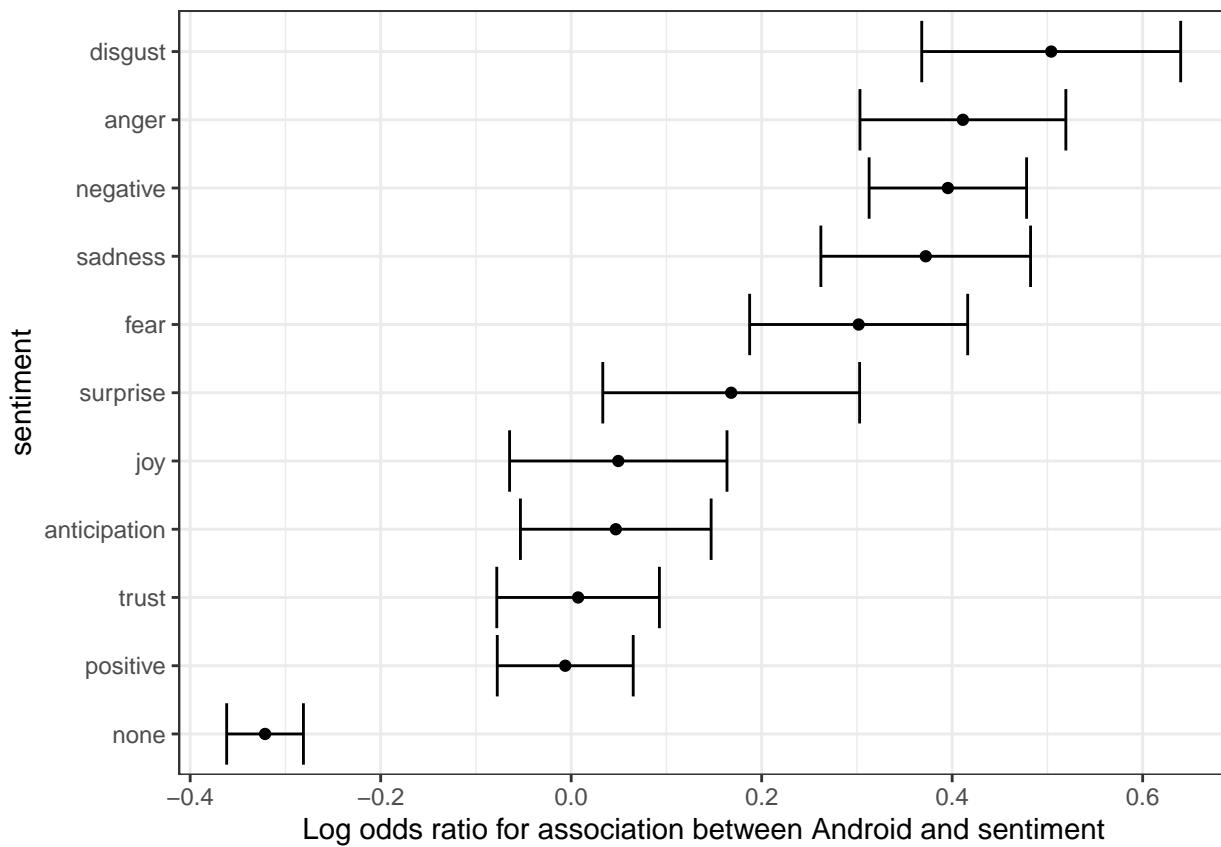
```

Una visualización gráfica muestra algunos sentimientos que están claramente sobrerepresentados:

```

log_or %>%
  mutate(sentiment = reorder(sentiment, log_or),) %>%
  ggplot(aes(x = sentiment, ymin = conf.low, ymax = conf.high)) +
  geom_errorbar() +
  geom_point(aes(sentiment, log_or)) +
  ylab("Log odds ratio for association between Android and sentiment") +
  coord_flip()

```



Si estamos interesados en explorar qué palabras específicas están impulsando estas diferencias, podemos volver a nuestro objeto android\_iphone\_or:

```
android_iphone_or %>% inner_join(nrc) %>%
  filter(sentiment == "disgust" & Android + iPhone > 10) %>%
  arrange(desc(or))
```

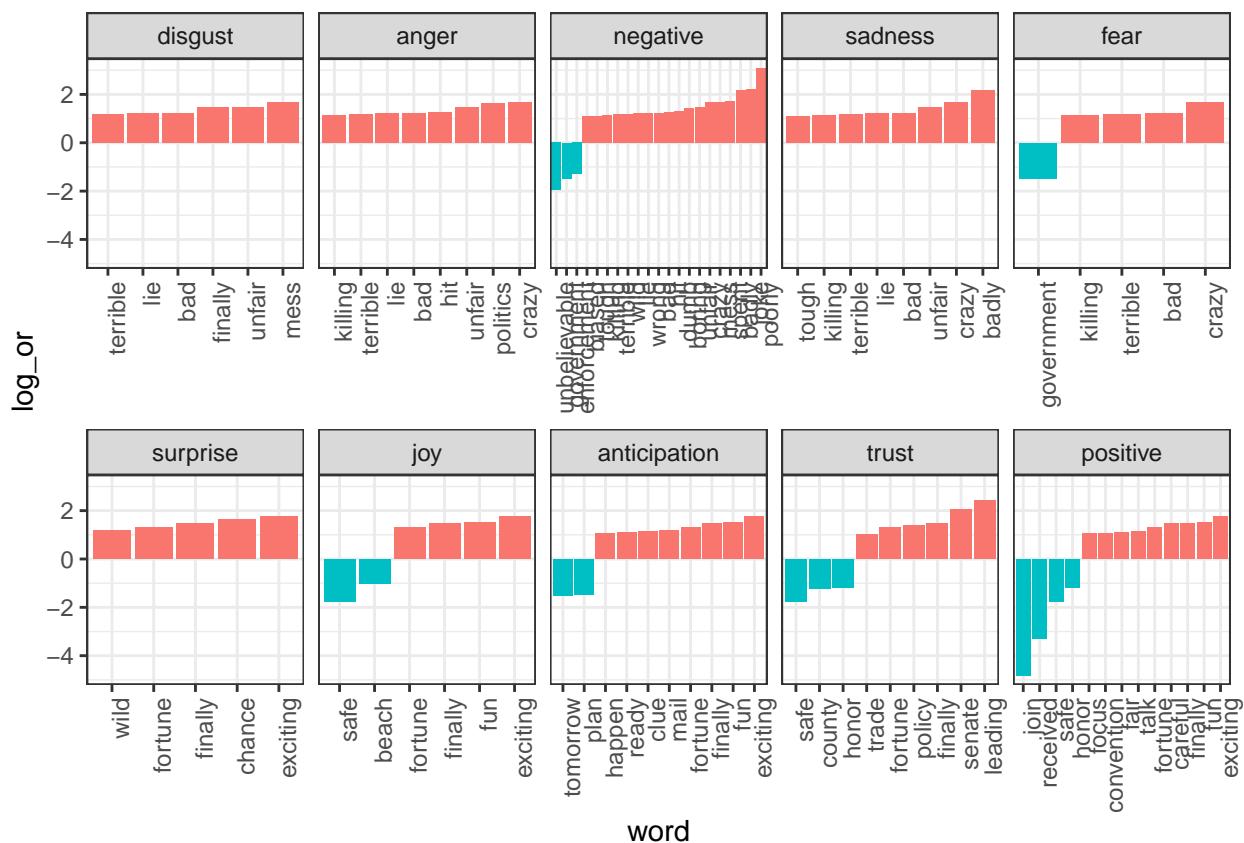
```
## Joining, by = "word"
##          word Android iPhone      or sentiment
## 1      mess      15     2 5.410  disgust
## 2 finally     12     2 4.362  disgust
## 3 unfair      12     2 4.362  disgust
## 4      bad     104    26 3.455  disgust
## 5      lie      13     3 3.365  disgust
## 6 terrible     31     8 3.236  disgust
## 7      lying     9     3 2.368  disgust
## 8      waste     12     5 1.982  disgust
## 9      phony     21     9 1.975  disgust
## 10     illegal    32    14 1.956  disgust
## 11      nasty     14     6 1.946  disgust
## 12     pathetic    11     5 1.824  disgust
## 13     horrible    14     7 1.686  disgust
## 14     disaster    21    11 1.631  disgust
## 15     winning    14     9 1.331  disgust
## 16      liar      6     5 1.031  disgust
## 17 dishonorable   37    32 1.006  disgust
```

```

## 18      john      24      21 0.994  disgust
## 19    dying      6       6 0.872  disgust
## 20 terrorism      9       9 0.872  disgust

android_iphone_or %>% inner_join(nrc, by = "word") %>%
  mutate(sentiment = factor(sentiment, levels = log_or$sentiment)) %>%
  mutate(log_or = log(or)) %>%
  filter(Android + iPhone > 10 & abs(log_or)>1) %>%
  mutate(word = reorder(word, log_or)) %>%
  ggplot(aes(word, log_or, fill = log_or < 0)) +
  facet_wrap(~sentiment, scales = "free_x", nrow = 2) +
  geom_bar(stat="identity", show.legend = FALSE) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))

```



## 7. Regresión Lineal

### 7.1. Introducción a las regresiones

#### 7.1.1. Ejemplo: Baseball

Como ejemplo, supóngase un equipo de béisbol con un presupuesto limitado y en donde nos enfocaremos en predecir las carreras anotadas. El análisis se dividirá en dos:

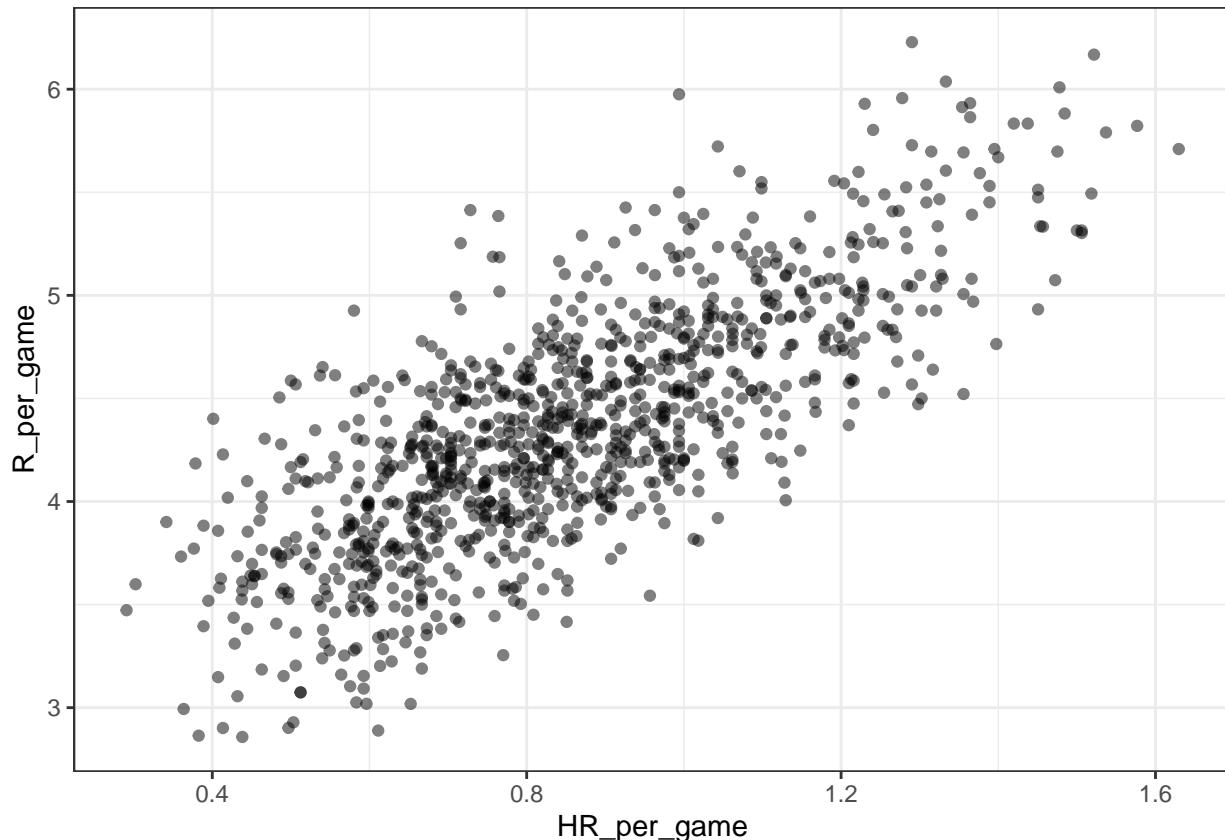
1. Determinar qué estadísticas registradas de los jugadores pueden predecir las carreras.
2. Examinar si los jugadores estuvieron subvalorados con base en el primer análisis.

¿Los equipos que conectan más *home runs* anotan más carreras? Observamos la relación entre *home runs* y carreras anotadas entre 1961 y 2001:

```
library(Lahman)
library(tidyverse)
library(dslabs)

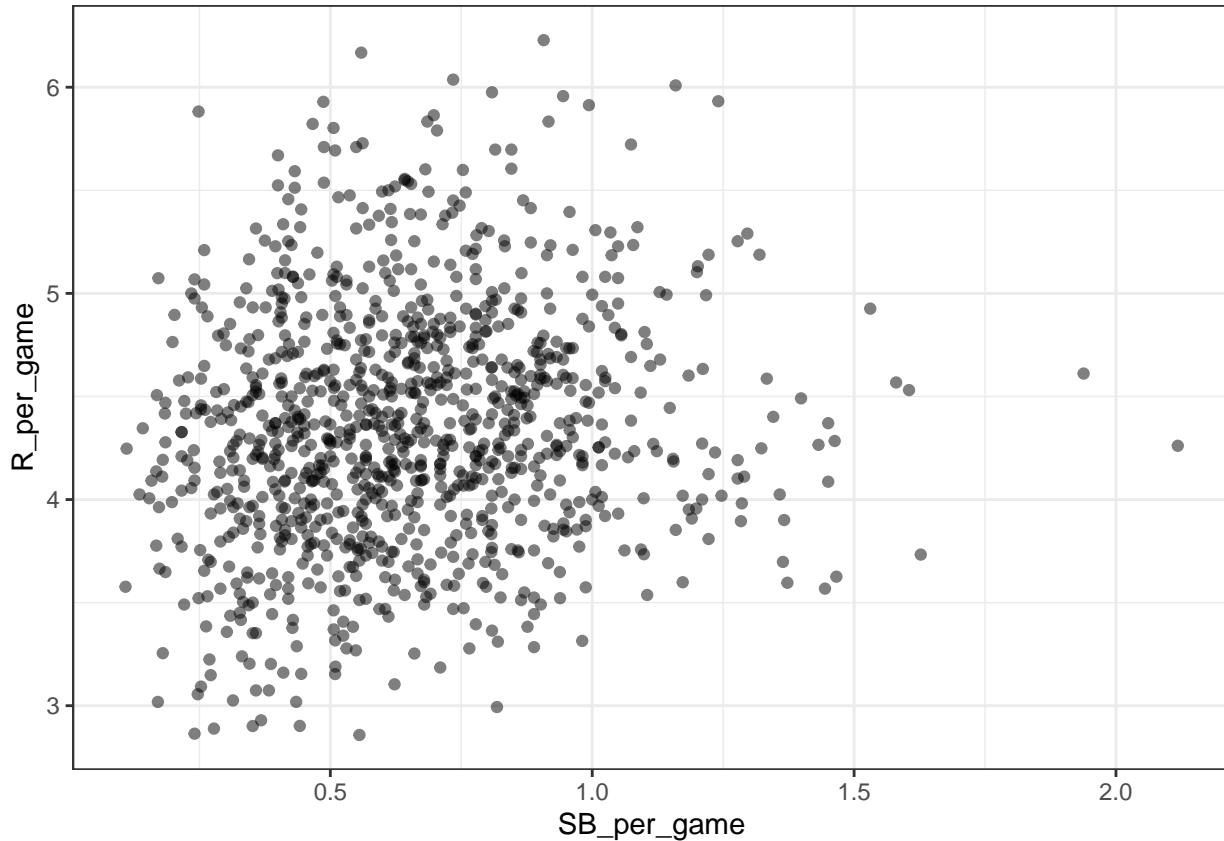
ds_theme_set()

Teams %>%
  filter(yearID %in% 1961:2001) %>%
  mutate(HR_per_game = HR/G, R_per_game = R/G) %>%
  ggplot(aes(HR_per_game, R_per_game)) +
  geom_point(alpha = 0.5)
```



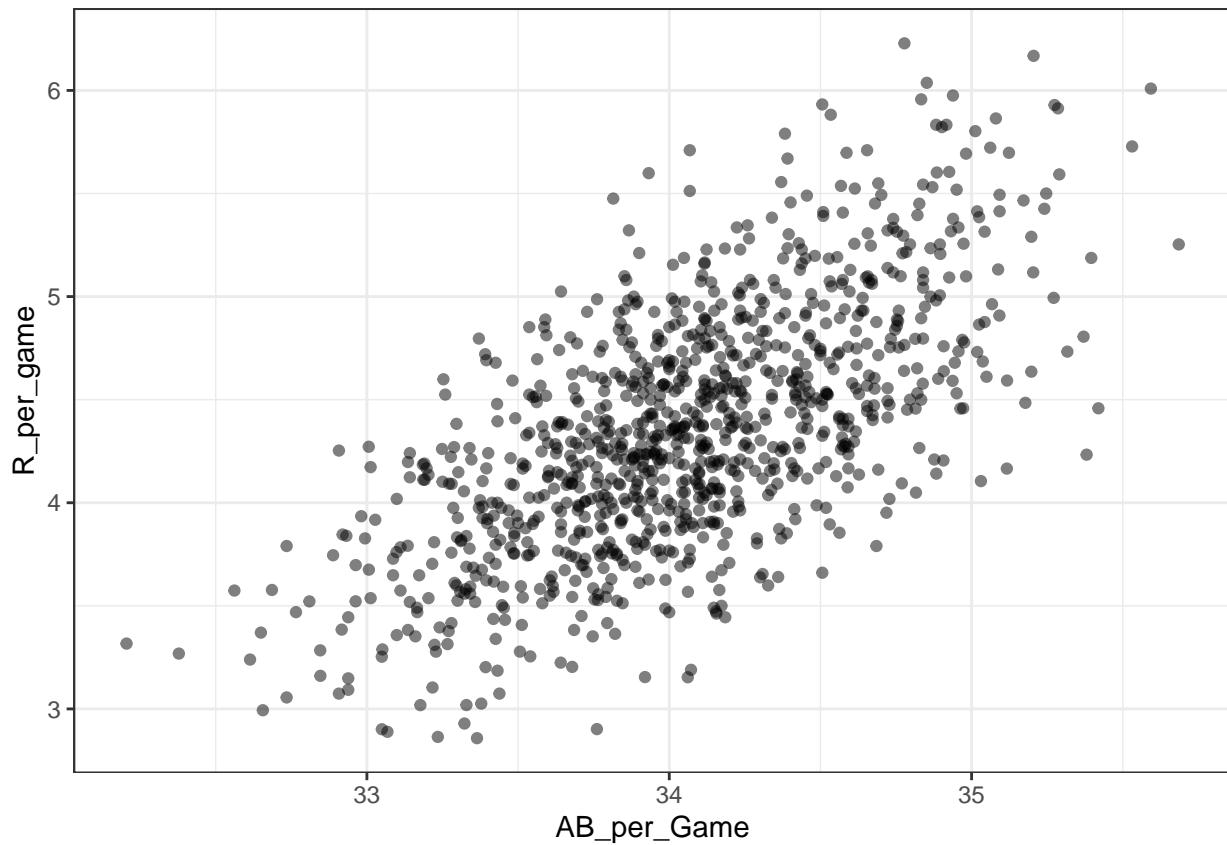
El gráfico anterior muestra una relación fuerte: los equipos con más *home runs* tienden a anotar más carreras. Ahora, analicemos la relación entre las bases robadas y las victorias:

```
Teams %>%
  filter(yearID %in% 1961:2001) %>%
  mutate(SB_per_game = SB/G, R_per_game = R/G) %>%
  ggplot(aes(SB_per_game, R_per_game)) +
  geom_point(alpha = 0.5)
```



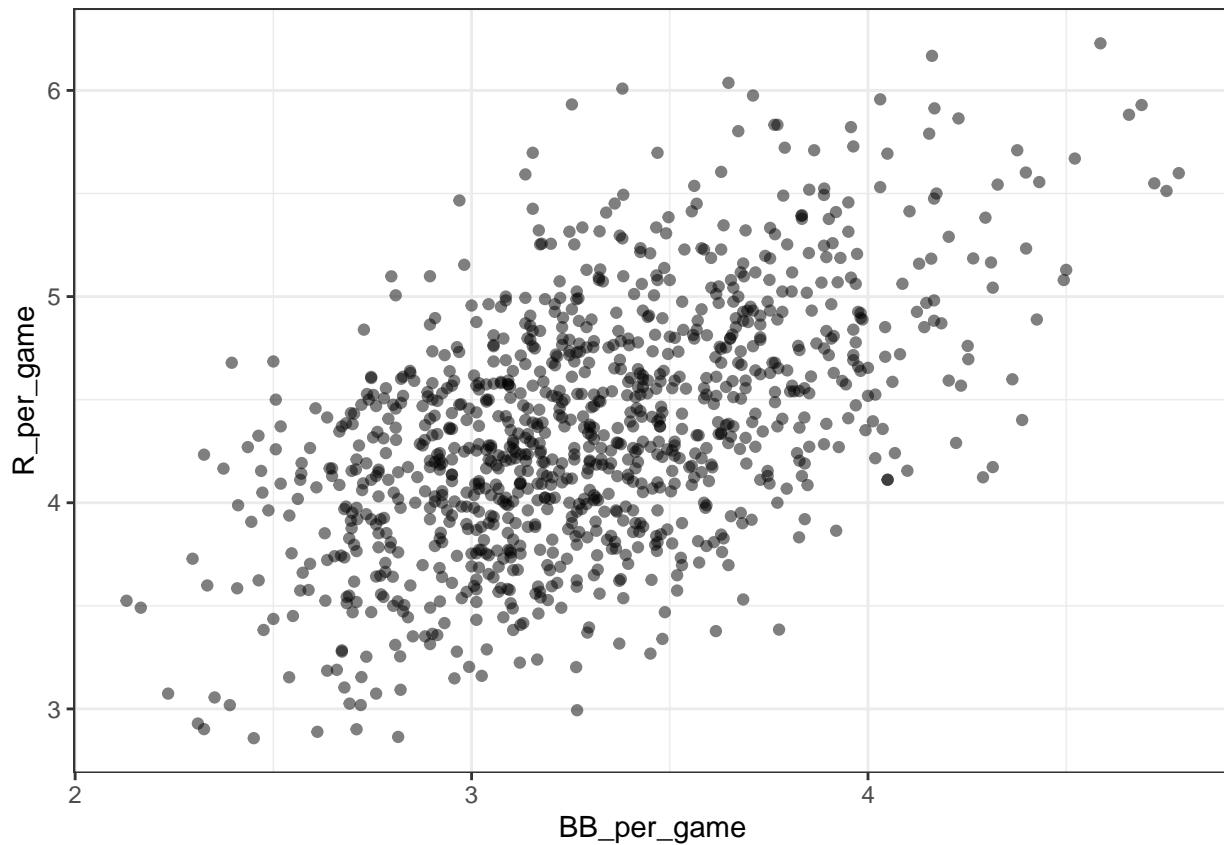
Se observa que la relación no es clara. Ahora, observemos la relación entre turnos al bate y carreras:

```
Teams %>%
  filter(yearID %in% 1961:2001) %>%
  mutate(AB_per_Game = AB/G, R_per_game = R/G) %>%
  ggplot(aes(AB_per_Game, R_per_game)) +
  geom_point(alpha = 0.5)
```



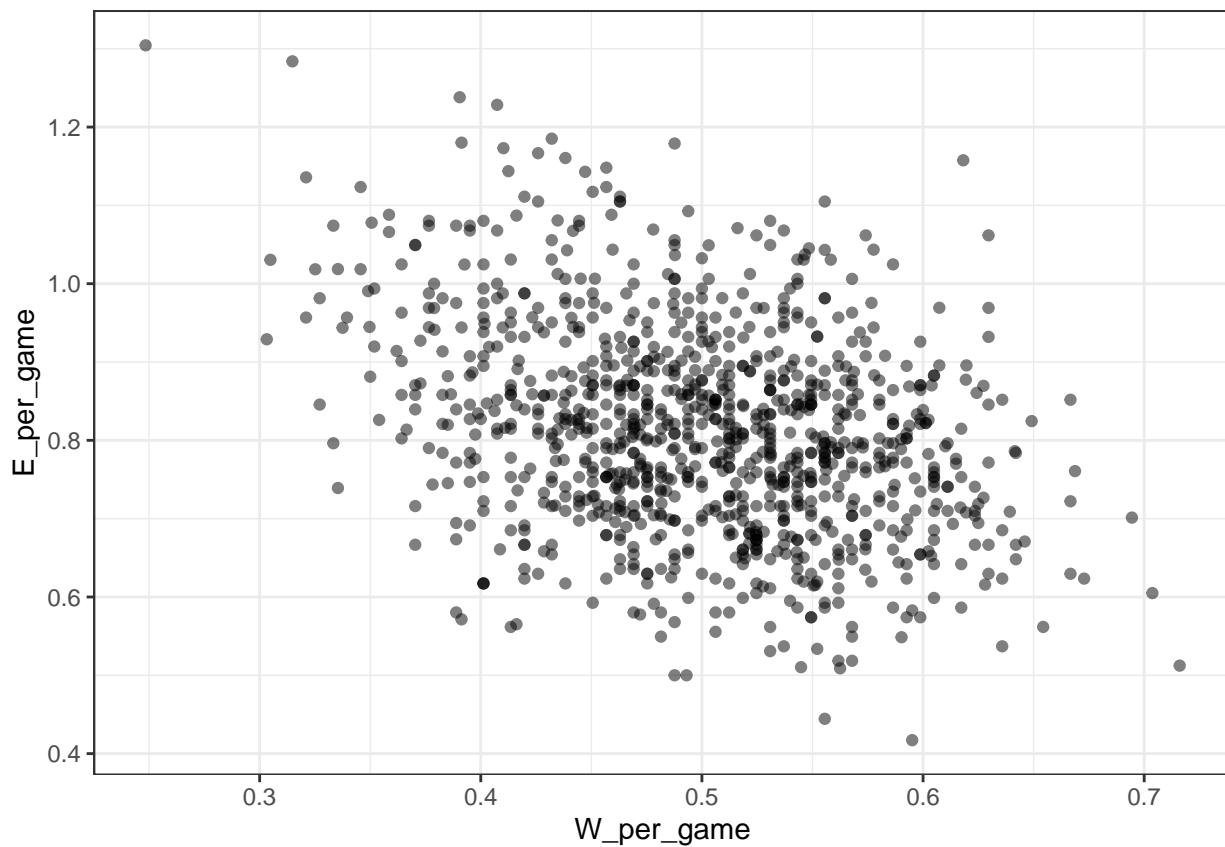
Grafiquese la relación entre bases por bola y carreras:

```
Teams %>%
  filter(yearID %in% 1961:2001) %>%
  mutate(BB_per_game = BB/G, R_per_game = R/G) %>%
  ggplot(aes(BB_per_game, R_per_game)) +
  geom_point(alpha = 0.5)
```



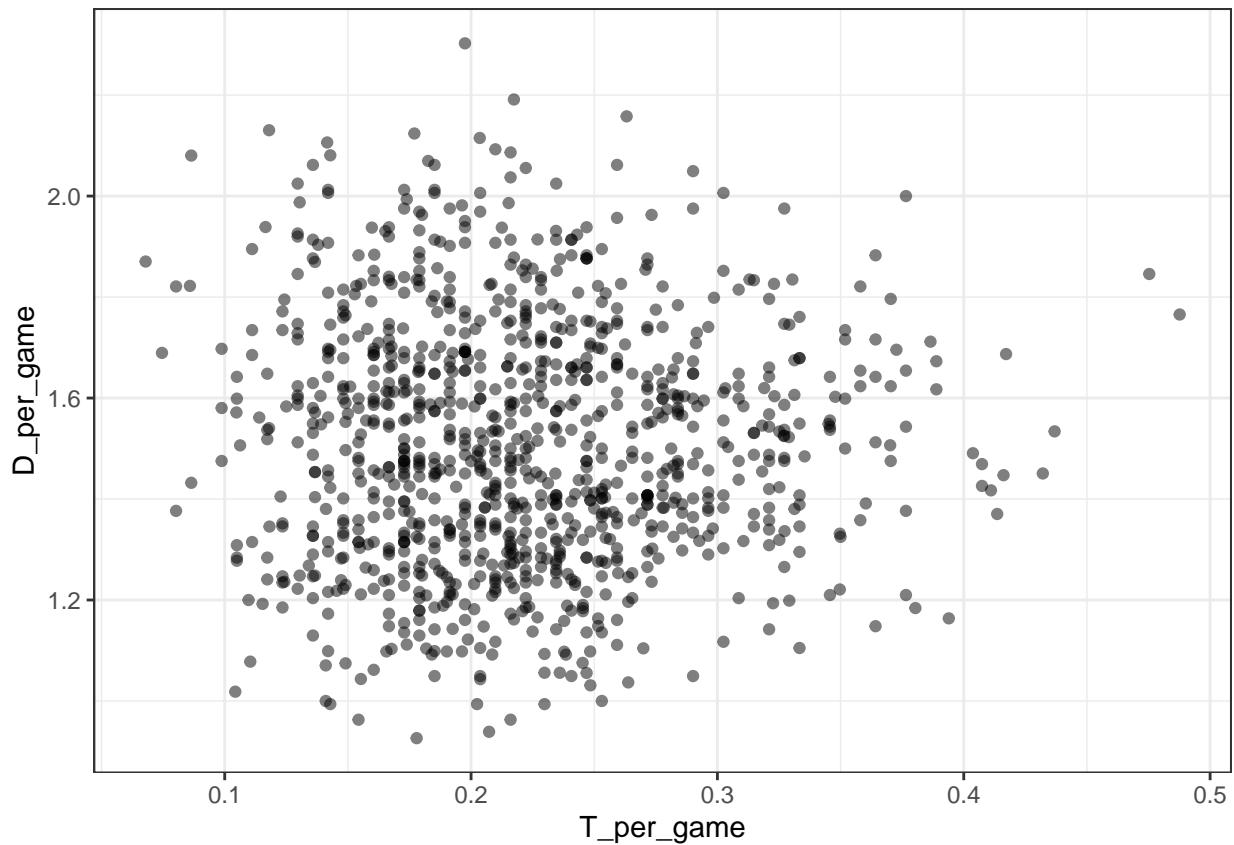
A pesar de que no es una relación tan fuerte como la primera, esta sí existe aparentemente. Ahora, analicemos la relación entre las victorias y los errores de campo:

```
Teams %>%
  filter(yearID %in% 1961:2001) %>%
  mutate(W_per_game = W/G, E_per_game = E/G) %>%
  ggplot(aes(W_per_game, E_per_game)) +
  geom_point(alpha = 0.5)
```



Finalmente, considérese la relación entre triples y dobles:

```
Teams %>%
  filter(yearID %in% 1961:2001) %>%
  mutate(T_per_game = X3B/G, D_per_game = X2B/G) %>%
  ggplot(aes(T_per_game, D_per_game)) +
  geom_point(alpha = 0.5)
```



### 7.1.2. Correlación

Es común que durante nuestro análisis de datos nos interesemos en la relación entre dos o más variables. Considérese, por ejemplo, una base de datos sobre genética, la cual fue construida para estudiar la variación de la herencia de características humanas, específicamente la altura. Con ella se trató de responder a la pregunta: ¿Qué tanto de la altura de un hijo se puede predecir con la altura del padre?

```
library(HistData)

data("GaltonFamilies")

galton_heights <- GaltonFamilies %>%
  filter(childNum == 1 & gender == "male") %>%
  select(father, childHeight) %>%
  rename(son = childHeight)
```

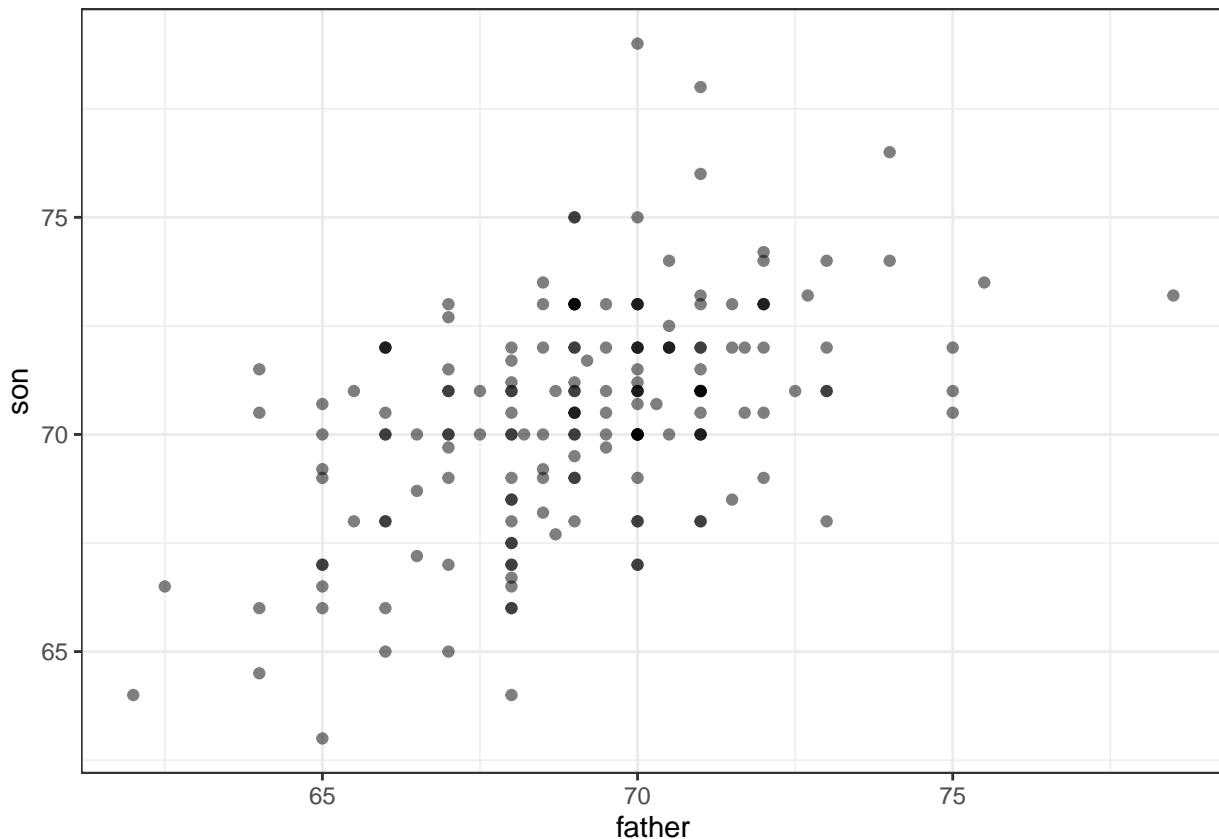
Supóngase que queremos resumir estos datos; dado que ambas variables se aproximan a una distribución normal, podemos usar las dos medias y desviaciones estándar como estadísticas descriptivas:

```
galton_heights %>%
  summarize(mean(father), sd(father), mean(son), sd(son))

##   mean(father) sd(father) mean(son) sd(son)
## 1      69.1      2.55     70.5      2.56
```

Sin embargo, este resumen estadístico no describe una característica sumamente relevante que se puede observar en la siguiente gráfica, la tendencia y relación positivas entre las alturas del padre e hijo:

```
galton_heights %>%
  ggplot(aes(father, son)) +
  geom_point(alpha = 0.5)
```



El **coeficiente de correlación** está definido por una lista de pares  $(x_1, y_1), \dots, (x_n, y_n)$ , obtenidos como el producto de sus valores estandarizados:  $\left(\frac{x_i - \mu_x}{\sigma_x}\right) \left(\frac{y_i - \mu_y}{\sigma_y}\right)$ ; esto es:

$$\rho = \frac{1}{n} \sum_{i=1}^n \left( \frac{x_i - \mu_x}{\sigma_x} \right) \left( \frac{y_i - \mu_y}{\sigma_y} \right)$$

Este coeficiente esencialmente transmite cómo dos variables se mueven juntas y siempre cumple  $\rho \in [-1, 1]$  (nótese que, en caso de que x, y no estén correlacionados,  $\rho = 0$ , si las variable se mueven en la misma dirección,  $\rho > 0$  y si se mueven en direcciones opuestas,  $\rho < 0$ ).

Así, calcúlese la correlación entre las alturas de padre e hijo:

```
galton_heights %>%
  summarize(cor(father, son))

##   cor(father, son)
## 1      0.501
```

Antes de pasar a la definición de una regresión, es pertinente recordar a la variabilidad aleatoria: en la mayoría de las aplicaciones de ciencia de datos, no observamos la población, sino una muestra. Como en el caso del promedio y la desviación estándar, la correlación muestral es el estimador más común de la correlación

poblacional. Esto implica que la correlación que calculamos es una variable aleatoria. Por ejemplo, supóngase que nuestra muestra es de solo 25:

```
set.seed(0)

R <- sample_n(galton_heights, 25, replace = TRUE) %>%
  summarize(cor(father, son))
```

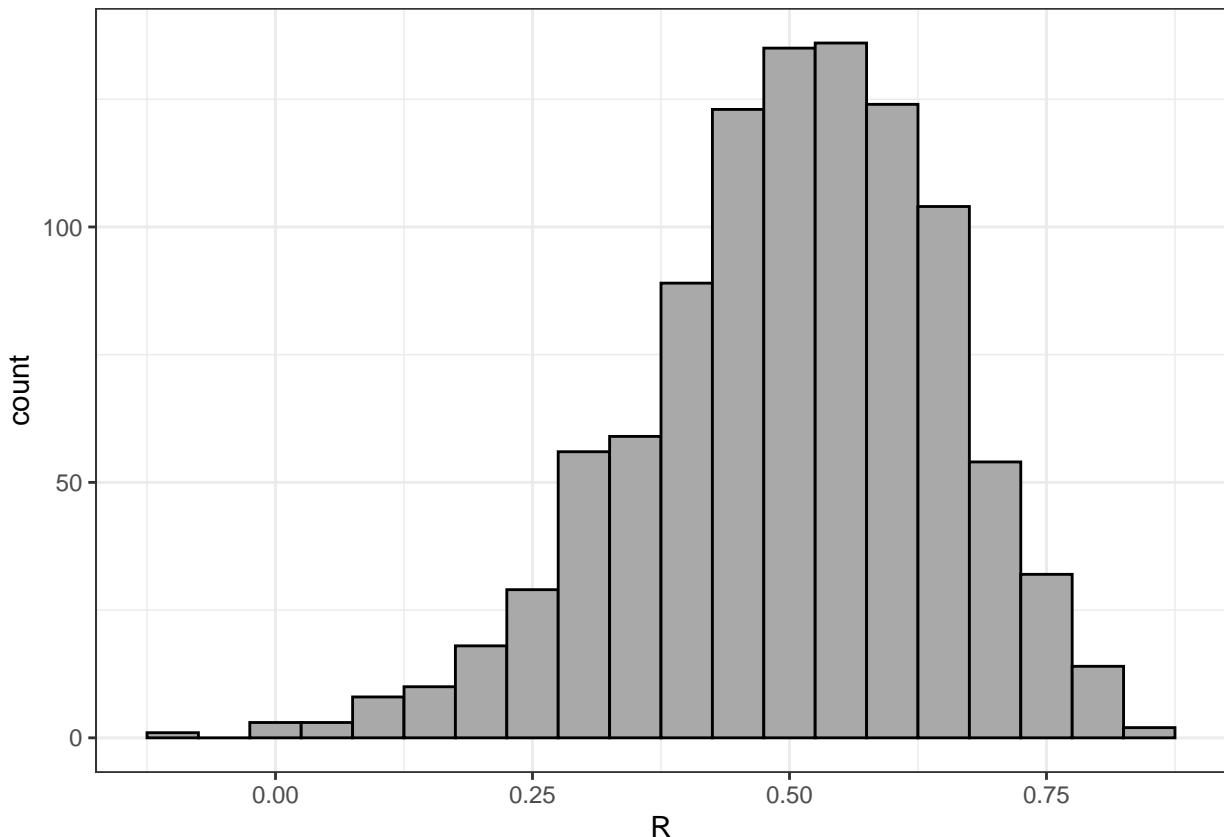
Puede realizarse una simulación Monte Carlo para observar la distribución de esta variable aleatoria:

```
B <- 1000

N <- 25

R <- replicate(B, {
  sample_n(galton_heights, N, replace = TRUE) %>%
    summarize(r = cor(father, son)) %>%
    .$r
})

data.frame(R) %>%
  ggplot(aes(R)) +
  geom_histogram(binwidth = 0.05, color = "black", fill = "darkgray")
```



```
mean(R)
```

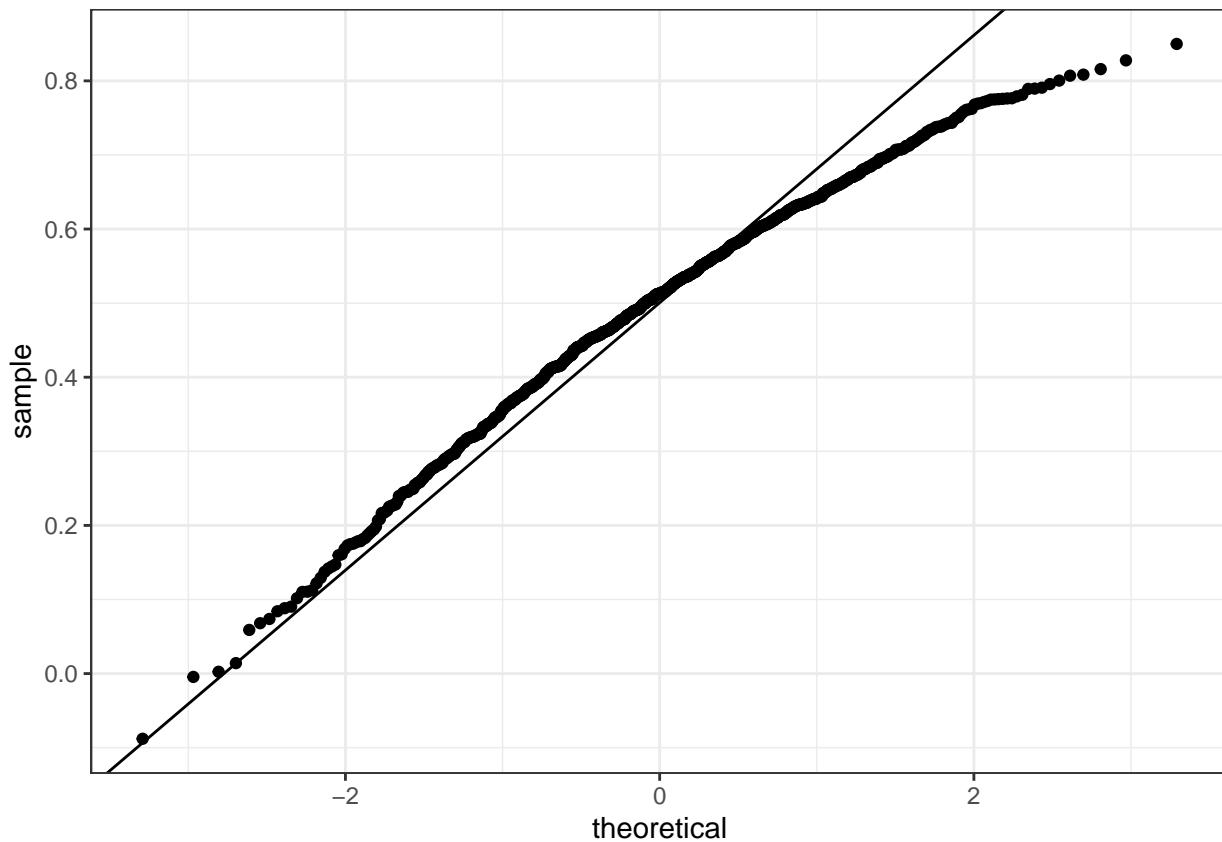
```
## [1] 0.501
```

```
sd(R)
```

```
## [1] 0.147
```

Nótese que dado que nuestra correlación muestral es un promedio de realizaciones independientes, el Teorema del Límite Central sí aplica. Por tanto, para un tamaño de muestra suficientemente grande N,  $R \sim N\left(\rho, \sqrt{\frac{1-\rho^2}{N-2}}\right)$ . En nuestro ejemplo,  $N = 25$ , lo que parece no ser suficiente:

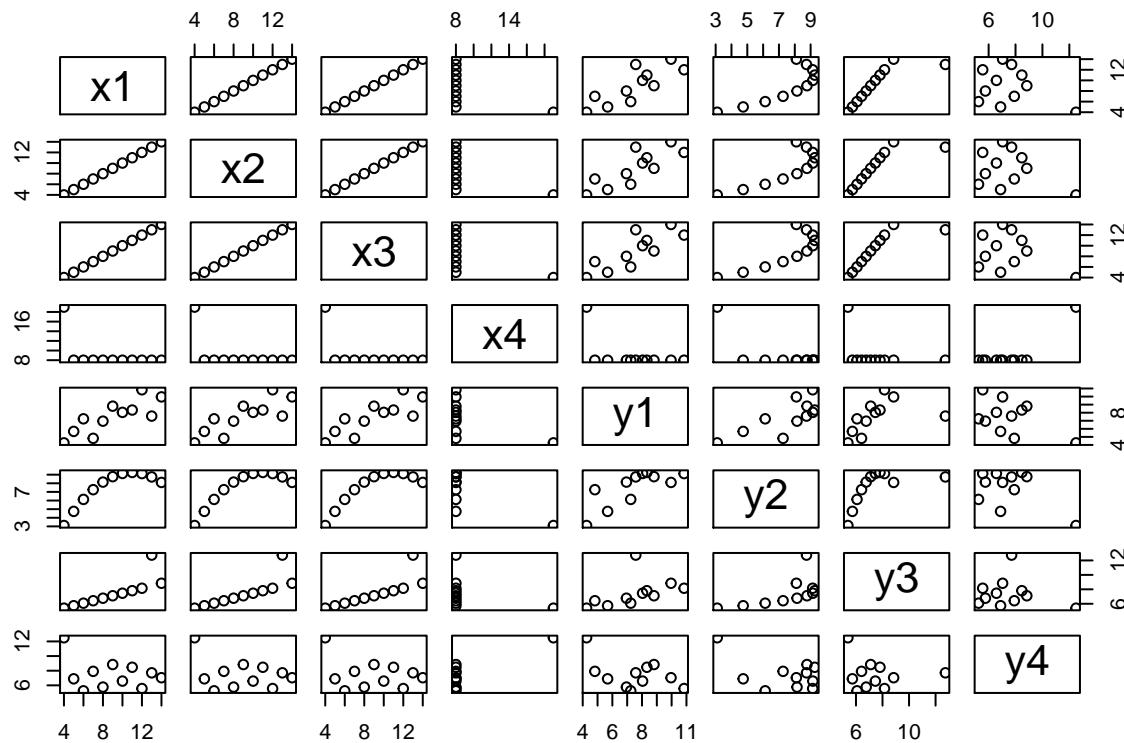
```
data.frame(R) %>%
  ggplot(aes(sample = R)) +
  stat_qq() +
  geom_abline(intercept = mean(R), slope = sqrt((1-mean(R)^2)/(N-2)))
```



### 7.1.3. Estratificación y varianza

A veces, el coeficiente de correlación no es un buen resumen de la relación entre dos variables. Un ejemplo para ilustrar esto es el conjunto de datos artificiales conocido como el Cuarteto de Anscombe:

```
plot(anscombe)
```



Donde todas las relaciones tienen un coeficiente  $\rho = 0,82$ . Para entender cuándo la correlación es significativa como estadística descriptiva, retómese el ejemplo de las alturas de padre e hijo; estratificaremos la altura de los padres. Específicamente, condicionaremos la altura del hijo a que la altura del padre es de aproximadamente 72 pulgadas:

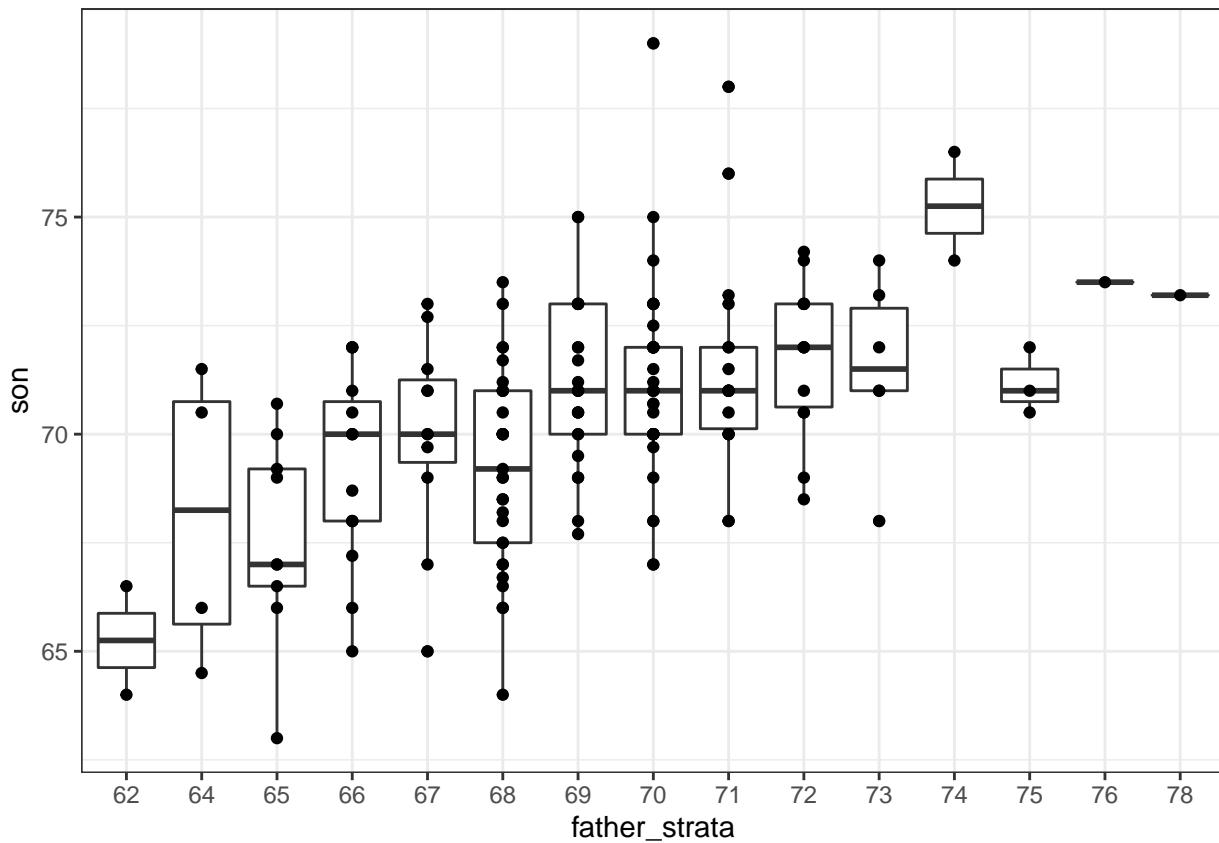
```
conditional_avg <- galton_heights %>%
  filter(round(father) == 72) %>%
  summarize(avg = mean(son)) %>%
  .$avg

conditional_avg
```

```
## [1] 71.8
```

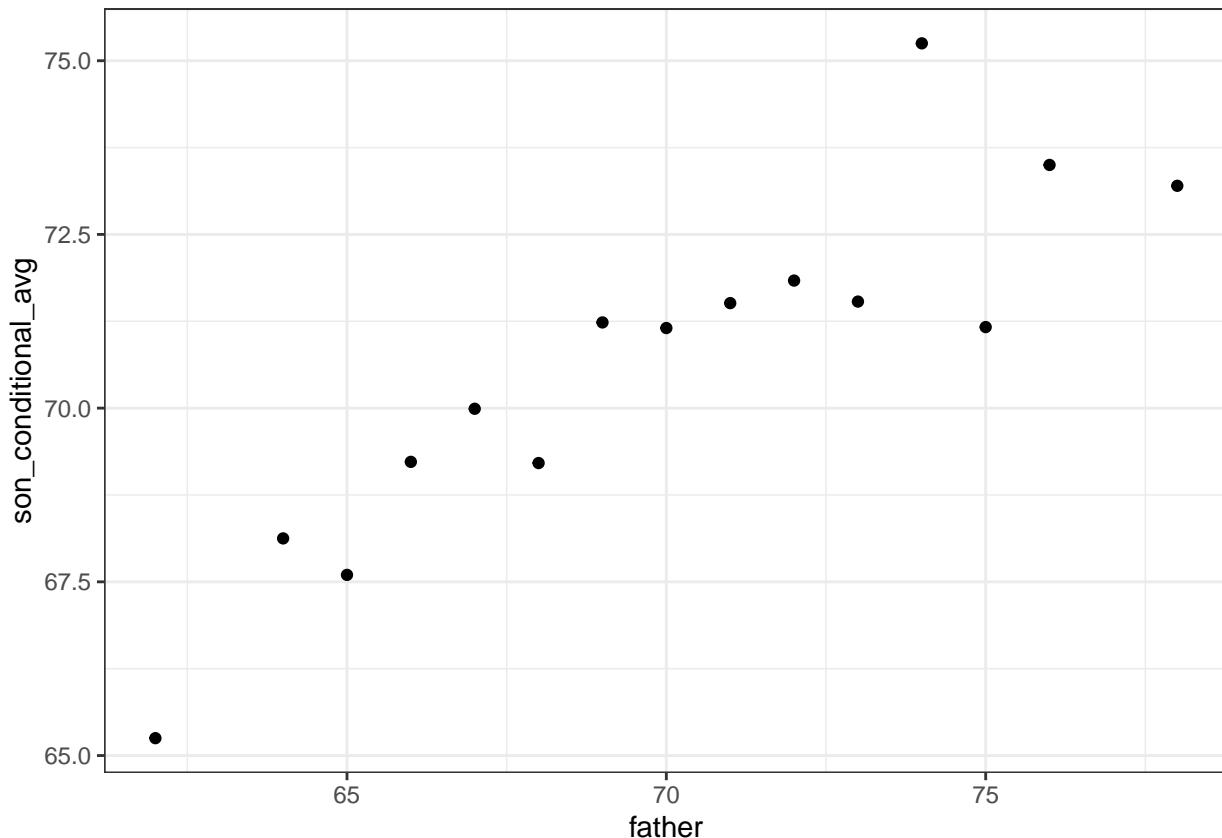
Una estratificación en boxplots nos permite observar la distribución de cada grupo:

```
galton_heights %>%
  mutate(father_strata = factor(round(father))) %>%
  ggplot(aes(father_strata, son)) +
  geom_boxplot() +
  geom_point()
```



Donde aparentemente las medias siguen una relación lineal con una pendiente  $\approx 0,5$  (que es similar al coeficiente de correlación que habíamos calculado). Para comprobarlo:

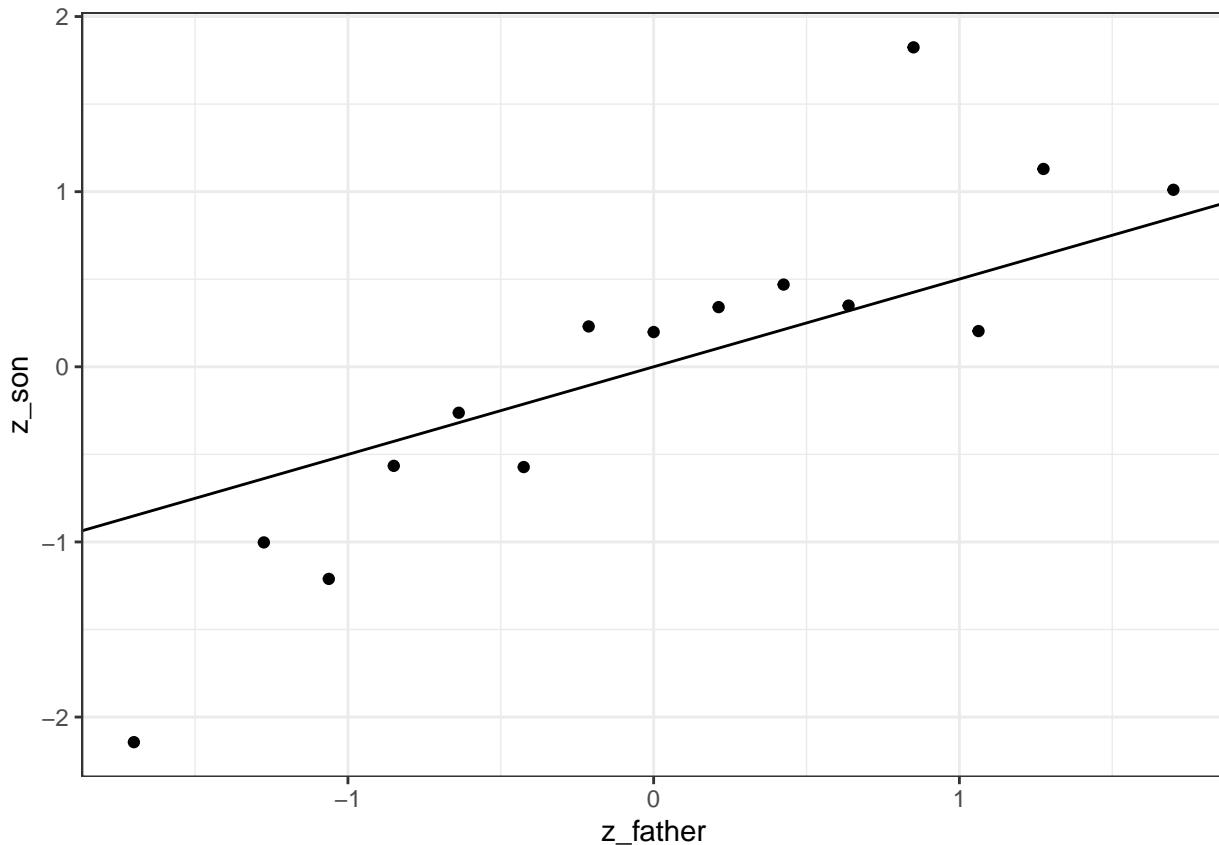
```
galton_heights %>%
  mutate(father = round(father)) %>%
  group_by(father) %>%
  summarize(son_conditional_avg = mean(son)) %>%
  ggplot(aes(father, son_conditional_avg)) +
  geom_point()
```



Para ver esta conexión, grafíquese las alturas estandarizadas de padre e hijo con una línea con pendiente igual a  $\rho$ :

```
r <- galton_heights %>%
  summarize(r = cor(father, son)) %>%
  .$r

galton_heights %>%
  mutate(father = round(father)) %>%
  group_by(father) %>%
  summarize(son = mean(son)) %>%
  mutate(z_father = scale(father), z_son = scale(son)) %>%
  ggplot(aes(z_father, z_son)) +
  geom_point() +
  geom_abline(intercept = 0, slope = r)
```



Donde la línea es lo que se conoce como **línea de regresión**: para cada incremento en una desviación estándar  $\sigma_x$  por encima del promedio  $\mu_x$ ,  $y$  crece  $\rho$  desviaciones estándar  $\sigma_y$  por encima del promedio  $\mu_y$ . Así, la línea de regresión tiene la siguiente fórmula:

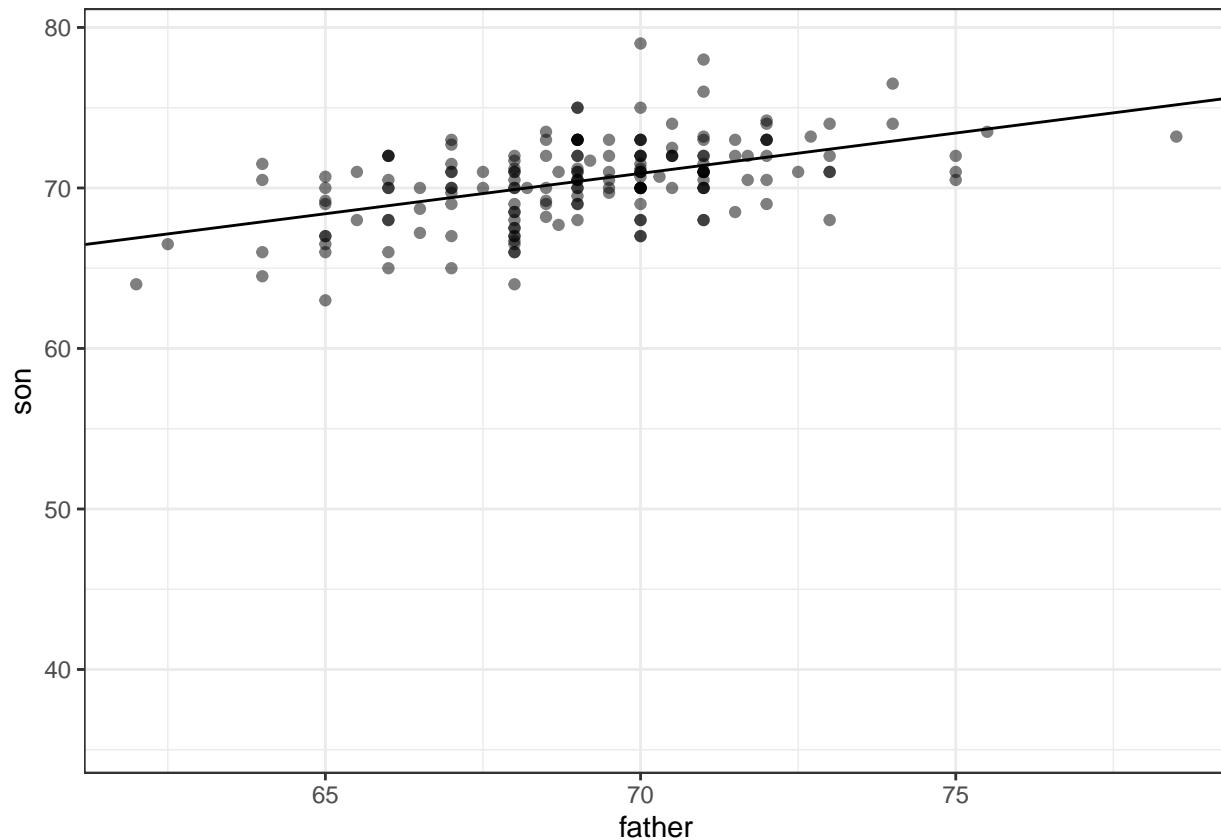
$$\left( \frac{y_i - \mu_y}{\sigma_y} \right) = \rho \left( \frac{x_i - \mu_x}{\sigma_x} \right)$$

Que implica a que la línea de regresión tiene pendiente  $m = \rho \frac{\sigma_y}{\sigma_x}$  e intercepto  $b = \mu_y - m\mu_x$ .

Observemos los datos originales de padres e hijos y agreguemos la línea de regresión:

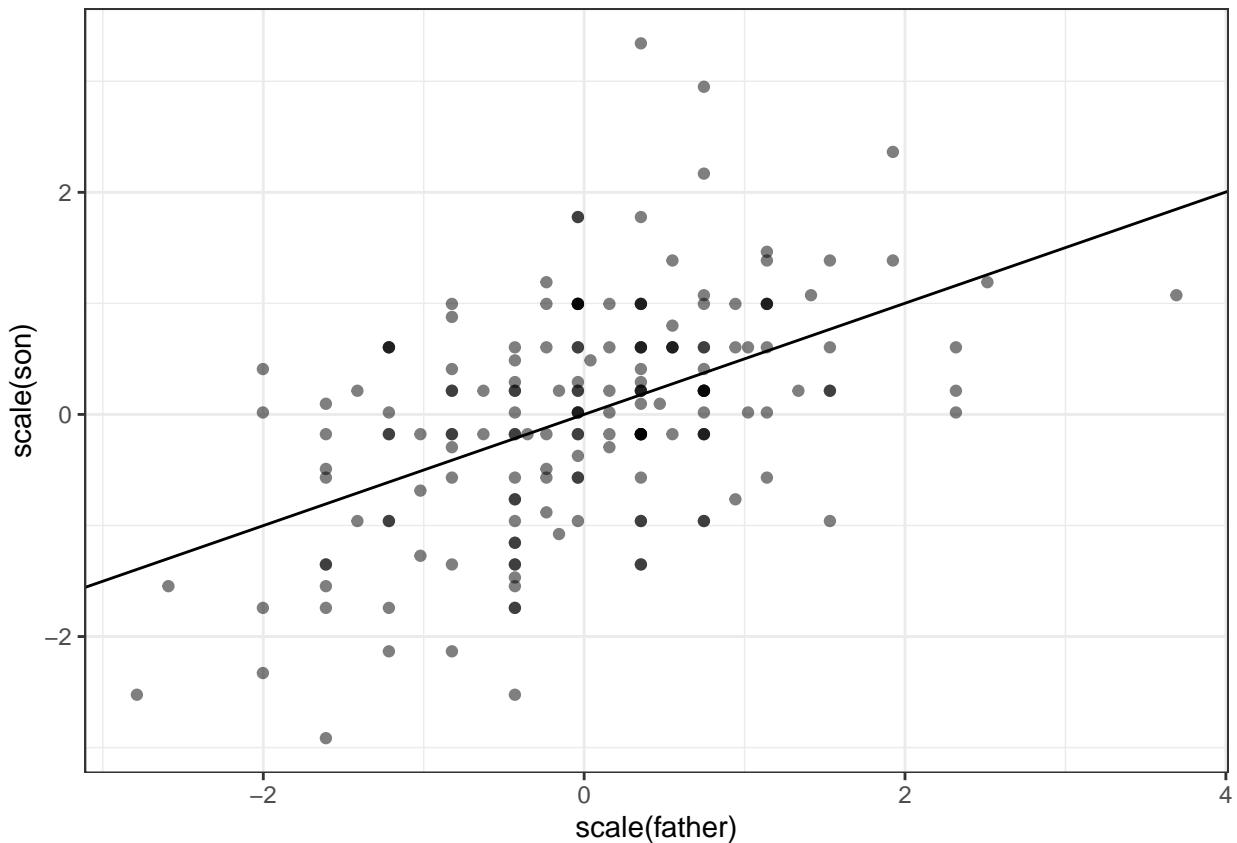
```
mu_x <- mean(galton_heights$father)
mu_y <- mean(galton_heights$son)
s_x <- sd(galton_heights$father)
s_y <- sd(galton_heights$son)
r <- cor(galton_heights$father, galton_heights$son)
m <- r*s_y/s_x
b <- mu_y-m*mu_x
```

```
galton_heights %>%
  ggplot(aes(father, son)) +
  geom_point(alpha = 0.5) +
  geom_abline(intercept = b, slope = m)
```



Y en unidades estandarizadas:

```
galton_heights %>%
  ggplot(aes(scale(father), scale(son))) +
  geom_point(alpha=0.5) +
  geom_abline(intercept = 0, slope = r)
```



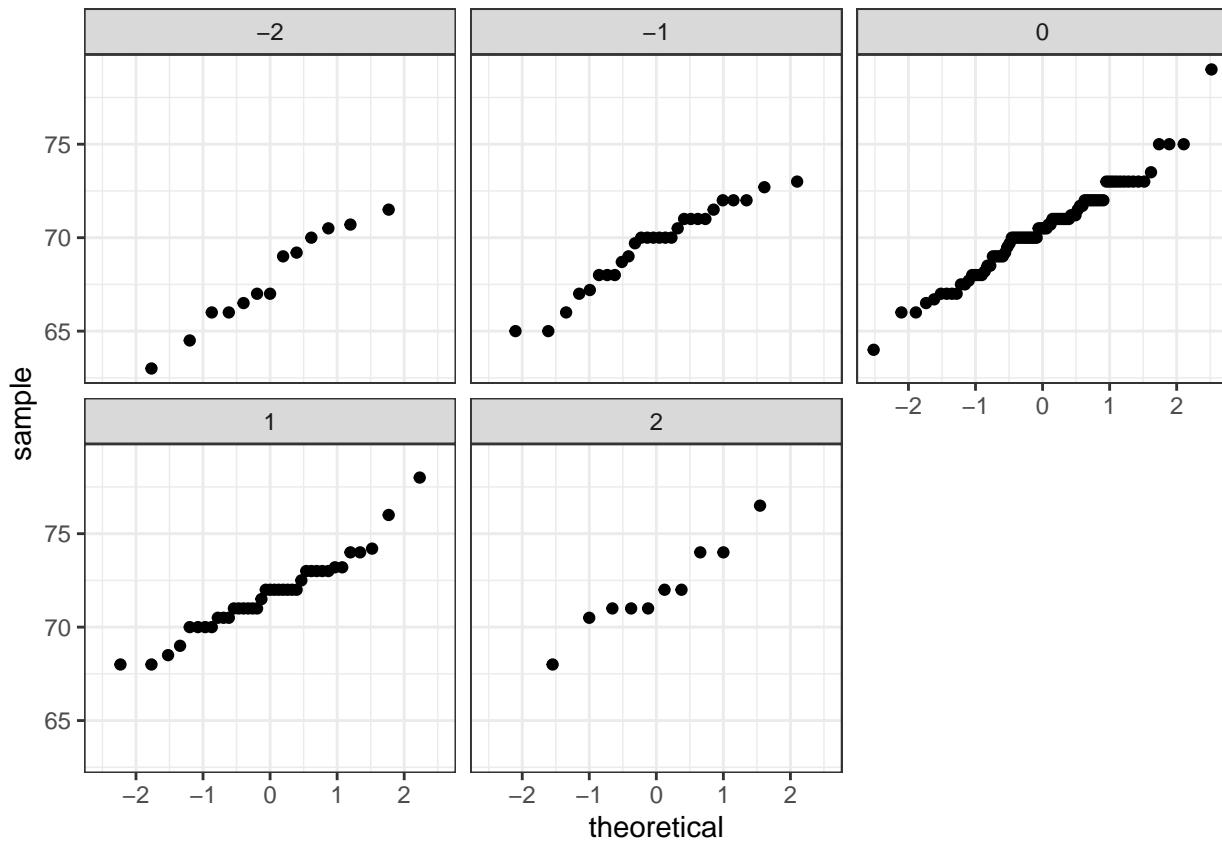
**7.1.3.1. Distribución normal bivariada** Cuando un par de variables aleatorias son aproximadas por una distribución normal bivariada, el *scatterplot* luce como óvalos: si son delgados, hay una correlación alta; si se asemeja más a un círculo, hay poca o nula correlación.

De manera más formal, si  $X$  y  $Y$  son variables aleatorias normalmente distribuidas, y para cualquier grupo de  $X$ ,  $X = x$ ,  $Y$  es aproximadamente normal en ese grupo, entonces el par es aproximadamente normal bivariado.

**KEY:** Así, computar una línea de regresión es equivalente a calcular las esperanzas condicionales  $E[Y|X = x]$ .

A continuación se estratifican las alturas de los hijos mediante las alturas estandarizadas de los padres:

```
galton_heights %>%
  mutate(z_father = round((father - mean(father))/sd(father))) %>%
  filter(z_father %in% -2:2) %>%
  ggplot() +
  stat_qq(aes(sample = son)) +
  facet_wrap(~z_father)
```



De hecho, puede demostrarse que si dos variables aleatorias son normales bivariadas, entonces se cumple:

$$\mathbf{E}[\mathbf{Y}|\mathbf{X} = \mathbf{x}] = \mu_{\mathbf{y}} + \rho \frac{\mathbf{X} - \mu_{\mathbf{x}}}{\sigma_{\mathbf{x}}} \sigma_{\mathbf{y}}$$

Que es idéntica a la línea de regresión que revisamos más arriba. En resumen, **si nuestros datos son aproximadamente bivariados, entonces la esperanza condicional está dada por la línea de regresión.**

Adicionalmente, condicional en una variable aleatoria X nos ayuda a reducir la varianza de la respuesta de Y. La desviación estándar de una distribución condicional es:

$$\text{SD}(\mathbf{Y}|\mathbf{X} = \mathbf{x}) = \sigma_{\mathbf{y}} \sqrt{1 - \rho^2}$$

Que implica que la varianza condicional es:

$$\text{Var}(\mathbf{Y}|\mathbf{X} = \mathbf{x}) = \sigma_{\mathbf{y}}^2 (1 - \rho^2)$$

Nótese que  $\sigma_{\mathbf{y}}^2 (1 - \rho^2) < \sigma_{\mathbf{y}}^2$

Retomando el ejemplo de las alturas, nuestros cálculos implican que  $E[Y|X = x] = 35,7 + 0,5x$ . ¿Qué ocurriría si quisieramos hacer lo opuesto, predecir la altura del padre con base en la del hijo? Esto NO se calcula con la función inversa de la esperanza condicional obtenida; en su lugar, hay que calcular  $E[X|Y = y]$ :

```
m_2 <- r * s_x / s_y  
b_2 <- mu_x - m_2*mu_y
```

Esto es, el valor esperado de la altura del padre dada la altura del hijo es  $E[X|Y = y] = 34 + 0,5y$ , una **línea de regresión diferente**.

### 7.1.4. Assessment 13

In the second part of this assessment, you'll analyze a set of mother and daughter heights, also from GaltonFamilies. Define female\_heights, a set of mother and daughter heights sampled from GaltonFamilies, as follows:

```
set.seed(1989, sample.kind="Rounding")

## Warning in set.seed(1989, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used

library(HistData)
data("GaltonFamilies")

female_heights <- GaltonFamilies %>%
  filter(gender == "female") %>%
  group_by(family) %>%
  sample_n(1) %>%
  ungroup() %>%
  select(mother, childHeight) %>%
  rename(daughter = childHeight)
```

Calculate the mean and standard deviation of mothers' heights, the mean and standard deviation of daughters' heights, and the correlation coefficient between mother and daughter heights.

```
mean(female_heights$mother)

## [1] 64.1

sd(female_heights$mother)

## [1] 2.29

mean(female_heights$daughter)

## [1] 64.3

sd(female_heights$daughter)

## [1] 2.39

cor(female_heights$mother, female_heights$daughter)

## [1] 0.325
```

Calculate the slope and intercept of the regression line predicting daughters' heights given mothers' heights. Given an increase in mother's height by 1 inch, how many inches is the daughter's height expected to change?

```
lm(female_heights$daughter~female_heights$mother)

##
## Call:
## lm(formula = female_heights$daughter ~ female_heights$mother)
##
## Coefficients:
## (Intercept)  female_heights$mother
##             42.517                 0.339
```

What percent of the variability in daughter heights is explained by the mother's height?

```
summary(lm(female_heights$daughter~female_heights$mother))

##
## Call:
## lm(formula = female_heights$daughter ~ female_heights$mother)
##
## Residuals:
##   Min     1Q Median     3Q    Max 
## -7.077 -1.235  0.102  1.432  5.414 
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 42.517     4.812    8.84  1.1e-15 ***
## female_heights$mother 0.339      0.075    4.53  1.1e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.27 on 174 degrees of freedom
## Multiple R-squared:  0.105, Adjusted R-squared:  0.1 
## F-statistic: 20.5 on 1 and 174 DF,  p-value: 1.11e-05
```

A mother has a height of 60 inches. Using the regression formula, what is the conditional expected value of her daughter's height given the mother's height?

42.5170 + 0.3393\*60

## [1] 62.9

## 7.2. Modelos lineales

### 7.2.1. Introducción

Nótese la correlación entre *home runs*, bases por bola y *singles*:

```
Teams %>%
  filter(yearID %in% 1961:2001) %>%
  mutate(Singles = (H-HR-X2B-X3B)/G, BB = BB/G, HR = HR/G) %>%
  summarize(cor(BB, HR), cor(Singles, HR), cor(BB, Singles))

##   cor(BB, HR) cor(Singles, HR) cor(BB, Singles)
## 1      0.404        -0.174       -0.056
```

Donde se observa que la primera es bastante alta en comparación con las otras. Podría interpretarse que más bases por bolas implican más carreras; sin embargo, recordemos que **correlación no implica causalidad**. En este ejemplo específico, los jugadores que conectan más *home runs* también obtienen más bases por bolas derivado de las precauciones tomadas por los pitchers; así, un equipo con más carreras es explicado por la mayor cantidad de *home runs*, no por las bases por bolas.

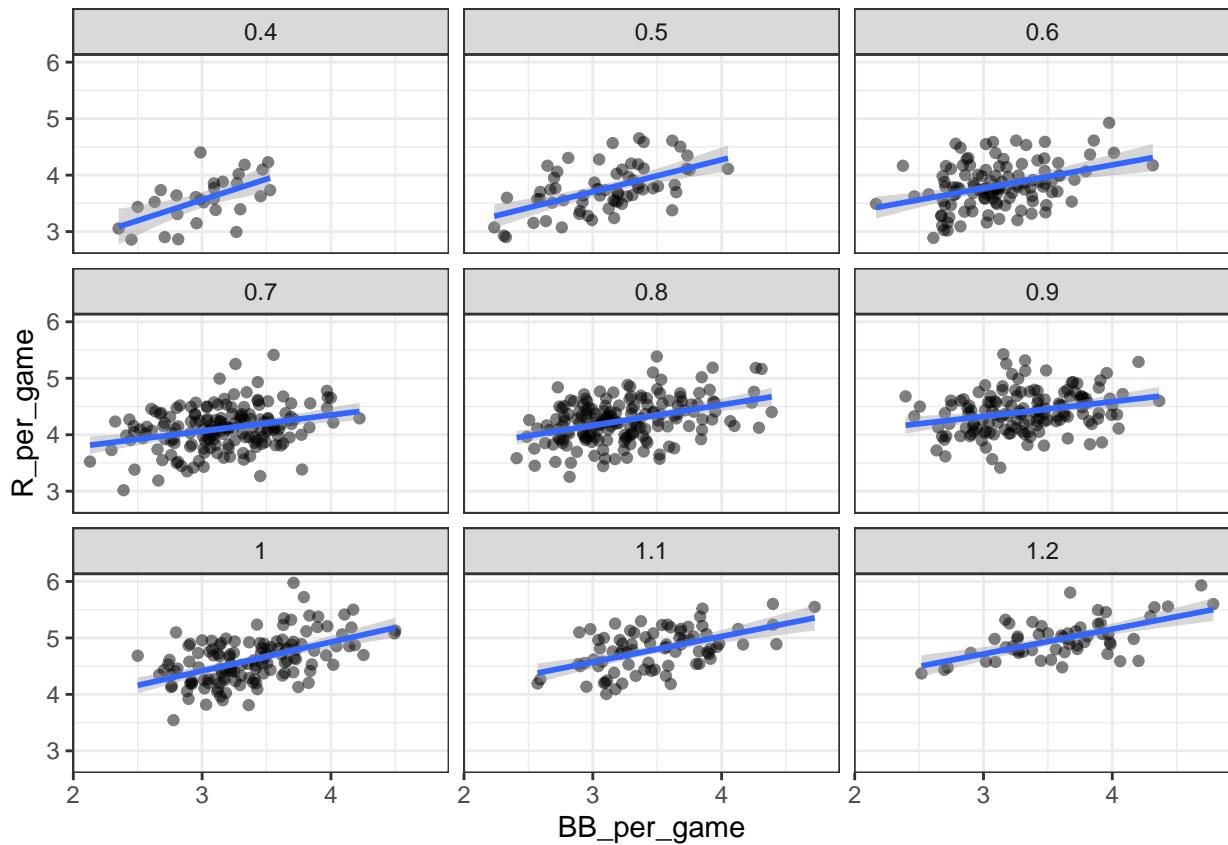
**7.2.1.1. Regresión multivariada** Para tratar de determinar si las bases por bola son útiles para la anotación de carreras, un primer enfoque sugiere fijar los *home runs* en un valor específico y después examinar la relación entre carreras y bases por bola.

En primer lugar, estratificaremos los datos de *home runs* por juego y redondearemos a la decena más cercana:

```
dat <- Teams %>%
  filter(yearID %in% 1961:2001) %>%
  mutate(HR_strata = round(HR/G, 1),
         BB_per_game = BB/G,
         R_per_game = R/G) %>%
  filter(HR_strata >= 0.4 & HR_strata <= 1.2)

dat %>%
  ggplot(aes(BB_per_game, R_per_game)) +
  geom_point(alpha = 0.5) +
  geom_smooth(method = "lm") +
  facet_wrap(~HR_strata)

## `geom_smooth()` using formula 'y ~ x'
```



Podemos extraer las pendientes individuales con el siguiente código:

```
dat %>%
  group_by(HR_strata) %>%
  summarize(slope = cor(BB_per_game, R_per_game)*sd(R_per_game)/sd(BB_per_game))

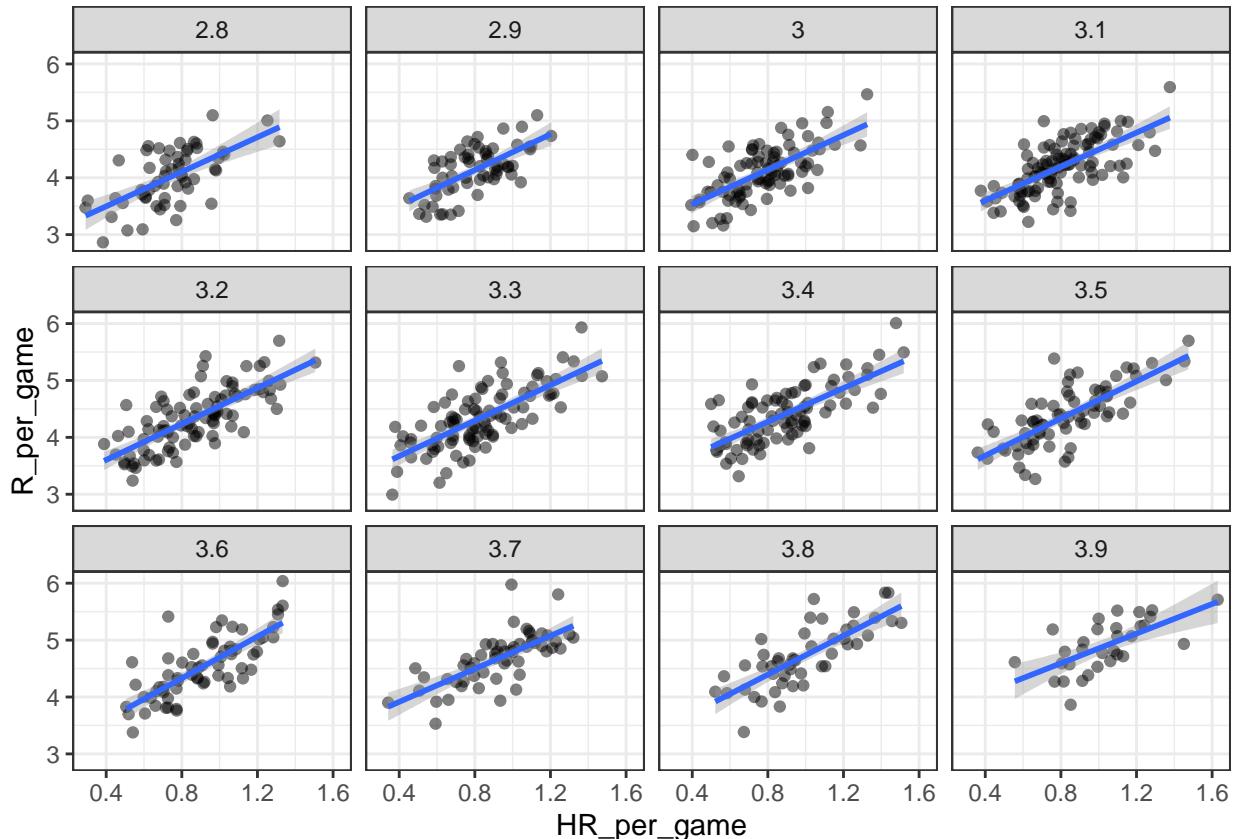
## # A tibble: 9 x 2
##   HR_strata slope
##       <dbl> <dbl>
## 1      0.4  0.734
## 2      0.5  0.566
## 3      0.6  0.412
## 4      0.7  0.285
## 5      0.8  0.365
## 6      0.9  0.261
## 7      1    0.512
## 8      1.1  0.454
## 9      1.2  0.440
```

Realícese el mismo análisis pero ahora con bases por bola:

```
dat <- Teams %>%
  filter(yearID %in% 1961:2001) %>%
  mutate(BB_strata=round(BB/G,1),
        HR_per_game = HR/G,
        R_per_game = R/G) %>%
  filter(BB_strata >= 2.8 & BB_strata <= 3.9)
```

```
dat %>%
  ggplot(aes(HR_per_game, R_per_game)) +
  geom_point(alpha = 0.5) +
  geom_smooth(method = "lm") +
  facet_wrap(~BB_strata)
```

## `geom\_smooth()` using formula 'y ~ x'



Donde las pendientes son las siguientes:

```
dat %>%
  group_by(BB_strata) %>%
  summarize(slope = cor(HR_per_game, R_per_game)*sd(R_per_game)/sd(HR_per_game))

## # A tibble: 12 x 2
##   BB_strata slope
##       <dbl> <dbl>
## 1      2.8  1.52
## 2      2.9  1.57
## 3      3     1.52
## 4      3.1  1.49
## 5      3.2  1.58
## 6      3.3  1.56
## 7      3.4  1.48
## 8      3.5  1.63
## 9      3.6  1.83
## 10     3.7  1.45
```

```
## 11      3.8  1.70
## 12      3.9  1.30
```

Dado que el estimado original era 1.84, puede verse que las pendientes no varían demasiado con respecto a este. Esto implica que las carreras se pueden explicar mediante un modelo lineal, como a continuación se describe.

En la práctica, es común escribir explícitamente un modelo que describa la relación entre dos variables o más mediante lo que se conoce como **modelo lineal**. Es importante recalcar que “lineal” no se refiere a líneas, sino al hecho de que la esperanza condicional es una combinación lineal de cantidades conocidas.

Retomando el ejemplo de las alturas, sean las alturas de los padres  $x_1, \dots, x_n$ ; así, el modelo que predice las alturas de los hijos en función de las de los padres es:

$$Y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$$

Donde se asume que los errores  $\varepsilon_i$  son independientes unos de otros y  $\varepsilon \sim N(0, \sigma)$ ; este término de error puede incluir otros efectos en la variable explicada como el efecto de los genes maternos, factores ambientales y demás aleatoriedades biológicas.

Una de las ventajas de los modelos lineales es su fácil interpretabilidad; en nuestro ejemplo: la altura predicha de los hijos crece en  $\beta_1$  por cada pulgada que se incrementa la altura del padre  $x$ . En muchos casos, como este, el intercepto es más difícil de interpretar.

### 7.2.2. MCO

Para que nuestros modelos estimados sean útiles, debemos estimar los parámetros desconocidos. El enfoque estándar es encontrar los valores que minimizan la distancia del modelo ajustado con los datos. Para esto, utilizamos la ecuación de mínimos cuadrados:

$$\text{RSS} = \sum_{i=1}^n [Y_i - (\beta_0 + \beta_1 x_i)]^2$$

Donde RSS: residual sum of squares. Una vez que encontramos los valores que minimizan RSS, los llamamos MCO y se denotan como  $\hat{\beta}_1, \dots, \hat{\beta}_k$ . Para nuestro ejemplo:

```
rss <- function(beta0, beta1, data){
  resid <- galton_heights$son - (beta0+beta1*galton_heights$father)
  return(sum(resid^2))
}
```

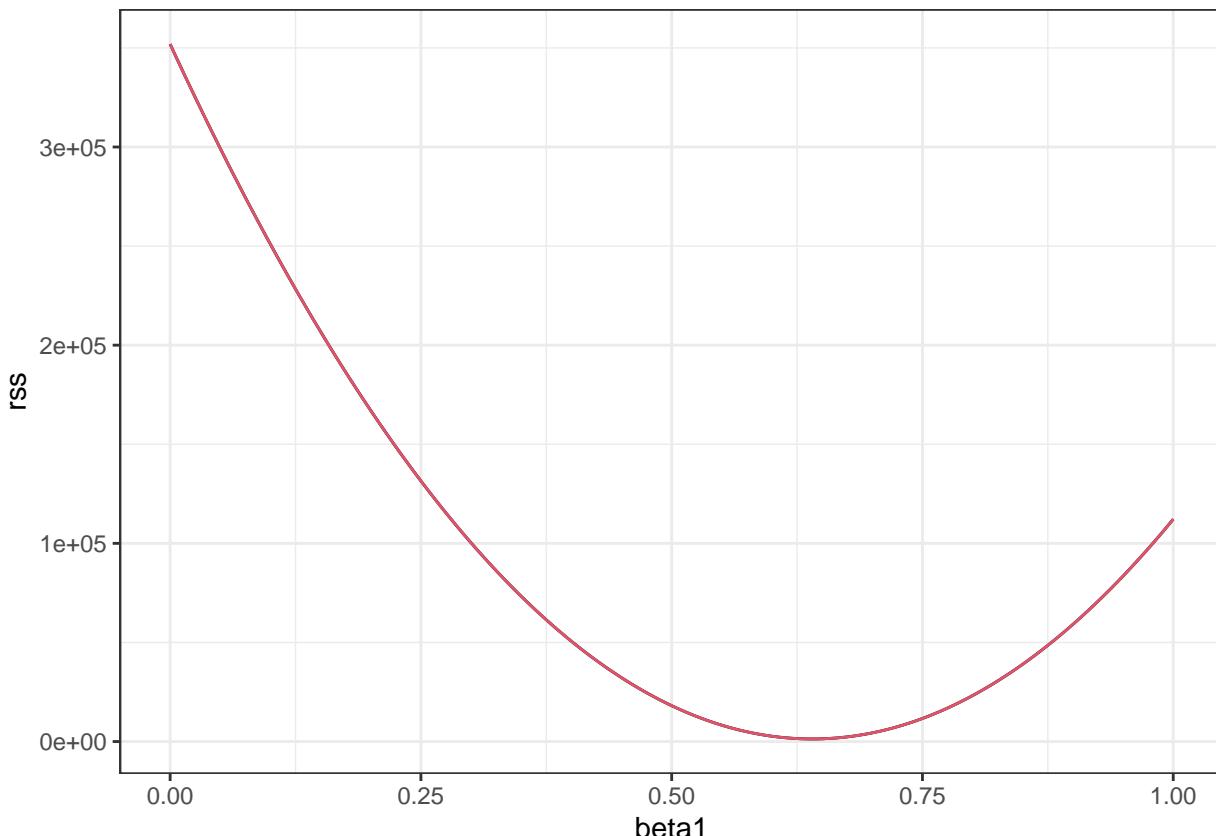
Si fijamos  $\beta_0 = 25$ , podemos graficar RSS como función de  $\beta_1$ :

```
library(HistData)
data("GaltonFamilies")
set.seed(1983)
galton_heights <- GaltonFamilies %>%
  filter(gender == "male") %>%
  group_by(family) %>%
  sample_n(1) %>%
  ungroup() %>%
  select(father, childHeight) %>%
  rename(son = childHeight)

beta1 = seq(0,1, len = nrow(galton_heights))
```

```
results <- data.frame(beta1= beta1,
                      rss = sapply(beta1, rss, beta0=25))

results %>%
  ggplot(aes(beta1, rss)) +
  geom_line() +
  geom_line(aes(beta1, rss), col =2)
```



Para poder estimar ambos valores simultáneamente (sin tener que fijar ninguno), necesitamos de cálculo. No obstante, la función de R, “lm()”, permite ajustar el modelo lineal que indiquemos y arroja los estimadores pertinentes:

```
fit <- lm(son ~ father, data = galton_heights)

fit

##
## Call:
## lm(formula = son ~ father, data = galton_heights)
##
## Coefficients:
## (Intercept)      father
##       38.765        0.441
```

Si queremos obtener más información del modelo, podemos usar “summary()”:

```
summary(fit)
```

```

## 
## Call:
## lm(formula = son ~ father, data = galton_heights)
## 
## Residuals:
##    Min     1Q Median     3Q    Max 
## -9.423 -1.702  0.033  1.567  9.357 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 38.7646    5.4109   7.16  2.0e-11 ***
## father       0.4411    0.0783   5.64  6.7e-08 ***  
## --- 
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 
## 
## Residual standard error: 2.66 on 177 degrees of freedom 
## Multiple R-squared:  0.152, Adjusted R-squared:  0.147 
## F-statistic: 31.8 on 1 and 177 DF,  p-value: 6.72e-08

```

Como es evidente, los estimadores MCO son variables aleatorias. Para comprobarlo, corramos una simulación Monte Carlo (donde asumiremos que “galton\_heights” es toda la población a partir de la cual tomaremos muestras de 50):

```

B <- 1000

N <- 50

lse <- replicate(B, {
  sample_n(galton_heights, N, replace = TRUE) %>%
    lm(son ~ father, data = .) %>%
    .$coef
})

lse <- data.frame(beta0 = lse[,1], beta_1 = lse[,2])

```

Podemos observar la variabilidad de los estimadores graficando su distribución:

```

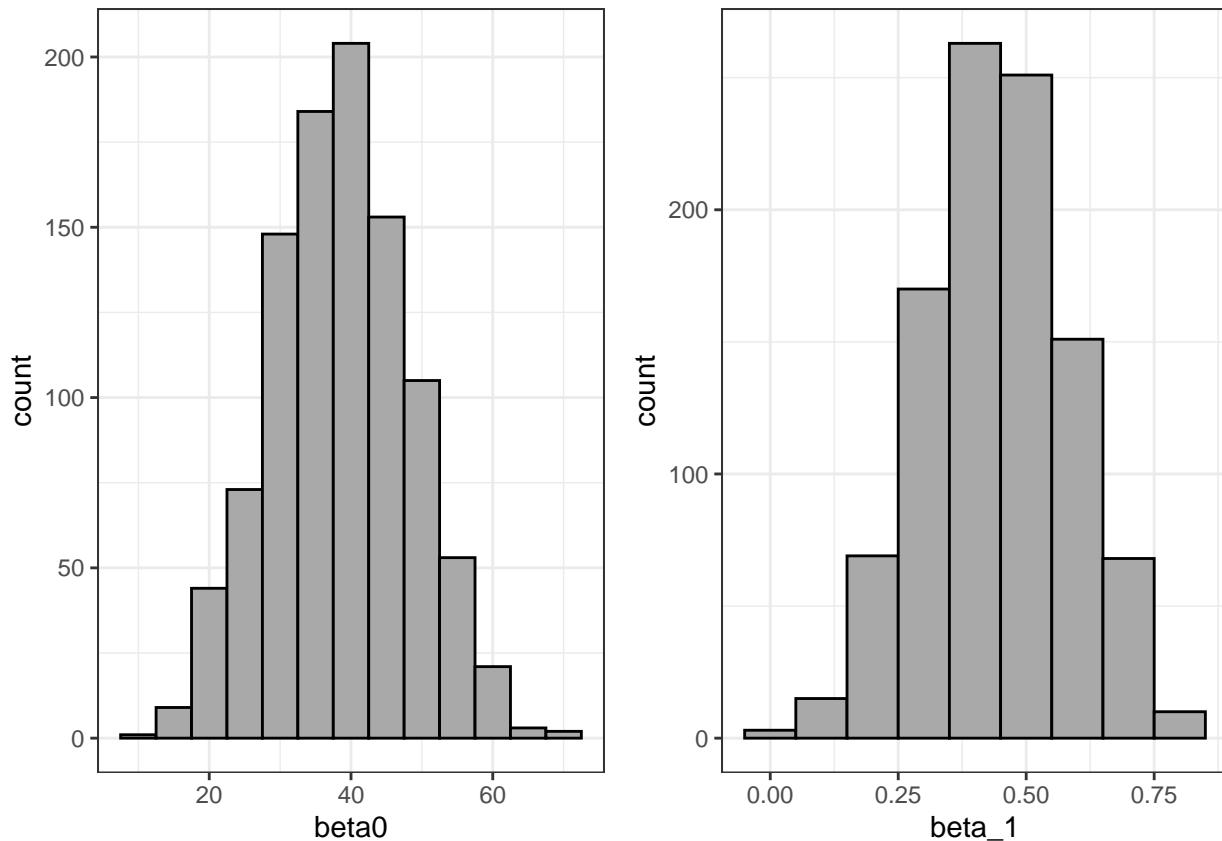
library(gridExtra)

p1 <- lse %>%
  ggplot(aes(beta0)) +
  geom_histogram(binwidth = 5, color = "black", fill = "darkgray")

p2 <- lse %>%
  ggplot(aes(beta_1)) +
  geom_histogram(binwidth = 0.1, fill = "darkgray", color = "black")

grid.arrange(p1, p2, ncol = 2)

```



Es obvio que su distribución asemeja a una normal, esto debido a que **para n suficientemente grandes, los estimadores MCO serán aproximadamente normales con valores esperados  $\beta_0$  y  $\beta_1$ .**

Nótese que el resumen del modelo arroja un valor conocido como el estadístico t, el cual está basado en el supuesto de que  $\varepsilon \sim N$ . Bajo este supuesto, la teoría matemática nos dice que:

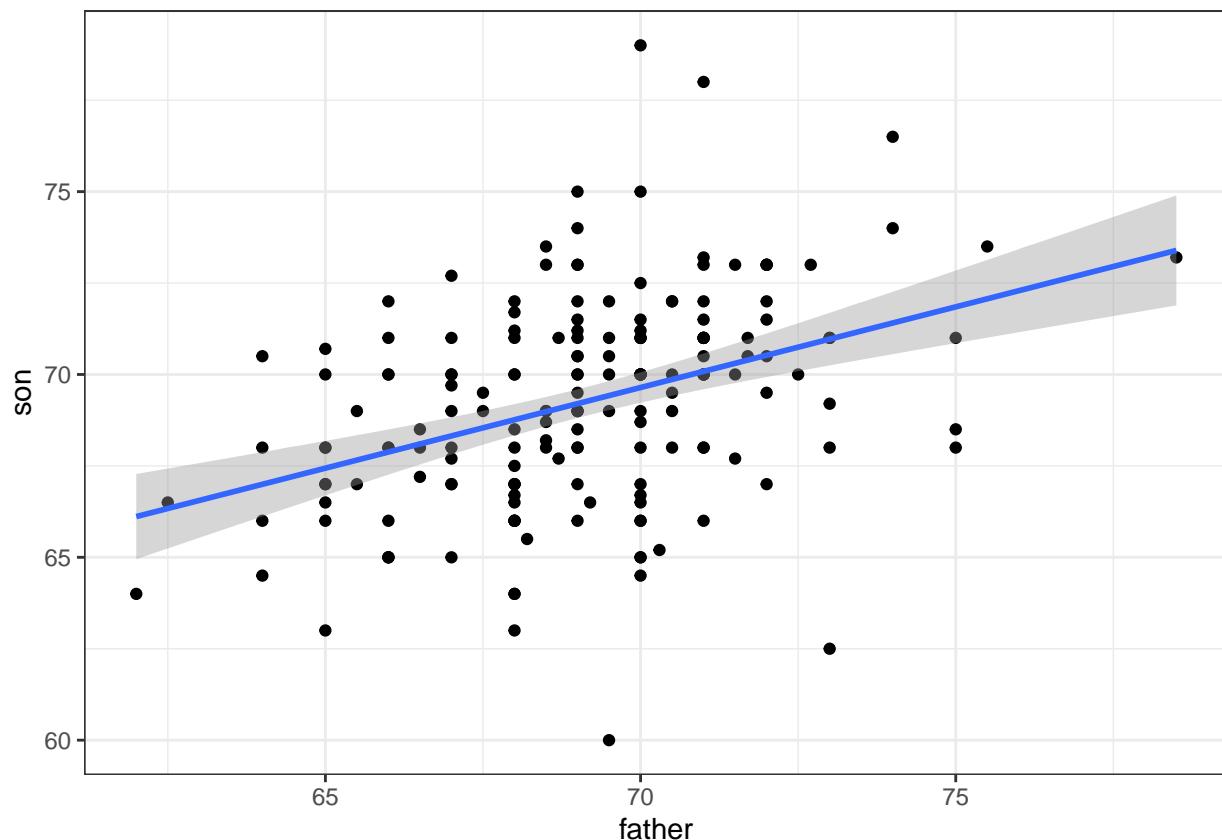
$$\frac{\hat{\beta}_k}{\text{SE}(\hat{\beta}_k)} \sim t_{n-p}$$

Por otro lado, los valores p prueban la hipótesis nula  $H_0 : \beta_k = 0$ .

Así, una vez ajustado nuestro modelo, podemos obtener predicciones sobre Y sustituyendo valores. Nótese que el argumento “method =”lm de “geom\_smooth()” ya incluye la graficación de intervalos de confianza de los estimadores:

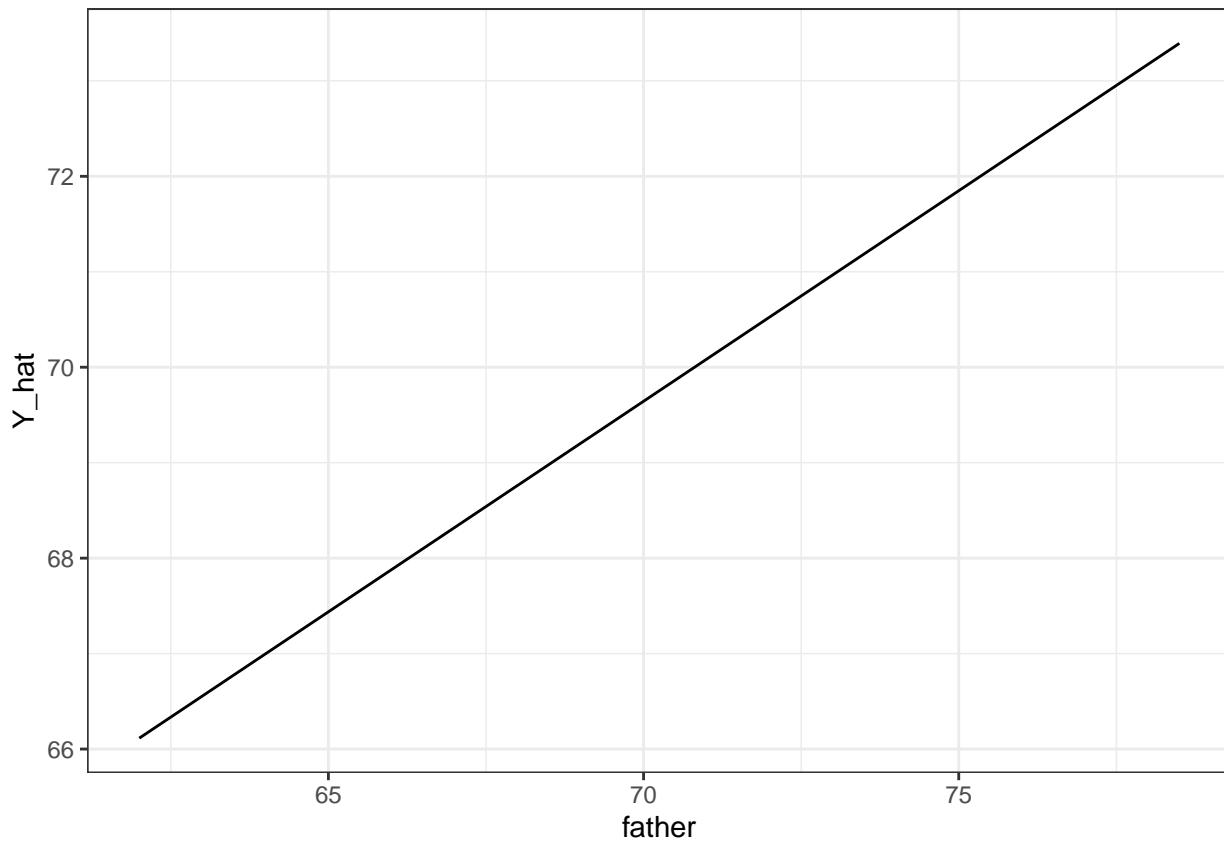
```
galton_heights %>%
  ggplot(aes(father, son)) +
  geom_point() +
  geom_smooth(method = "lm")

## `geom_smooth()` using formula 'y ~ x'
```



La función “predict()” toma a lm como input y arroja predicciones sobre Y:

```
galton_heights %>%
  mutate(Y_hat = predict(lm(son~father, data = .))) %>%
  ggplot(aes(father, Y_hat)) +
  geom_line()
```



Además, podemos extraer información como los errores estándar, a partir de la cual se construyen intervalos de confianza, de la función “predict()”:

```
fit <- galton_heights %>%
  lm(son~father, data = .)

Y_hat <- predict(fit, se.fit = TRUE)

names(Y_hat)

## [1] "fit"           "se.fit"        "df"            "residual.scale"
```

### 7.2.3. Assessment 14

Load the Lahman library and filter the Teams data frame to the years 1961-2001. Run a linear model in R predicting the number of runs per game based on both the number of bases on balls per game and the number of home runs per game. What is the coefficient for bases on balls?

```
library(Lahman)

Teams %>%
  filter(yearID %in% 1961:2001) %>%
  mutate(BB_per_game = BB/G, HR_per_game = HR/G, R_per_game = R/G) %>%
  lm(R_per_game ~ BB_per_game + HR_per_game, data = .)
```

```
##
## Call:
## lm(formula = R_per_game ~ BB_per_game + HR_per_game, data = .)
##
## Coefficients:
## (Intercept) BB_per_game HR_per_game
##           1.744        0.387        1.561
```

We run a Monte Carlo simulation where we repeatedly take samples of  $N = 100$  from the Galton heights data and compute the regression slope coefficients for each sample:

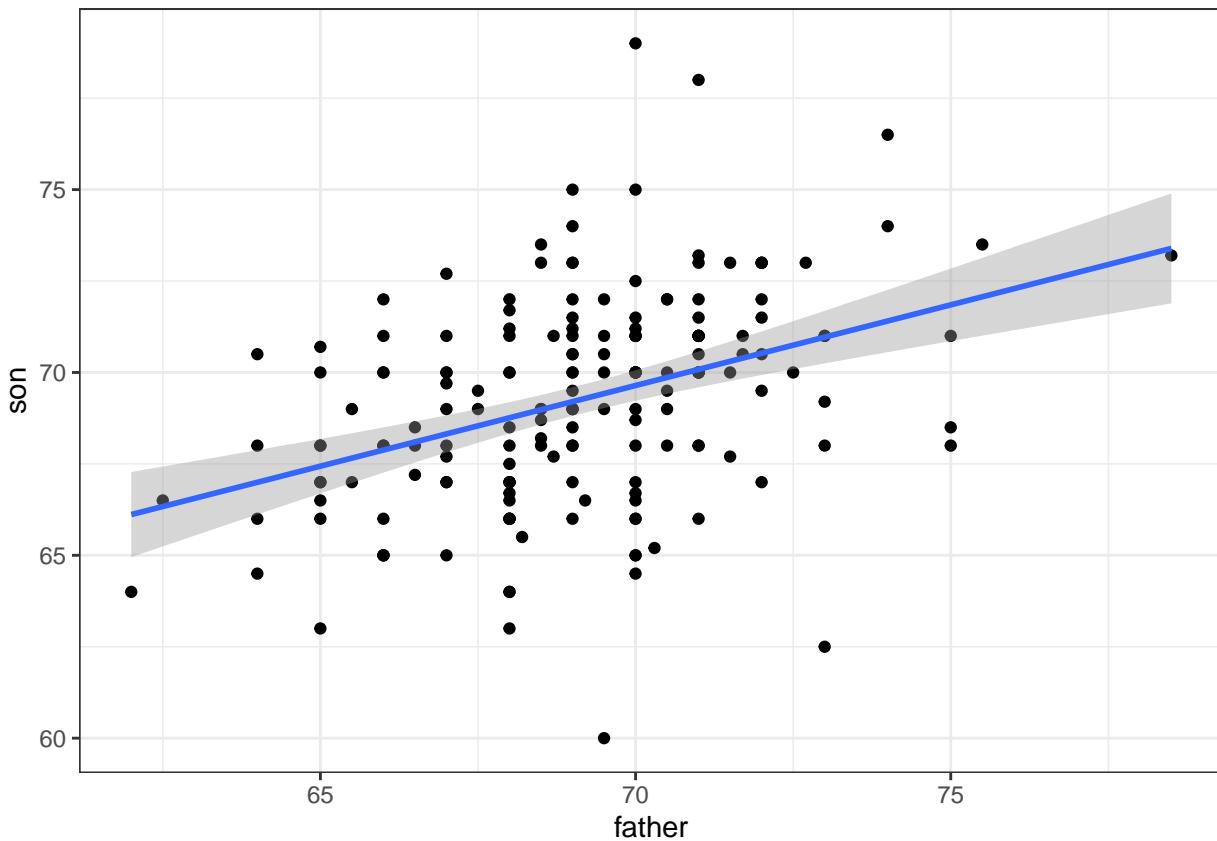
```
B <- 1000
N <- 100
lse <- replicate(B, {
  sample_n(galton_heights, N, replace = TRUE) %>%
  lm(son ~ father, data = .) %>% .$coef
})

lse <- data.frame(beta_0 = lse[, 1], beta_1 = lse[, 2])
```

Which R code(s) below would properly plot the predictions and confidence intervals for our linear model of sons' heights?

```
#1
galton_heights %>% ggplot(aes(father, son)) +
  geom_point() +
  geom_smooth(method = "lm")

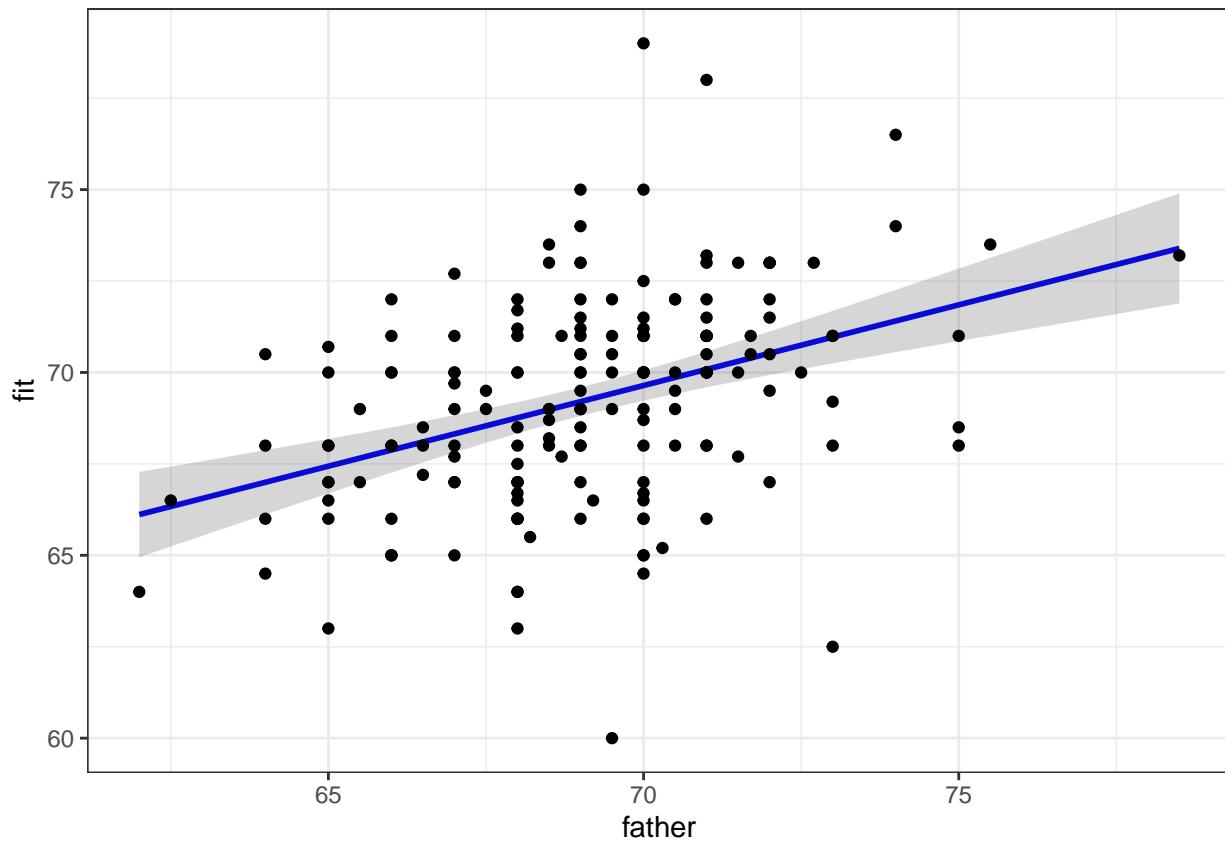
## `geom_smooth()` using formula 'y ~ x'
```



```
#2
model <- lm(son ~ father, data = galton_heights)
predictions <- predict(model, interval = c("confidence"), level = 0.95)
data <- as.tibble(predictions) %>% bind_cols(father = galton_heights$father)

## Warning: `as.tibble()` was deprecated in tibble 2.0.0.
## Please use `as_tibble()` instead.
## The signature and semantics have changed, see `?as_tibble`.

ggplot(data, aes(x = father, y = fit)) +
  geom_line(color = "blue", size = 1) +
  geom_ribbon(aes(ymin=lwr, ymax=upr), alpha=0.2) +
  geom_point(data = galton_heights, aes(x = father, y = son))
```



In Questions 7 and 8, you'll look again at female heights from GaltonFamilies. Define female\_heights, a set of mother and daughter heights sampled from GaltonFamilies, as follows:

```
set.seed(1989) #if you are using R 3.5 or earlier
set.seed(1989, sample.kind="Rounding") #if you are using R 3.6 or later
```

```
## Warning in set.seed(1989, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
library(HistData)
data("GaltonFamilies")
options(digits = 3)      # report 3 significant digits

female_heights <- GaltonFamilies %>%
  filter(gender == "female") %>%
  group_by(family) %>%
  sample_n(1) %>%
  ungroup() %>%
  select(mother, childHeight) %>%
  rename(daughter = childHeight)
```

Fit a linear regression model predicting the mothers' heights using daughters' heights.

```
fit <- lm(mother~daughter, data =female_heights)

fit
##
```

```

## Call:
## lm(formula = mother ~ daughter, data = female_heights)
##
## Coefficients:
## (Intercept)    daughter
##           44.18          0.31

```

Predict mothers' heights using the model. What is the predicted height of the first mother in the dataset?

```

predict(fit, interval = c("confidence"), level = 0.95)[1,]

##   fit lwr upr
## 65.6 64.9 66.3
female_heights[1,]

## # A tibble: 1 x 2
##   mother daughter
##     <dbl>    <dbl>
## 1       67      69

```

We have shown how BB and singles have similar predictive power for scoring runs. Another way to compare the usefulness of these baseball metrics is by assessing how stable they are across the years. Because we have to pick players based on their previous performances, we will prefer metrics that are more stable. In these exercises, we will compare the stability of singles and BBs.

Before we get started, we want to generate two tables: one for 2002 and another for the average of 1999-2001 seasons. We want to define per plate appearance statistics, keeping only players with more than 100 plate appearances. Here is how we create the 2002 table:

```

library(Lahman)

bat_02 <- Batting %>%
  filter(yearID == 2002) %>%
  mutate(pa = AB + BB, singles = (H - X2B - X3B - HR)/pa, bb = BB/pa) %>%
  filter(pa >= 100) %>%
  select(playerID, singles, bb)

```

Now compute a similar table but with rates computed over 1999-2001. Keep only rows from 1999-2001 where players have 100 or more plate appearances, calculate each player's single rate and BB rate per stint (where each row is one stint - a player can have multiple stints within a season), then calculate the average single rate (mean\_singles) and average BB rate (mean\_bb) per player over the three year period. How many players had a single rate mean\_singles of greater than 0.2 per plate appearance over 1999-2001?

```

bat_9901 <- Batting %>%
  filter(yearID %in% 1999:2001) %>%
  mutate(pa = AB + BB, singles = (H - X2B - X3B - HR)/pa, bb = BB/pa) %>%filter(pa >= 100) %>%
  select(playerID, singles, bb)

bat_9901 <- bat_9901 %>%
  group_by(playerID) %>%
  summarise(mean_singles=mean(singles),mean_bb=mean(bb))

```

```
bat_9901 %>%
  filter(mean_singles >.2) %>%
  nrow()
```

```
## [1] 46
```

```
bat_9901 %>%
  filter(mean_bb >.2) %>%
  nrow()
```

```
## [1] 3
```

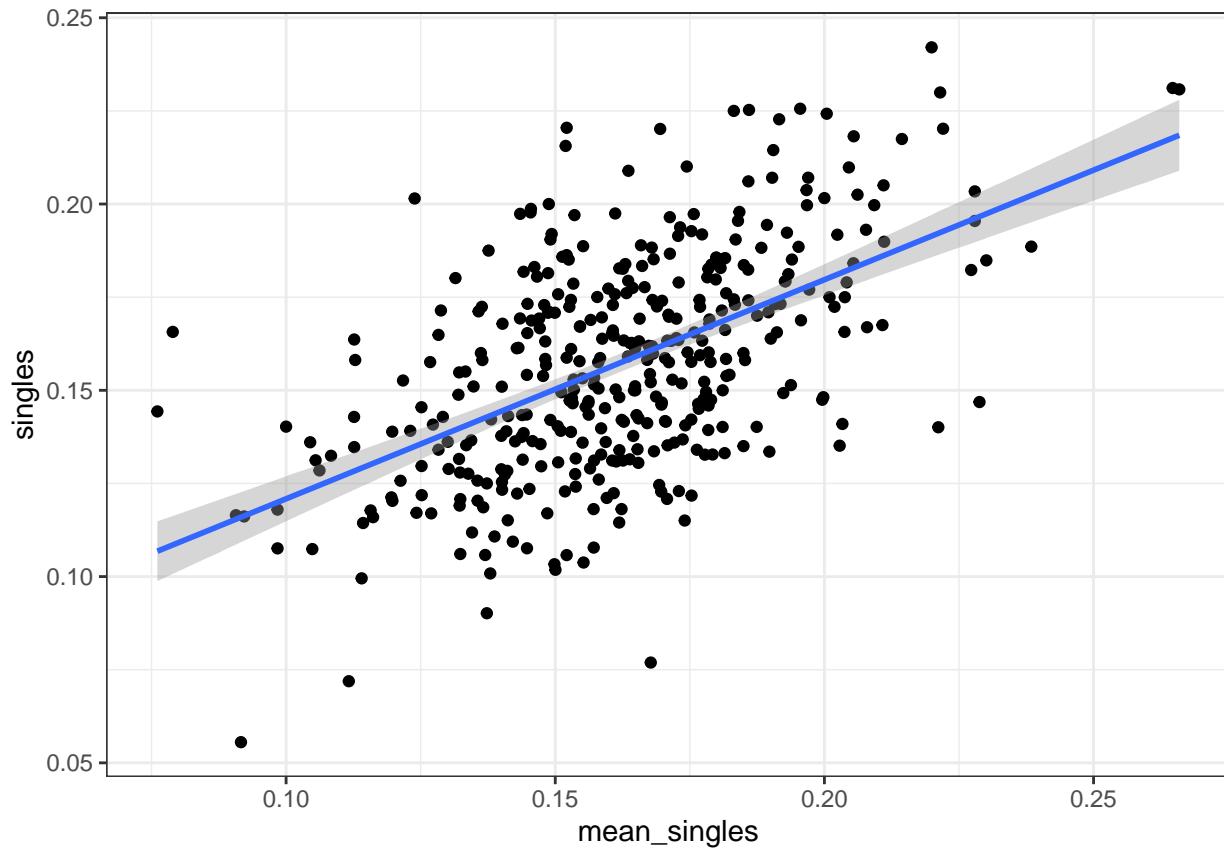
Use `inner_join()` to combine the `bat_02` table with the table of 1999-2001 rate averages you created in the previous question. What is the correlation between 2002 singles rates and 1999-2001 average singles rates?

```
new_bat <- inner_join(bat_02,bat_9901,by="playerID")  
  
new_bat %>%
  summarise(singles_r=cor(singles,mean_singles),bb_r=cor(bb,mean_bb))
```

```
##   singles_r   bb_r
## 1      0.551 0.717
```

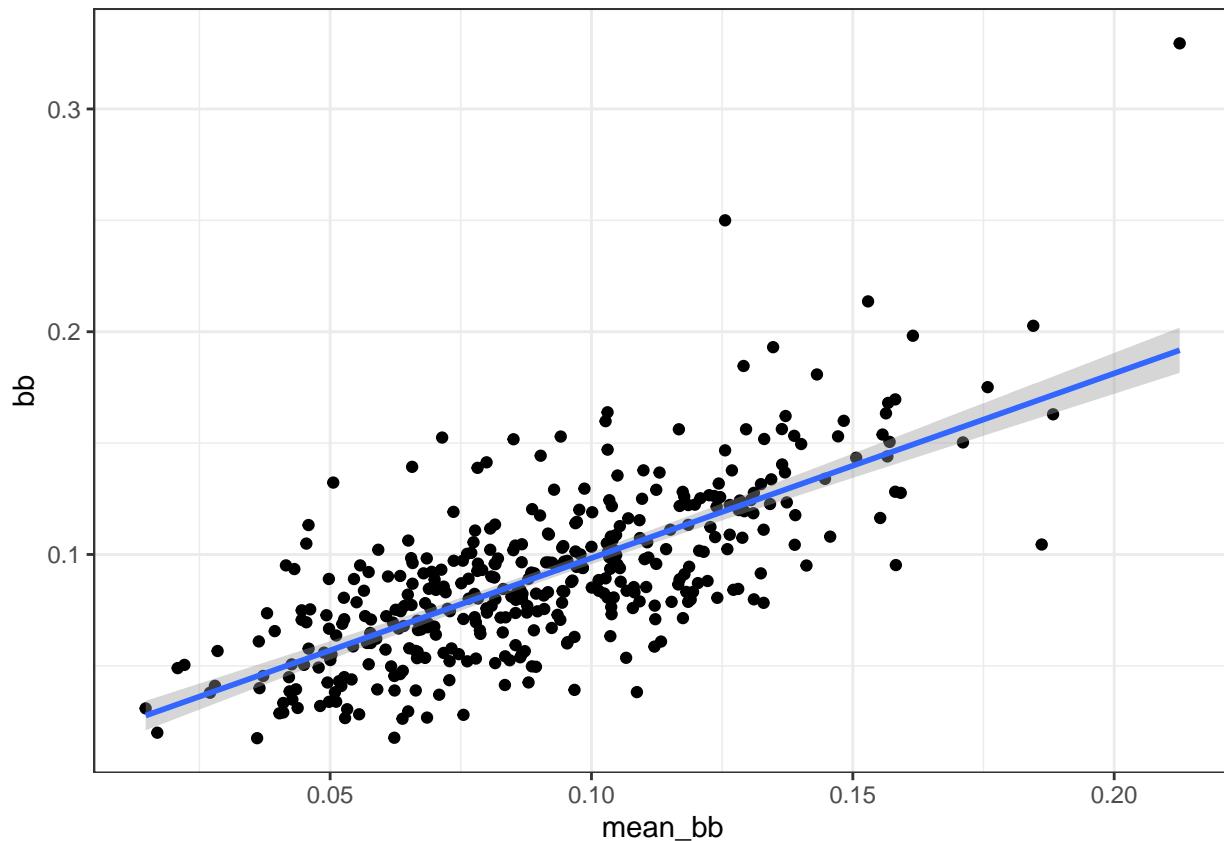
Make scatterplots of `mean_singles` versus `singles` and `mean_bb` versus `bb`.

```
new_bat %>%
  ggplot(aes(mean_singles, singles)) +
  geom_point() +
  geom_smooth(method = "lm")  
  
## `geom_smooth()` using formula 'y ~ x'
```



```
new_bat %>%
  ggplot(aes(mean_bb, bb)) +
  geom_point() +
  geom_smooth(method = "lm")

## `geom_smooth()` using formula 'y ~ x'
```



Fit a linear model to predict 2002 singles given 1999-2001 mean\_singles.

```
new_bat %>%
  lm(singles~mean_singles, data =.)
```

```
##
## Call:
## lm(formula = singles ~ mean_singles, data = .)
##
## Coefficients:
## (Intercept)  mean_singles
##          0.0621      0.5881
```

Fit a linear model to predict 2002 bb given 1999-2001 mean\_bb.

```
new_bat %>%
  lm(bb~mean_bb, data =.)
```

```
##
## Call:
## lm(formula = bb ~ mean_bb, data = .)
##
## Coefficients:
## (Intercept)      mean_bb
##          0.0155      0.8290
```

### 7.2.4. Tibbles, do y broom

**7.2.4.1. Tibbles** Retómese el ejemplo de béisbol, donde construimos la siguiente base de datos:

```
library(Lahman)

library(tidyverse)

dat <- Teams %>%
  filter(yearID %in% 1961:2001) %>%
  mutate(HR = round(HR/G, 1),
        BB = BB/G,
        R = R/G) %>%
  select(HR, BB, R) %>%
  filter(HR >= 0.4 & HR <=1.2)
```

Así, calculamos la pendiente de las líneas de regresión para predecir carreras por bases por bola en diferentes estratos de *home runs*:

```
dat %>%
  group_by(HR) %>%
  summarize(slope = cor(BB, R)*sd(R)/sd(BB))

## # A tibble: 9 x 2
##       HR     slope
##   <dbl>    <dbl>
## 1 0.4  0.734
## 2 0.5  0.566
## 3 0.6  0.412
## 4 0.7  0.285
## 5 0.8  0.365
## 6 0.9  0.261
## 7 1    0.512
## 8 1.1  0.454
## 9 1.2  0.440
```

Donde observamos que las pendientes son bastante similares y las diferencias probablemente se deben a variación aleatoria. Para reforzar este argumento, podemos construir intervalos de confianza para cada pendiente..

Si usamos la función “lm()” para esto, no obtenemos lo que buscamos, dado que esta no funciona con *tibbles* agrupadas:

```
dat %>%
  group_by(HR) %>%
  lm(R ~ BB, data = .) %>%
  .$coef

## (Intercept)      BB
## 2.198        0.638
```

Como ejemplo, considérese el siguiente resultado de una agrupación:

```
dat %>%
  group_by(HR) %>%
  head()

## # A tibble: 6 x 3
## # Groups:   HR [5]
```

```

##      HR      BB      R
##    <dbl> <dbl> <dbl>
## 1   0.9   3.56  4.24
## 2   0.7   3.97  4.47
## 3   0.8   3.37  4.69
## 4   1.1   3.46  4.42
## 5   1     2.75  4.61
## 6   0.9   3.06  4.58

dat %>%
  group_by(HR) %>%
  class()

## [1] "grouped_df" "tbl_df"       "tbl"          "data.frame"

```

Una *tibble* es un tipo especial de *data frame*, como las arrojadas con las funciones “`group_by()`” y “`summarize()`”: de hecho, la primera arroja una *tibble* agrupada, “`grouped_df`”. Por su parte, funciones como “`select()`”, “`filter()`”, “`mutate()`” y “`arrange()`” arrojarán el objeto que toman (*data frame* → *data frame* y *tibble* → *tibble*).

**Nota:** las *tibbles* son la forma de organización de datos predeterminada del tidyverse.

### ¿Cuáles son las diferencias entre una *tibble* y un *data frame*?

1. Las *tibbles* despliegan mejor la información:

```
Teams[1:10, 1:5]
```

```

##   yearID lgID teamID franchID divID
## 1   1871   NA    BS1     BNA   <NA>
## 2   1871   NA    CH1     CNA   <NA>
## 3   1871   NA    CL1     CFC   <NA>
## 4   1871   NA    FW1     KEK   <NA>
## 5   1871   NA    NY2     NNA   <NA>
## 6   1871   NA    PH1     PNA   <NA>
## 7   1871   NA    RC1     ROK   <NA>
## 8   1871   NA    TRO     TRO   <NA>
## 9   1871   NA    WS3     OLY   <NA>
## 10  1872   NA    BL1     BLC   <NA>

as_tibble(Teams[1:10, 1:5])

```

```

## # A tibble: 10 x 5
##   yearID lgID teamID franchID divID
##   <int> <fct> <fct>   <fct>   <chr>
## 1   1871 NA    BS1     BNA     <NA>
## 2   1871 NA    CH1     CNA     <NA>
## 3   1871 NA    CL1     CFC     <NA>
## 4   1871 NA    FW1     KEK     <NA>
## 5   1871 NA    NY2     NNA     <NA>
## 6   1871 NA    PH1     PNA     <NA>
## 7   1871 NA    RC1     ROK     <NA>
## 8   1871 NA    TRO     TRO     <NA>
## 9   1871 NA    WS3     OLY     <NA>
## 10  1872 NA    BL1     BLC     <NA>

```

2. Si se hace un subconjunto de un *data frame*, podríamos no obtener un *data frame*. Si se hace un subconjunto de una *tibble*, siempre obtendremos una *tibble*:

```
class(Teams[,20])
## [1] "integer"
class(as_tibble(Teams[,20]))
## [1] "tbl_df"     "tbl"        "data.frame"

3. Si queremos acceder a una columna que no existe, tibble nos avisará, mientras que un data frame no:
```

```
Teams$hr
```

```
## NULL
as_tibble(Teams)$hr

## Warning: Unknown or uninitialized column: `hr`.

## NULL
```

4. Las *tibbles* pueden tener entradas complejas, como listas o funciones:

```
tibble(id = c(1, 2, 3), func = c(mean, median, sd))

## # A tibble: 3 x 2
##       id   func
##   <dbl> <list>
## 1     1 <fn>
## 2     2 <fn>
## 3     3 <fn>
```

5. Las *tibbles* pueden ser agrupadas.

**7.2.4.2. do** A continuación describiremos la función “do()”. Como mencionamos, las funciones de tidyverse saben cómo interpretar las *tibbles* agrupadas, además que, mediante el operador de pipa, arroja *data frames*. Por su parte, la mayoría de las funciones base de R no reconocen *tibbles* ni arrojan *data frames*, como es el caso de la función “lm()”.

Así, la función “do()” sirve como puente entre las funciones base de R y el tidyverse: siempre reconoce *tibbles* y siempre arroja *data frames*.

Usemos la función “do()” para ajustar una línea de regresión para cada estrato de *home runs*:

```
dat %>%
  group_by(HR) %>%
  do(pendiente = lm(R~BB, data = .))

## # A tibble: 9 x 2
## # Rowwise:
##       HR   pendiente
##   <dbl> <lm>
## 1  0.4  0.4 <lm>
## 2  0.5  0.5 <lm>
## 3  0.6  0.6 <lm>
## 4  0.7  0.7 <lm>
## 5  0.8  0.8 <lm>
## 6  0.9  0.9 <lm>
## 7  1.0  1.0 <lm>
## 8  1.1  1.1 <lm>
## 9  1.2  1.2 <lm>
```

Nótese que hay que nombrar la columna, la cual contiene objetos creados a partir de “lm()”, lo cual no es muy útil. Para corregir nuestro *data frame*, el resultado de la función dentro de “do()” debe ser también un *data frame*. Así, primero crearemos una función para extraer la pendiente de “lm()”:

```
get_slope <- function(data){
  fit <- lm(R ~ BB, data = data)
  data.frame(slope = fit$coefficients[2],
             se = summary(fit)$coefficient[2,2])
}
```

Para finalmente usar “do()” y extraer los datos deseados (pendiente y error estándar):

```
dat %>%
  group_by(HR) %>%
  do(get_slope(.))
```

```
## # A tibble: 9 x 3
## # Groups:   HR [9]
##       HR slope     se
##   <dbl> <dbl> <dbl>
## 1 0.4  0.734 0.208
## 2 0.5  0.566 0.110
## 3 0.6  0.412 0.0974
## 4 0.7  0.285 0.0705
## 5 0.8  0.365 0.0653
## 6 0.9  0.261 0.0751
## 7 1    0.512 0.0751
## 8 1.1  0.454 0.0855
## 9 1.2  0.440 0.0801
```

Nótese que no se nombraron las columnas, dado que los datos utilizados ya eran un *data frame*. Si las nombramos, obtenemos una *tibble* bastante compleja:

```
dat %>%
  group_by(HR) %>%
  do(slope = get_slope(.))

## # A tibble: 9 x 2
## # Rowwise:
##       HR slope
##   <dbl> <list>
## 1 0.4 <df [1 x 2]>
## 2 0.5 <df [1 x 2]>
## 3 0.6 <df [1 x 2]>
## 4 0.7 <df [1 x 2]>
## 5 0.8 <df [1 x 2]>
## 6 0.9 <df [1 x 2]>
## 7 1   <df [1 x 2]>
## 8 1.1 <df [1 x 2]>
## 9 1.2 <df [1 x 2]>
```

Si, por ejemplo, el *data frame* resultante tiene más de un renglón, se concatenará apropiadamente:

```
get_lse <- function(data){
  fit <- lm(R ~ BB, data = data)
  data.frame(term = names(fit$coefficients),
             slope = fit$coefficients,
             se = summary(fit)$coefficient[,2])
```

```

}

dat %>%
  group_by(HR) %>%
  do(get_lse(.))

## # A tibble: 18 x 4
## # Groups:   HR [9]
##       HR term      slope     se
##       <dbl> <chr>    <dbl>  <dbl>
## 1 0.4 (Intercept) 1.36  0.631
## 2 0.4 BB          0.734  0.208
## 3 0.5 (Intercept) 2.01  0.344
## 4 0.5 BB          0.566  0.110
## 5 0.6 (Intercept) 2.53  0.305
## 6 0.6 BB          0.412  0.0974
## 7 0.7 (Intercept) 3.21  0.225
## 8 0.7 BB          0.285  0.0705
## 9 0.8 (Intercept) 3.07  0.213
## 10 0.8 BB         0.365  0.0653
## 11 0.9 (Intercept) 3.54  0.251
## 12 0.9 BB         0.261  0.0751
## 13 1 (Intercept)  2.88  0.256
## 14 1 BB           0.512  0.0751
## 15 1.1 (Intercept) 3.21  0.300
## 16 1.1 BB          0.454  0.0855
## 17 1.2 (Intercept) 3.40  0.291
## 18 1.2 BB          0.440  0.0801

```

**7.2.4.3. broom** En ocasiones, lo anterior puede ser complicado: la librería broom lo facilita. Esta librería tiene tres funciones principales, las cuales extraen información de los objetos creados por “lm()” y los arrojan en un *data frame* adecuado para tidyverse:

1. `tidy()`: arroja los estimadores y relacina la información en un *data frame*

```

library(broom)

fit <- lm(R ~ BB, data = dat)

tidy(fit)

## # A tibble: 2 x 5
##   term      estimate std.error statistic p.value
##   <chr>    <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)  2.20     0.113     19.4 1.12e-70
## 2 BB          0.638     0.0344    18.5 1.35e-65

```

Pueden agregarse datos que nos interesen como intervalos de confianza:

```

tidy(fit, conf.int = TRUE)

## # A tibble: 2 x 7
##   term      estimate std.error statistic p.value conf.low conf.high
##   <chr>    <dbl>     <dbl>     <dbl>    <dbl>    <dbl>    <dbl>
## 1 (Intercept)  2.20     0.113     19.4 1.12e-70    1.98     2.42
## 2 BB          0.638     0.0344    18.5 1.35e-65    0.570    0.705

```

Así, podemos encadenar inmediatamente con “do()”

```
dat %>%
  group_by(HR) %>%
  do(tidy(lm(R~BB, data = .), conf.int = TRUE))

## # A tibble: 18 x 8
## # Groups:   HR [9]
##       HR term      estimate std.error statistic p.value conf.low conf.high
##     <dbl> <chr>      <dbl>     <dbl>     <dbl>    <dbl>     <dbl>     <dbl>
## 1 0.4 (Intercept)  1.36      0.631     2.16  4.05e- 2  0.0631    2.66
## 2 0.4 BB           0.734     0.208     3.54  1.54e- 3  0.308     1.16
## 3 0.5 (Intercept)  2.01      0.344     5.84  2.07e- 7  1.32      2.69
## 4 0.5 BB           0.566     0.110     5.14  3.02e- 6  0.346     0.786
## 5 0.6 (Intercept)  2.53      0.305     8.32  2.43e-13 1.93      3.14
## 6 0.6 BB           0.412     0.0974    4.23  4.80e- 5  0.219     0.605
## 7 0.7 (Intercept)  3.21      0.225    14.3   1.49e-30 2.76      3.65
## 8 0.7 BB           0.285     0.0705    4.05  7.93e- 5  0.146     0.425
## 9 0.8 (Intercept)  3.07      0.213    14.4   5.40e-31 2.65      3.49
## 10 0.8 BB          0.365     0.0653    5.59  9.13e- 8  0.236     0.494
## 11 0.9 (Intercept) 3.54      0.251    14.1   8.77e-29 3.05      4.04
## 12 0.9 BB          0.261     0.0751    3.47  6.85e- 4  0.112     0.409
## 13 1 (Intercept)   2.88      0.256    11.3   6.62e-21 2.37      3.39
## 14 1 BB            0.512     0.0751    6.81  3.28e-10  0.363     0.660
## 15 1.1 (Intercept) 3.21      0.300    10.7   6.46e-17 2.61      3.81
## 16 1.1 BB          0.454     0.0855    5.31  1.03e- 6  0.284     0.624
## 17 1.2 (Intercept) 3.40      0.291    11.7   2.33e-16 2.81      3.98
## 18 1.2 BB          0.440     0.0801    5.50  1.07e- 6  0.280     0.601
```

Además de poder encadenar con “filter()” y “select()” también, para mostrar lo que queremos, una tabla con las pendientes y los valores mínimos y máximos de los intervalos de confianza para cada estrato:

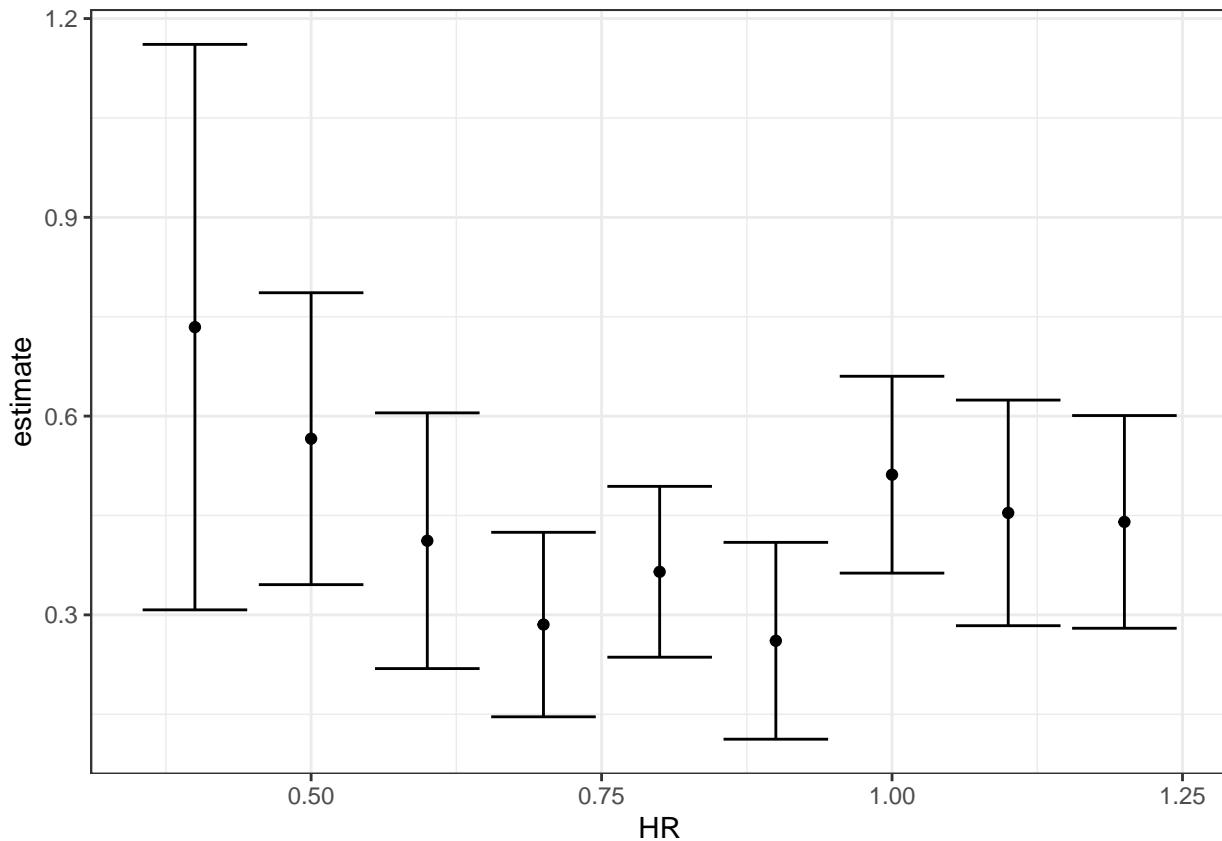
```
dat %>%
  group_by(HR) %>%
  do(tidy(lm(R ~ BB, data = .), conf.int = TRUE)) %>%
  filter(term == "BB") %>%
  select(HR, estimate, conf.low, conf.high)

## # A tibble: 9 x 4
## # Groups:   HR [9]
##       HR estimate conf.low conf.high
##     <dbl>     <dbl>    <dbl>     <dbl>
## 1 0.4      0.734    0.308     1.16
## 2 0.5      0.566    0.346     0.786
## 3 0.6      0.412    0.219     0.605
## 4 0.7      0.285    0.146     0.425
## 5 0.8      0.365    0.236     0.494
## 6 0.9      0.261    0.112     0.409
## 7 1        0.512    0.363     0.660
## 8 1.1      0.454    0.284     0.624
## 9 1.2      0.440    0.280     0.601
```

Además, una tabla de esta manera hace muy fácil la visualización mediante ggplot:

```
dat %>%
  group_by(HR) %>%
  do(tidy(lm(R ~ BB, data = .), conf.int = TRUE)) %>%
```

```
filter(term == "BB") %>%
  select(HR, estimate, conf.low, conf.high) %>%
  ggplot(aes(HR, y = estimate, ymin = conf.low, ymax = conf.high)) +
  geom_errorbar() +
  geom_point()
```



Donde puede verse que los intervalos se traslanan, lo que fortalece el supuesto de que la pendiente no cambian con la estratificación de *home runs*.

2. `glance()`: resultados específicos del modelo:

```
glance(fit)
```

```
## # A tibble: 1 x 12
##   r.squared adj.r.squared sigma statistic p.value    df logLik    AIC    BIC
##       <dbl>          <dbl>  <dbl>     <dbl> <dbl> <dbl> <dbl> <dbl>
## 1     0.266        0.265  0.454     343. 1.35e-65     1 -596. 1199. 1214.
## # ... with 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```

3. `augment()`: resultados de observaciones específicas

### 7.2.5. Assessment 15

We have investigated the relationship between fathers' heights and sons' heights. But what about other parent-child relationships? Does one parent's height have a stronger association with child height? How does the child's gender affect this relationship in heights? Are any differences that we observe statistically significant?

The galton dataset is a sample of one male and one female child from each family in the GaltonFamilies dataset. The pair column denotes whether the pair is father and daughter, father and son, mother and daughter, or mother and son.

Create the galton dataset using the code below:

```
library(tidyverse)
library(HistData)

data("GaltonFamilies")

set.seed(1, sample.kind = "Rounding")

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used

galton <- GaltonFamilies %>%
  group_by(family, gender) %>%
  sample_n(1) %>%
  ungroup() %>%
  gather(parent, parentHeight, father:mother) %>%
  mutate(child = ifelse(gender == "female", "daughter", "son")) %>%
  unite(pair, c("parent", "child"))

galton[1:10,]

## # A tibble: 10 x 8
##   family midparentHeight children childNum gender childHeight pair
##   <fct>          <dbl>     <int>    <int> <fct>      <dbl> <chr>
## 1 001            75.4       4        2 female      69.2 father_daughter
## 2 001            75.4       4        1 male       73.2 father_son
## 3 002            73.7       4        4 female      65.5 father_daughter
## 4 002            73.7       4        2 male       72.5 father_son
## 5 003            72.1       2        2 female      68.0 father_daughter
## 6 003            72.1       2        1 male       71.0 father_son
## 7 004            72.1       5        5 female      63.0 father_daughter
## 8 004            72.1       5        2 male       68.5 father_son
## 9 005            69.1       6        5 female      62.5 father_daughter
## 10 005           69.1       6        1 male       72.0 father_son
## # ... with 1 more variable: parentHeight <dbl>
```

Group by pair and summarize the number of observations in each group. How many father-daughter pairs are in the dataset?

```
table(galton$pair)

##
##   father_daughter   father_son   mother_daughter   mother_son
##                 176          179             176          179
```

Calculate the correlation coefficients for fathers and daughters, fathers and sons, mothers and daughters and mothers and sons. Which pair has the strongest/weakest correlation in heights?

```
f_d <- subset(galton, pair == "father_daughter")

f_s <- subset(galton, pair == "father_son")

m_d <- subset(galton, pair == "mother_daughter")
```

```
m_s <- subset(galton, pair == "mother_son")

cor(f_d$childHeight, f_d$parentHeight)

## [1] 0.401

cor(f_s$childHeight, f_s$parentHeight)

## [1] 0.43

cor(m_d$childHeight, m_d$parentHeight)

## [1] 0.383

cor(m_s$childHeight, m_s$parentHeight)

## [1] 0.343
```

Use `lm()` and the `broom` package to fit regression lines for each parent-child pair type. Compute the least squares estimates, standard errors, confidence intervals and p-values for the `parentHeight` coefficient for each pair.

What is the estimate of the father-daughter coefficient?

```
f_d_fit <- lm(childHeight ~ parentHeight, data = f_d)

f_d_fit

## 
## Call:
## lm(formula = childHeight ~ parentHeight, data = f_d)
## 
## Coefficients:
## (Intercept) parentHeight
##          40.134          0.345
```

For every 1-inch increase in mother's height, how many inches does the typical son's height increase?

```
m_s_fit <- lm(childHeight ~ parentHeight, data = m_s)

m_s_fit

## 
## Call:
## lm(formula = childHeight ~ parentHeight, data = m_s)
## 
## Coefficients:
## (Intercept) parentHeight
##          44.878          0.381
```

Which sets of parent-child heights are significantly correlated at a p-value cut off of .05?

```
cor.test(f_d$childHeight, f_d$parentHeight)$p.value

## [1] 3.56e-08

cor.test(f_s$childHeight, f_s$parentHeight)$p.value

## [1] 1.94e-09
```

```
cor.test(m_d$childHeight, m_d$parentHeight)$p.value  
## [1] 1.56e-07  
cor.test(m_s$childHeight, m_s$parentHeight)$p.value  
## [1] 2.59e-06
```

### 7.2.6. Regresiones y baseball

Al tratar de responder qué tan bien explican las bases por bola a las carreras, la exploración de datos nos llevó al siguiente modelo:

$$E[R|BB = x_1, HR = x_2] = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

Dado que lo anterior se distribuye aproximadamente normal, podemos derivar un modelo lineal de la siguiente manera:

$$Y_i = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \varepsilon_i$$

Donde  $Y_i$ : carreras por juego,  $x_1$ : bases por bola por juego y  $x_2$ : *home runs* por juego. A continuación se muestra el código para ajustar el modelo de regresión simple:

```
library(tidyverse)
library(broom)
library(kableExtra)
library(Lahman)
library(stargazer)

fit <- Teams %>%
  filter(yearID %in% 1961:2001) %>%
  mutate(BB = BB/G, HR = HR/G, R = R/G) %>%
  lm(R ~ BB + HR, data = .)
```

Recordemos que con “tidy()” podemos visualizar un resumen sintético:

```
tidy(fit, conf.int = TRUE)
```

```
## # A tibble: 3 x 7
##   term      estimate std.error statistic p.value conf.low conf.high
##   <chr>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
## 1 (Intercept) 1.74     0.0824    21.2 7.62e- 83     1.58     1.91
## 2 BB          0.387    0.0270    14.3 1.20e- 42     0.334    0.440
## 3 HR          1.56     0.0490    31.9 1.78e-155    1.47     1.66
```

**Nota:** El paquete stargazer permite construir tablas de regresión sumamente atractivas:

Modelo ajustado de Carreras vs. bases por bola y home runs

Con dos variables explicativas	
	Carreras Modelo 1
Bases por bola	0.387*** (0.027)
Home runs	1.560*** (0.049)
Constant	1.740*** (0.082)
Observations	1,026
R <sup>2</sup>	0.650
Adjusted R <sup>2</sup>	0.650
Residual Std. Error	0.348 (df = 1023)
F Statistic	951.000*** (df = 2; 1023)
Nivel de significancia (P-valor)	*p<0.1; **p<0.05; ***p<0.01

No obstante, para elegir un jugador, también tenemos que tomar en cuenta el número de *singles*, *doubles* y *triples* conectados. Asumiremos (arriesgadamente) que todas estas variables se distribuyen conjuntamente de manera normal. Así, este nuevo modelo está definido como:

$$Y_i = \beta_0 + \beta_1 x_{i,1} - \beta_2 x_{i,2} + \beta_3 x_{i,3} + \beta_4 x_{i,4} + \beta_5 x_{i,5} + \varepsilon_i$$

```

fit <- Teams %>%
  filter(yearID %in% 1961:2001) %>%
  mutate(BB = BB/G,
        singles = (H-X2B-X3B-HR)/G,
        doubles = X2B/G,
        triples = X3B/G,
        HR = HR/G,
        R = R/G) %>%
  lm(R ~ BB + singles + doubles + triples + HR, data = .)

coefs <- tidy(fit, conf.int = TRUE)

coefs

## # A tibble: 6 x 7
##   term      estimate std.error statistic  p.value conf.low conf.high
##   <chr>     <dbl>     <dbl>     <dbl>    <dbl>    <dbl>    <dbl>
## 1 (Intercept) -2.77     0.0862   -32.1 4.76e-157   -2.94    -2.60
## 2 BB          0.371     0.0117    31.6 1.87e-153    0.348    0.394
## 3 singles     0.519     0.0127    40.8 8.67e-217    0.494    0.544
## 4 doubles     0.771     0.0226    34.1 8.44e-171    0.727    0.816
## 5 triples     1.24      0.0768    16.1 2.12e- 52    1.09     1.39
## 6 HR          1.44      0.0243    59.3 0            1.40     1.49

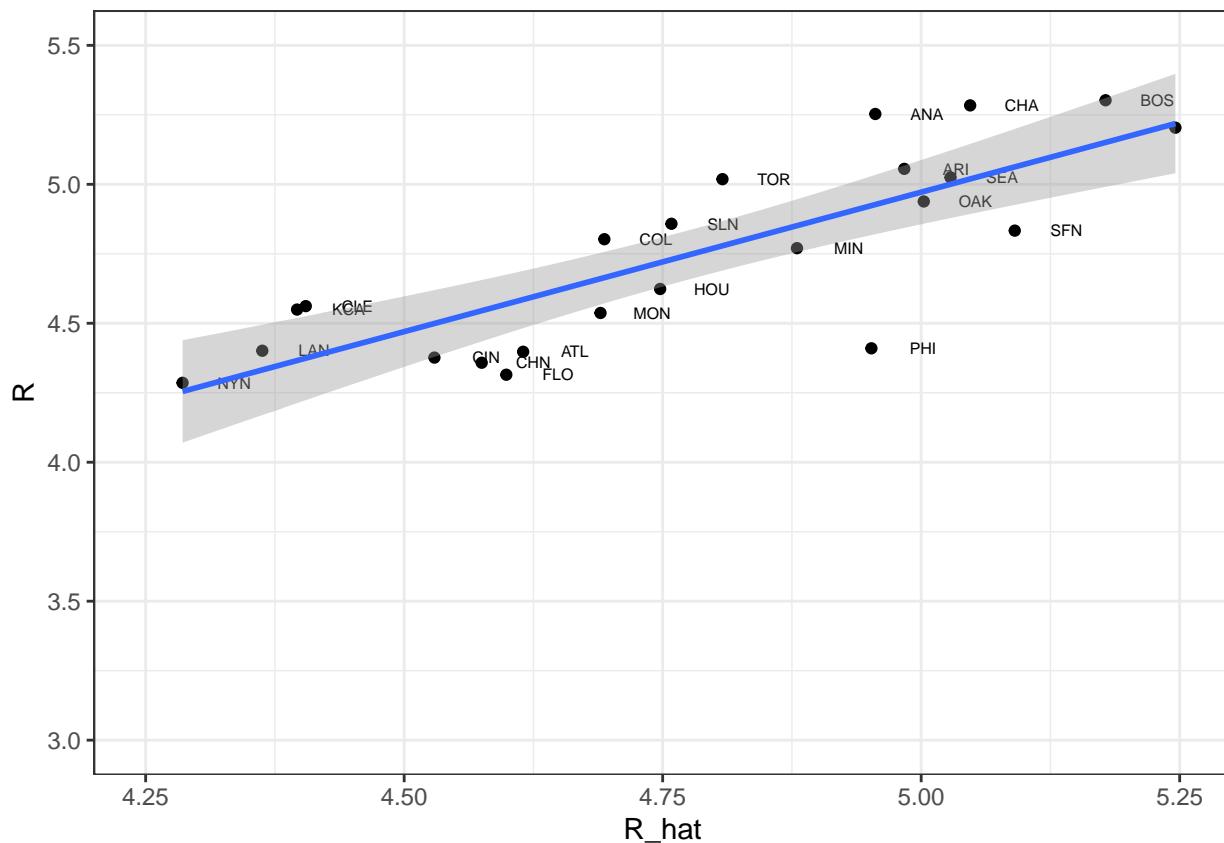
```

Para comprobar la fiabilidad de nuestro modelo, predeciremos las carreras para cada equipo en 2002 mediante la función “predict()”:

```

Teams %>%
  filter(yearID %in% 2002) %>%
  mutate(BB = BB/G,
        singles = (H-X2B-X3B-HR)/G,
        doubles = X2B/G,
        triples = X3B/G,
        HR = HR/G,
        R = R/G) %>%
  mutate(R_hat = predict(fit, newdata = .)) %>%
  ggplot(aes(R_hat, R, label = teamID)) +
  ylim(3,5.5) +
  xlim(4.25, 5.25) +
  geom_point() +
  geom_text(nudge_x = 0.05, cex = 2) +
  geom_smooth(method = "lm")

```



Modelo ajustado de Carreras vs. bases por bola, hits (singles, doubles y triples) y home runs

Con cinco variables explicativas	
	Carreras Modelo 2
Bases por bola	0.371*** (0.012)
Singles	0.519*** (0.013)
Doubles	0.771*** (0.023)
Triples	1.240*** (0.077)
Home runs	1.440*** (0.024)
Constant	-2.770*** (0.086)
Observations	1,026
R <sup>2</sup>	0.936
Adjusted R <sup>2</sup>	0.935
Residual Std. Error	0.150 (df = 1020)
F Statistic	2,961.000*** (df = 5; 1020)
Nivel de significancia (P-valor)	*p<0.1; **p<0.05; ***p<0.01

Así, la línea de regresión está definida como:

$$Y_i = -2,770 + 0,371BB + 0,519singles + 0,771doubles + 1,240triples + 1,440HR$$

Nótese que hemos calculado el análisis para los equipos, no jugadores. Sin embargo, existe una estadística que nos puede ayudar porque toma en cuenta las oportunidades por jugador: la *per-plate-appearance rate*. Para poder comparar la tasa del equipo con la del jugador, computemos el número promedio de *team plate appearance per game* con el siguiente código:

```

pa_per_game <- Batting %>%
  filter(yearID == 2002) %>%
  group_by(teamID) %>%
  summarize(pa_per_game = sum(AB + BB)/max(G)) %>%
  .$pa_per_game %>%
  mean

pa_per_game

## [1] 38.7

```

Ahora estamos listo para utilizar nuestra métrica y computar los *per-plate-appearance rates* para jugadores en 2002 utilizando datos de 1999-2001 (recuérdese que estamos simulando la elección de un jugador en 2002):

```

players <- Batting %>%
  filter(yearID %in% 1999:2001) %>%
  group_by(playerID) %>%
  mutate(PA = BB + AB) %>%
  summarize(G = sum(PA)/pa_per_game,
            BB = sum(BB)/G,
            singles = sum(H-X2B-X3B-HR)/G,
            doubles= sum(X2B)/G,
            triples = sum(X3B)/G,
            HR = sum(HR)/G,
            AVG = sum(H)/sum(AB),
            PA = sum(PA)) %>%
  filter(PA >= 300) %>%
  select(-G) %>%
  mutate(R_hat = predict(fit, newdata = .))

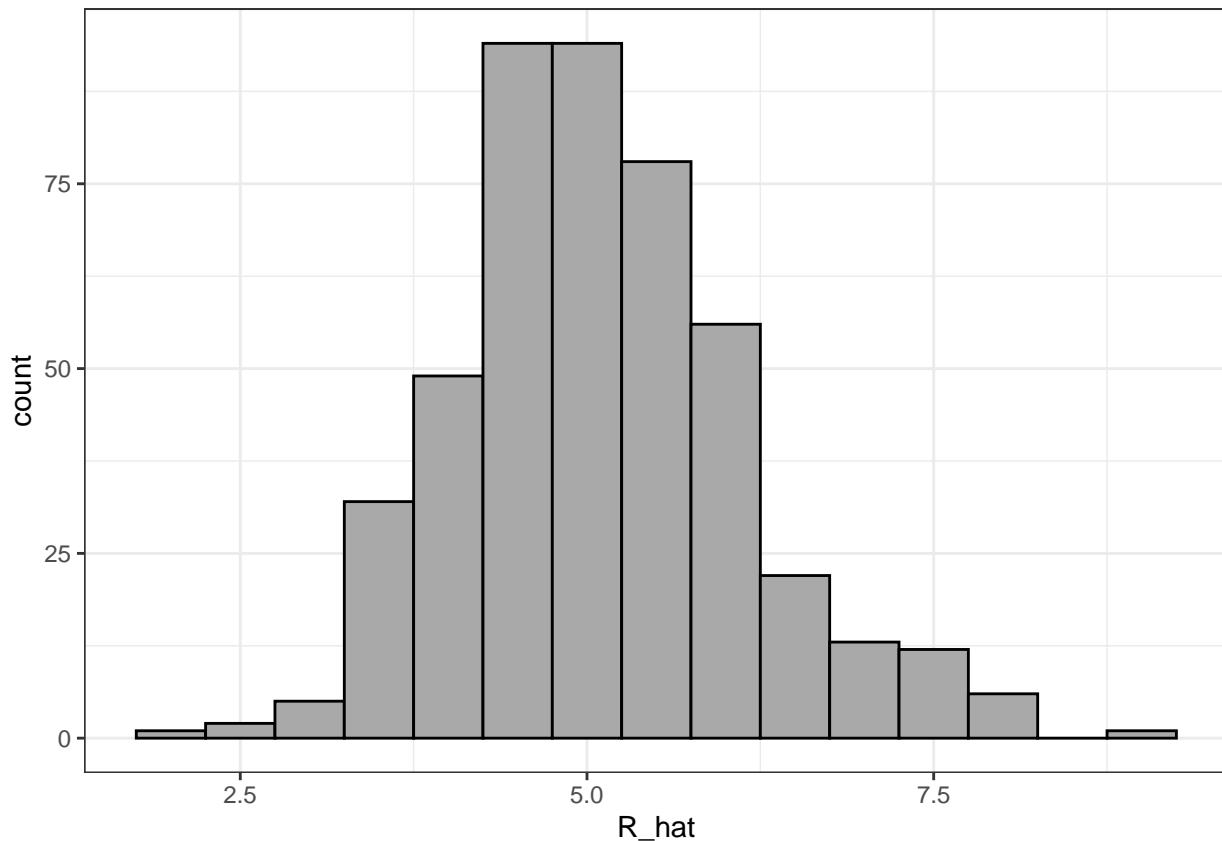
```

Donde la interpretación es la siguiente: cada estadística representa el número de carreras que un equipo anotaría a partir de la estadística correspondiente si todos los jugadores fueran como el jugador en cuestión. La distribución muestra una variación amplia entre jugadores, como se ve en el gráfico siguiente:

```

players %>%
  ggplot(aes(R_hat)) +
  geom_histogram(binwidth = 0.5, color = "black", fill = "darkgray")

```



Además, es importante conocer los salarios de los jugadores, dado que suponemos un presupuesto limitado, así como las posiciones de los jugadores.

Primero, añádanse los salarios para 2002:

```
players <- Salaries %>%
  filter(yearID == 2002) %>%
  select(playerID, salary) %>%
  right_join(players, by = "playerID")
```

Ahora, añadamos las posiciones defensivas (dado que los jugadores juegan más de una posición al año, tomaremos aquella que juegan más con la función “top\_n()”; para asegurar que solo tomamos un jugador por posición, y en caso de empate, tomaremos la primera fila; finalmente, removeremos a los *outfielders*, la cual es una generalización de 3 posiciones y a los *pitchers*):

```
players <- Fielding %>%
  filter(yearID == 2002) %>%
  filter(!POS %in% c("OF", "P")) %>%
  group_by(playerID) %>%
  top_n(1, G) %>%
  filter(row_number(G)==1) %>%
  ungroup() %>%
  select(playerID, POS) %>%
  right_join(players, by = "playerID") %>%
  filter(!is.na(POS) & !is.na(salary))
```

Finalmente, añadiremos nombres y apellidos:

```
players <- Master %>%
  select(playerID, nameFirst, nameLast, debut) %>%
  right_join(players, by = "playerID")
```

Así, ya tenemos una tabla con nuestra predicción de carreras para cada jugador:

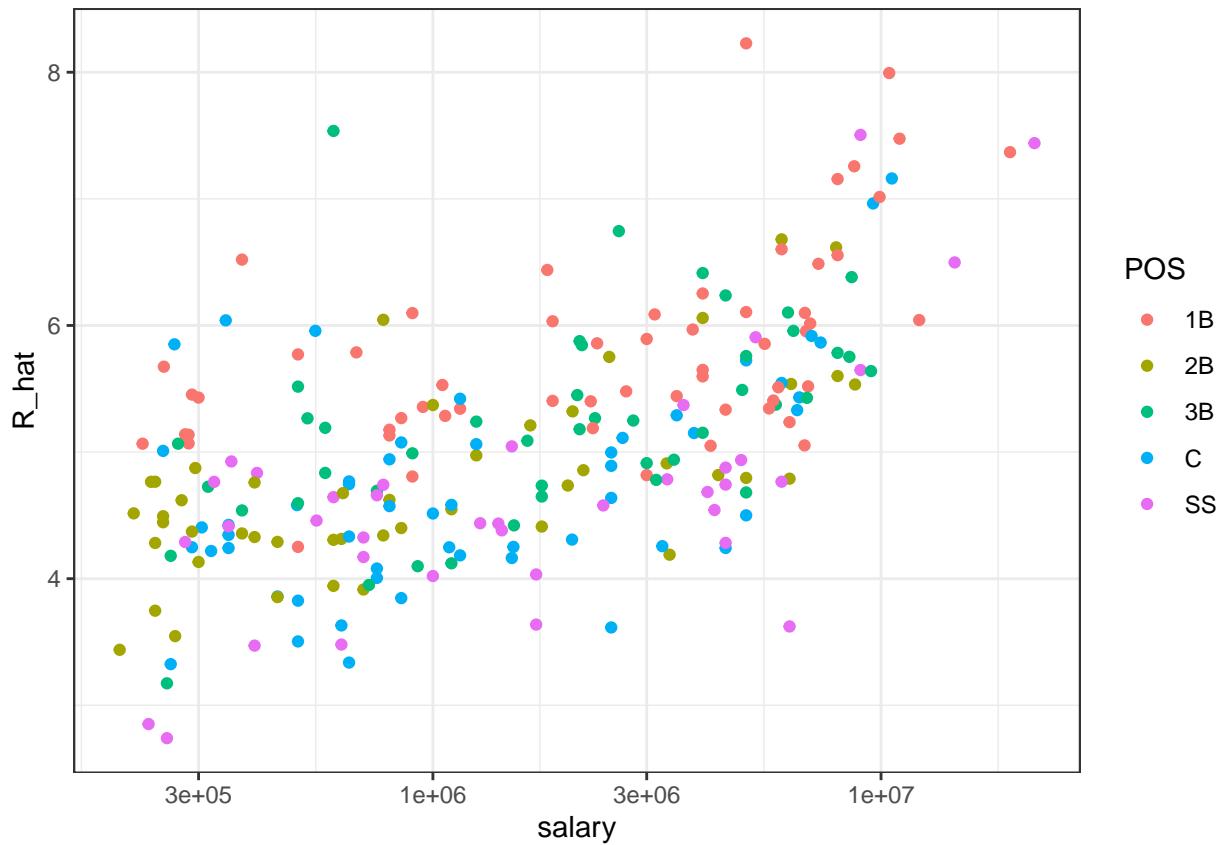
```
players %>%
  select(nameFirst, nameLast, POS, salary, R_hat) %>%
  arrange(desc(R_hat)) %>%
  top_n(10)
```

```
## Selecting by R_hat

##   nameFirst   nameLast POS    salary R_hat
## 1      Todd     Helton  1B 5000000  8.23
## 2     Jason    Giambi  1B 10428571  7.99
## 3    Albert    Pujols  3B  600000  7.54
## 4    Nomar Garciaparra SS 9000000  7.51
## 5      Jeff   Bagwell  1B 11000000  7.48
## 6      Alex Rodriguez SS 22000000  7.44
## 7     Carlos   Delgado  1B 19400000  7.37
## 8    Rafael  Palmeiro  1B  8712986  7.26
## 9      Mike   Piazza   C 10571429  7.16
## 10     Jim     Thome  1B 8000000  7.16
```

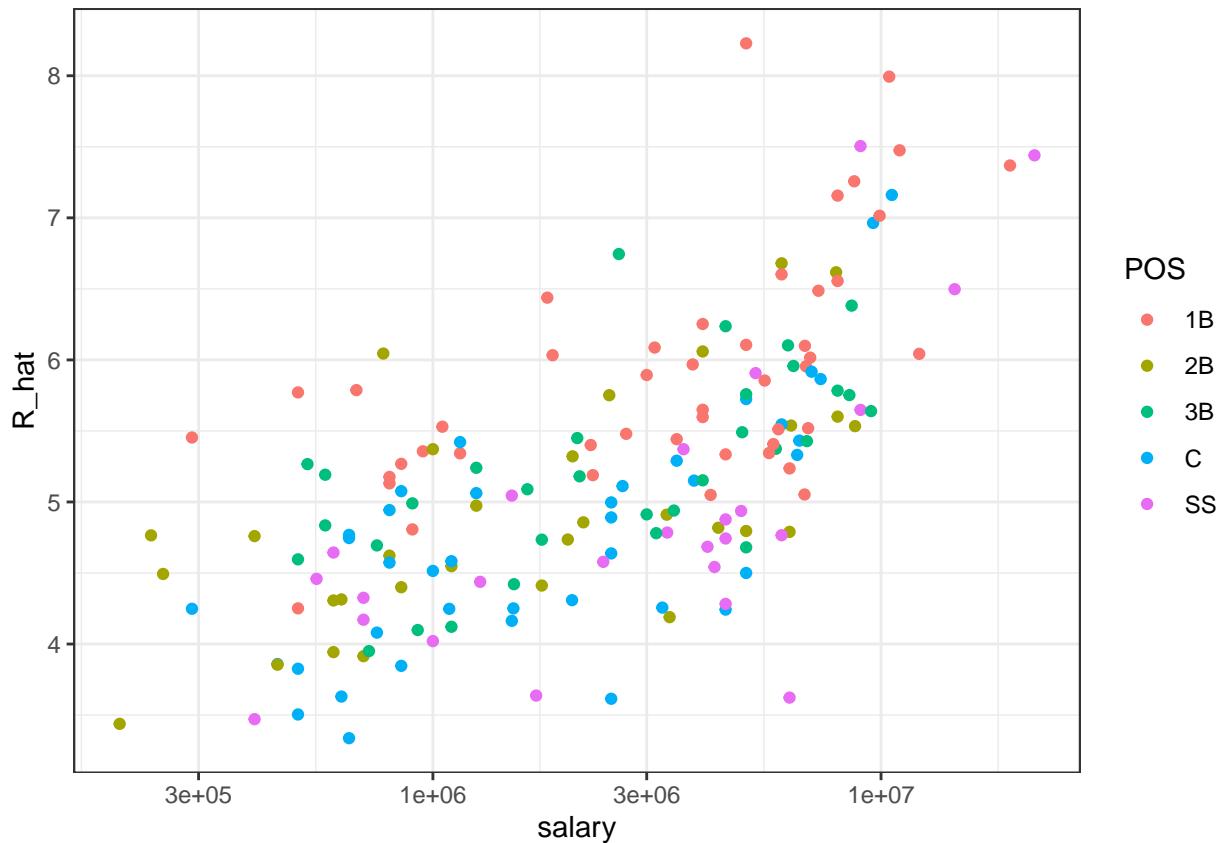
Donde podemos comprobar que los jugadores con una predicción alta de carreras tienen salarios altos:

```
players %>%
  ggplot(aes(salary, R_hat, color = POS)) +
  geom_point() +
  scale_x_log10()
```



Sin embargo, se observan algunos jugadores con altas predicciones y de bajo costo, lo cual sería idóneo para nuestro equipo. Lamentablemente, muchos de estos son *rookies* por lo que no son elegibles para renegociar un contrato en 2002. Así, obsérvese la gráfica de jugadores que debutaron antes de 1997:

```
players %>%
  filter(debut < 1998) %>%
  ggplot(aes(salary, R_hat, color = POS)) +
  geom_point() +
  scale_x_log10()
```



**Nota:** Una manera de escoger a nuestros jugadores es mediante **programación lineal** con el siguiente procedimiento:

```
library(reshape2)
```

```
## Warning: package 'reshape2' was built under R version 4.0.5
##
## Attaching package: 'reshape2'
## The following object is masked from 'package:tidyverse':
##   smiths
library(lpSolve)

players <- players %>%
  filter(debut <= "1997-01-01" & debut > "1988-01-01")

constraint_matrix <- acast(players, POS ~ playerID, fun.aggregate = length)

## Using R_hat as value column: use value.var to override.
npos <- nrow(constraint_matrix)

constraint_matrix <- rbind(constraint_matrix, salary = players$salary)

constraint_dir <- c(rep("==", npos), "<=")
```

```
constraint_limit <- c(rep(1, npos), 50*10^6)

lp_solution <- lp("max", players$R_hat,
                  constraint_matrix, constraint_dir, constraint_limit,
                  all.int = TRUE)
```

Donde el algoritmo escoge los siguiente nueve jugadores:

```
our_team <- players %>%
  filter(lp_solution$solution == 1) %>%
  arrange(desc(R_hat))

our_team %>% select(nameFirst, nameLast, POS, salary, R_hat)

##   nameFirst   nameLast POS    salary R_hat
## 1 Jason      Giambi  1B 10428571  7.99
## 2 Nomar      GarciaParra SS 9000000  7.51
## 3 Mike       Piazza   C 10571429  7.16
## 4 Phil       Nevin   3B 2600000  6.75
## 5 Jeff       Kent    2B 6000000  6.68
```

Nótese que estos jugadores tienen una estadística de BB y HR por encima del promedio, aunque no para *singles*:

```
my_scale <- function(x)(x - median(x))/mad(x)
players %>%
  mutate(BB = my_scale(BB),
        singles = my_scale(singles),
        doubles = my_scale(doubles),
        triples = my_scale(triples),
        HR = my_scale(HR),
        AVG = my_scale(AVG),
        R_hat = my_scale(R_hat)) %>%
  filter(playerID %in% our_team$playerID) %>%
  select(nameFirst, nameLast, BB, singles, doubles, triples, HR, AVG, R_hat) %>%
  arrange(desc(R_hat))

##   nameFirst   nameLast BB singles doubles triples     HR    AVG R_hat
## 1 Jason      Giambi  3.118 -0.5382   0.908 -0.633 1.905 2.496  3.67
## 2 Nomar      GarciaParra 0.154  1.6114   3.141  0.467 0.917 3.795  3.06
## 3 Mike       Piazza   0.459 -0.0811  -0.188 -1.266 2.476 1.584  2.63
## 4 Phil       Nevin   0.649 -0.6745   0.809 -1.076 1.938 0.989  2.11
## 5 Jeff       Kent    0.732 -0.2916   2.178  1.176 0.883 1.548  2.03
```

**7.2.6.1. Falacia de regresión** El *sophomore slump* se conoce como la baja de rendimiento de atletas o estudiantes durante su segundo año de juego o estudio, comparado con el primero. Ahora, ¿los datos confirman la existencia de este fenómeno? Primero, crearemos una tabla con el playerID, nombres y la posición más jugada de los *rookies*:

```
library(Lahman)
```

```
playerInfo <- Fielding %>%
  group_by(playerID) %>%
  arrange(desc(G)) %>%
  slice(1) %>%
  ungroup %>%
```

```
left_join(Master, by="playerID") %>%
  select(playerID, nameFirst, nameLast, POS)
```

Ahora, construiremos una tabla con solo los ganadores del premio *Rookie of the Year* (sin contar *Pitchers*, los cuales no reciben ningún premio) y añadiremos sus estadísticas de bateo:

```
ROY <- AwardsPlayers %>%
  filter(awardID == "Rookie of the Year") %>%
  left_join(playerInfo, by = "playerID") %>%
  rename(rookie_year = yearID) %>%
  right_join(Batting, by = "playerID") %>%
  mutate(AVG = H/AB) %>%
  filter(POS != "P")
```

Ahora, filtraremos solo a los *rookies* en su temporada de *sophomores*:

```
ROY <- ROY %>%
  filter(yearID == rookie_year | yearID == rookie_year+1) %>%
  group_by(playerID) %>%
  mutate(rookie = ifelse(yearID == min(yearID), "rookie", "sophomore")) %>%
  filter(n() == 2) %>%
  ungroup %>%
  select(playerID, rookie_year, rookie, nameFirst, nameLast, AVG)
```

Finalmente, usaremos “`spread()`” para separar por columnas a *rookies* y *sophomores*:

```
ROY <- ROY %>%
  spread(rookie, AVG) %>%
  arrange(desc(rookie))
```

Si se observa con cuidado, parece ser que el *sophomore slump* efectivamente ocurre. ¿Por qué ocurre esto? Considérense las temporadas 2013 y 2014 y los jugadores que batearon al menos 130 veces (mínimo requerido para ganar el ROY):

```
two_years <- Batting %>%
  filter(yearID %in% 2013:2014) %>%
  group_by(playerID, yearID) %>%
  filter(sum(AB) >= 130) %>%
  summarize(AVG = sum(H)/sum(AB)) %>%
  ungroup %>%
  spread(yearID, AVG) %>%
  filter(!is.na(`2013`) & !is.na(`2014`)) %>%
  left_join(playerInfo, by="playerID") %>%
  filter(POS!="P") %>%
  select(-POS) %>%
  arrange(desc(`2013`)) %>%
  select(nameFirst, nameLast, `2013`, `2014`)
```

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

```
two_years[1:10,]
```

```
## # A tibble: 10 x 4
##   nameFirst nameLast `2013` `2014`
##   <chr>     <chr>    <dbl>   <dbl>
## 1 Miguel    Cabrera    0.348   0.313
## 2 Hanley   Ramirez    0.345   0.283
```

```
## 3 Michael Cuddyer 0.331 0.332
## 4 Scooter Gennett 0.324 0.289
## 5 Joe Mauer 0.324 0.277
## 6 Mike Trout 0.323 0.287
## 7 Chris Johnson 0.321 0.263
## 8 Freddie Freeman 0.319 0.288
## 9 Yasiel Puig 0.319 0.296
## 10 Yadier Molina 0.319 0.282
```

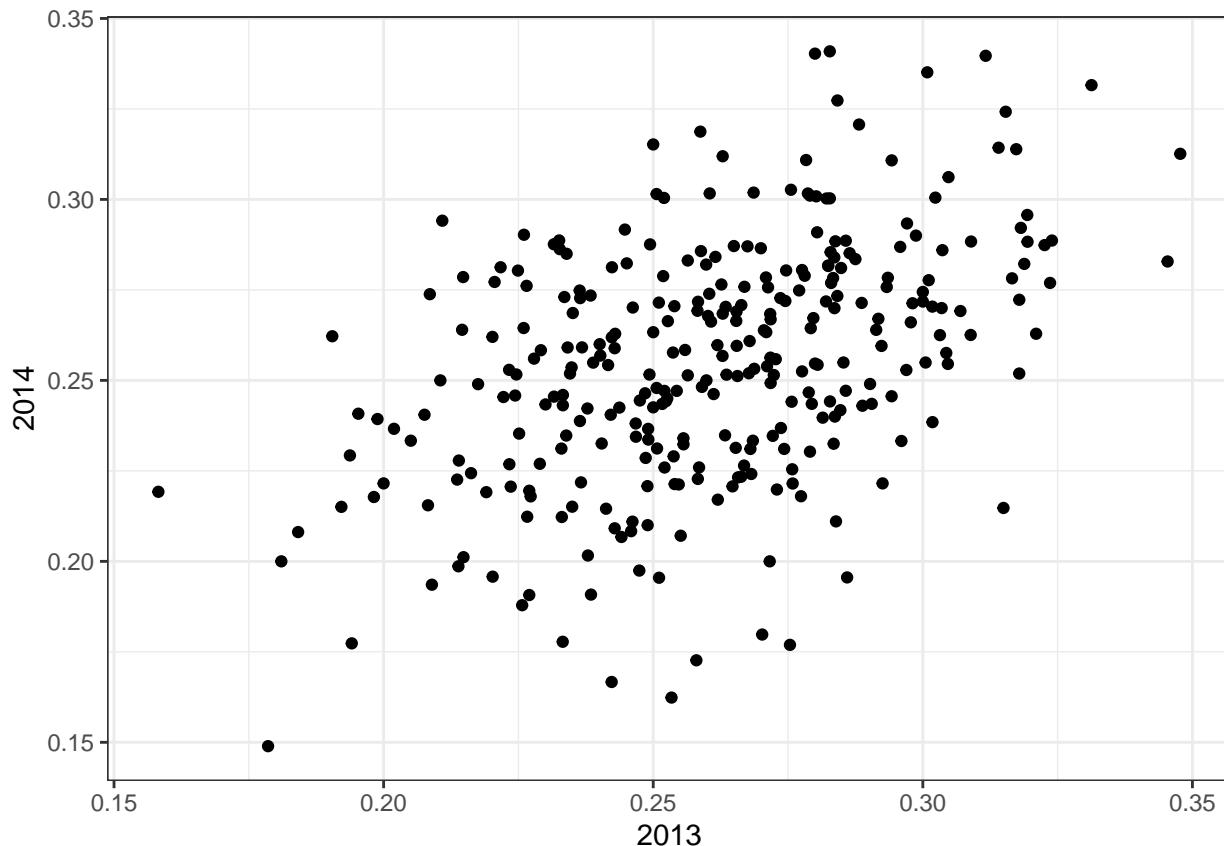
Puede verse que los jugadores con mejores estadísticas también disminuyeron su rendimiento el siguiente año. Nótese también qué ocurrió con los peores rendimiento en 2013:

```
arrange(two_years, `2013`)[1:10,]
```

```
## # A tibble: 10 x 4
##   nameFirst nameLast `2013` `2014`
##   <chr>     <chr>    <dbl>   <dbl>
## 1 Danny     Espinosa  0.158   0.219
## 2 Dan       Uggla     0.179   0.149
## 3 Jeff      Mathis    0.181   0.2
## 4 B. J.     Upton     0.184   0.208
## 5 Adam      Rosales   0.190   0.262
## 6 Aaron     Hicks     0.192   0.215
## 7 Chris     Colabello 0.194   0.229
## 8 J. P.     Arencibia  0.194   0.177
## 9 Tyler     Flowers   0.195   0.241
## 10 Ryan     Hanigan   0.198   0.218
```

Así, la estadística concluye que realmente no existe ningún *sophomore slump*, dado que la correlación del rendimiento en dos años separados es alta pero no perfecta; a continuación se muestran los rendimientos de 2013 y 2014:

```
two_years %>%
  ggplot(aes(`2013`, `2014`)) +
  geom_point()
```



Donde existe una relación positiva, pero no perfecta:

```
summarize(two_years, cor(`2013`, `2014`))
```

```
## # A tibble: 1 x 1
##   `cor(`2013`, `2014`)`<dbl>
## 1 0.460
```

En conclusión, como la correlación no es perfecta, la regresión nos dice que, en promedio, esperamos que los mejores jugadores de 2013 rindan ligeramente peor en 2014. Este fenómeno se conoce como **regresión a la media** y ocurre cuando una variable es extrema en una primera medición y tiende a estar más cerca de la media en su segunda medición.

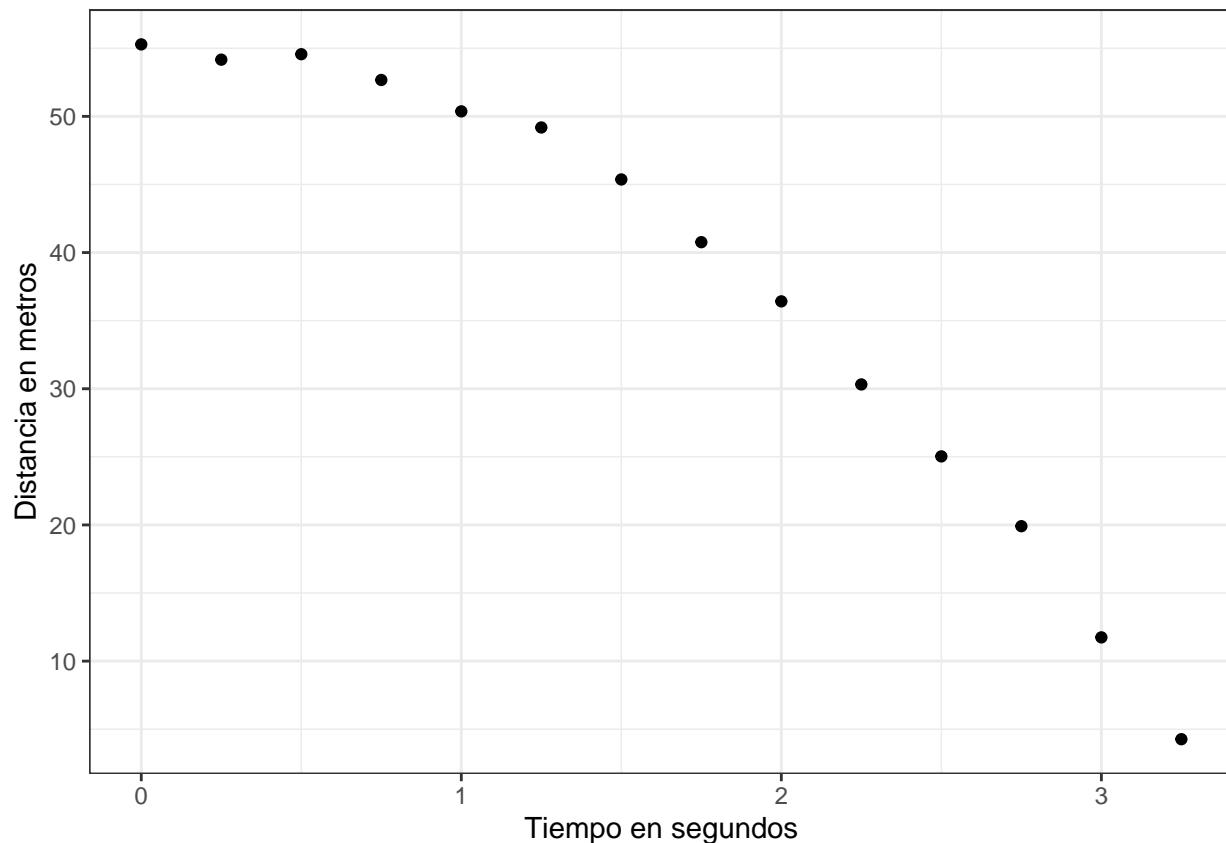
**7.2.6.2. Modelos de errores de medición** Hasta ahora, nuestros ejemplos de regresión lineal han sido aplicados a dos o más variables aleatorias, donde asumimos que los pares son bivariados normales.

Otro uso de la regresión lineal son los **modelos de errores de medición**, donde es común tener una covariable no aleatoria (como el tiempo). La aleatoriedad es introducida mediante el error de medición y no por el muestreo o la variación natural.

Para ejemplificar estos modelos, usaremos un caso relacionado con la física. Supóngase que se es un científico del siglo XVII tratando de describir la velocidad de un objeto que cae. Así, se realiza un experimento en la Torre de Pisa, donde se deja caer un objeto desde lo alto y se registra el tiempo cuando pasa por diferentes pisos de la torre. R contiene datos sobre este ejemplo en la base de datos “falling\_object”:

```
library(dslabs)
falling_object <- rfalling_object()
```

```
falling_object %>%
  ggplot(aes(time, observed_distance)) +
  geom_point() +
  ylab("Distancia en metros") +
  xlab("Tiempo en segundos")
```



No conocemos la ecuación exacta para describir la gráfica anterior, aunque podemos deducir que se trata de una parábola definida de la siguiente manera:

$$f(x) = \beta_0 + \beta_1 x + \beta_2 x^2$$

Si bien los datos no conforman una parábola exacta, podemos asumir que esto se debe a errores de medición. Para considerar este obstáculo, reformulemos el modelo así:

$$Y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \varepsilon_i , \quad i = 1, \dots, n$$

Donde  $Y_i$ : distancia que ha recorrido el objeto,  $x_i$ : el tiempo en segundos y  $\varepsilon$ : el error de medición. Es importante notar que los errores de medición son independientes unos de otros y que  $E[\varepsilon] = 0$ , así como que nuestro modelo sigue siendo lineal (porque  $\beta_k$  son lineales). Así, nuestro modelo estimado es:

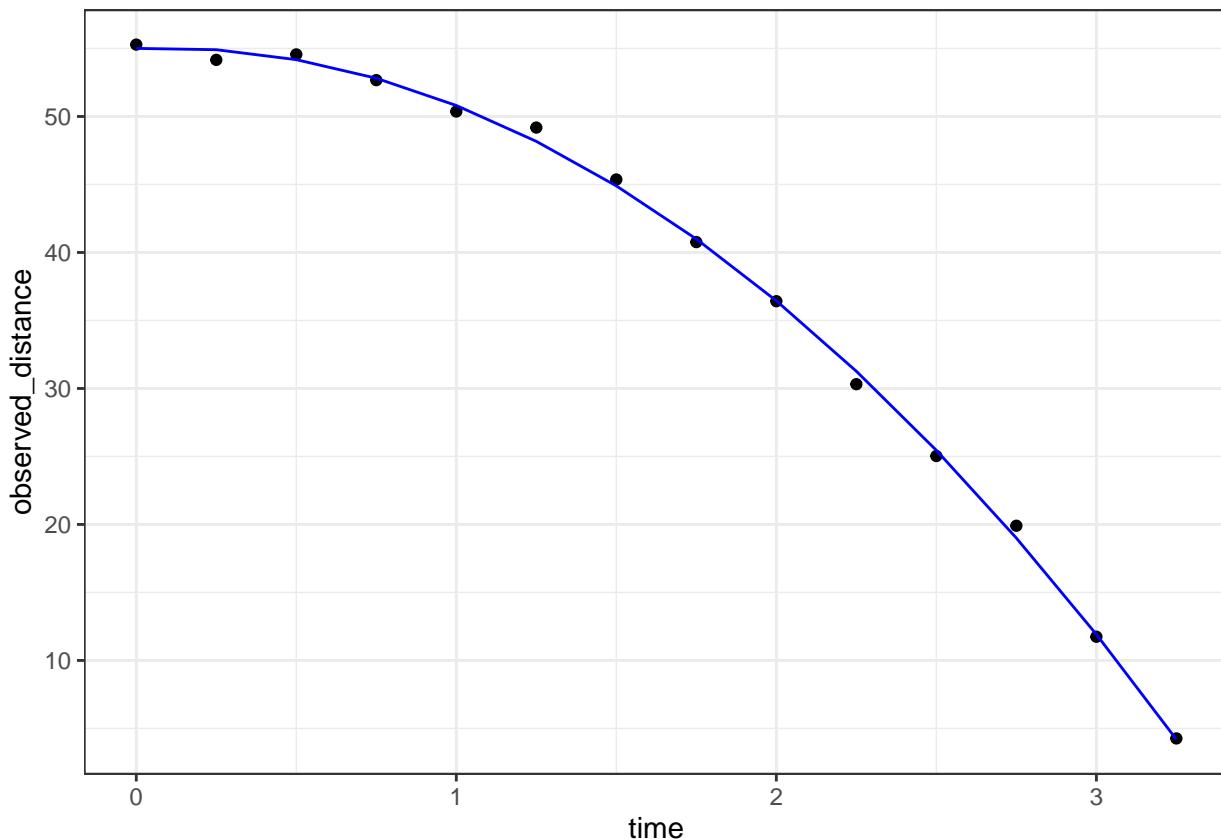
```
fit <- falling_object %>%
  mutate(time_sq = time^2) %>%
  lm(observed_distance~time+time_sq, data=.)
```

```
tidy(fit, conf.int = TRUE)
```

```
## # A tibble: 3 x 7
##   term      estimate std.error statistic p.value conf.low conf.high
##   <chr>     <dbl>     <dbl>     <dbl>    <dbl>     <dbl>     <dbl>
## 1 (Intercept) 55.0      0.434    127. 9.15e-19  54.1      56.0
## 2 time        0.888     0.620     1.43 1.80e- 1 -0.476     2.25
## 3 time_sq     -5.08     0.184    -27.7 1.61e-11 -5.49     -4.68
```

Para comprobar que la parábola estimada se ajusta a los datos, utilizaremos la función “augment()”:

```
augment(fit) %>%
  ggplot() +
  geom_point(aes(time, observed_distance)) +
  geom_line(aes(time, .fitted), col = "blue")
```



Considérese que la ecuación de la trayectoria de un objeto que cae es la siguiente:  $d = h_0 + v_0 t - 0,5 * 9,8t^2$ . Donde  $h_0$ : es la altura inicial;  $v_0$ : la velocidad inicial. Una comprobación a los datos demuestra que son consistentes con esta ecuación, dado que los valores reales sí entran en los intervalos de confianza calculados (comenzamos desde una altura de 56 metros en la Torre de Pisa y una velocidad de 0)

### 7.2.7. Assessment 16

Use the Teams data frame from the Lahman package. Fit a multivariate linear regression model to obtain the effects of BB and HR on Runs (R) in 1971. Use the tidy() function in the broom package to obtain the results in a data frame.

```
library(Lahman)
library(broom)
```

```

data(Teams)

dat <- Teams %>%
  filter(yearID == 1971)

modelo <- dat %>%
  lm(R ~ BB + HR, data = .)

tidy(modelo)

## # A tibble: 3 x 5
##   term      estimate std.error statistic p.value
##   <chr>     <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept) 257.      112.      2.31  0.0314
## 2 BB          0.414     0.210     1.97  0.0625
## 3 HR          1.30      0.431     3.01  0.00673

```

Modelo estimado de Carreras vs. bases por bola y home runs, 1971

Con dos variables explicativas	
	Carreras Modelo 3
Bases por bola	0.414* (0.210)
Home runs	1.290*** (0.431)
Constant	257.000** (112.000)
Observations	24
R <sup>2</sup>	0.436
Adjusted R <sup>2</sup>	0.382
Residual Std. Error	61.800 (df = 21)
F Statistic	8.110*** (df = 2; 21)

Nivel de significancia (P-valor) \*p<0.1; \*\*p<0.05; \*\*\*p<0.01

Repeat the above exercise to find the effects of BB and HR on runs (R) for every year from 1961 to 2018 using do() and the broom package. Make a scatterplot of the estimate for the effect of BB on runs over time and add a trend line with confidence intervals.

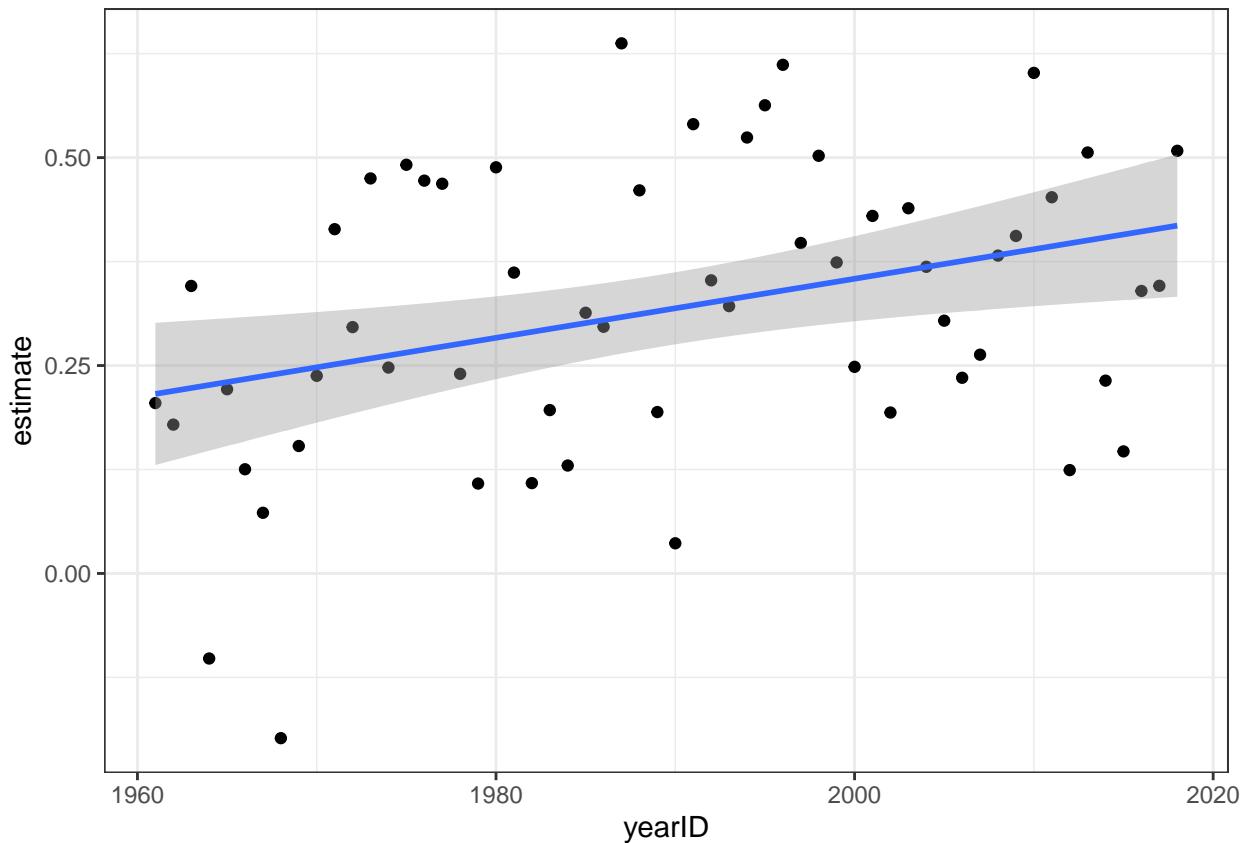
```

res <- Teams %>%
  filter(yearID %in% 1961:2018) %>%
  group_by(yearID) %>%
  do(tidy(lm(R ~ BB + HR, data = .))) %>%
  ungroup()

res %>%
  filter(term == "BB") %>%
  ggplot(aes(yearID, estimate)) +
  geom_point() +
  geom_smooth(method = "lm")

## `geom_smooth()` using formula 'y ~ x'

```



Fit a linear model on the results from Question 10 to determine the effect of year on the impact of BB.

```
res <- res %>%
  filter(term == "BB")

summary(lm(estimate ~ yearID, data = res))

##
## Call:
## lm(formula = estimate ~ yearID, data = res)
##
## Residuals:
##     Min      1Q      Median      3Q      Max 
## -0.4388 -0.1077 -0.0043  0.1185  0.3292 
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) -6.74629   2.57102  -2.62   0.0112 *  
## yearID       0.00355   0.00129   2.75   0.0081 ** 
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.165 on 56 degrees of freedom
## Multiple R-squared:  0.119, Adjusted R-squared:  0.103 
## F-statistic: 7.55 on 1 and 56 DF,  p-value: 0.00807
```

### 7.2.8. Assessment 17

Game attendance in baseball varies partly as a function of how well a team is playing. Load the Lahman library. The Teams data frame contains an attendance column. This is the total attendance for the season. To calculate average attendance, divide by the number of games played, as follows:

```
library(tidyverse)
library(broom)
library(Lahman)

Teams_small <- Teams %>%
  filter(yearID %in% 1961:2002) %>%
  mutate(avg_attendance = attendance/G, R = R/G, HR = HR/G)
```

Use runs (R) per game to predict average attendance. For every 1 run scored per game, average attendance increases by how much?

```
fit <- lm(avg_attendance ~ R, data = Teams_small)

fit

##
## Call:
## lm(formula = avg_attendance ~ R, data = Teams_small)
##
## Coefficients:
## (Intercept)          R
##           -7330        4158
```

Use home runs (HR) per game to predict average attendance. For every 1 home run hit per game, average attendance increases by how much?

```
fit <- lm(avg_attendance ~ HR, data = Teams_small)

fit

##
## Call:
## lm(formula = avg_attendance ~ HR, data = Teams_small)
##
## Coefficients:
## (Intercept)          HR
##           3685        8285
```

Use number of wins to predict average attendance; do not normalize for number of games. For every game won in a season, how much does average attendance increase? Suppose a team won zero games in a season. Predict the average attendance.

```
fit <- lm(avg_attendance ~ W, data = Teams_small)

fit

##
## Call:
## lm(formula = avg_attendance ~ W, data = Teams_small)
##
## Coefficients:
## (Intercept)          W
```

```
##          1038          123
```

Use year to predict average attendance. How much does average attendance increase each year?

```
fit <- lm(avg_attendance ~ yearID, data = Teams_small)
```

```
fit
```

```
##  
## Call:  
## lm(formula = avg_attendance ~ yearID, data = Teams_small)  
##  
## Coefficients:  
## (Intercept)      yearID  
## -462567        239
```

Game wins, runs per game and home runs per game are positively correlated with attendance. We saw in the course material that runs per game and home runs per game are correlated with each other. Are wins and runs per game or wins and home runs per game correlated? Use the Teams\_small data once again.

What is the correlation coefficient for wins and runs per game?

```
Teams_small %>%  
  summarize(cor(W, R))
```

```
##   cor(W, R)  
## 1  0.419
```

What is the correlation coefficient for wins and home runs per game?

```
Teams_small %>%  
  summarize(cor(W, HR))
```

```
##   cor(W, HR)  
## 1  0.277
```

Stratify Teams\_small by wins: divide number of wins by 10 and then round to the nearest integer. Keep only strata 5 through 10, which have 20 or more data points.

```
teams_strata <- Teams_small %>%  
  mutate(W_strata = round(W/10, 0)) %>%  
  filter(W_strata >= 5 & W_strata <= 10)
```

```
teams_strata %>%  
  filter(W_strata == "8") %>%  
  nrow()
```

```
## [1] 346
```

Calculate the slope of the regression line predicting average attendance given runs per game for each of the win strata.

```
slope_table1 <- teams_strata %>%  
  mutate(R_per_game = R/G) %>%  
  group_by(W_strata) %>%  
  summarize(slope = cor(R/G, avg_attendance)*sd(avg_attendance)/sd(R/G))
```

```
slope_table1$W_strata[which.max(slope_table1$slope)]
```

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

Calculate the slope of the regression line predicting average attendance given HR per game for each of the win strata. Which win stratum has the largest regression line slope?

```
slope_table2 <- teams_strata %>%
  mutate(HR_per_game = HR/G) %>%
  group_by(W_strata) %>%
  summarize(slope = cor(avg_attendance, HR_per_game))

slope_table2$W_strata[which.max(slope_table2$slope)]
```

## [1] 5

Fit a multivariate regression determining the effects of runs per game, home runs per game, wins, and year on average attendance. Use the original Teams\_small wins column, not the win strata from question 3.

What is the estimate of the effect of runs/HR/W per game on average attendance?

```
Teams_small <- Teams %>%
  filter(yearID %in% 1961:2001) %>%
  mutate(avg_attendance = attendance / G)

fit <- Teams_small %>%
  mutate(R_per_game = R/G, HR_per_game = HR/G) %>%
  lm(avg_attendance ~ R_per_game + HR_per_game + W + yearID, data = .)

tidy(fit)
```

```
## # A tibble: 5 x 5
##   term      estimate std.error statistic p.value
##   <chr>     <dbl>    <dbl>     <dbl>    <dbl>
## 1 (Intercept) -456674.  21815.    -20.9   3.00e-81
## 2 R_per_game     322.    331.     0.972  3.31e- 1
## 3 HR_per_game    1798.   690.     2.61   9.24e- 3
## 4 W            117.     9.88    11.8   2.79e-30
## 5 yearID       230.    11.2     20.6   7.10e-79
```

Use the multivariate regression model from Question 4. Suppose a team averaged 5 runs per game, 1.2 home runs per game, and won 80 games in a season.

El modelo es:

$$Y_i = -456674.4085 + 321.7864 \text{Runs} + 1798.3917 \text{HomeRuns} + 116.6691 \text{Wins} + 229.6320 \text{Year}$$

What would this team's average attendance be in 2002?

```
-456674.4085 + 321.7864*5+1798.3917*1.2+116.6691*80+229.6320*2002
```

## [1] 16149

What would this team's average attendance be in 1960?

```
-456674.4085 + 321.7864*5+1798.3917*1.2+116.6691*80+229.6320*1960
```

## [1] 6505

Use your model from Question 4 to predict average attendance for teams in 2002 in the original Teams data frame. What is the correlation between the predicted attendance and actual attendance?

```
Teams_small <- Teams %>%
  filter(yearID %in% 1961:2001) %>%
  mutate(avg_attendance = attendance/G)

fit <- Teams_small %>%
  mutate(R_pg = R/G, HR_pg = HR/G) %>%
  lm(avg_attendance ~ R_pg + HR_pg + W + yearID, data = .)

avg_2002 <- Teams %>%
  filter(yearID == 2002) %>% mutate(avg_2002 = attendance/G, R_pg = R/G, HR_pg = HR/G)

predict <- predict(fit, avg_2002, se.fit = TRUE)

cor(predict$fit, avg_2002$avg_2002)

## [1] 0.519
```

## 7.3. Confounding

### 7.3.1. Correlación y causalidad

**7.3.1.1. Correlaciones espurias** Una simulación Monte Carlo puede mostrar cómo el *data dredging* resulta en la identificación de correlaciones altas entre variables teóricamente no correlacionadas:

```
library(tidyverse)

N <- 25

G <- 1000000

sim_data <- tibble(group = rep(1:G, each = N), X = rnorm(N*G), Y = rnorm(N*G))
```

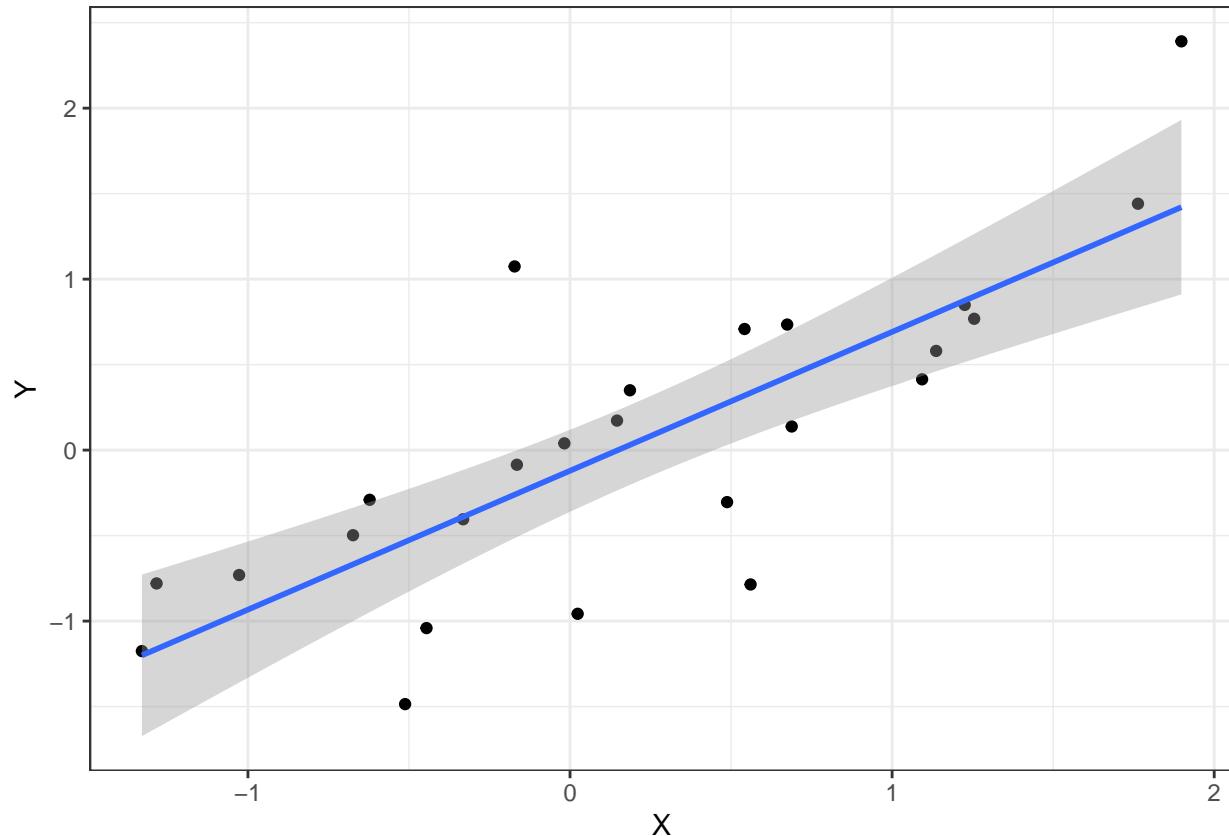
Ahora, computemos la correlación de x y y para cada grupo:

```
res <- sim_data %>%
  group_by(group) %>%
  summarize(r = cor(X, Y)) %>%
  arrange(desc(r))
```

Si graficamos el grupo con la correlación más alta:

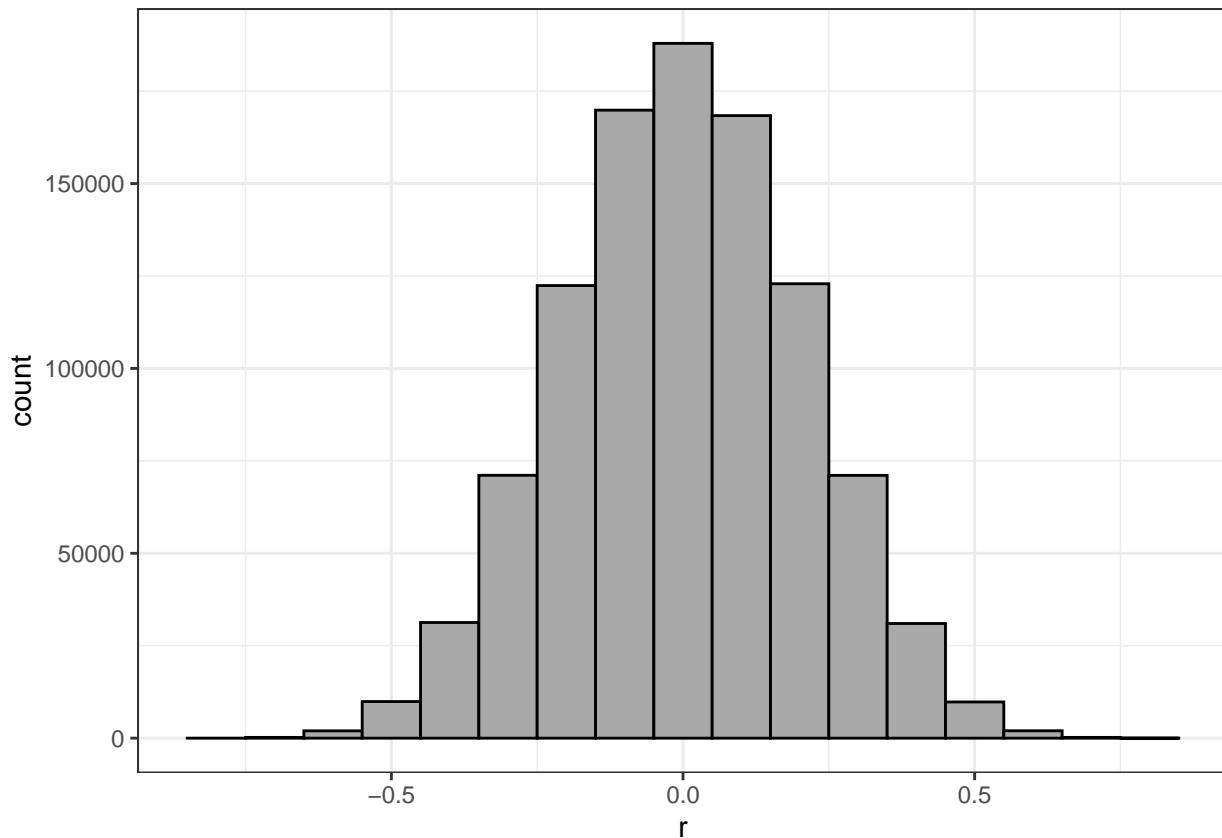
```
sim_data %>%
  filter(group == res$group[which.max(res$r)]) %>%
  ggplot(aes(X, Y)) +
  geom_point() +
  geom_smooth(method = "lm")

## `geom_smooth()` using formula 'y ~ x'
```



Así, analícese la distribución que siguen todas las correlaciones, la cual es evidentemente normal con media 0 y error estándar de 0.2

```
res %>%
  ggplot(aes(x=r)) +
  geom_histogram(binwidth = 0.1, color = "black", fill = "darkgray")
```



Nótese que si realizamos una regresión del grupo más e interpretamos el p-value concluiríamos que existe una correlación evidente:

```
library(broom)

sim_data %>%
  filter(group == res$group[which.max(res$r)]) %>%
  do(tidy(lm(Y ~ X, data = .)))

## # A tibble: 2 x 5
##   term      estimate std.error statistic  p.value
##   <chr>      <dbl>     <dbl>     <dbl>      <dbl>
## 1 (Intercept) -0.121     0.116    -1.04  0.309
## 2 X          0.812     0.130     6.27 0.00000213
```

Lo anterior se conoce como *p-hacking* y es el fenómeno cuando los científicos de datos tienden a ponderar mejor a los resultados significativos, dado que son los que normalmente se publican, y despreciar los resultados no significativos.

**7.3.1.2. Outliers** Otro ejemplo que puede provocar la identificación errónea de una correlación son los *outliers*. Supóngase que en una muestra de observaciones nos olvidamos de estandarizar un valor, sea el 23:

```
set.seed(1)

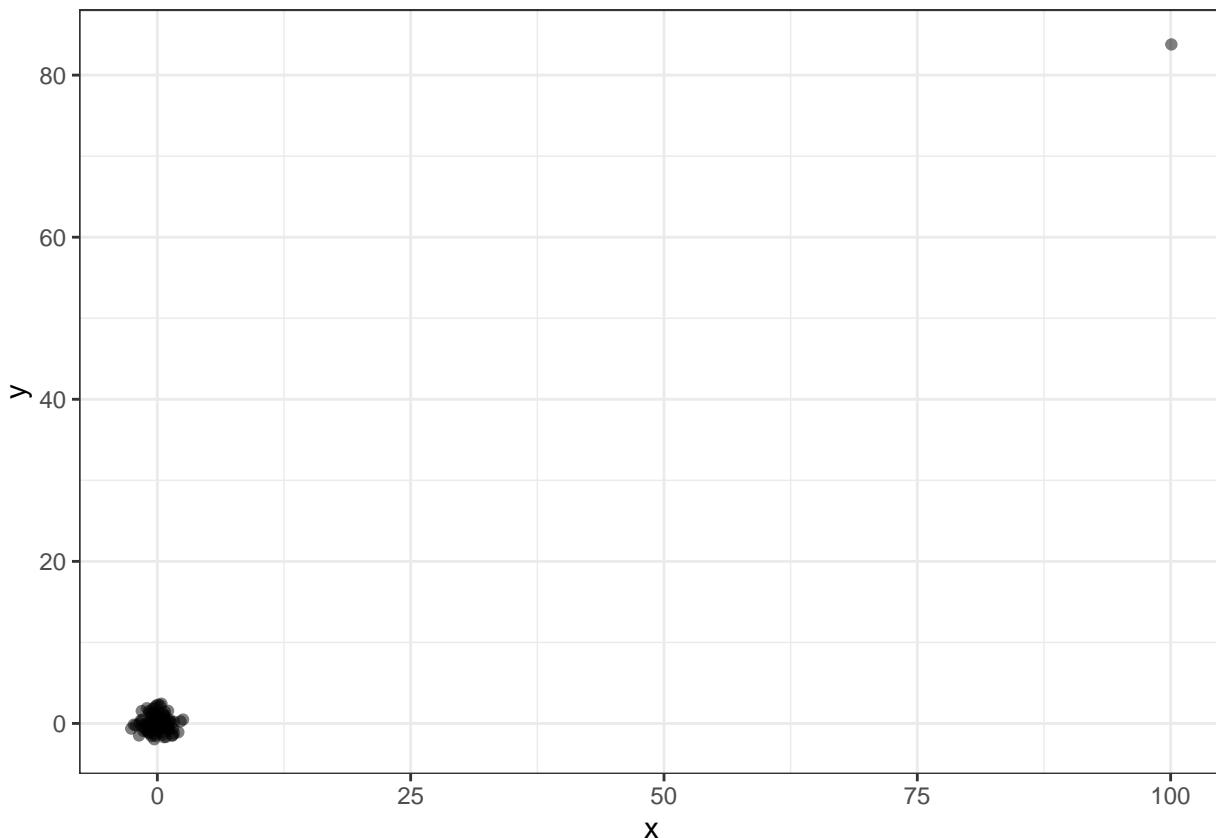
x <- rnorm(100, 100, 1)

y <- rnorm(100, 84, 1)
```

```
x[-23] <- scale(x[-23])  
y[-23] <- scale(y[-23])
```

Así, los datos lucen así:

```
tibble(x,y) %>%  
  ggplot(aes(x,y)) +  
  geom_point(alpha = 0.5)
```



No es sorprendente que la correlación sea muy alta:

```
cor(x,y)
```

```
## [1] 0.988
```

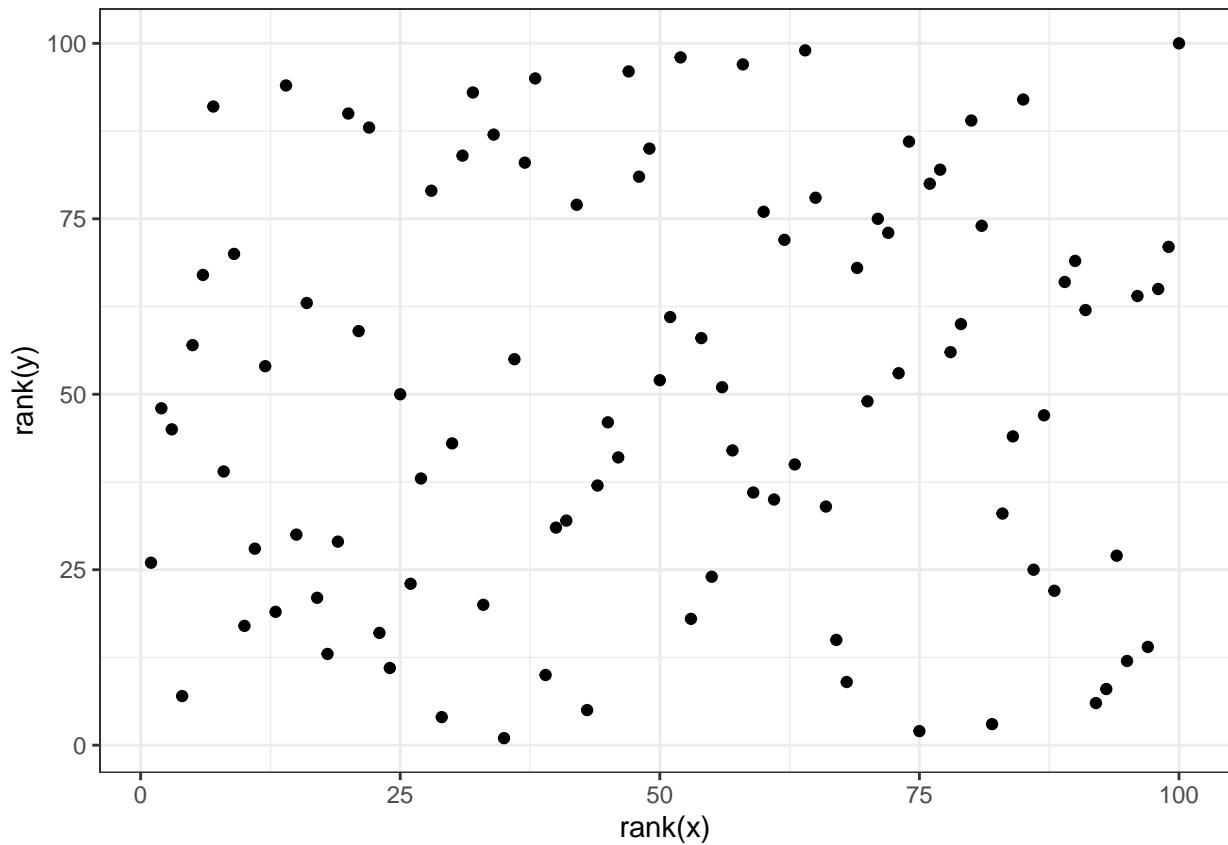
Pero obsérvese qué ocurre si removemos esa observación:

```
cor(x[-23], y[-23])
```

```
## [1] -0.00107
```

Así, para evitar este problema, es recomendable detectar y remover los *outliers*. Sin embargo, existe una alternativa: la **correlación Spearman**, la cual calcula la correlación de los rangos de los valores en vez de que los valores mismos.

```
qplot(rank(x), rank(y))
```



```
cor(rank(x), rank(y))
```

```
## [1] 0.0658
```

La correlación Spearman también puede calcularse con la función “cor()” de la siguiente manera:

```
cor(x, y, method = "spearman")
```

```
## [1] 0.0658
```

**7.3.1.3. Inversión de causa y efecto** Otra manera de la cual las asociaciones pueden ser confundidas con causalidad es cuando la causa y efecto son invertidos.

Un efecto es afirmar que las asesorías hace que los estudiantes rindan peor porque los resultados de un examen muestran peores resultados que aquellos que no recibieron asesorías.

Retómese el ejemplo de las alturas de padres e hijos:

```
library(HistData)
data("GaltonFamilies")
GaltonFamilies %>%
  filter(childNum == 1 & gender == "male") %>%
  select(father, childHeight) %>%
  rename(son = childHeight) %>%
  do(tidy(lm(father ~ son, data = .)))
## # A tibble: 2 x 5
```

```
##   term      estimate std.error statistic p.value
##   <chr>     <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept) 34.0      4.57      7.44 4.31e-12
## 2 son         0.499     0.0648     7.70 9.47e-13
```

Donde es evidentemente incorrecto interpretar que la altura del hijo causa a la del padre. Así, a pesar de que el modelo es técnicamente correcto, la interpretación no lo es.

**7.3.1.4. Factores de confusión** Si X y Y están correlacionados, llamamos a Z un **counfounder** si cambios en Z causan cambios en X y Y.

Como ejemplo, considérense los datos sobre admisiones de UC Berkeley en 1973.

```
library(dslabs)

data(admissions)

admissions

##   major gender admitted applicants
## 1     A   men       62      825
## 2     B   men       63      560
## 3     C   men       37      325
## 4     D   men       33      417
## 5     E   men       28      191
## 6     F   men        6      373
## 7     A women      82      108
## 8     B women      68       25
## 9     C women      34      593
## 10    D women      35      375
## 11    E women      24      393
## 12    F women       7      341
```

Donde el porcentaje de mujeres aceptadas fue significativamente menor:

```
admissions %>%
  group_by(gender) %>%
  summarize(percentage = round(sum(admitted*applicants)/sum(applicants),1))
```

```
## # A tibble: 2 x 2
##   gender percentage
##   <chr>     <dbl>
## 1 men       44.5
## 2 women     30.3
```

Un test estadístico de Chi cuadrada claramente rechaza la hipótesis de que el género y las admisiones son independientes:

```
admissions %>% group_by(gender) %>%
  summarize(total_admitted = round(sum(admitted / 100 * applicants)),
            not_admitted = sum(applicants) - sum(total_admitted)) %>%
  select(-gender) %>%
  do(tidy(chisq.test(.)))

## # A tibble: 1 x 4
##   statistic p.value parameter method
##       <dbl>    <dbl>     <int> <chr>
## 1     91.6 1.06e-21          1 Pearson's Chi-squared test with Yates' continuit-
```

Sin embargo, una inspección detallada muestra un resultado paradójico. A continuación se presentan los resultados del porcentaje de admisiones por carrera:

```
admissions %>%
  select(major, gender, admitted) %>%
  spread(gender, admitted) %>%
  mutate(women_minus_men = women - men)

##   major men women women_minus_men
## 1     A   62     82            20
## 2     B   63     68             5
## 3     C   37     34            -3
## 4     D   33     35             2
## 5     E   28     24            -4
## 6     F    6      7              1
```

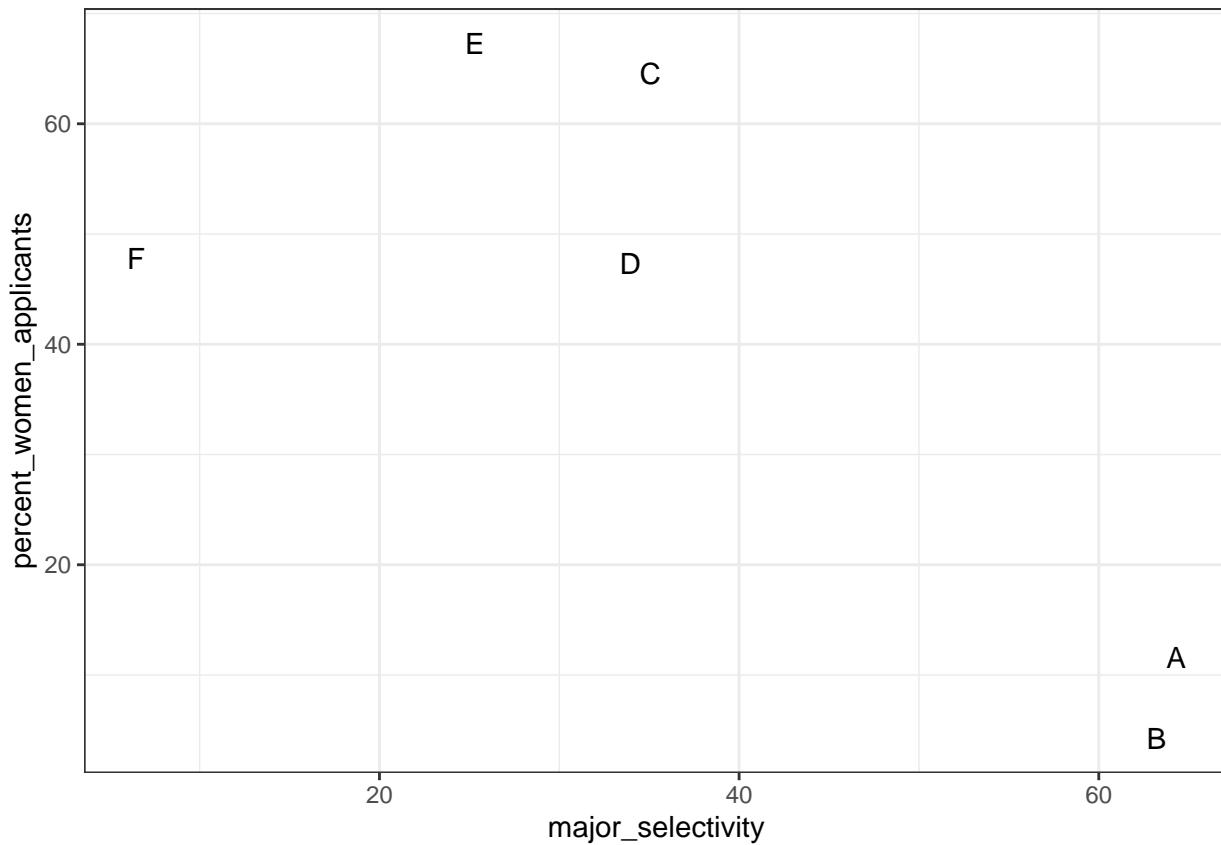
Se observa que 4 de 6 carreras favorecieron las admisiones de mujeres, lo que sugiere que cuando los datos se agrupan por carrera, la dependencia entre género y admisiones desaparece.

Definamos 3 variables, X: 1 para hombres, 0 para mujeres; Y: 1 para admitidos, 0 para rechazados; Z: cuantifica qué tan selectiva es la carrera. Un sesgo de género implicaría que ocurre:

$$Pr[Y = 1|X = 1] > Pr[Y = 1|X = 0]$$

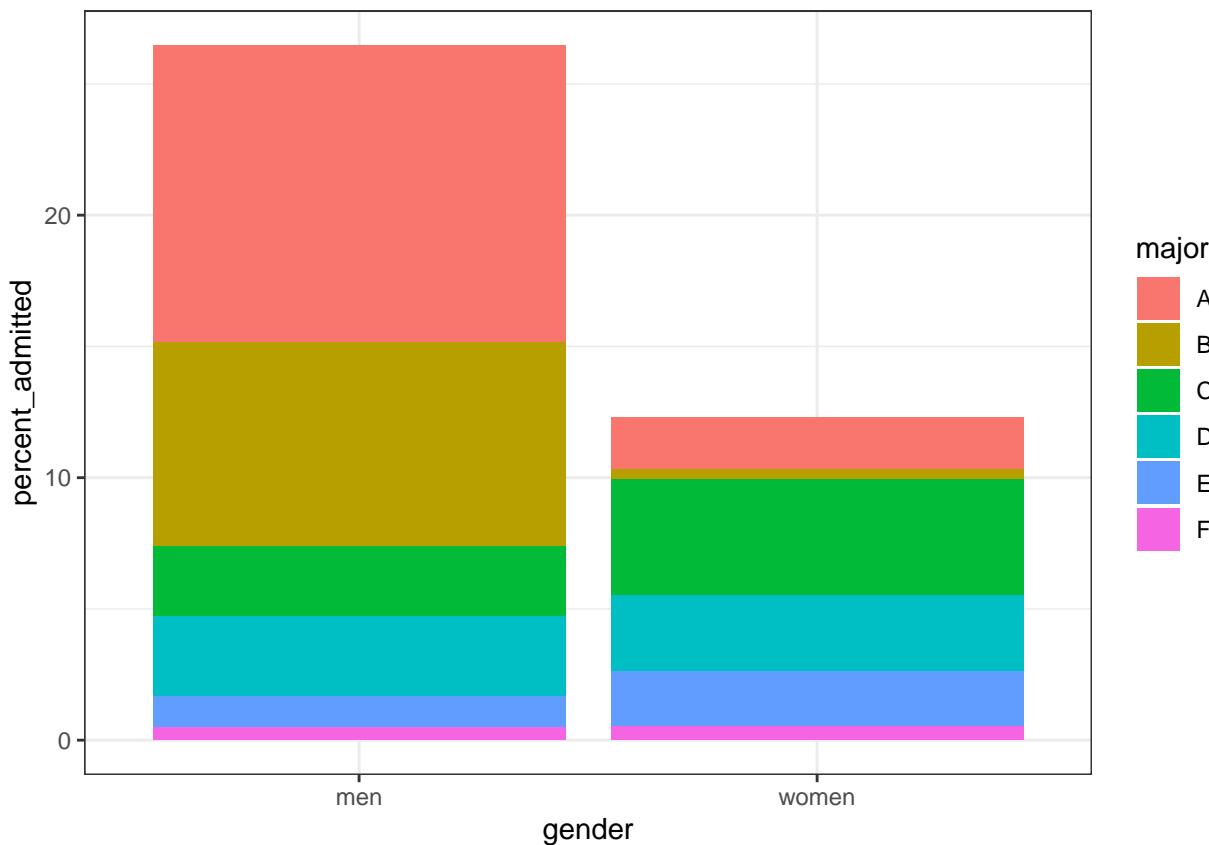
Sin embargo, Z es un *counfounder* relevante. ¿Está la selectividad relacionada con el género? Para ver esto, grafíquese el porcentaje total de admitidos en una carrera contra el porcentaje de mujeres que aplicaron:

```
admissions %>%
  group_by(major) %>%
  summarize(major_selectivity = sum(admitted*applicants)/sum(applicants),
            percent_women_applicants = sum(applicants*(gender == "women")/sum(applicants))*100) %>%
  ggplot(aes(major_selectivity, percent_women_applicants, label = major)) +
  geom_text()
```



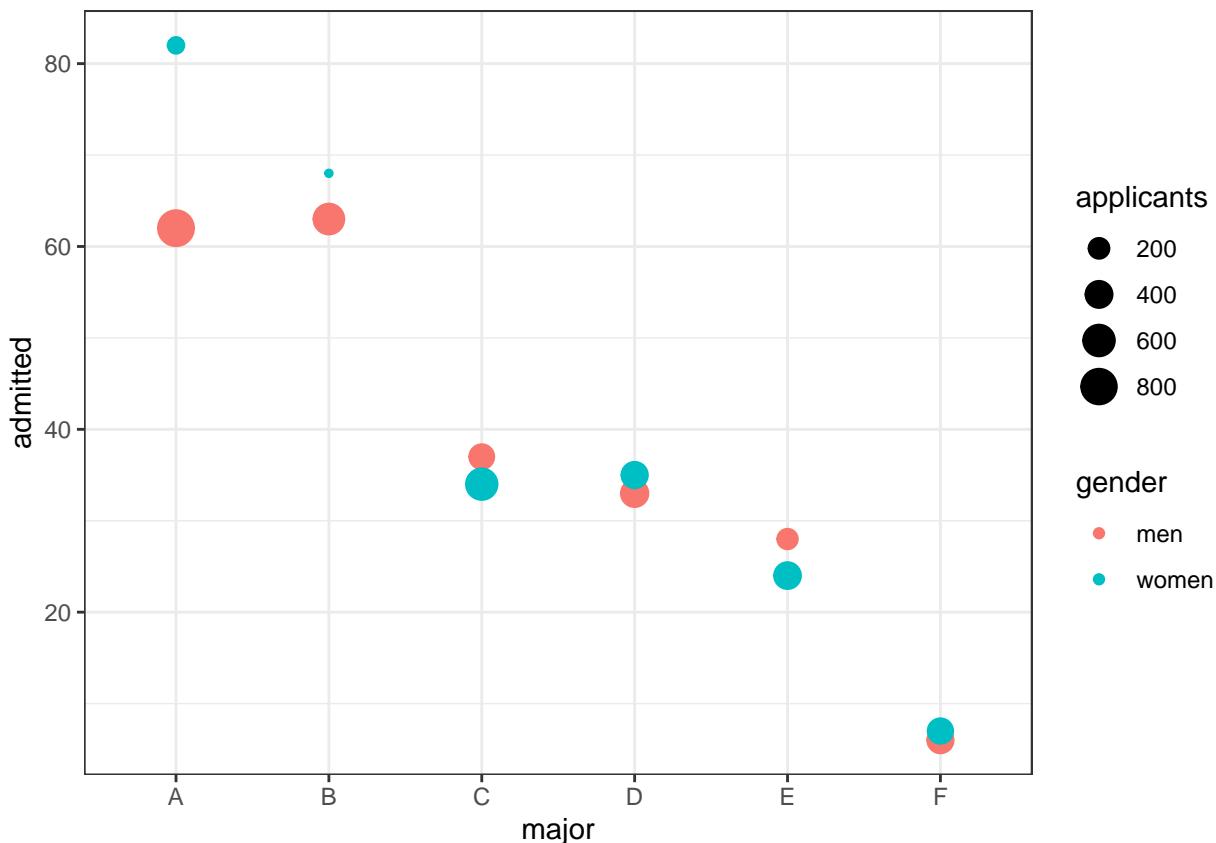
El gráfico sugiere que las mujeres fueron más propensas a aplicar a las dos carreras más selectivas, por lo que es sensato afirmar que el género y la selectividad están correlacionadas. El siguiente gráfico muestra el porcentaje de aplicantes aceptados por género:

```
admissions %>%
  mutate(percent_admitted = admitted*applicants/sum(applicants)) %>%
  ggplot(aes(gender, y = percent_admitted, fill = major)) +
  geom_bar(stat = "identity", position= "stack")
```



En el siguiente gráfico podemos ver que si condicionamos o estratificamos por carrera, entonces controlamos al *counfounder* y el efecto problemático desaparece:

```
admissions %>%  
  ggplot(aes(major, admitted, col = gender, size = applicants)) +  
  geom_point()
```



Si estratificamos por carrera, computamos la diferencia y luego el promedio, encontramos que la diferencia porcentual es realmente pequeña

```
admissions %>%
  group_by(gender) %>%
  summarize(average = mean(admitted))

## # A tibble: 2 x 2
##   gender     average
##   <chr>      <dbl>
## 1 men        38.2
## 2 women      41.7
```

Este fenómeno se conoce como la paradoja Simpson.

**7.3.1.5. Paradoja Simpson** Este fenómeno ocurre cuando observamos que el signo de la correlación se da la vuelta cuando comparamos la base de datos entera con estratos específicos.

### 7.3.2. Assessment 18

For this set of exercises, we examine the data from a 2014 PNAS paper that analyzed success rates from funding agencies in the Netherlands External link and concluded:

“our results reveal gender bias favoring male applicants over female applicants in the prioritization of their “quality of researcher” (but not “quality of proposal”) evaluations and success rates, as well as in the language used in instructional and evaluation materials.”

A response was published a few months later titled No evidence that gender contributes to personal research funding success in The Netherlands: A reaction to Van der Lee and Ellemers

External link, which concluded:

However, the overall gender effect borders on statistical significance, despite the large sample. Moreover, their conclusion could be a prime example of Simpson's paradox; if a higher percentage of women apply for grants in more competitive scientific disciplines (i.e., with low application success rates for both men and women), then an analysis across all disciplines could incorrectly show "evidence" of gender inequality.

Who is right here: the original paper or the response? Here, you will examine the data and come to your own conclusion.

The main evidence for the conclusion of the original paper comes down to a comparison of the percentages. The information we need was originally in Table S1 in the paper, which we include in dslabs:

```
library(dslabs)
data("research_funding_rates")
research_funding_rates
```

	discipline	applications_total	applications_men	applications_women	
## 1	Chemical sciences	122	83	39	
## 2	Physical sciences	174	135	39	
## 3	Physics	76	67	9	
## 4	Humanities	396	230	166	
## 5	Technical sciences	251	189	62	
## 6	Interdisciplinary	183	105	78	
## 7	Earth/life sciences	282	156	126	
## 8	Social sciences	834	425	409	
## 9	Medical sciences	505	245	260	
##					
	awards_total	awards_men	awards_women	success_rates_total	success_rates_men
## 1	32	22	10	26.2	26.5
## 2	35	26	9	20.1	19.3
## 3	20	18	2	26.3	26.9
## 4	65	33	32	16.4	14.3
## 5	43	30	13	17.1	15.9
## 6	29	12	17	15.8	11.4
## 7	56	38	18	19.9	24.4
## 8	112	65	47	13.4	15.3
## 9	75	46	29	14.9	18.8
##					
	success_rates_women				
## 1	25.6				
## 2	23.1				
## 3	22.2				
## 4	19.3				
## 5	21.0				
## 6	21.8				
## 7	14.3				
## 8	11.5				
## 9	11.2				

Construct a two-by-two table of gender (men/women) by award status (awarded/not) using the total numbers across all disciplines.

```
research_funding_rates %>%
  summarize(men_not_awarded = sum(applications_men)-sum(awards_men),
            women_not_awarded = sum(applications_women)-sum(awards_women))
```

```
##   men_not_awarded women_not_awarded
## 1           1345            1011
```

Use the two-by-two table from Question 1 to compute the percentages of men awarded versus women awarded.

```
research_funding_rates %>%
  summarize(percentage_men = sum(awards_men)/sum(applications_men),
            percentage_women= sum(awards_women)/sum(applications_women))

##   percentage_men percentage_women
## 1       0.177        0.149
```

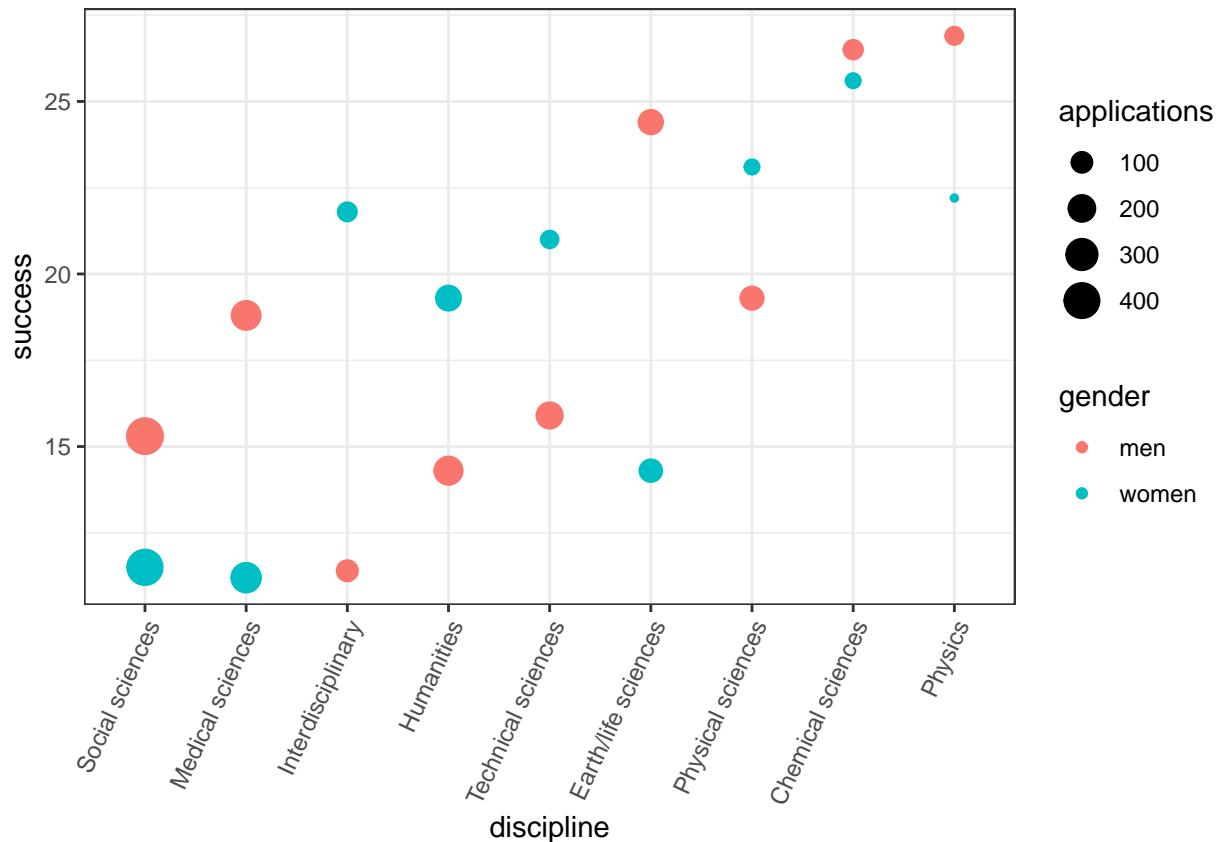
There may be an association between gender and funding. But can we infer causation here? Is gender bias causing this observed difference? The response to the original paper claims that what we see here is similar to the UC Berkeley admissions example. Specifically they state that this “could be a prime example of Simpson’s paradox; if a higher percentage of women apply for grants in more competitive scientific disciplines, then an analysis across all disciplines could incorrectly show ‘evidence’ of gender inequality.” To settle this dispute, use this dataset with number of applications, awards, and success rate for each gender:

```
dat <- research_funding_rates %>%
  mutate(discipline = reorder(discipline, success_rates_total)) %>%
  rename(success_total = success_rates_total,
         success_men = success_rates_men,
         success_women = success_rates_women) %>%
  gather(key, value, -discipline) %>%
  separate(key, c("type", "gender")) %>%
  spread(type, value) %>%
  filter(gender != "total")
dat

##      discipline gender applications awards success
## 1 Social sciences men        425     65    15.3
## 2 Social sciences women      409     47    11.5
## 3 Medical sciences men      245     46    18.8
## 4 Medical sciences women     260     29    11.2
## 5 Interdisciplinary men      105     12    11.4
## 6 Interdisciplinary women     78     17    21.8
## 7 Humanities men            230     33    14.3
## 8 Humanities women          166     32    19.3
## 9 Technical sciences men    189     30    15.9
## 10 Technical sciences women   62     13    21.0
## 11 Earth/life sciences men   156     38    24.4
## 12 Earth/life sciences women  126     18    14.3
## 13 Physical sciences men     135     26    19.3
## 14 Physical sciences women    39      9    23.1
## 15 Chemical sciences men     83     22    26.5
## 16 Chemical sciences women    39     10    25.6
## 17 Physics men               67     18    26.9
## 18 Physics women              9      2    22.2
```

To check if this is a case of Simpson’s paradox, plot the success rates versus disciplines, which have been ordered by overall success, with colors to denote the genders and size to denote the number of applications.

```
dat %>%
  ggplot(aes(discipline, success, col = gender, size = applications)) +
  geom_point() +
  theme(axis.text.x = element_text(angle = 65, hjust = 1))
```



## 8. Machine Learning

**NOTA:** A LO LARGO DE ESTA SECCIÓN, SE ESCRIBIERON ALGUNOS CÓDIGOS EN FORMA DE COMENTARIO (#) CON EL OBJETIVO DE QUE EL TIEMPO DE LA REPLICABILIDAD DEL CÓDIGO Y EL TEJIDO DEL DOCUMENTO EN R MARKDOWN FUERA RELATIVAMENTE CORTO. QUITAR ESTOS COMENTARIOS DISMINUIRÁ CONSIDERABLEMENTE LA RAPIDEZ PARA LLEVAR A CABO ESTAS TAREAS.

### 8.1. Introducción

#### 8.1.1. Notación

En el aprendizaje automático o *machine learning*, los datos se presentan en la forma del resultado que queremos predecir y las características que usaremos para la predicción. Nuestro objetivo será **construir un algoritmo que tome valores característicos como entradas y arroje una predicción para un resultado desconocido**.

Usaremos  $Y$  para denotar el resultado y  $X_1, \dots, X_p$  para las características (a veces conocidas como predictores o covariables). Los problemas de predicción pueden ser divididos en:

1. Resultados categóricos:  $Y$  puede ser de cualquiera de  $K$  clases con  $k = 1, \dots, K$ .
2. Resultados continuos: donde  $Y \in \mathbb{R}$

Así, la configuración inicial es como se muestra a continuación:

outcome	feature_1	feature_2	feature_3	...	feature_K
?	$X_1$	$X_2$	$X_3$	...	$X_K$

Para construir un modelo o algoritmo que provea una predicción para cualquier conjunto de valores  $X_1 = x_1 \dots X_5 = x_5$ , recolectamos los datos para los cuales sí conocemos el resultado y obtenemos una tabla como la que sigue:

outcome	feature_1	feature_2	feature_3	...	feature_K
$Y_1$	$X_{1,1}$	$X_{1,2}$	$X_{1,3}$	...	$X_{1,K}$
$Y_2$	$X_{2,1}$	$X_{2,2}$	$X_{2,3}$	...	$X_{2,K}$
$Y_3$	$X_{3,1}$	$X_{3,2}$	$X_{3,3}$	...	$X_{3,K}$
$Y_4$	$X_{4,1}$	$X_{4,2}$	$X_{4,3}$	...	$X_{4,K}$
...	...	...	...	...	...
$Y_m$	$X_{m,1}$	$X_{m,2}$	$X_{m,3}$	...	$X_{m,K}$

Para denotar la predicción, escribimos  $\hat{Y}$ ; así, queremos que esta predicción iguale al resultado real. Si nos encontramos con un problema categórico, nos referiremos a la tarea de aprendizaje automático como **clasificación**, donde nuestras predicciones también serán categóricas y podrán ser correctas o incorrectas. Si, por el contrario, se trata de un resultado continuo, la tarea se denominará como **predicción**, donde esta no será correcta o incorrecta, sino que trataremos de minimizar el error (diferencia entre la predicción y el resultado actual).

#### 8.1.2. Ejemplo

Considérese el siguiente ejemplo: en una oficina postal, la primera clasificación que se hace a las cartas es por código postal. Originalmente, esta tarea se realizaba a mano; sin embargo, hoy en día, y gracias a la utilización de algoritmos, una máquina puede leerlos y clasificarlos pertinente. A continuación construiremos un algoritmo para leer dígitos.

El primer paso es preguntarse ¿cuáles son los posibles resultados y cuáles son las características? Si queremos evaluar una imagen del número 159, hay que considerar que cada número está representado por varios pixeles (sean 784), cada uno coloreado con un tono que va desde el 0 (blanco) hasta el 255 (negro)..

Así, para cada imagen digitalizada  $i$  tenemos un resultado categórico  $Y_i$  que puede tomar 10 valores,  $Y_i \in \{0, 9\}$ . Además, tenemos 784 características  $X_{i,1}, \dots, X_{i,784}$ ; así, el vector de predictores es  $\mathbf{X}_i = X_{i,1}, \dots, X_{i,784}$  (nótese que utilizamos letras mayúsculas porque pueden pensarse a los predictores como variables aleatorias; escribiremos con minúsculas cuando nos refiramos a valores observados).

## 8.2. Básicos de Machine Learning

### 8.2.1. Evaluación de algoritmos de Machine Learning

En esta sección, se introducirá la librería caret, el cual cuenta con diversas funciones para la construcción y evaluación de métodos de aprendizaje automático:

```
library(caret)
```

Comenzaremos con un ejemplo simple: predecir el sexo de individuos a partir de su altura observada:

```
library(dslabs)
```

```
library(tidyverse)
```

```
data(heights)
```

Comencemos definiendo los resultados y predictores:

```
y <- heights$sex
```

```
x <- heights$height
```

Es evidente que se trata de un resultado categórico, dado que  $y$  solo puede tomar dos valores,  $y \in \{F, M\}$ ; por otro lado, solo tenemos un predictor, la altura,  $x \in \mathbb{R}$ .

Para simular un proceso de evaluación de los datos, típicamente se separan a los datos en dos y se pretende desconocer el resultado de alguno de estos dos conjuntos. Posteriormente, al evaluar el algoritmo, eliminamos este supuesto, pero solo después de haberlo construido.

Nos referiremos a los grupos de los cuales sí conocemos el resultado  $y$  que usaremos para desarrollar el algoritmo como **conjunto de preparación** y al grupo del cual pretendemos no conocer el resultado como **conjunto de prueba**. Para la separación de estos dos conjuntos, se suele realizar una división aleatoria de los datos; esto puede lograrse mediante la función “createDataPartition()” del paquete caret. El argumento “times” de esta función permite definir cuántas muestras aleatorias de índices queremos, el argumento “p” define la proporción del índice representado y “la lista de argumentos son usados para decidir”list”se usa para decidir si queremos que los índices sean arrojados como listas o no:

```
set.seed(2, sample.kind = "Rounding")
```

```
## Warning in set.seed(2, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
test_index <- createDataPartition(y, times = 1, p = 0.5, list = FALSE)
```

Así, podemos crear los dos conjuntos de datos a partir de este índice como se muestra a continuación:

```
train_set <- heights[-test_index, ]
```

```
test_set <- heights[test_index, ]
```

Ahora, desarrollaremos el algoritmo solo a partir del conjunto de preparación; una vez lo tengamos, lo fijaremos y usaremos para evaluar el conjunto de prueba.

**La manera más fácil de evaluar un algoritmo cuando el resultado es categórico es simplemente reportando la proporción de casos que fueron correctamente reportados en el conjunto de prueba; a esto se le conoce como precisión general.**

Para fines de comparabilidad, construiremos dos algoritmos y compararemos su precisión general. Defínase el primero como sigue (nótese que en este caso ignoramos al predictor y solo estamos adivinando el sexo):

```
y_hat <- sample(c("Male", "Female"),
                  length(test_index), replace = TRUE)
```

Al respecto, las funciones utilizadas por el paquete caret recomiendan o requieren que los resultados categóricos sean codificados como factores:

```
y_hat <- sample(c("Male", "Female"), length(test_index), replace = TRUE) %>%
  factor(levels = levels(test_set$sex))
```

Donde la precisión general es:

```
mean(y_hat == test_set$sex)
```

```
## [1] 0.524
```

No es sorprendente que nuestra precisión sea alrededor de 50 %, dado que estamos adivinando. ¿Puede mejorarse el algoritmo? El análisis exploratorio de datos sugiere que sí, dado que, en promedio, las alturas de los hombres son mayores que las de mujeres, como se puede comprobar a continuación:

```
heights %>%
  group_by(sex) %>%
  summarize(mean(height), sd(height))
```

```
## # A tibble: 2 x 3
##   sex     `mean(height)` `sd(height)`
##   <fct>        <dbl>      <dbl>
## 1 Female       64.9       3.76
## 2 Male         69.3       3.61
```

Así, predeciremos si la altura de los hombres se encuentra a dos desviaciones estándar de la media masculina mediante el siguiente código:

```
y_hat <- ifelse(x > 62, "Male", "Female") %>%
  factor(levels = levels(test_set$sex))
```

Donde nuestra precisión general aumenta considerablemente:

```
mean(y == y_hat)
```

```
## [1] 0.793
```

Nótese que utilizamos un límite de 62 pulgadas arbitrariamente, por lo que es sensato suponer que podríamos obtener un mejor resultado con otros valores (recuérdese que estas modificaciones deben hacerse en el conjunto de práctica y después evaluarse en el conjunto de prueba):

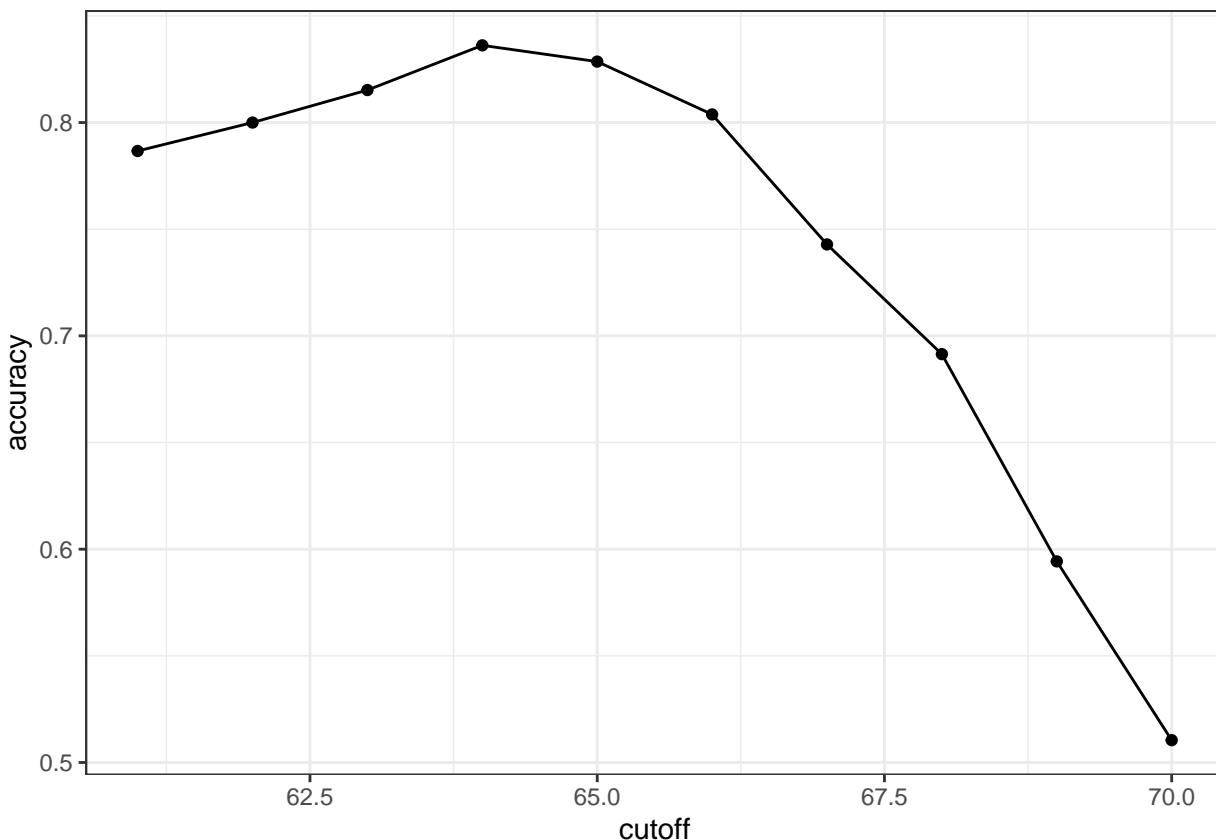
```
cutoff <- seq(61, 70)

accuracy <- map_dbl(cutoff, function(x){
  y_hat <- ifelse(train_set$height > x, "Male", "Female") %>%
    factor(levels = levels(test_set$sex))

  mean(y_hat == train_set$sex)
})
```

Observemos la gráfica de la precisión general en el conjunto de práctica para el conjunto de valores definido:

```
data.frame(cutoff, accuracy) %>%
  ggplot(aes(cutoff, accuracy)) +
  geom_point() +
  geom_line()
```



Donde vemos que el valor máximo es alrededor de 83.6 %:

```
max(accuracy)
```

```
## [1] 0.836
```

El cual está maximizado con el límite de altura de 64 pulgadas:

```
best_cutoff <- cutoff[which.max(accuracy)]
```

```
best_cutoff
```

```
## [1] 64
```

Ahora, estamos listos para probar este algoritmo en nuestro conjunto de prueba y comprobar que no se trate de una predicción demasiado optimista o sesgada:

```
y_hat <- ifelse(test_set$height > best_cutoff, "Male", "Female") %>%
  factor(levels = levels(test_set$sex))

y_hat <- factor(y_hat)

mean(y_hat == test_set$sex)

## [1] 0.817
```

Donde obtenemos una precisión de 81.7 %; si bien es ligeramente menor al resultado de nuestro conjunto de práctica, es sustantivamente mejor que adivinar

**8.2.1.1. Matriz de confusión** Utilizamos el límite de altura de 64 pulgadas para escoger nuestro mejor algoritmo en el ejemplo de arriba; sin embargo, dado que la altura media de las mujeres es de 65 pulgadas, esta regla de decisión resulta contraintuitivo.

Esto se debe a que la precisión general puede ser una medida engañosa. Para exemplificar esto, construiremos lo que se conoce como **matriz de confusión**, la cual tabula cada combinación de predicción y valor actual:

```
table(predicted = y_hat, actual = test_set$sex)
```

```
##           actual
## predicted Female Male
##   Female      50   27
##   Male        69  379
```

Si computamos la precisión por sexo, vemos que esta difiere bastante en cada caso:

```
test_set %>%
  mutate(y_hat = y_hat) %>%
  group_by(sex) %>%
  summarize(accuracy = mean(y_hat == sex))
```

```
## # A tibble: 2 x 2
##   sex     accuracy
##   <fct>    <dbl>
## 1 Female    0.420
## 2 Male      0.933
```

Esta falta de balance entre sexos sugiere que estamos prediciendo que muchas mujeres son hombres. ¿Por qué, entonces, nuestra precisión general es tan alta? Esto ocurre por la **prevalecia**, dado que hay más hombres que mujeres en la base de datos (77.3%):

```
prev <- mean(y == "Male")  
  
prev  
  
## [1] 0.773
```

Así, los errores que comete nuestro algoritmo con las mujeres son matizados por la gran cantidad de aciertos que ocurren al predecir hombres. Esto puede suponer grandes problemas en el aprendizaje automático: si nuestro conjunto de práctica está sesgado de algún modo, es probable que desarrollemos un algoritmo también sesgado.

Existen varias métricas que podemos usar para evaluar un algoritmo de tal forma que la prevalencia no influya nuestras conclusiones; todas ellas pueden derivarse de la matriz de confusión:

1. Sensibilidad: también conocida como la tasa real positiva, es la proporción de resultados realmente positivos correctamente identificados como tal (i.e.  $\hat{Y} = 1$  siempre que  $Y = 1$ ).
2. Especificidad: también conocida como la tasa real negativa, es la proporción de resultados realmente negativos correctamente identificados como tal (i.e.  $\hat{Y} = 0$  siempre que  $Y = 0$ ).

De manera intuitiva, una alta sensibilidad implica que si  $Y = 1 \Rightarrow \hat{Y} = 1$ ; por su parte, una alta especificidad implica que si  $Y = 0 \Rightarrow \hat{Y} = 0$  o que si  $\hat{Y} = 1 \Rightarrow Y = 1$ .

Así, la matriz de confusión se define como sigue:

	Positivo real	Negativo real
Positivo predicho	Positivo reales (TP)	Positivo falso (FP)
Negativo predicho	Negativo falsos (FN)	Negativo reales (TN)

De manera más formal, la sensibilidad (que también puede ser llamada *true positive rate (TPR)*):

$$\text{Sensibilidad} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Por su parte, la especificidad (o *true negative rate (TNR)*):

$$\text{Especificidad} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$

A veces, la especificidad también se define como *positive predicted value(PPV)*:

$$\text{Especificidad} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Para todo esto, la función “confusionMatrix()” del paquete caret computa todas las métricas una vez que hemos definido qué es un positivo. Esta función espera factores como entradas y el primer nivel se considera como el resultado positivo,  $Y = 1$ :

```
confusionMatrix(data = y_hat, reference = test_set$sex)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction Female Male
##     Female      50    27
##     Male        69   379
##
##           Accuracy : 0.817
##                 95% CI : (0.781, 0.849)
##     No Information Rate : 0.773
##     P-Value [Acc > NIR] : 0.00835
##
##           Kappa : 0.404
##
## McNemar's Test P-Value : 2.86e-05
##
##           Sensitivity : 0.4202
##           Specificity : 0.9335
##     Pos Pred Value : 0.6494
##     Neg Pred Value : 0.8460
##           Prevalence : 0.2267
##     Detection Rate : 0.0952
##     Detection Prevalence : 0.1467
##     Balanced Accuracy : 0.6768
##
##     'Positive' Class : Female
##
```

Donde vemos que la especificidad es sustancialmente alta a pesar de que la sensibilidad no lo es, como habíamos dicho, debido a la prevalencia.

**8.2.1.2. Precisión balanceada y puntaje  $F_1$**  Resultaría muy conveniente tener un resumen de un solo número, sobre todo para propósitos de optimización. Una métrica preferida sobre la precisión general es el promedio de la especificidad y la sensibilidad, llamada **precisión balanceada**. Dado que estas dos métricas son un ratio, este número se calculará de la siguiente manera:

$$\mathbf{F_1 - score} = \frac{1}{\frac{1}{2} \left( \frac{1}{\text{recall}} + \frac{1}{\text{precision}} \right)} = 2 \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$

Dependiendo del contexto, la sensibilidad o especificidad pueden ser más importantes de maximizar. Por tanto, defínase  $\beta$  para representar cuán más importante es la sensibilidad comparada con la especificidad considerando un promedio armónico usando la siguiente fórmula:

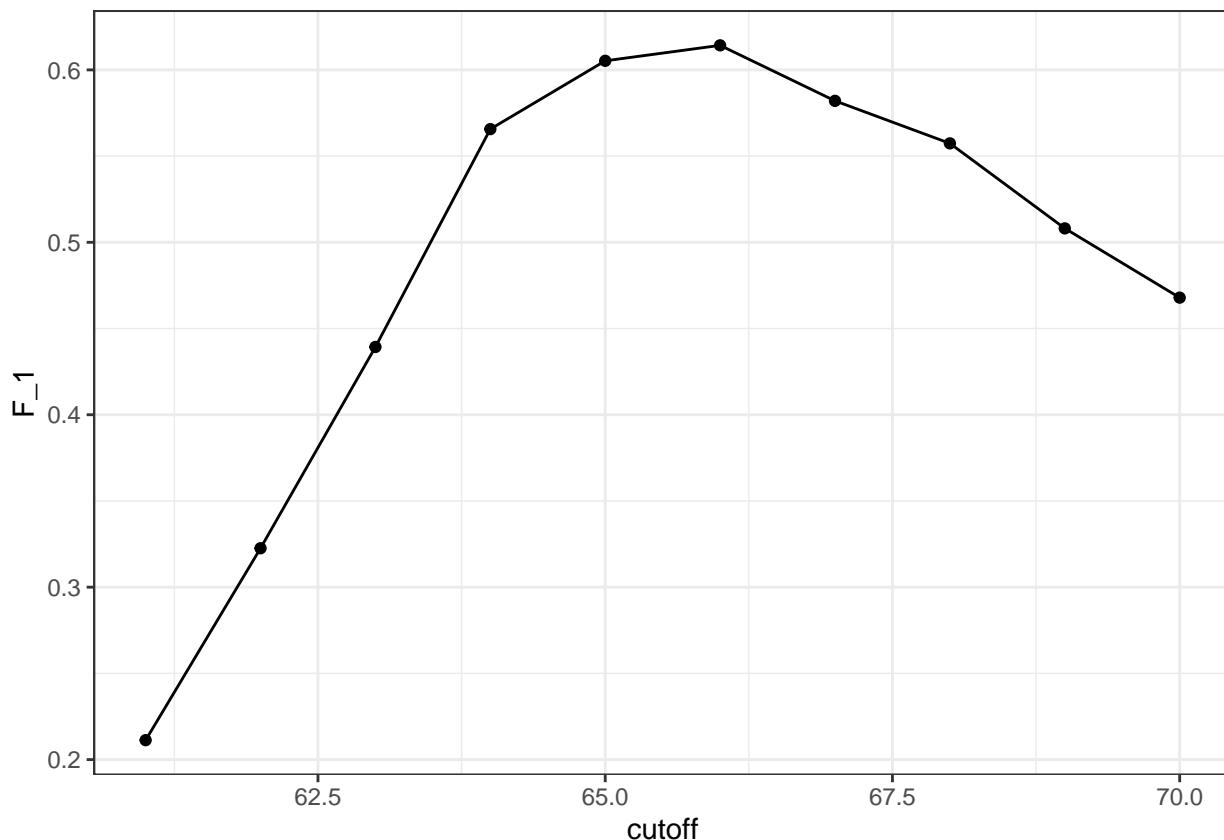
$$\frac{1}{\frac{\beta^2}{1+\beta^2} \frac{1}{\text{recall}} + \frac{1}{1+\beta^2} \frac{1}{\text{precision}}}$$

La función “F\_meas()” del paquete caret computa un resumen asumiendo  $\beta = 1$ . Reconstruyamos nuestro algoritmo de predicción pero ahora maximizando el puntaje F en lugar de la precisión general:

```
cutoff <- seq(61, 70)

F_1 <- map_dbl(cutoff, function(x){
  y_hat <- ifelse(train_set$height > x, "Male", "Female") %>%
    factor(levels = levels(test_set$sex))
  F_meas(data = y_hat, reference = factor(train_set$sex))
})

data.frame(cutoff, F_1) %>%
  ggplot(aes(cutoff, F_1)) +
  geom_point() +
  geom_line()
```



Donde ahora vemos que se maximiza en 61.4 % cuando usamos 66 pulgadas.

```
max(F_1)
## [1] 0.614
best_cutoff <- cutoff[which.max(F_1)]
best_cutoff
## [1] 66
```

Intuitivamente, el límite de 66 pulgadas tiene más sentido que el que habíamos computado de 64; además, balancea la especificidad y la sensibilidad en nuestra matriz de confusión:

```
y_hat <- ifelse(test_set$height > best_cutoff, "Male", "Female") %>%
  factor(levels = levels(test_set$sex))

confusionMatrix(data = y_hat, reference = test_set$sex)

## Confusion Matrix and Statistics
##
##          Reference
## Prediction Female Male
##     Female      81   67
##     Male        38  339
##
##          Accuracy : 0.8
## 95% CI : (0.763, 0.833)
```

```

##      No Information Rate : 0.773
##      P-Value [Acc > NIR] : 0.07819
##
##                  Kappa : 0.475
##
## McNemar's Test P-Value : 0.00629
##
##      Sensitivity : 0.681
##      Specificity : 0.835
##      Pos Pred Value : 0.547
##      Neg Pred Value : 0.899
##      Prevalence : 0.227
##      Detection Rate : 0.154
##      Detection Prevalence : 0.282
##      Balanced Accuracy : 0.758
##
##      'Positive' Class : Female
##

```

Con esto concluimos nuestro primer algoritmo de aprendizaje automático, el cual toma como predictor a la altura y predice que un individuo es mujer si mide 66 pulgadas o menos.

**8.2.1.3. Prevalencia en la práctica** Al utilizar datos reales, podemos encontrar prevalencias cercanas a 0 ó 1. Para ejemplificar esto, considérese un médico especializado en una enfermedad que desarrolla un algoritmo para predecir quién tiene dicha enfermedad.

Así, al desarrollar el algoritmo, encontramos que tiene una alta sensibilidad, lo cual implica que si el paciente tiene la enfermedad, el algoritmo la predecirá correctamente en casi todos los casos. Por otro lado, le hacemos notar al doctor que la mitad de los pacientes estudiados en la base de datos tienen la enfermedad, esto es,  $Pr(\hat{Y} = 1) = 1/2$ .

En respuesta, el médico asegura que lo importante es la precisión de la prueba,  $Pr(Y = 1|\hat{Y} = 1)$ . Utilizando el Teorema de Bayes, podemos conectar estas dos medidas:

$$Pr(Y = 1|\hat{Y} = 1) = Pr(\hat{Y} = 1|Y = 1) \frac{Pr(Y = 1)}{Pr(\hat{Y} = 1)}$$

También sabemos que la prevalencia de la enfermedad es 5 en 1 000; como mencionamos, la prevalencia en la base de datos es de 50 %. Esto implica que:

$$\frac{Pr(Y = 1)}{Pr(\hat{Y} = 1)} = \frac{5/1000}{1/2} = 1/100$$

Es decir, la precisión del algoritmo es menor a 0.01, por lo que este no es útil para el médico.

**8.2.1.4. ROC y curvas *precision-recall*** Al comparar dos métodos de predicción, por ejemplo, adivinar vs. un límite de altura, miramos a las métricas de precisión y  $F_1$ . Para el segundo método, el cual tuvo un rendimiento positivo, consideramos límites de altura; por su parte, para el primero solo consideramos adivinar con probabilidades iguales (nótese que si hubiéramos adivinado “hombre” con mayor probabilidad, hubiéramos tenido un mejor resultado, dada la prevalencia de la muestra):

```

p <- 0.9

n <- length(test_index)
y_hat <- sample(c("Male", "Female"), n, replace = TRUE, prob=c(p, 1-p)) %>%

```

```

factor(levels = levels(test_set$sex))

mean(y_hat == test_set$sex)

## [1] 0.718

```

Si bien la predicción aumentó considerablemente, esto ocurre a expensas de una baja sensibilidad. Así, gráficamente podemos comparar diferentes valores y enfoques: la más común es la *receiver operating characteristic (ROC) curve*, la cual grafica sensibilidad (TPR) vs. 1 - especificidad o FPR.

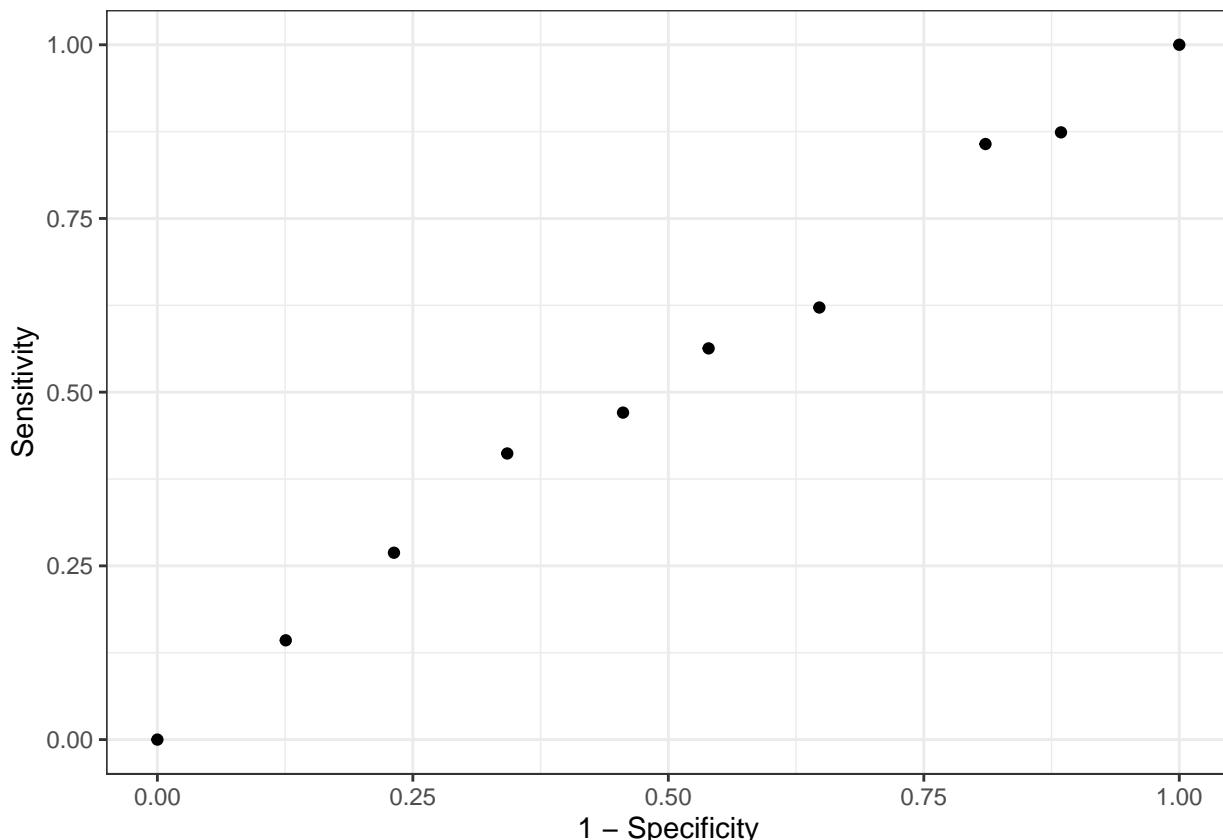
Así, construyase una curva ROC para el enfoque de adivinar:

```

probs <- seq(0, 1, length.out = 10)
guessing <- map_df(probs, function(p){
  y_hat <-
    sample(c("Male", "Female"), n, replace = TRUE, prob=c(p, 1-p)) %>%
    factor(levels = c("Female", "Male"))
  list(method = "Guessing",
       FPR = 1 - specificity(y_hat, test_set$sex),
       TPR = sensitivity(y_hat, test_set$sex))
})

guessing %>%
  qplot(FPR, TPR, data = ., xlab = "1 - Specificity", ylab = "Sensitivity")

```

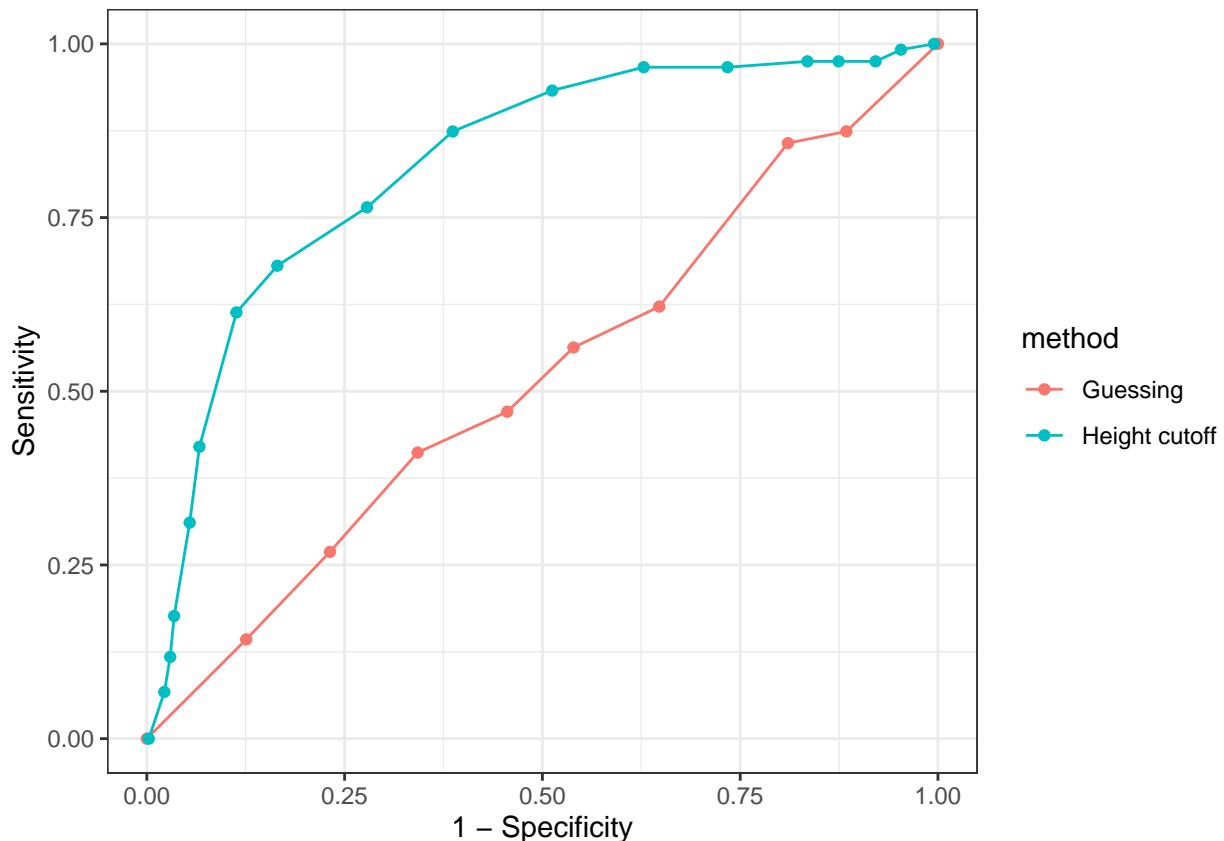


¿Cómo se comporta el segundo enfoque, de limitar altura?

```
cutoffs <- c(50, seq(60, 75), 80)

height_cutoff <- map_df(cutoffs, function(x){
  y_hat <- ifelse(test_set$height > x, "Male", "Female") %>%
    factor(levels = c("Female", "Male"))
  list(method = "Height cutoff",
       FPR = 1-specificity(y_hat, test_set$sex),
       TPR = sensitivity(y_hat, test_set$sex))
})

bind_rows(guessing, height_cutoff) %>%
  ggplot(aes(FPR, TPR, color = method)) +
  geom_line() +
  geom_point() +
  xlab("1 - Specificity") +
  ylab("Sensitivity")
```



Podemos ver que obtenemos una sensibilidad mayor con la limitación de altura, lo que implica que se trata de un mejor método. Nótese que al graficar curvas ROC, puede ser útil agregar etiquetas de los límites impuestos:

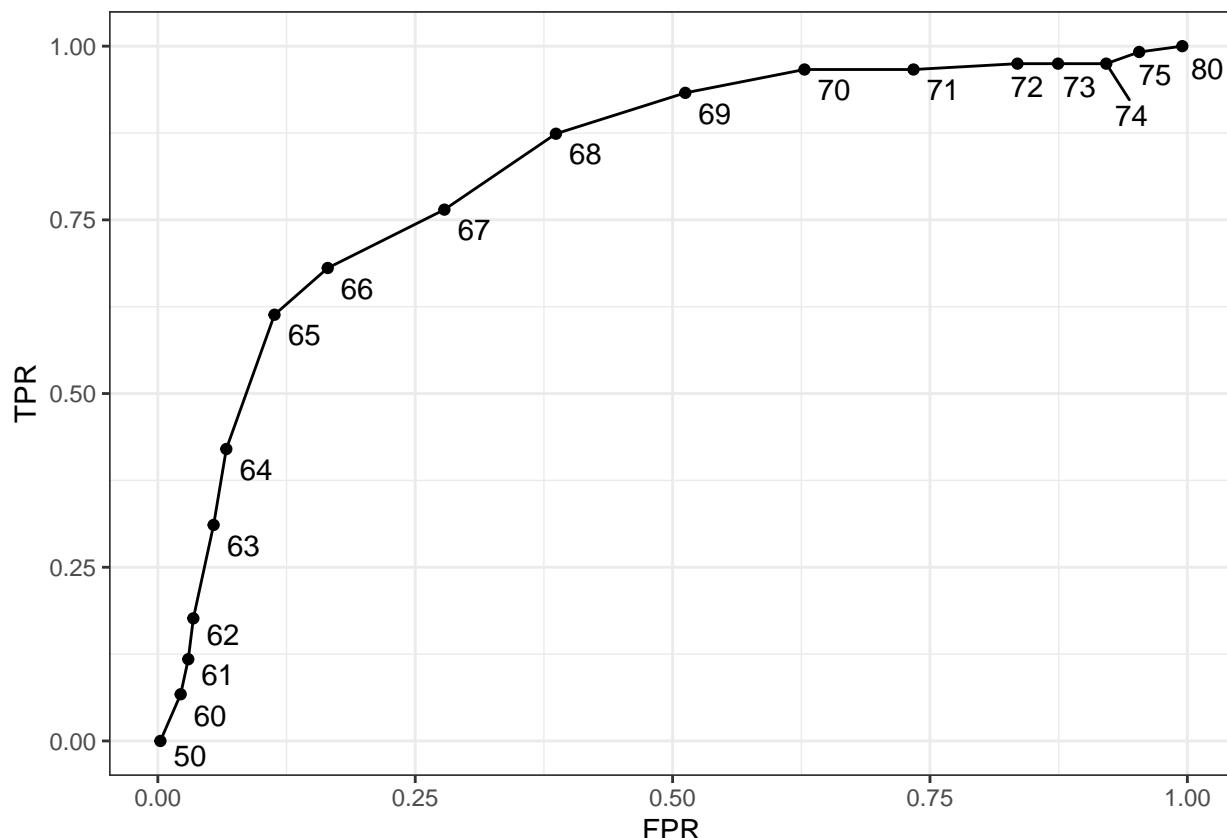
```
library(ggrepel)

map_df(cutoffs, function(x){
  y_hat <- ifelse(test_set$height > x, "Male", "Female") %>%
    factor(levels = c("Female", "Male"))
  list(method = "Height cutoff",
       cutoff = x,
```

```

FPR = 1-specificity(y_hat, test_set$sex),
TPR = sensitivity(y_hat, test_set$sex)
}) %>%
ggplot(aes(FPR, TPR, label = cutoff)) +
geom_line() +
geom_point() +
geom_text_repel(nudge_x = 0.01, nudge_y = -0.01)

```



Así, es clara la utilidad de las curvas ROC; no obstante, mantienen un problema ya conocido: no depende de la prevalencia de los datos. En contextos donde la prevalencia es relevante, podemos recurrir a un *precision-recall plot*, donde la idea es similar pero graficamos precisión vs. *recall*.

```

guessing <- map_df(probs, function(p){
  y_hat <- sample(c("Male", "Female"), length(test_index),
                 replace = TRUE, prob=c(p, 1-p)) %>%
    factor(levels = c("Female", "Male"))
  list(method = "Guess",
       recall = sensitivity(y_hat, test_set$sex),
       precision = precision(y_hat, test_set$sex))
})

height_cutoff <- map_df(cutoffs, function(x){
  y_hat <- ifelse(test_set$height > x, "Male", "Female") %>%
    factor(levels = c("Female", "Male"))
  list(method = "Height cutoff",
       recall = sensitivity(y_hat, test_set$sex),
       precision = precision(y_hat, test_set$sex))
})

```

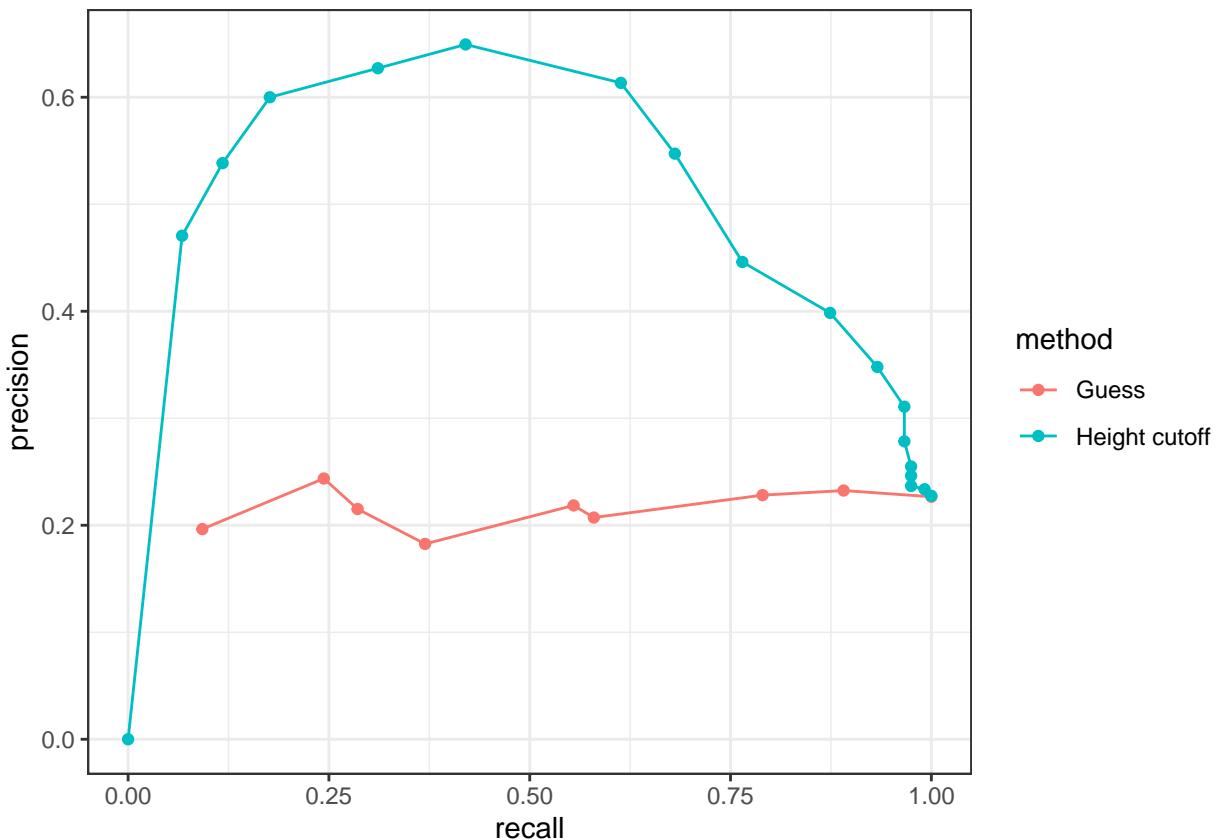
```

    recall = sensitivity(y_hat, test_set$sex),
    precision = precision(y_hat, test_set$sex))
})

bind_rows(guessing, height_cutoff) %>%
  ggplot(aes(recall, precision, color = method)) +
  geom_line() +
  geom_point()

## Warning: Removed 1 row(s) containing missing values (geom_path).
## Warning: Removed 1 rows containing missing values (geom_point).

```



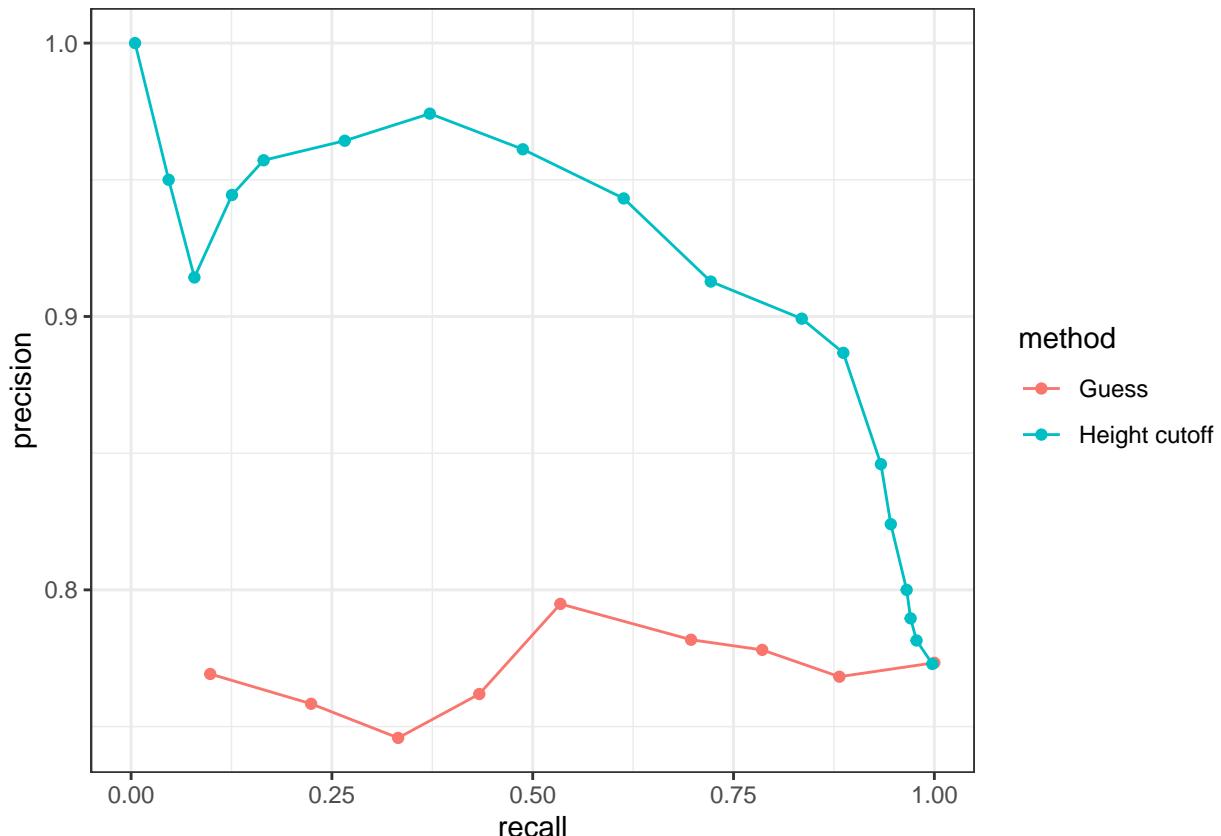
```

guessing <- map_df(probs, function(p){
  y_hat <- sample(c("Male", "Female"), length(test_index), replace = TRUE,
                 prob=c(p, 1-p)) %>%
    factor(levels = c("Male", "Female"))
  list(method = "Guess",
       recall = sensitivity(y_hat, relevel(test_set$sex, "Male", "Female")),
       precision = precision(y_hat, relevel(test_set$sex, "Male", "Female")))
})

```

Observamos que la precisión de adivinar no es muy alta, debido a que la prevalencia es baja. Si modificamos a que nuestros positivos sean ahora hombres:

```
height_cutoff <- map_df(cutoffs, function(x){  
  y_hat <- ifelse(test_set$height > x, "Male", "Female") %>%  
    factor(levels = c("Male", "Female"))  
  list(method = "Height cutoff",  
       recall = sensitivity(y_hat, relevel(test_set$sex, "Male", "Female")),  
       precision = precision(y_hat, relevel(test_set$sex, "Male", "Female")))  
})  
  
bind_rows(guessing, height_cutoff) %>%  
  ggplot(aes(recall, precision, color = method)) +  
  geom_line() +  
  geom_point()  
  
## Warning: Removed 1 row(s) containing missing values (geom_path).  
## Warning: Removed 1 rows containing missing values (geom_point).
```



### 8.2.2. Assessment 19

The reported\_heights and heights datasets were collected from three classes taught in the Departments of Computer Science and Biostatistics, as well as remotely through the Extension School. The Biostatistics class was taught in 2016 along with an online version offered by the Extension School. On 2016-01-25 at 8:15 AM, during one of the lectures, the instructors asked student to fill in the sex and height questionnaire that populated the reported\_heights dataset. The online students filled out the survey during the next few days, after the lecture was posted online. We can use this insight to define a variable which we will call type, to denote the type of student, inclass or online.

```
library(dslabs)
library(dplyr)
library(lubridate)
data(reported_heights)

dat <- mutate(reported_heights, date_time = ymd_hms(time_stamp)) %>%
  filter(date_time >= make_date(2016, 01, 25) & date_time < make_date(2016, 02, 1)) %>%
  mutate(type = ifelse(day(date_time) == 25 & hour(date_time) == 8 & between(minute(date_time), 15, 30),
                        "online", "inclass"))
  select(sex, type)

y <- factor(dat$sex, c("Female", "Male"))
x <- dat$type
```

The type column of dat indicates whether students took classes in person (“inclass”) or online (“online”). What proportion of the inclass group is female? What proportion of the online group is female?

```
dat %>%
  group_by(type) %>%
  summarize(prope_female = mean(sex == "Female"))

## # A tibble: 2 x 2
##   type    prope_female
##   <chr>      <dbl>
## 1 inclass     0.667
## 2 online      0.378
```

In the course videos, height cutoffs were used to predict sex. Instead of height, use the type variable to predict sex. Assume that for each class type the students are either all male or all female, based on the most prevalent sex in each class type you calculated in Q1. Report the accuracy of your prediction of sex based on type. You do not need to split the data into training and test sets.

```
y_hat <- ifelse(x == "online", "Male", "Female") %>%
  factor(levels = levels(y))

mean(y_hat==y)

## [1] 0.633

Write a line of code using the table() function to show the confusion matrix between y_hat and y. Use the exact format function(a, b) for your answer and do not name the columns and rows. Your answer should have exactly one space. Enter the line of code below.

table(y_hat, y)

##
```

```
## y_hat      Female Male
##   Female      26    13
##   Male       42    69
```

What is the sensitivity of this prediction? You can use the sensitivity() function from the caret package. Enter your answer as a percentage or decimal (eg “50 %” or “0.50”) to at least the hundredths place.

```
library(caret)
```

```
sensitivity(y_hat, y)
```

```
## [1] 0.382
```

What is the specificity of this prediction? You can use the specificity() function from the caret package. Enter your answer as a percentage or decimal (eg “50 %” or “0.50”) to at least the hundredths place.

```
specificity(y_hat, y)
```

```
## [1] 0.841
```

What is the prevalence (% of females) in the dat dataset defined above? Enter your answer as a percentage or decimal (eg “50 %” or “0.50”) to at least the hundredths place.

```
mean(y == "Female")
```

```
## [1] 0.453
```

We will practice building a machine learning algorithm using a new dataset, iris, that provides multiple predictors for us to use to train. To start, we will remove the setosa species and we will focus on the versicolor and virginica iris species using the following code:

```
library(caret)
```

```
data(iris)
```

```
iris <- iris[-which(iris$Species=="setosa"),]
y <- iris$Species
```

First let us create an even split of the data into train and test partitions using createDataPartition() from the caret package. The code with a missing line is given below:

```
set.seed(2, sample.kind="Rounding")
```

```
## Warning in set.seed(2, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
```

```
## used
```

```
test_index <- createDataPartition(y,times=1,p=0.5,list=FALSE)
```

```
## Warning in createDataPartition(y, times = 1, p = 0.5, list = FALSE): Some
```

```
## classes have no records ( setosa ) and these will be ignored
```

```
test <- iris[test_index,]
```

```
train <- iris[-test_index,]
```

Next we will figure out the singular feature in the dataset that yields the greatest overall accuracy when predicting species. You can use the code from the introduction and from Q7 to start your analysis.

Using only the train iris dataset, for each feature, perform a simple search to find the cutoff that produces the highest accuracy, predicting virginica if greater than the cutoff and versicolor

otherwise. Use the seq function over the range of each feature by intervals of 0.1 for this search.

```
foo <- function(x){  
  rangedValues <- seq(range(x)[1],range(x)[2],by=0.1)  
  sapply(rangedValues,function(i){  
    y_hat <- ifelse(x>i,'virginica','versicolor')  
    mean(y_hat==train$Species)  
  })  
}  
predictions <- apply(train[,-5],2,foo)  
sapply(predictions,max)
```

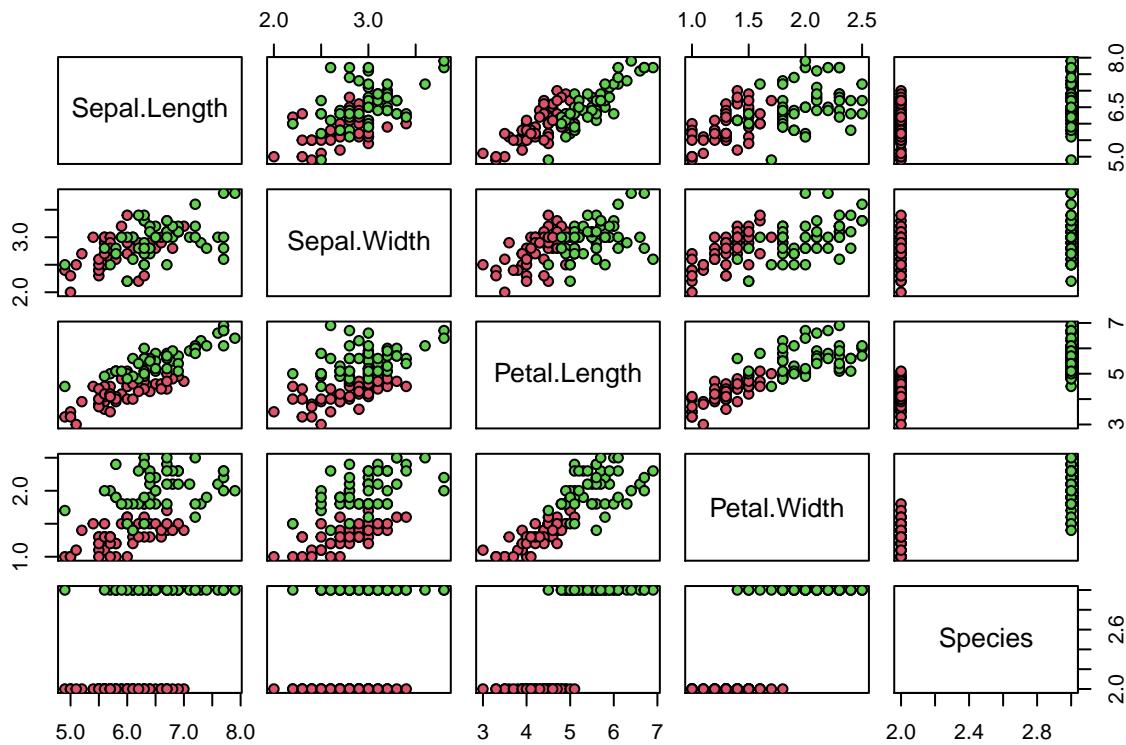
```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width  
##          0.70        0.62        0.96        0.94
```

For the feature selected in Q8, use the smart cutoff value from the training data to calculate overall accuracy in the test data. What is the overall accuracy?

```
predictions <- foo(train[,3])  
  
rangedValues <- seq(range(train[,3])[1],range(train[,3])[2],by=0.1)  
  
cutoffs <- rangedValues[which(predictions==max(predictions))]  
  
y_hat <- ifelse(test[,3]>cutoffs[1],'virginica','versicolor')  
  
mean(y_hat==test$Species)  
  
## [1] 0.9
```

Now we will perform some exploratory data analysis on the data.

```
plot(iris,pch=21,bg=iris$Species)
```



Notice that Petal.Length and Petal.Width in combination could potentially be more information than either feature alone. Optimize the the cutoffs for Petal.Length and Petal.Width separately in the train dataset by using the seq function with increments of 0.1. Then, report the overall accuracy when applied to the test dataset by creating a rule that predicts virginica if Petal.Length is greater than the length cutoff OR Petal.Width is greater than the width cutoff, and versicolor otherwise. What is the overall accuracy for the test data now?

```
set.seed(2)

test_index <- createDataPartition(y, times=1, p=0.5, list=FALSE)

## Warning in createDataPartition(y, times = 1, p = 0.5, list = FALSE): Some
## classes have no records ( setosa ) and these will be ignored

test <- iris[test_index,]
train <- iris[-test_index,]

petalLengthRange <- seq(range(train$Petal.Length) [1], range(train$Petal.Length) [2], by=0.1)

petalWidthRange <- seq(range(train$Petal.Width) [1], range(train$Petal.Width) [2], by=0.1)

length_predictions <- sapply(petalLengthRange, function(i){
  y_hat <- ifelse(train$Petal.Length>i, 'virginica', 'versicolor')
  mean(y_hat==train$Species)
})

length_cutoff <- petalLengthRange[which.max(length_predictions)]

width_predictions <- sapply(petalWidthRange, function(i){
```

```
y_hat <- ifelse(train$Petal.Width>i, 'virginica', 'versicolor')
mean(y_hat==train$Species)
})
width_cutoff <- petalWidthRange[which.max(width_predictions)]  

y_hat <- ifelse(test$Petal.Length>length_cutoff |
                  test$Petal.Width>width_cutoff, 'virginica', 'versicolor')

mean(y_hat==test$Species)

## [1] 0.88
```

### 8.2.3. Probabilidades condicionales y función de pérdida

Al momento de lidiar con problemas de aprendizaje automático, la representación probabilística de estos puede ser muy útil.

Considérese un ejemplo de datos categóricos:  $(X_1 = x_1, \dots, X_p = x_p)$  son nuestros datos observados que van de  $x_1, \dots, x_p$  con covariables  $X_1, \dots, X_p$ . Esto no implica que nuestro resultado tomará un resultado específico, sino que este implica una probabilidad específica.

Específicamente, denotamos probabilidades condicionales para cada clase  $k$ :

$$\Pr(\mathbf{Y} = \mathbf{k} | \mathbf{X}_1 = \mathbf{x}_1, \dots, \mathbf{X}_p = \mathbf{x}_p) \quad \forall \quad \mathbf{k} = 1, \dots, K$$

Que implica que la probabilidad de la clase  $k$  es:

$$p_k(x) = \Pr(\mathbf{Y} = k | \mathbf{X} = x) \quad \forall \quad k = 1, \dots, K$$

Para cualquier conjunto de predictores  $X$ , predeciremos la clase  $k$  con la probabilidad más grande de entre  $p_1(x), \dots, p_K(x)$ :

$$\hat{Y} = \max_k p_k(x)$$

Donde el reto es encontrar estas  $p_k$ : mientras mejores sean las predicciones de  $\hat{p}_k(x)$ , mejor será nuestro algoritmo y predictor  $\hat{Y} = \max_k \hat{p}_k(x)$ . Lo anterior dependerá de dos cosas:

1. Qué tan cerca está  $\max_k p_k(x)$  de 1 (nótese que esta condición es inherente a la naturaleza del problema y no podemos cambiarla)
2. Qué tan cerca está la probabilidad estimada de la real,  $\hat{p}_k(x)$  de  $p_k(x)$ .

Con datos categóricos, puede pensarse a la probabilidad condicional  $Pr(Y = 1 | X = x)$  como la proporción de 1s en el estrato de población para el cual  $X = x$ .

Muchos de los algoritmos que revisaremos en las siguientes secciones pueden ser aplicados a datos continuos o categóricos gracias a la conexión entre probabilidades y esperanzas condicionales.

Así, con datos categóricos, la esperanza es equivalente a la probabilidad de obtener un 1, dado que el promedio es simplemente la proporción de 1s.

En el caso continuo, las métricas que hemos revisado (especificidad, sensibilidad, precisión y  $F_1$ ) no son adecuadas. Así, como alternativa, definiremos una **función de pérdida**; la más común es la función de pérdida cuadrática. Sea  $\hat{Y}$  el predictor y  $Y$  el resultado real, entonces la función de pérdida es:

$$(\hat{Y} - Y)^2$$

Si tenemos  $n$  observaciones, entonces:

$$\frac{1}{N} \sum_{i=1}^N (\hat{Y}_i - Y_i)^2$$

Nótese que si los resultados son binarios, el Error Cuadrático Medio (MSE) y la raíz del error cuadrático medio (RMSE) son equivalentes a  $1 - \text{precisión}$ , dado que  $(\hat{y} - y)^2$  es 0 si la predicción es correcta.

En general, el objetivo es construir un algoritmo que minimice el promedio de la pérdida cuadrática a través de muchas muestras aleatorias, lo que implica que sea lo más cercano posible a 0. Dado que nuestras muestras son aleatorias, entonces el MSE es una variable aleatoria:

$$\mathbf{E} \left[ \frac{1}{N} \sum_{i=1}^N (\hat{Y}_i - Y_i)^2 \right]$$

Entonces, ¿por qué es relevante la esperanza condicional para el aprendizaje automático? Esto es debido a una característica relevante: **el valor esperado minimiza la pérdida cuadrática esperada**:

$$\hat{Y} = \mathbf{E}[Y|X = x] \text{ minimiza } \mathbf{E}\{(\hat{Y} - Y)^2 | X = x\}$$

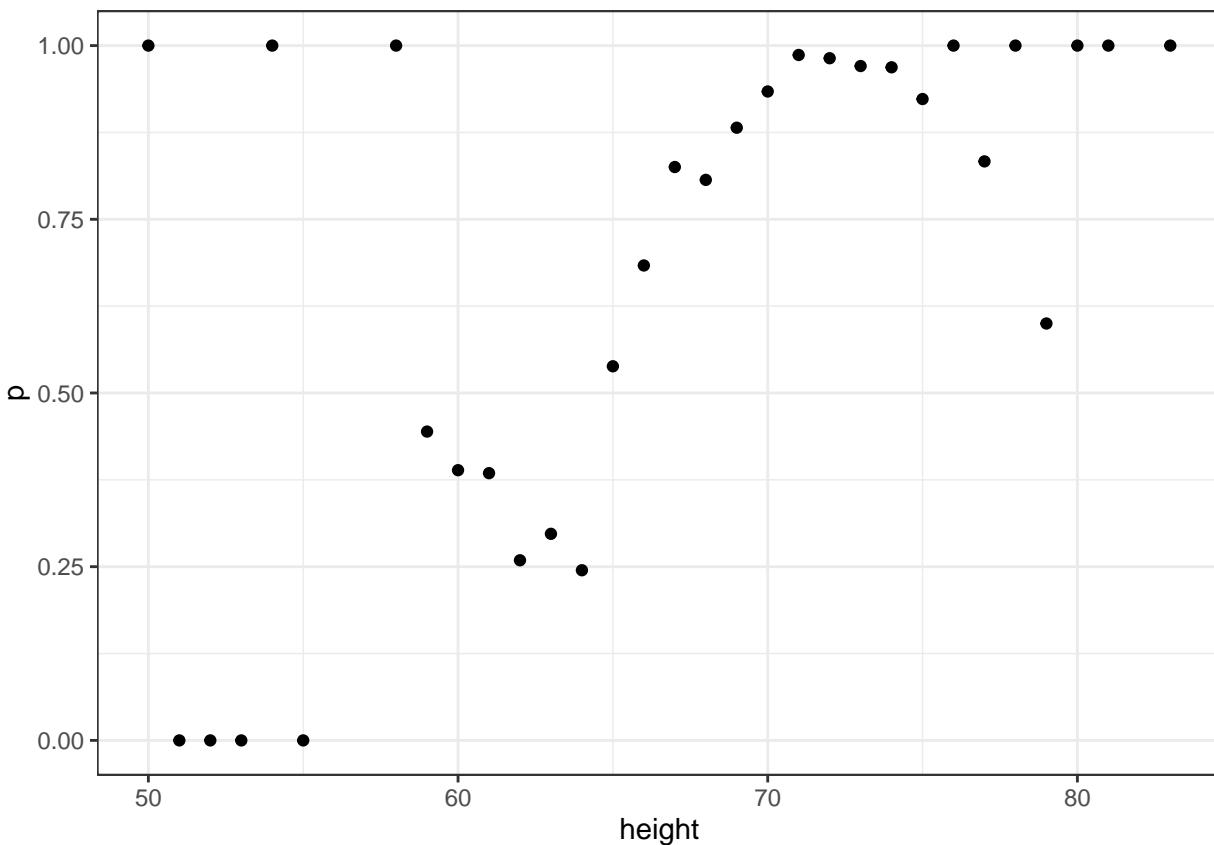
De manera sucinta: **el principal objetivo del aprendizaje automático es utilizar datos para estimar probabilidades condicionales**  $f(x) = E[Y|X = x]$  para cualquier conjunto de características  $x = (x_1, \dots, x_p)$ . La principal forma en que los algoritmos de aprendizaje automático difieren unos de otros es en su enfoque para estimar esta expectativa.

#### 8.2.4. Assessment 20

We are now going to write code to compute conditional probabilities for being male in the heights dataset. Round the heights to the closest inch. Plot the estimated conditional probability  $P(x) = \Pr(\text{Male}|\text{height} = x)$  for each  $x$ . Part of the code is provided here:

```
library(dslabs)
library(tidyverse)
data("heights")

heights %>%
  mutate(height = round(height)) %>%
  group_by(height) %>%
  summarize(p = mean(sex == "Male")) %>%
  qplot(height, p, data = .)
```

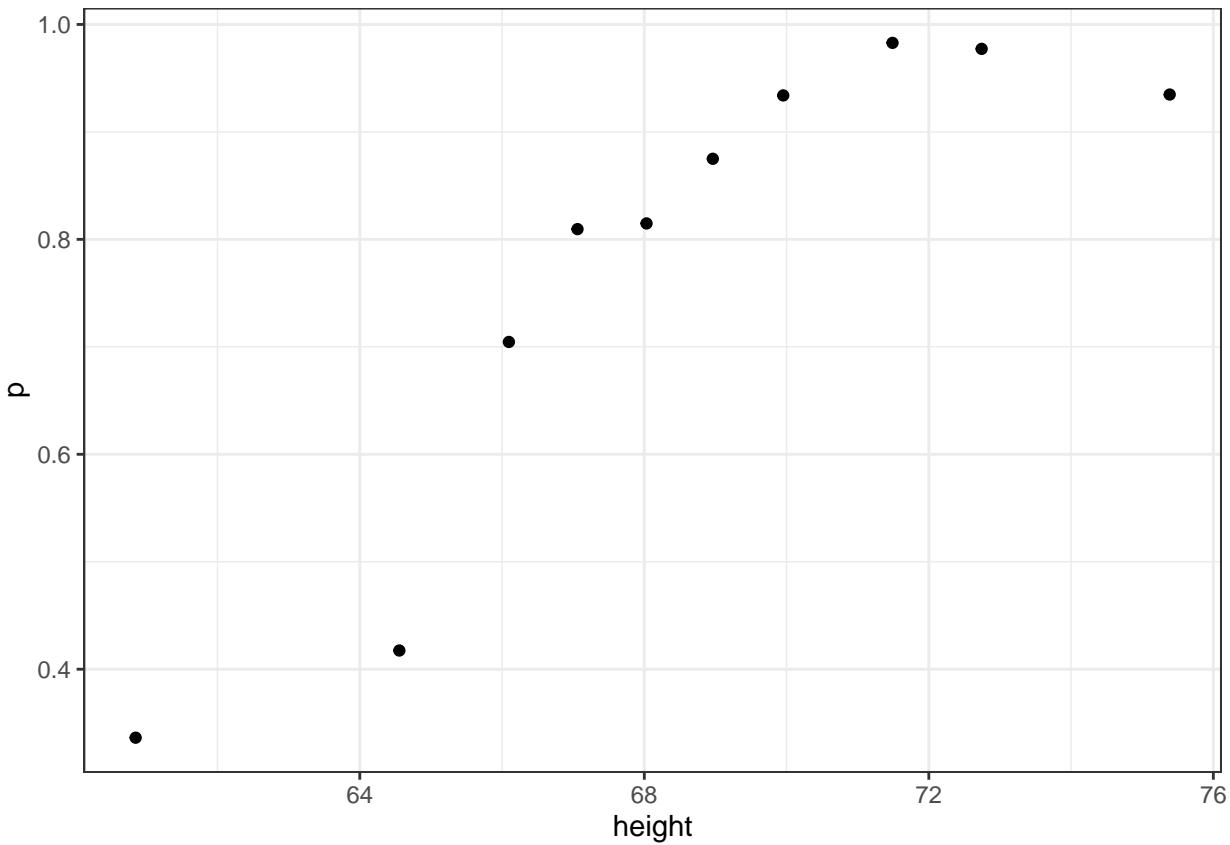


In the plot we just made in Q6 we see high variability for low values of height. This is because we have few data points. This time use the quantile and the `cut()` function to assure each group has the same number of points. Note that for any numeric vector  $x$ , you can create groups based on quantiles like this: `cut(x, quantile(x, seq(0, 1, 0.1)))`, `include.lowest = TRUE`). Part of the code is provided here:

```
ps <- seq(0, 1, 0.1)

heights %>%
  mutate(g = cut(height, quantile(height, ps), include.lowest = TRUE)) %>%
  group_by(g) %>%
  summarize(p = mean(sex == "Male"), height = mean(height)) %>%
```

```
qplot(height, p, data = .)
```



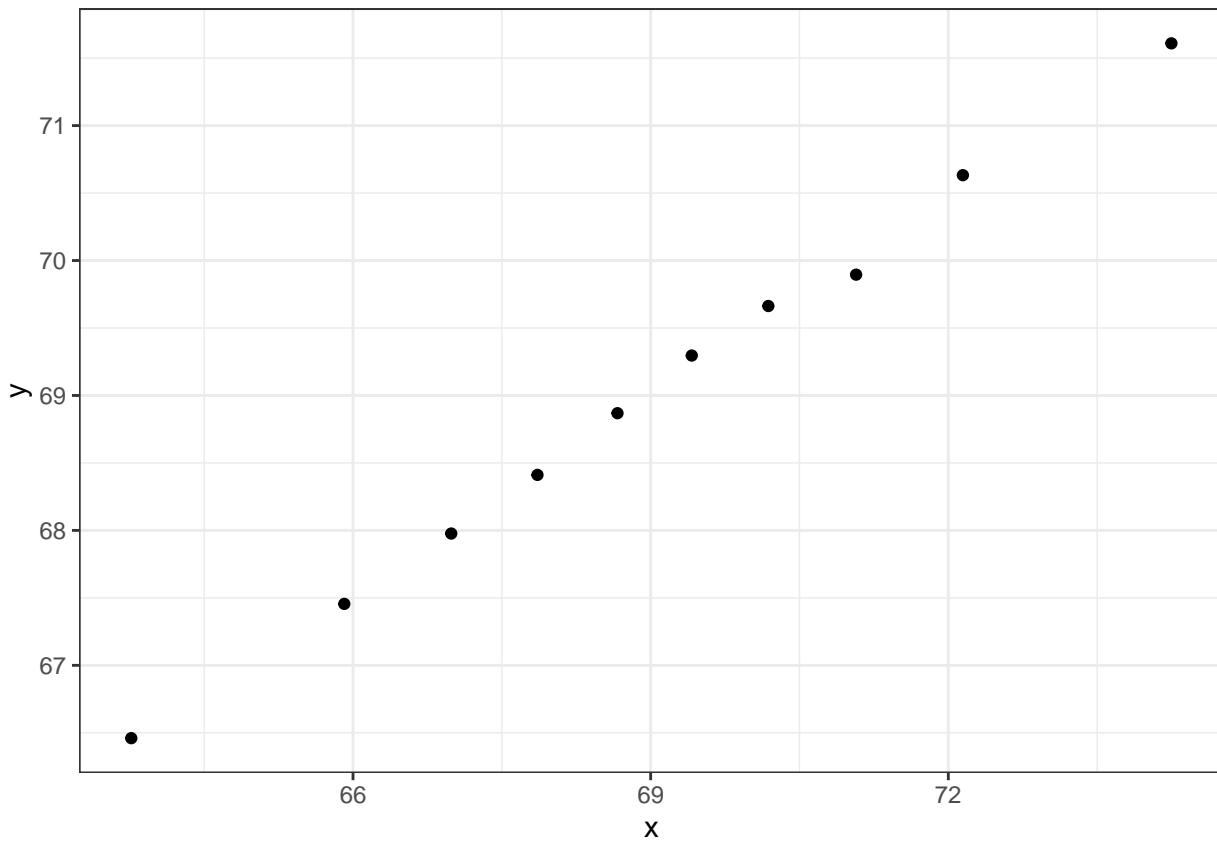
You can generate data from a bivariate normal distribution using the MASS package using the following code:

```
Sigma <- 9*matrix(c(1,0.5,0.5,1), 2, 2)

dat <- MASS::mvrnorm(n = 10000, c(69, 69), Sigma) %>%
  data.frame() %>%
  setNames(c("x", "y"))
```

And you can make a quick plot using `plot(dat)`. Using an approach similar to that used in the previous exercise, let's estimate the conditional expectations and make a plot. Part of the code has again been provided for you:

```
ps <- seq(0, 1, 0.1)
dat %>%
  mutate(g = cut(x, quantile(x, ps), include.lowest = TRUE)) %>%
  group_by(g) %>%
  summarize(y = mean(y), x = mean(x)) %>%
  qplot(x, y, data = .)
```



## 8.3. Regresiones lineales para predicción, suavización y trabajo con matrices

### 8.3.1. Regresiones lineales para predicciones

La utilización de regresiones lineales para hacer predicciones puede ser consideradas como un algoritmo de aprendizaje automático. Para exemplificar lo anterior, retomaremos el ejemplo de alturas de Galton:

```
library(HistData)
library(tidyverse)

galton_heights <- GaltonFamilies %>% filter(childNum == 1 & gender == "male") %>%
  select(father, childHeight) %>%
  rename(son = childHeight)
```

Supóngase que queremos construir un algoritmo que prediga la altura de los hijos,  $Y$ , utilizando la altura del padre,  $X$ . Como recordaremos, el primer paso es definir los conjuntos de práctica y prueba mediante el siguiente código:

```
library(caret)

y <- galton_heights$son

text_index <- createDataPartition(y, times = 1, p = 0.5, list = FALSE)

train_set <- galton_heights %>%
  slice(-text_index)

test_set <- galton_heights %>%
  slice(text_index)
```

En caso de que estuviéramos adivinando y sin tomar en cuenta la altura de los padres, nuestra predicción sería como sigue:

```
avg <- mean(train_set$son)
```

```
avg
```

```
## [1] 70.5
```

Donde nuestra pérdida al cuadrado es:

```
mean((avg - test_set$son)^2)
```

```
## [1] 7.03
```

Recuérdese que si un par  $(X, Y)$  sigue una distribución normal bivariada (como es el caso de las alturas de hijos y padres), entonces la esperanza condicional es equivalente a la línea de regresión:

$$f(x) = E[Y|X = x] = \beta_0 + \beta_1 x$$

Así, mediante MCO, nuestro modelo estimado tiene los parámetros:

```
fit <- lm(son ~ father, data = train_set)

fit$coef

## (Intercept)      father
##       39.671      0.447
```

Que implica que nuestro modelo estimado es:

$$\hat{f}(x) = 38,26 + 0,45x$$

Podemos ver que esto es una mejora de la pérdida que calculamos con solo adivinar:

```
y_hat <- fit$coef[1]+fit$coef[2]*test_set$father  
  
mean((y_hat-test_set$son)^2)  
  
## [1] 5.11
```

La función “predict()” es sumamente útil en aplicaciones de *machine learning*; dicha función toma un objeto estimado de las funciones “lm()” o “glm()”, así como un *data frame* con los nuevos predictores para los cuales predecir. Así, en nuestro ejemplo, en lugar de escribir la fórmula de nuestra línea de regresión, podemos usar “predict()” de la siguiente forma:

```
y_hat <- predict(fit, test_set)
```

Con lo cual obtenemos la misma pérdida:

```
mean((y_hat- test_set$son)^2)  
  
## [1] 5.11
```

### 8.3.2. Assessment 21

Create a data set using the following code:

```
library(tidyverse)  
library(caret)  
  
  
set.seed(1, sample.kind="Rounding")  
  
n <- 100  
  
Sigma <- 9*matrix(c(1.0, 0.5, 0.5, 1.0), 2, 2)  
  
dat <- MASS::mvrnorm(n = 100, c(69, 69), Sigma) %>%  
  data.frame() %>% setNames(c("x", "y"))
```

We will build 100 linear models using the data above and calculate the mean and standard deviation of the combined models. First, set the seed to 1 again (make sure to use sample.kind=“Rounding” if your R is version 3.6 or later). Then, within a replicate() loop, (1) partition the dataset into test and training sets with p = 0.5 and using dat\$y to generate your indices, (2) train a linear model predicting y from x, (3) generate predictions on the test set, and (4) calculate the RMSE of that model. Then, report the mean and standard deviation (SD) of the RMSEs from all 100 models.

```
set.seed(1, sample.kind="Rounding")  
  
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler  
## used  
  
rmse <- replicate(100,{  
  test_index <- createDataPartition(dat$y, times = 1, p = 0.5, list = FALSE)  
  train_set <- dat %>%
```

```

  slice(-test_index)
  test_set <- dat %>%
    slice(test_index)
  fit <- lm(y ~ x, data = train_set)
  y_hat <- predict(fit, newdata = test_set)
  sqrt(mean((y_hat-test_set$y)^2))
}

mean(rmse)

## [1] 2.49
sd(rmse)

## [1] 0.124

```

Now we will repeat the exercise above but using larger datasets. Write a function that takes a size n, then (1) builds a dataset using the code provided at the top of Q1 but with n observations instead of 100 and without the set.seed(1), (2) runs the replicate() loop that you wrote to answer Q1, which builds 100 linear models and returns a vector of RMSEs, and (3) calculates the mean and standard deviation of the 100 RMSEs.

Set the seed to 1 (if using R 3.6 or later, use the argument sample.kind="Rounding") and then use sapply() or map() to apply your new function to n <- c(100, 500, 1000, 5000, 10000).

Note: You only need to set the seed once before running your function; do not set a seed within your function. Also be sure to use sapply() or map() as you will get different answers running the simulations individually due to setting the seed.

```

set.seed(1, sample.kind="Rounding")

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used

n <- c(100, 500, 1000, 5000, 10000)

res <- sapply(n, function(n){
  Sigma <- 9*matrix(c(1.0, 0.5, 0.5, 1.0), 2, 2)
  dat <- MASS::mvrnorm(n, c(69, 69), Sigma) %>%
    data.frame() %>% setNames(c("x", "y"))
  rmse <- replicate(100, {
    test_index <- createDataPartition(dat$y, times = 1, p = 0.5, list = FALSE)
    train_set <- dat %>% slice(-test_index)
    test_set <- dat %>% slice(test_index)
    fit <- lm(y ~ x, data = train_set)
    y_hat <- predict(fit, newdata = test_set)
    sqrt(mean((y_hat-test_set$y)^2))
  })
  c(avg = mean(rmse), sd = sd(rmse))
})

res

##      [,1] [,2]  [,3]  [,4]  [,5]
## avg 2.498 2.72 2.5555 2.6248 2.6184
## sd  0.118 0.08 0.0456 0.0231 0.0169

```

Now repeat the exercise from Q1, this time making the correlation between x and y larger, as

in the following code:

```
set.seed(1, sample.kind="Rounding")

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
n <- 100

Sigma <- 9*matrix(c(1.0, 0.95, 0.95, 1.0), 2, 2)

dat <- MASS::mvrnorm(n = 100, c(69, 69), Sigma) %>%
  data.frame() %>%
  setNames(c("x", "y"))

set.seed(1)

rmse <- replicate(100, {
  test_index <- createDataPartition(dat$y, times = 1, p = 0.5, list = FALSE)
  train_set <- dat %>% slice(-test_index)
  test_set <- dat %>% slice(test_index)
  fit <- lm(y ~ x, data = train_set)
  y_hat <- predict(fit, newdata = test_set)
  sqrt(mean((y_hat-test_set$y)^2))
})

mean(rmse)

## [1] 0.91
sd(rmse)

## [1] 0.0624
```

Create a data set using the following code.

```
set.seed(1, sample.kind="Rounding")

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
Sigma <- matrix(c(1.0, 0.75, 0.75, 0.75, 1.0, 0.25, 0.75, 0.25, 1.0), 3, 3)

dat <- MASS::mvrnorm(n = 100, c(0, 0, 0), Sigma) %>%
  data.frame() %>%
  setNames(c("y", "x_1", "x_2"))
```

Note that  $y$  is correlated with both  $x_1$  and  $x_2$  but the two predictors are independent of each other, as seen by `cor(dat)`. Set the seed to 1, then use the `caret` package to partition into test and training sets with  $p = 0.5$ . Compare the RMSE when using just  $x_1$ , just  $x_2$  and both  $x_1$  and  $x_2$ . Train a single linear model for each (not 100 like in the previous questions).

```
set.seed(1, sample.kind = "Rounding")

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
test_index <- createDataPartition(dat$y, times = 1, p = 0.5, list = FALSE)
train_set <- dat %>% slice(-test_index)
test_set <- dat %>% slice(test_index)
```

```

fit <- lm(y ~ x_1, data = train_set)
y_hat <- predict(fit, newdata = test_set)
sqrt(mean((y_hat-test_set$y)^2))

## [1] 0.601

fit <- lm(y ~ x_2, data = train_set)
y_hat <- predict(fit, newdata = test_set)
sqrt(mean((y_hat-test_set$y)^2))

## [1] 0.631

fit <- lm(y ~ x_1 + x_2, data = train_set)
y_hat <- predict(fit, newdata = test_set)
sqrt(mean((y_hat-test_set$y)^2))

## [1] 0.307

```

Repeat the exercise from Q6 but now create an example in which `x_1` and `x_2` are highly correlated.

```

set.seed(1, sample.kind="Rounding")

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used

Sigma <- matrix(c(1.0, 0.75, 0.75, 0.75, 1.0, 0.95, 0.75, 0.95, 1.0), 3, 3)
dat <- MASS::mvrnorm(n = 100, c(0, 0, 0), Sigma) %>%
  data.frame() %>%
  setNames(c("y", "x_1", "x_2"))

```

Set the seed to 1, then use the caret package to partition into a test and training set of equal size. Compare the RMSE when using just `x_1`, just `x_2`, and both `x_1` and `x_2`.

```

set.seed(1)
test_index <- createDataPartition(dat$y, times = 1, p = 0.5, list = FALSE)
train_set <- dat %>%
  slice(-test_index)
test_set <- dat %>%
  slice(test_index)

fit <- lm(y ~ x_1, data = train_set)
y_hat <- predict(fit, newdata = test_set)
sqrt(mean((y_hat-test_set$y)^2))

## [1] 0.659

fit <- lm(y ~ x_2, data = train_set)
y_hat <- predict(fit, newdata = test_set)
sqrt(mean((y_hat-test_set$y)^2))

## [1] 0.64

fit <- lm(y ~ x_1 + x_2, data = train_set)
y_hat <- predict(fit, newdata = test_set)
sqrt(mean((y_hat-test_set$y)^2))

## [1] 0.66

```

**8.3.2.1. Regresiones para resultados categóricos** Para ejemplificar el uso de regresiones con resultados categóricos, retómese el ejemplo de predicción de sexo mediante alturas:

```
library(dslabs)

data(heights)

y <- heights$height

set.seed(2)

test_index <- createDataPartition(y, times = 1, p = 0.5, list= FALSE)

train_set <- heights %>%
  slice(-test_index)

test_set <- heights %>%
  slice(test_index)
```

Si definimos a los resultados como  $Y = 1$  para mujeres,  $Y = 0$  para hombres y  $X$  la altura, esto implica que nos interesa la probabilidad condicional  $PR(Y = 1|X = x)$ .

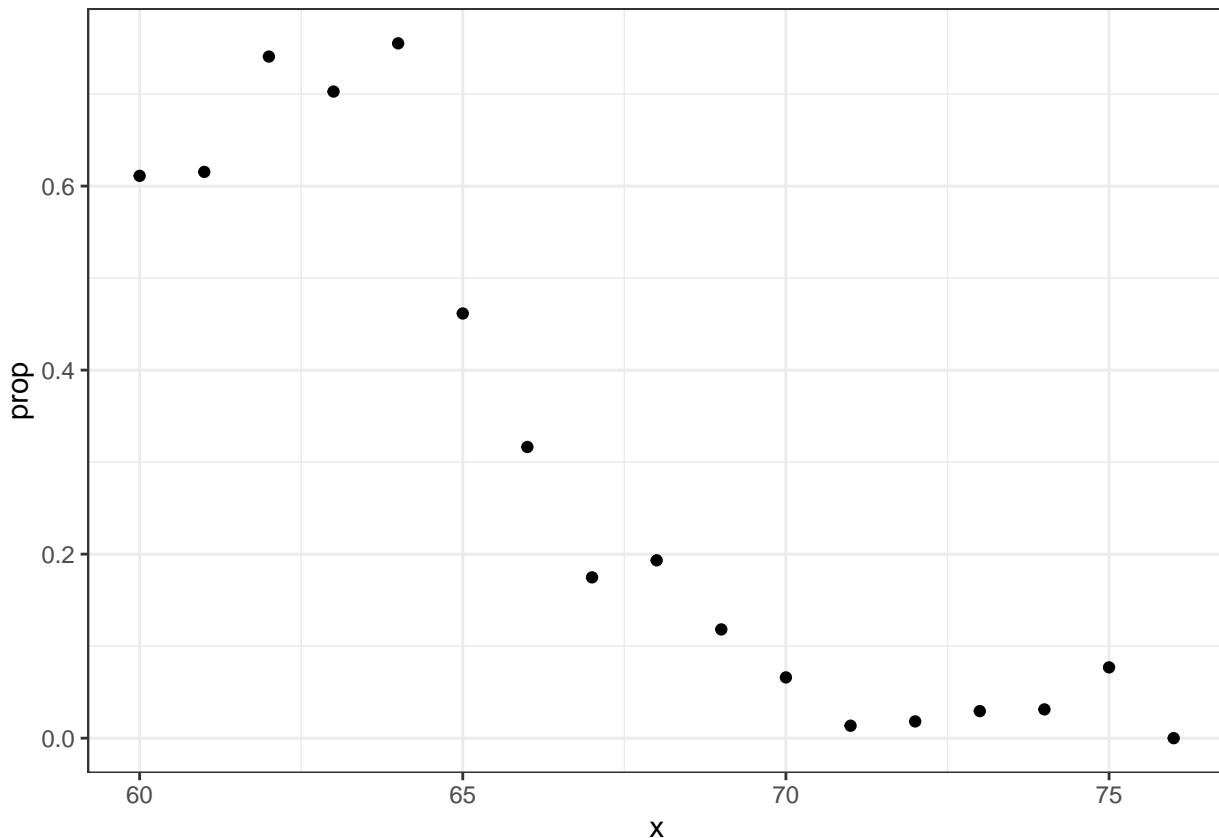
Como ejemplo, calculemos la predicción para un estudiante de altura de 66 pulgadas. Primero, redondeemos la base de datos a la pulgada más cercana y obtengamos el promedio de estos valores:

```
train_set %>%
  filter(round(height) == 66) %>%
  summarize(mean(sex == "Female"))

##   mean(sex == "Female")
## 1           0.242
```

Ahora, obtengamos este mismo estimado para varios valores de X:

```
heights %>%
  mutate(x = round(height)) %>%
  group_by(x) %>%
  filter(n() >= 10) %>%
  summarize(prop = mean(sex == "Female")) %>%
  ggplot(aes(x, prop)) +
  geom_point()
```



Nótese que los resultados del gráfico lucen similares a una línea, por lo que un enfoque de regresión lineal puede ser adecuado. Para ello, asumiremos que la probabilidad condicional es:

$$p(x) = Pr(Y = 1|X = x) = \beta_0 + \beta_1 x$$

```
lm_fit <- mutate(train_set, y = as.numeric(sex == "Female")) %>%
  lm(y ~ height, data =.)
```

Una vez obtenidos los estimadores, podemos definir una predicción real  $\hat{p}(x) = \hat{\beta}_0 + \hat{\beta}_1 x$ . Así, para formar una predicción, definimos una regla de decisión: predeciremos “mujer” si  $\hat{p}(x) > 0,5$ . Ahora podemos usar la matriz de confusión para observar nuestros resultados:

```
p_hat <- predict(lm_fit, test_set)

y_hat <- ifelse(p_hat > 0.5, "Female", "Male") %>%
  factor()

confusionMatrix(y_hat, test_set$sex)

## Confusion Matrix and Statistics
##
##             Reference
## Prediction Female Male
##     Female      20    15
##     Male        98   393
##
##             Accuracy : 0.785
```

```

##               95% CI : (0.748, 0.82)
##      No Information Rate : 0.776
##      P-Value [Acc > NIR] : 0.322
##
##                  Kappa : 0.177
##
## McNemar's Test P-Value : 1.22e-14
##
##      Sensitivity : 0.1695
##      Specificity : 0.9632
##      Pos Pred Value : 0.5714
##      Neg Pred Value : 0.8004
##      Prevalence : 0.2243
##      Detection Rate : 0.0380
##      Detection Prevalence : 0.0665
##      Balanced Accuracy : 0.5664
##
##      'Positive' Class : Female
##

```

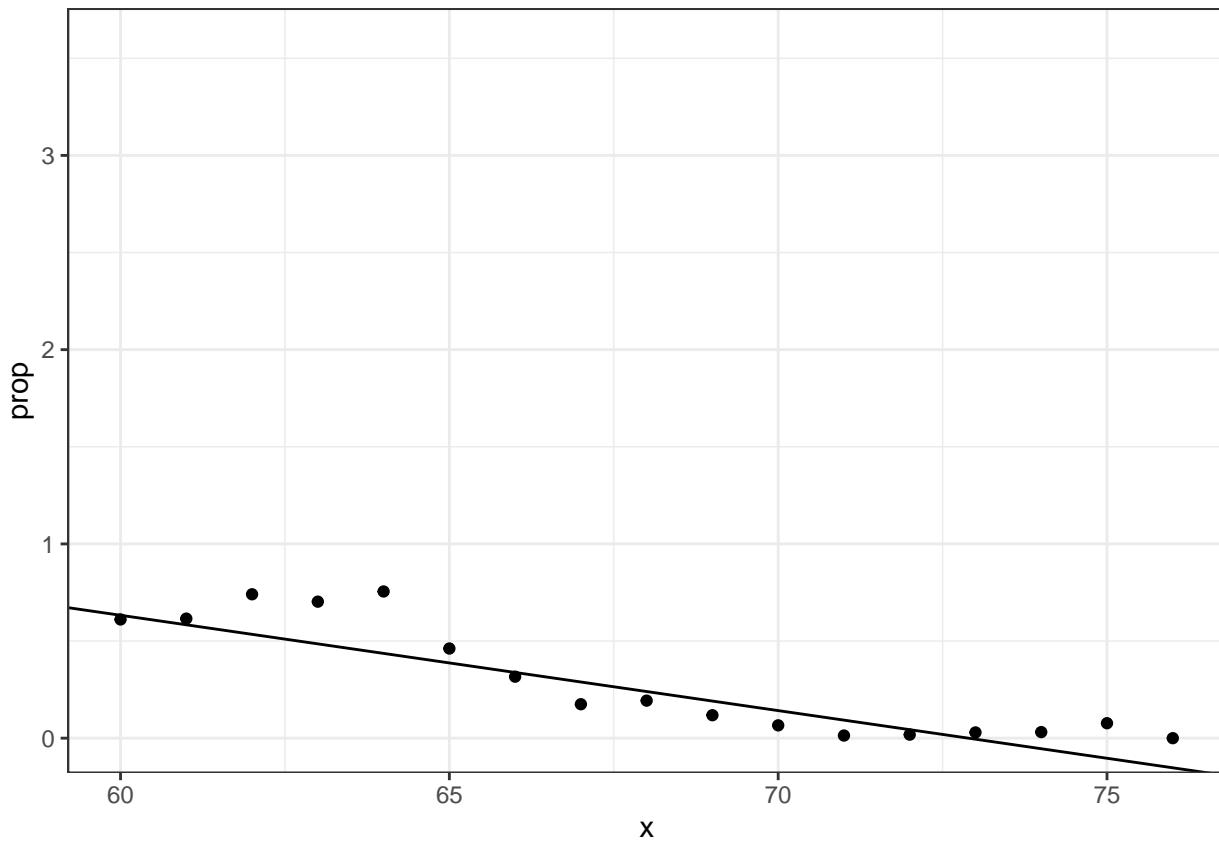
Observemos que obtuvimos una precisión de 78.5 %, sensibilidad de 16.9 % y especificidad de 96.3 %.

**8.3.2.2. Regresión logística** Nótese que nuestra función  $\beta_0 + \beta_1 x$  puede tomar cualquier valor, incluyendo negativos y mayores a 1. De hecho, nuestro estimado de la probabilidad condicional usando regresión lineal va de -0.4 a 1.12:

```

heights %>%
  mutate(x = round(height)) %>%
  group_by(x) %>%
  filter(n() >= 10) %>%
  summarize(prop = mean(sex == "Female")) %>%
  ggplot(aes(x, prop)) +
  geom_point() +
  geom_abline(intercept = lm_fit$coef[1], slope = lm_fit$coef[2])

```



```
range(p_hat)
```

```
## [1] -0.398 1.123
```

Para resolver este problema (recuérdese que una probabilidad solo existe en  $[0, 1]$ ), podemos utilizar regresiones logísticas, las cuales aseguran que nuestros estimados de probabilidades condicionales están en este intervalo. Recuérdese que este tipo de regresión está definida como:

$$g(p) = \log \frac{p}{1-p}$$

Donde  $p$  son las probabilidades de que algo ocurra. (si  $p = 0,5$ , entonces las probabilidades son 1 a 1; si  $p = 0,75$ , las probabilidades son 3 a 1).

Ahora, para ajustar nuestro modelo, ya no podemos utilizar MCO; en su lugar, es necesario recurrir a estimadores de **Máxima verosimilitud** o MLE. En R, esto se logra mediante la función “glm()”, las siglas de *generalized linear models*:

```
glm_fit <- train_set %>%
  mutate(y = as.numeric(sex == "Female")) %>%
  glm(y ~ height, data = ., family = "binomial")
```

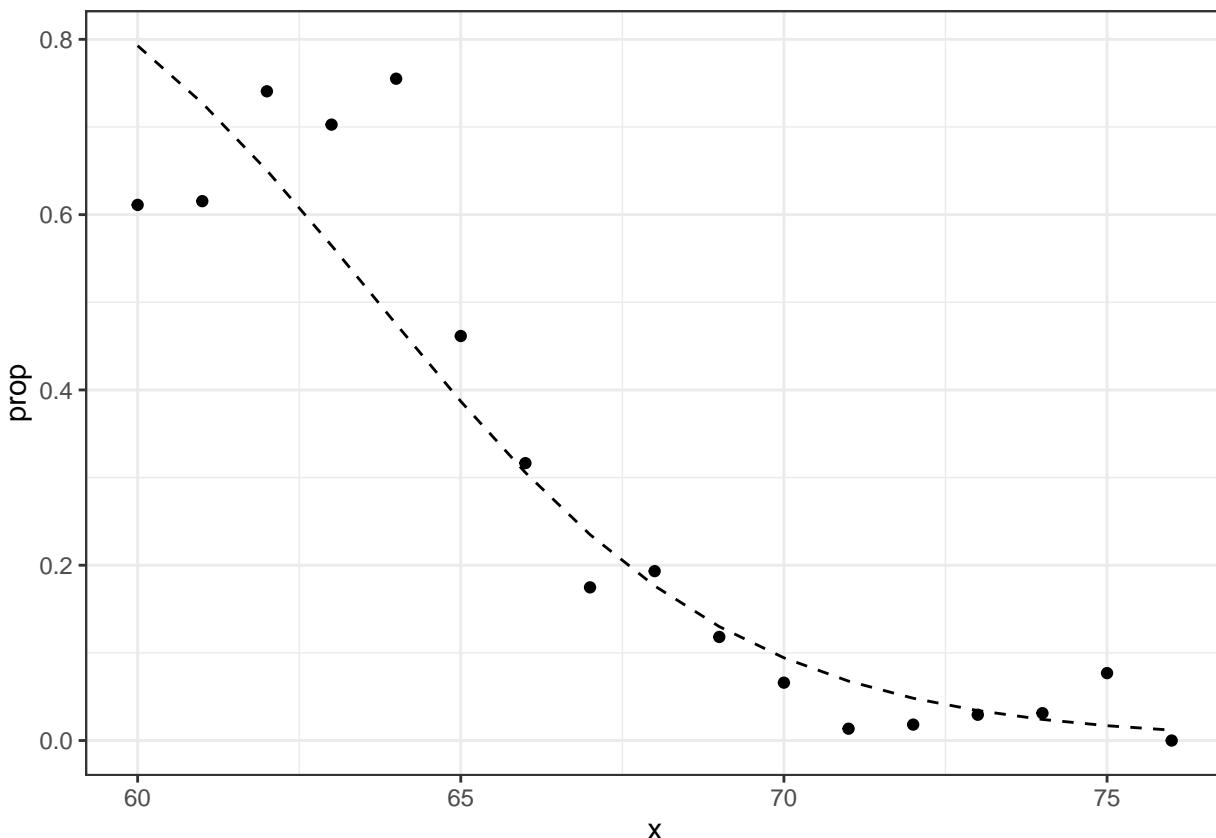
**Nota:** El argumento family es el que nos permite especificar un modelo logístico; dicho argumento puede tomar las siguientes formas:

-binomial(link = "logit") -gaussian(link = "identity") -Gamma(link = "inverse") -inverse.gaussian(link = "1/mu^2") -poisson(link = "log") -quasi(link = "identity", variance = "constant") -quasibinomial(link = "logit") -quasipoisson(link = "log")

```
p_hat_logit <- predict(glm_fit, newdata=test_set, type = "response")
```

Ahora, podemos visualizar qué tan bien nuestro modelo se ajustó:

```
tmp <- heights %>%
  mutate(x = round(height)) %>%
  group_by(x) %>%
  filter(n() >= 10) %>%
  summarize(prop = mean(sex == "Female"))
logistic_curve <- data.frame(x = seq(min(tmp$x), max(tmp$x))) %>%
  mutate(p_hat = plogis(glm_fit$coef[1] + glm_fit$coef[2]*x))
tmp %>%
  ggplot(aes(x, prop)) +
  geom_point() +
  geom_line(data = logistic_curve, mapping = aes(x, p_hat), lty = 2)
```



Esto es, el modelo logístico se ajusta a los datos mejor que una línea. Dado que tenemos un estimador de la probabilidad condicional, podemos obtener predicciones con el siguiente código:

```
y_hat_logit <- ifelse(p_hat_logit > 0.5, "Female", "Male") %>%
  factor()

confusionMatrix(y_hat_logit, test_set$sex)

## Confusion Matrix and Statistics
##
##          Reference
##          Prediction Female Male
##
```

```

##      Female     31    19
##      Male      87  389
##
##          Accuracy : 0.798
##             95% CI : (0.762, 0.832)
##      No Information Rate : 0.776
##      P-Value [Acc > NIR] : 0.114
##
##          Kappa : 0.272
##
##  Mcnemar's Test P-Value : 7.64e-11
##
##          Sensitivity : 0.2627
##          Specificity  : 0.9534
##      Pos Pred Value : 0.6200
##      Neg Pred Value : 0.8172
##          Prevalence  : 0.2243
##          Detection Rate : 0.0589
##      Detection Prevalence : 0.0951
##          Balanced Accuracy : 0.6081
##
##      'Positive' Class : Female
##

```

Donde podemos ver que, si bien la especificidad disminuyó ligeramente, nuestra precisión y sensibilidad aumentaron considerablemente.

**8.3.2.3. Caso de estudio: 2 o 7** En los ejemplos simples que hemos analizado, solo hemos utilizado un predictor. Así, retomemos el ejemplo de los dígitos en imágenes, donde teníamos 784 predictores; para fines ilustrativos, trabajaremos con un subconjunto de estos datos, con 2 predictores y 2 categorías. Así, queremos construir un algoritmo para determinar si un dígito es 2 o 7 a partir de nuestros dos predictores.

Nuestros predictores serán: (1) la proporción de píxeles oscuros en el cuadrante superior izquierdo u (2) proporción de píxeles oscuros en el cuadrante inferior derecho; además, trabajaremos con una muestra de 1 000 dígitos:

```
data("mnist_27")
```

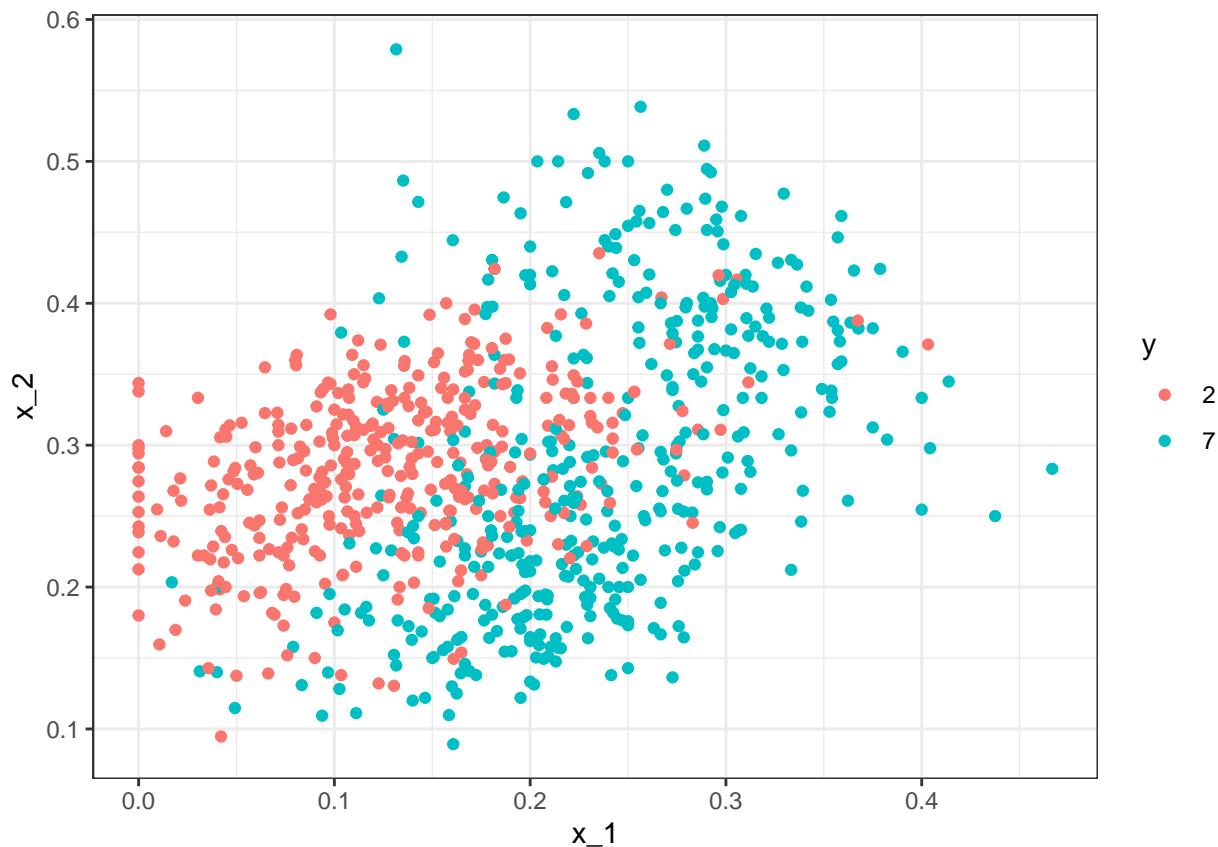
Podemos explorar estos datos graficando los dos predictores y usando colores para denotar las etiquetas:

```
data("mnist_27")
```

```

mnist_27$train %>%
  ggplot(aes(x_1, x_2, color = y)) +
  geom_point()

```

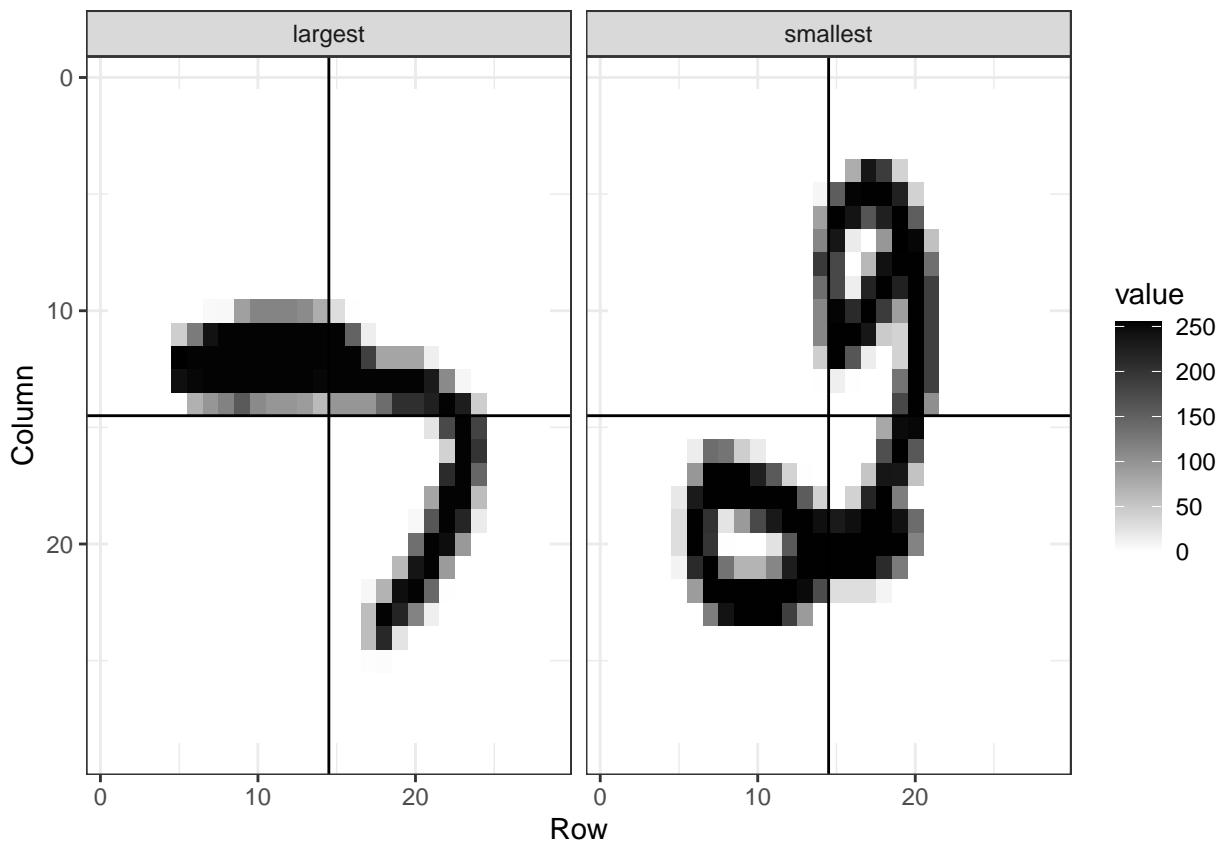


Donde pueden observarse algunos elementos relevantes: si  $x_1$ , el primer predictor (panel superior izquierdo) es grande, entonces el dígito probablemente sea 7; también, para valores pequeños y medianos del segundo predictor  $x_2$  (panel inferior derecho), el dígito es 2 en muchas ocasiones.

Obsérvense las imágenes de los dígitos con los valores más grandes y pequeños de  $x_1$ :

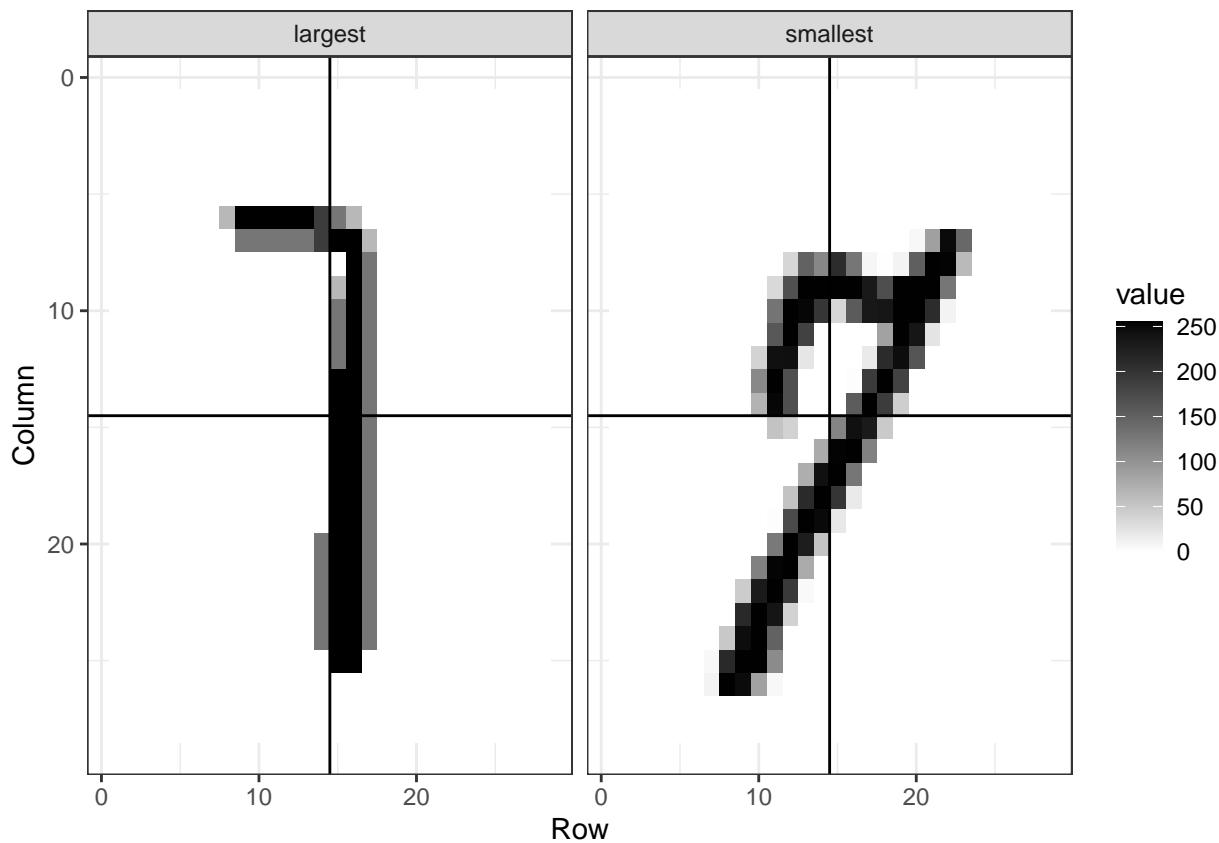
```
mnist <- read_mnist()

is <- mnist_27$index_train[c(which.min(mnist_27$train$x_1), which.max(mnist_27$train$x_1))]
titles <- c("smallest", "largest")
tmp <- lapply(1:2, function(i){
  expand.grid(Row=1:28, Column=1:28) %>%
    mutate(label=titles[i],
          value = mnist$train$images[is[i],])
})
tmp <- Reduce(rbind, tmp)
tmp %>% ggplot(aes(Row, Column, fill=value)) +
  geom_raster() +
  scale_y_reverse() +
  scale_fill_gradient(low="white", high="black") +
  facet_grid(~label) +
  geom_vline(xintercept = 14.5) +
  geom_hline(yintercept = 14.5)
```



Véase que el 7 de la izquierda tiene muchos píxeles oscuros en el cuadrante superior izquierdo ( $x_1$  es grande), mientras que el 2 de la derecha casi no tiene píxeles oscuros en ese cuadrante ( $x_1$  muy pequeña). Ahora, considérense las imágenes con los valores de  $x_2$  más altos y bajos:

```
is <- mnist_27$index_train[c(which.min(mnist_27$train$x_2), which.max(mnist_27$train$x_2))]
titles <- c("smallest", "largest")
tmp <- lapply(1:2, function(i){
  expand.grid(Row=1:28, Column=1:28) %>%
    mutate(label=titles[i],
          value = mnist$train$images[is[i],])
})
tmp <- Reduce(rbind, tmp)
tmp %>% ggplot(aes(Row, Column, fill=value)) +
  geom_raster() +
  scale_y_reverse() +
  scale_fill_gradient(low="white", high="black") +
  facet_grid(.~label) +
  geom_vline(xintercept = 14.5) +
  geom_hline(yintercept = 14.5)
```



Véase que ambos son 7s, donde el de la izquierda tiene muchos píxeles oscuros en el cuadrante inferior derecho ( $x_2$  grande) y el de la izquierda muy pocos ( $x_2$  pequeña).

Comencemos nuestro análisis mediante un algoritmo de regresión logística. La probabilidad condicional de que sea un 7 dado que los dos predictores serán una función lineal de  $x_1, x_2$  después de la transformación logística.

$$p(x_1, x_2) = \Pr(Y = 1 | X_1 = x_1, X_2 = x_2) = g^{-1}(\beta_0 + \beta_1 x_1 + \beta_2 x_2)$$

Donde  $g^{-1}$  es la función logística inversa:

$$g^{-1}(x) = \exp(x) / [1 + \exp(x)]$$

Así, el modelo puede estimarse mediante:

```
fit_glm <- glm(y ~ x_1 + x_2, data=mnist_27$train, family = "binomial")
```

Nuestra regla de decisión será predecir un 7 siempre que la probabilidad condicional sea mayor a 0.5:

```
p_hat_glm <- predict(fit_glm, mnist_27$test)
y_hat_glm <- factor(ifelse(p_hat_glm > 0.5, 7, 2))

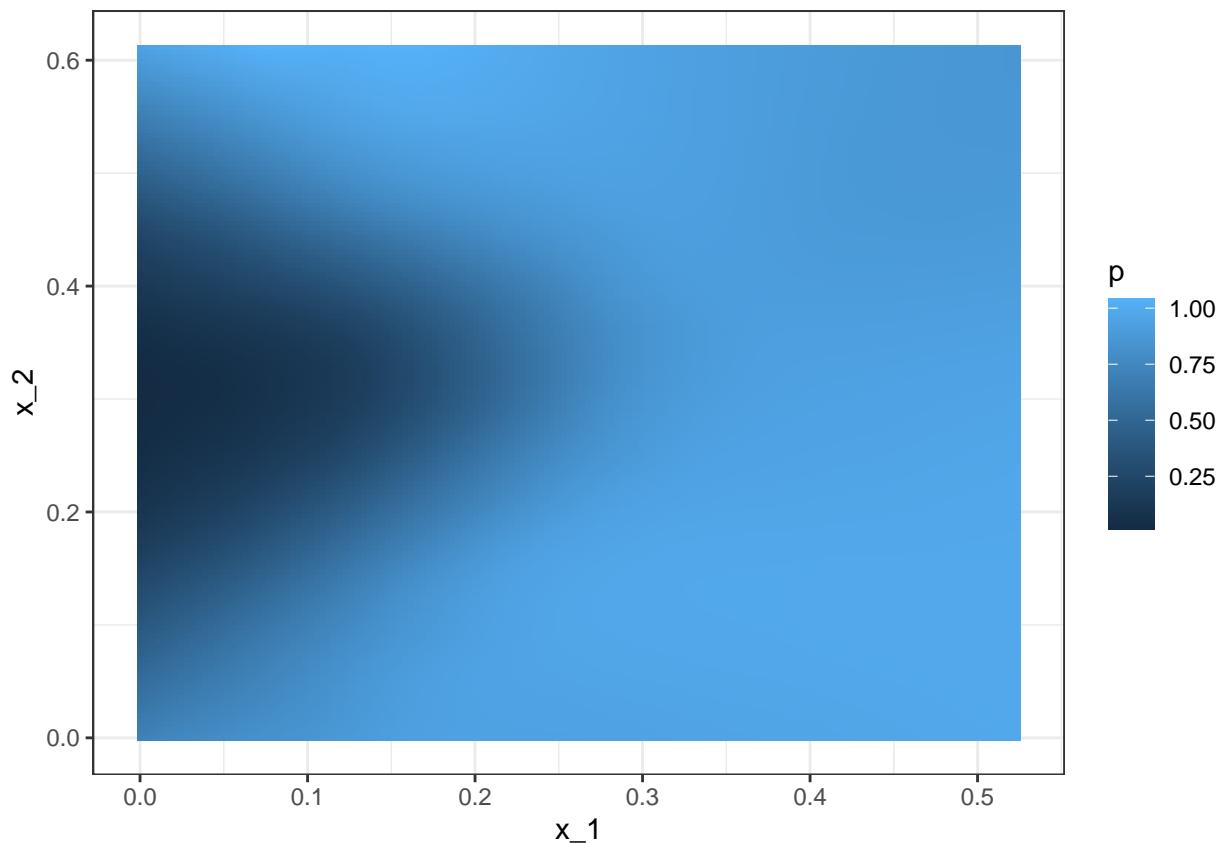
confusionMatrix(data = y_hat_glm, reference = mnist_27$test$y)
```

```
## Confusion Matrix and Statistics
##
##          Reference
```

```
## Prediction 2 7
##          2 92 34
##          7 14 60
##
##          Accuracy : 0.76
##          95% CI : (0.695, 0.817)
##          No Information Rate : 0.53
##          P-Value [Acc > NIR] : 1.67e-11
##
##          Kappa : 0.512
##
##          Mcnemar's Test P-Value : 0.0061
##
##          Sensitivity : 0.868
##          Specificity : 0.638
##          Pos Pred Value : 0.730
##          Neg Pred Value : 0.811
##          Prevalence : 0.530
##          Detection Rate : 0.460
##          Detection Prevalence : 0.630
##          Balanced Accuracy : 0.753
##
##          'Positive' Class : 2
##
```

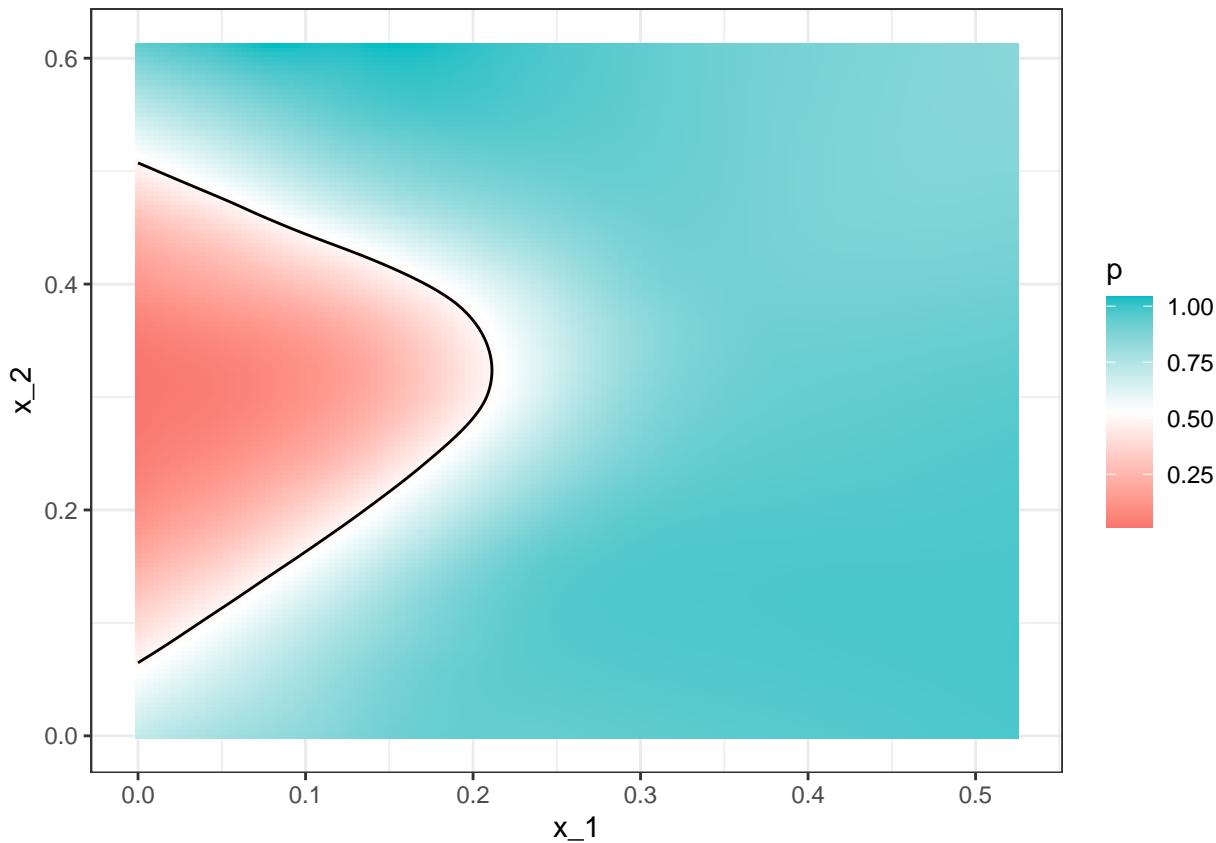
Donde vemos que obtuvimos una precisión de 76%; aunque parece un buen primer intento, consideremos la probabilidad condicional real de los datos a través del siguiente código:

```
mnist_27$true_p %>%
  ggplot(aes(x_1, x_2, fill = p)) +
  geom_raster()
```



**Nota:** Mejoraremos el gráfico anterior mediante la elección de mejores colores y el dibujo de una curva que separe a los pares  $(x_1, x_2)$  para los cuales la probabilidad condicional sea mayor o menor a 0.5:

```
mnist_27$true_p %>%
  ggplot(aes(x_1, x_2, z = p, fill = p)) +
  geom_raster() +
  scale_fill_gradientn(colors = c("#F8766D", "white", "#00BFC4")) +
  stat_contour(breaks = c(0.5), color = "black")
```



Donde podemos observar la probabilidad condicional real. Así, para comprender las limitaciones de las regresiones logísticas, podemos comparar la probabilidad condicional real y la estimada.

Compútese el límite que divide los valores de  $x_1, x_2$  que hacen que la probabilidad condicional estimada sea menor o mayor a 0.5 (en este límite, la probabilidad condicional será 0.5):

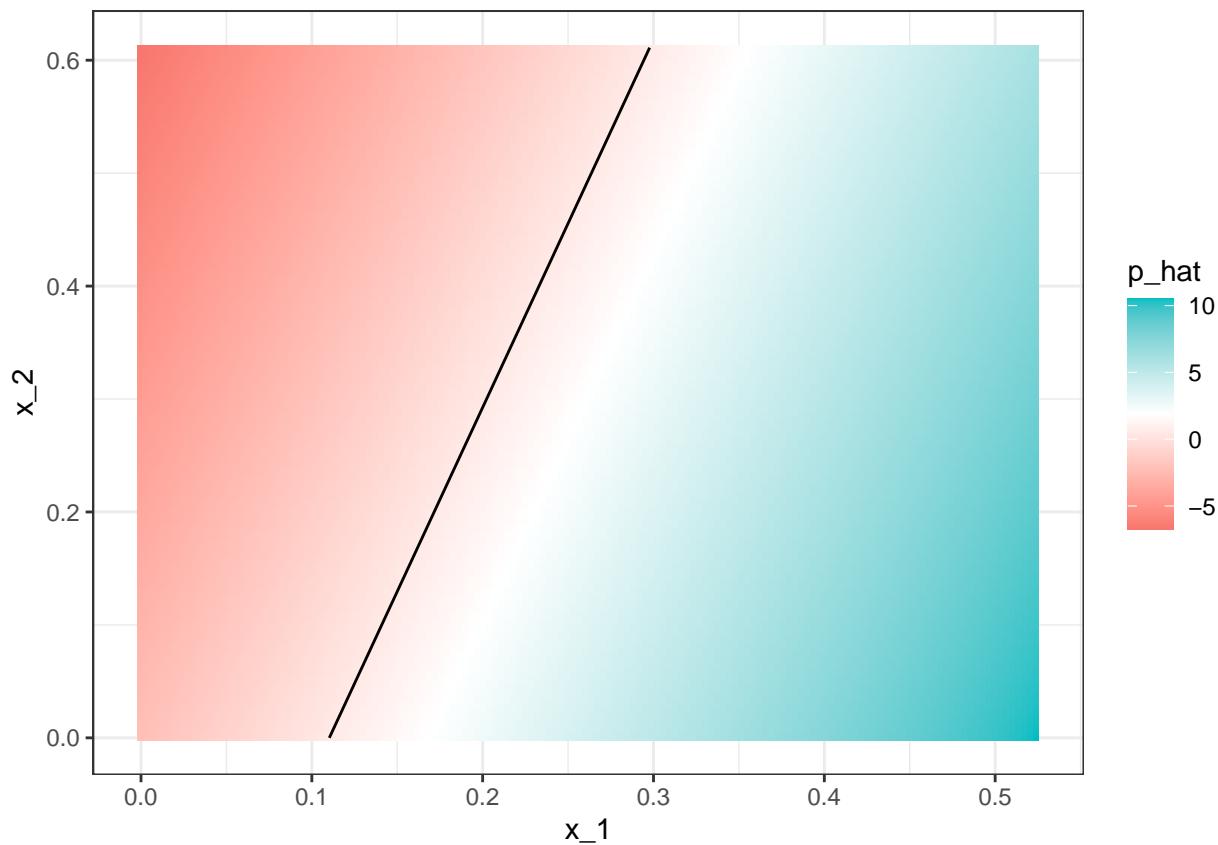
$$g^{-1}(\hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2) = 0,5$$

$$\Rightarrow \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 = g(0,5) = 0$$

$$\Rightarrow x_2 = -\hat{\beta}_0/\hat{\beta}_2 + -\hat{\beta}_1/\hat{\beta}_2 x_1$$

```
p_hat <- predict(fit_glm, newdata = mnist_27$true_p)

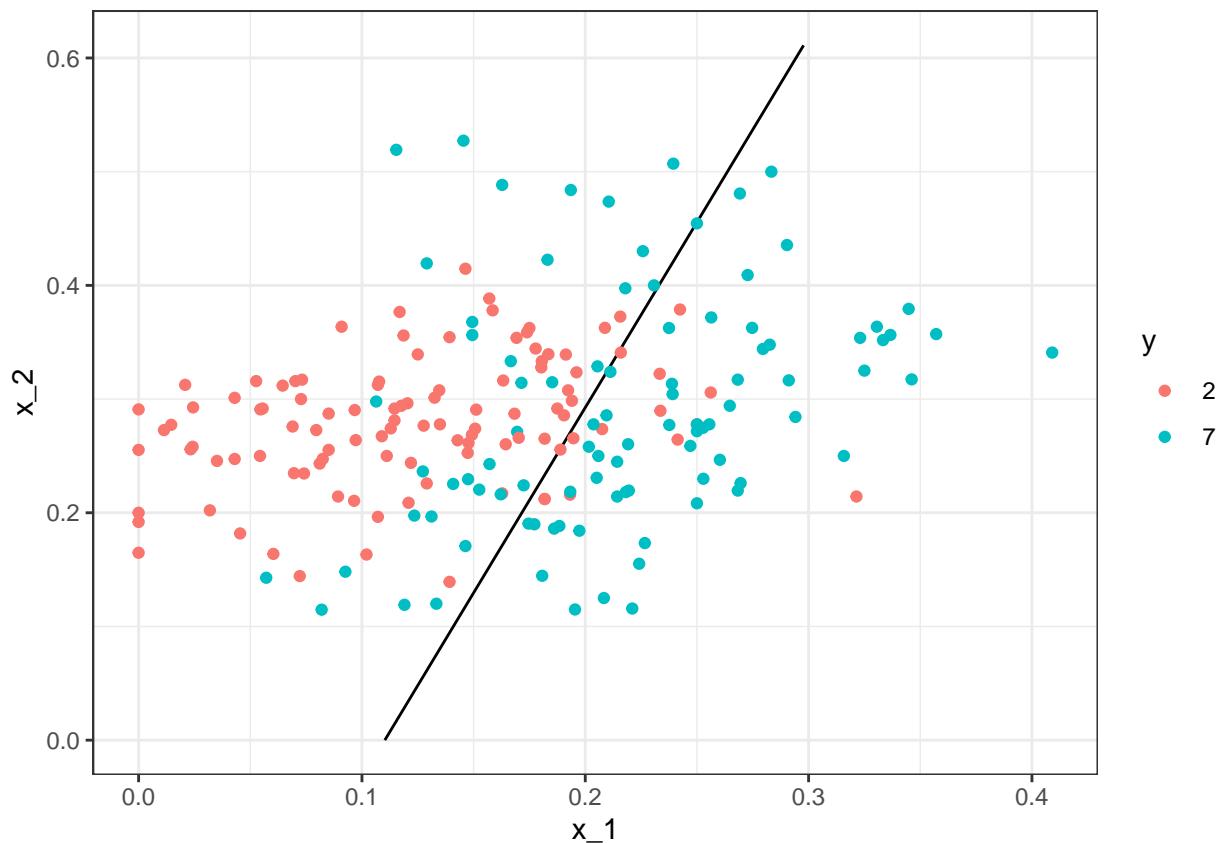
mnist_27$true_p %>%
  mutate(p_hat = p_hat) %>%
  ggplot(aes(x_1, x_2, z=p_hat, fill=p_hat)) +
  geom_raster() +
  scale_fill_gradientn(colors=c("#F8766D", "white", "#00BFC4")) +
  stat_contour(breaks=c(0.5), color="black")
```



Lo que implica que nuestra cota no puede ser diferente a una línea, lo que implica que nuestra regresión logística no puede capturar la naturaleza no lineal de la probabilidad condicional.

Para observar los errores, podemos graficar los datos de práctica con  $x_1$  y  $x_2$  y colorear las etiquetas:

```
p_hat <- predict(fit_glm, newdata = mnist_27$true_p)
mnist_27$true_p %>%
  mutate(p_hat = p_hat) %>%
  ggplot() +
  stat_contour(aes(x_1, x_2, z=p_hat), breaks=c(0.5), color="black") +
  geom_point(mapping = aes(x_1, x_2, color=y), data = mnist_27$test)
```



Puede verse que, dado que la regresión logística divide a los puntos con una línea, muchos puntos no podrán ser capturados por esta forma. Por tanto, en las siguientes secciones revisaremos métodos que nos permitirán

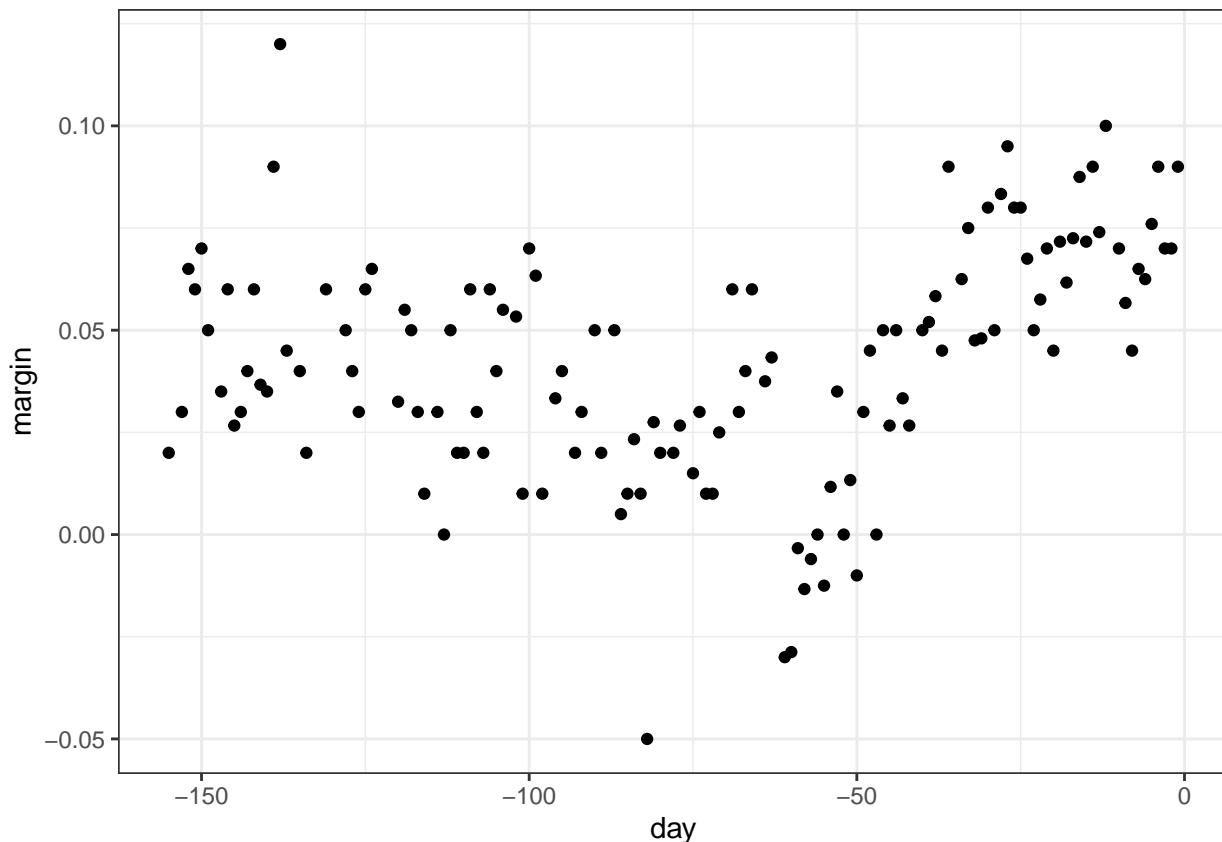
### 8.3.3. Suavización

La suavización es una técnica muy poderosa utilizada en muchas aplicaciones del análisis de datos (a veces también se le llama *curve fitting* o *low pass filtering*). Está diseñada para detectar tendencias en la presencia de datos ruidosos en casos donde la forma de la tendencia es desconocida.

Los conceptos de las técnicas de suavización son sumamente útiles en el aprendizaje automático dado que las probabilidades y esperanzas condicionales pueden ser pensadas como tendencias de formas desconocidas, las cuales estimamos en la presencia de incertidumbre.

Como ejemplo, considérese un problema con un solo predictor: trataremos de estimar la tendencia temporal en el voto popular de la elección de EE.UU. en 2008:

```
library(dslabs)
library(tidyverse)
data("polls_2008")
qplot(day, margin, data = polls_2008)
```



Para los fines de este ejemplo, considérese solamente la forma de la tendencia y no un pronóstico electoral que podamos hacer. Así, asumiremos que para cualquier día  $x$ , existe una preferencia electoral real  $f(x)$ , pero dada la incertidumbre creada por las encuestas, cada dato tiene un error  $\varepsilon$ ; esto implica que un modelo matemático para el margen de encuestas observado  $Y$  es:

$$Y_i = f(x_i) + \varepsilon_i$$

Para pensar esto como un problema de aprendizaje automático, considérese que queremos predecir  $y$  dado el día  $x$  mediante la estimación de probabilidades condicionales  $f(x) = E[Y|X = x]$ .

La idea general de una suavización de barras es agrupar puntos de datos en estratos en donde los valores de  $f(x)$  pueden ser asumidos constantes. En nuestro ejemplo, esto es equivalente a asumir que la opinión pública electoral se mantiene aproximadamente igual en un periodo de una semana.

Así, asúmase un día para ser el centro de la semana,  $x_0$ , entonces, para cualquier otro día  $x \ni |x - x_0| \leq 3,5$  asumiremos que  $f(x)$  es constante, sea  $f(x) = \mu$ . Este supuesto implica que  $E[Y_i|X_i = x_i] \approx \mu$  si  $|x_i - x_0| \leq 3,5$ . El tamaño del intervalo que satisface la segunda condición es conocido como *window size*, *bandwidth* o *span*.

Defínase  $A_0$  el conjunto de índices  $i \ni |x_i - x_0| \leq 3,5$  y  $N_0$  el número de índices en  $A_0$ , entonces nuestro estimador es:

$$\hat{f}(x_0) = \frac{1}{N_0} \sum_{i \in A_0} Y_i$$

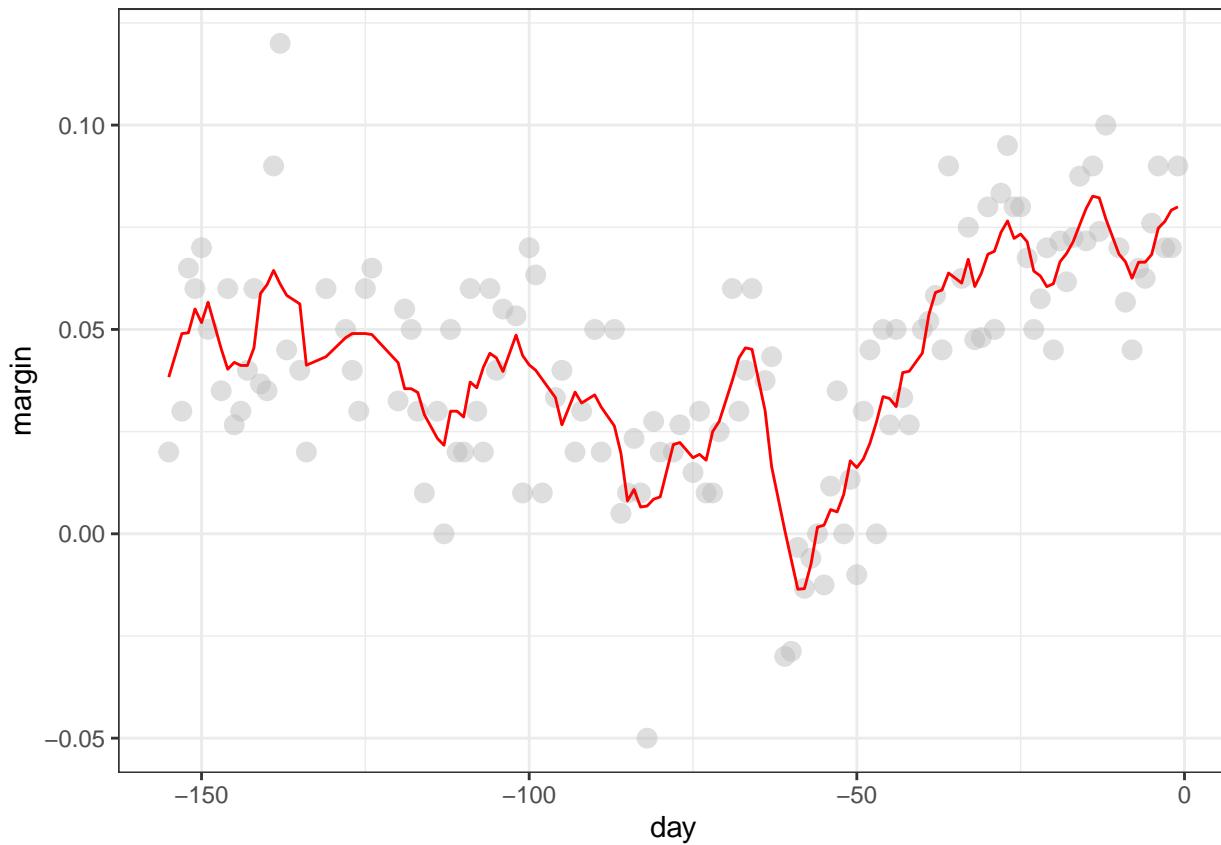
La idea de la suavización de barras es realizar este cálculo para cada valor de  $x$ . En el ejemplo de las encuestas, y para cada día, computaremos el promedio de valores dentro de una semana del día considerado.

Así, nuestro promedio móvil puede ser calculado y visualizado con el siguiente código:

```
span <- 7

fit <- with(polls_2008,
            ksmooth(day, margin, x.points = day, kernel = "box", bandwidth = span))

polls_2008 %>%
  mutate(smooth = fit$y) %>%
  ggplot(aes(day, margin)) +
  geom_point(size = 3, alpha = 0.5, color = "grey") +
  geom_line(aes(day, smooth), color = "red")
```



Nótese que el resultado final es muy ondulado; esto se debe a que cada vez que la ventana se mueve, dos puntos cambian, los cuales representan un porcentaje alto de los 7 que estamos considerando. Esto puede atenuarse si tomamos promedios ponderados que den al centro más peso que aquellos extremos.

Las funciones a partir de las cuales computamos estos pesos son llamadas **kernel** y parte de la siguiente fórmula:

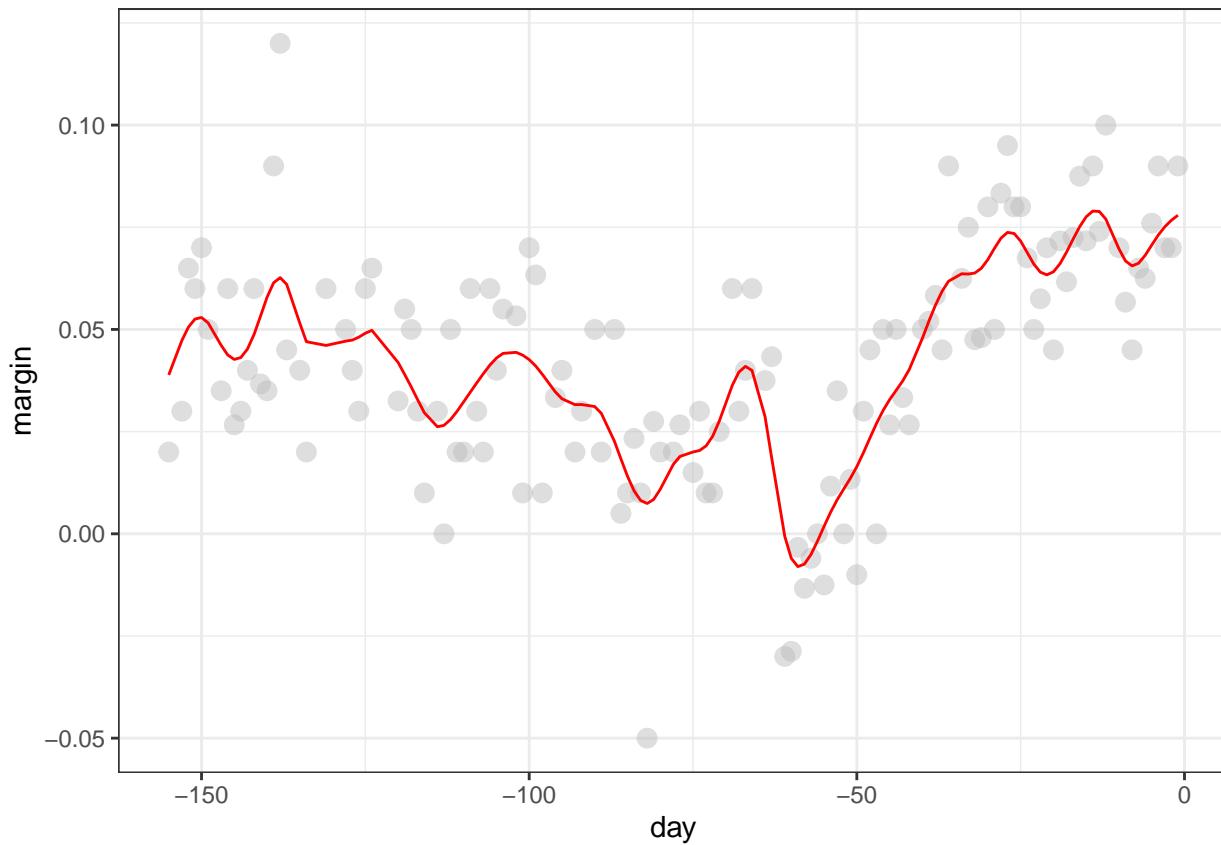
$$\hat{f}(x_0) = \sum_{i=1}^N w_0(x_i) Y_i$$

Así, la función “ksmooth()” nos permite suavizar la tendencia mediante el uso de densidades normales o Gaussianas que asignan pesos a los datos:

```
span <- 7

fit <- with(polls_2008,
            ksmooth(day, margin, x.points = day, kernel = "normal", bandwidth = span))

polls_2008 %>%
  mutate(smooth = fit$y) %>%
  ggplot(aes(day, margin)) +
  geom_point(size = 3, alpha = 0.5, color = "grey") +
  geom_line(aes(day, smooth), color = "red")
```



Si bien nuestra gráfica ha sido sustancialmente suavizada, existen intervalos donde sigue ondulando en exceso.

Una desventaja del enfoque de suavización de barras es que se necesitan ventanas muy pequeñas para que el supuesto aproximadamente normal se mantenga. Como resultado, terminamos con un número de datos pequeños para promediar, lo que puede llevar a estimaciones imprecisas de nuestra tendencia.

Como alternativa, está la regresión local ponderada, o ***local weighted regression (loess)***, la cual nos permite considerar ventanas más grandes. Este enfoque asume que la función es localmente lineal, en vez de asumir que sea aproximadamente constante dentro de una ventana.

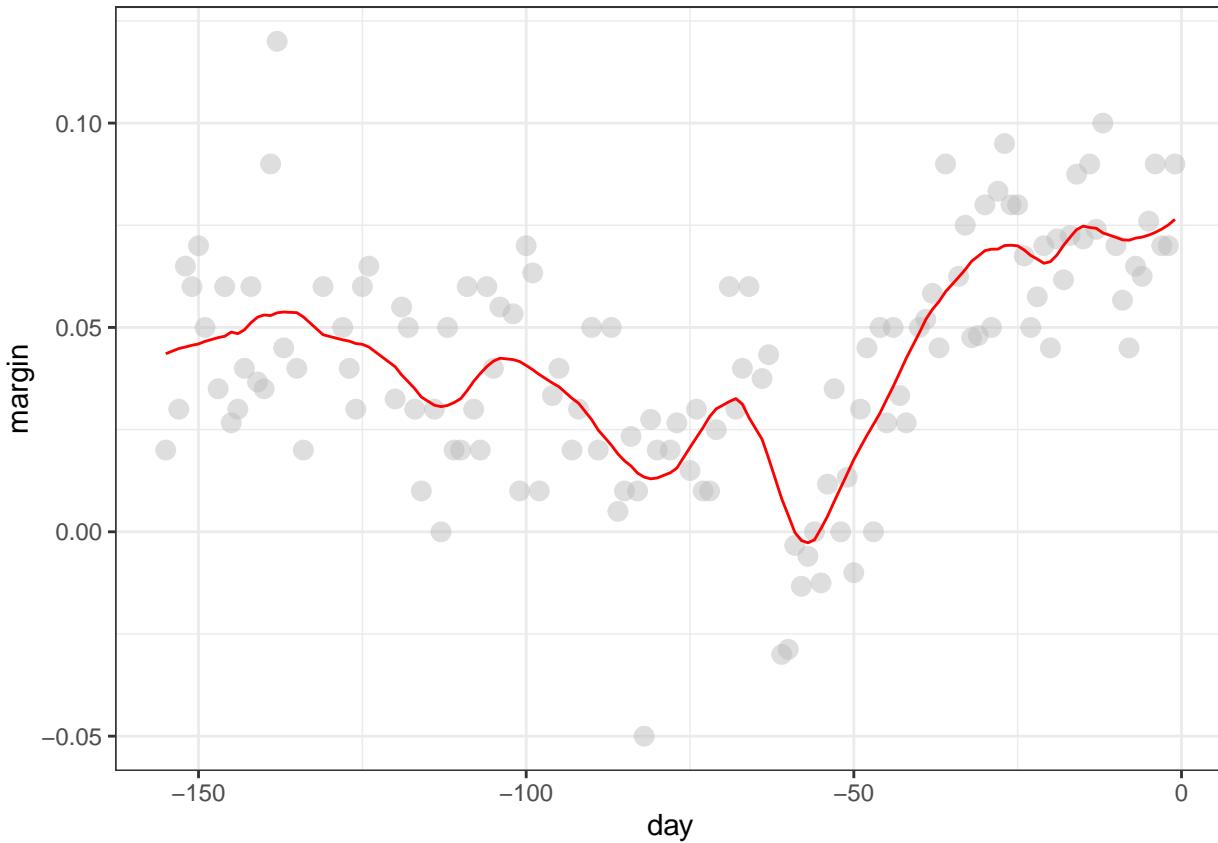
Para nuestro ejemplo, dejaremos de considerar una ventana de una semana y, como primera prueba, usaremos una de 3 semanas. Esto implica que  $E[Y_i|X_i = x_i] = \beta_0 + \beta_1(x_i - x_0)$  si  $|x_i - x_0| \leq 10.5$ . Así, nuestras estimaciones finales son las que siguen:

```
total_days <- diff(range(polls_2008$day))

span <- 21/total_days

fit <- loess(margin ~ day, degree = 1, span = span, data = polls_2008)

polls_2008 %>%
  mutate(smooth = fit$fitted) %>%
  ggplot(aes(day, margin)) +
  geom_point(size = 3, alpha = 0.5, color = "grey") +
  geom_line(aes(day, smooth), color = "red")
```



Una diferencia importante entre loess y la suavización de barras es que la primera mantiene el mismo número de puntos usados en el ajuste local (esto se controla vía el argumento “span”, el cual debe darse como proporción -si, por ejemplo, tenemos N puntos y la ventana es de 0.5, entonces, para cada  $X$ , loess usará  $0.5 \times N$  puntos cercanos-).

Otra diferencia es que loess minimiza la versión ponderada de los mínimos cuadrados:

$$\sum_{i=1}^N w_0(x_i)[Y_i - \{\beta_0 + \beta_1(x_i - x_0)\}]^2$$

La última diferencia es que loess tiene la opción de ajustar el modelo local de manera robusta: se implementa un algoritmo iterativo en donde, después de ajustar el modelo en una iteración, los *outliers* son detectados y ponderados hacia abajo en la siguiente iteración (esto puede usarse con el argumento “family = symmetric”).

Un elemento final y relevante de loess es que se asemeja a una parábola, lo que implica que no hay que mirar a la función de tan cerca como se hace en una aproximación lineal. Esto es, podemos ajustar paráboles en vez de líneas (en la función, el argumento “degree = 1” ajusta la estimación a polinomios de grado 1; por default, este argumento es 2, cuadrático). Para nuestro ejemplo, esto implica que  $E[Y_i|X_i = x_i] = \beta_0 + \beta_1(x_i - x_0) + \beta_2(x_i - x_0)^2$  si  $|x_i - x_0| \leq h$ :

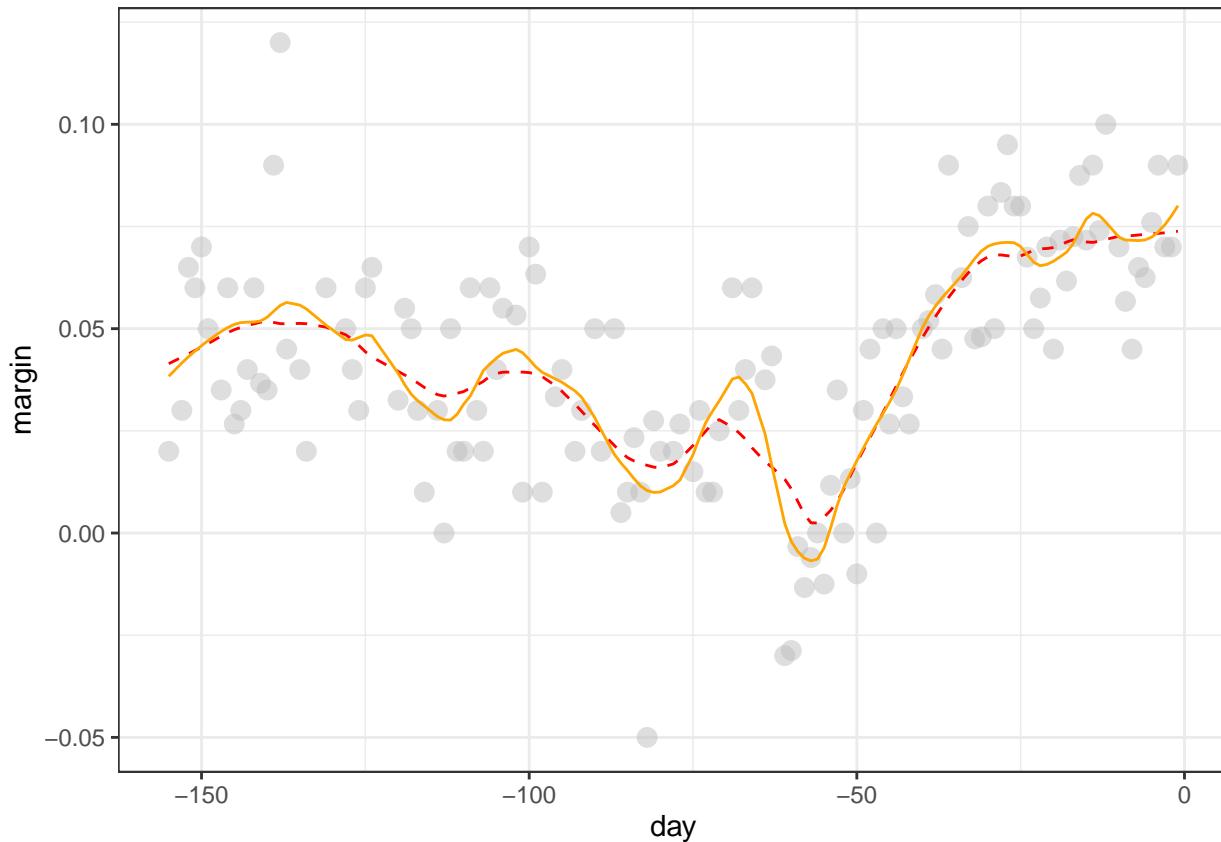
```
total_days <- diff(range(polls_2008$day))

span <- 28/total_days

fit_1 <- loess(margin ~ day, degree=1, span = span, data=polls_2008)

fit_2 <- loess(margin ~ day, span = span, data=polls_2008)
```

```
polls_2008 %>%
  mutate(smooth_1 = fit_1$fitted, smooth_2 = fit_2$fitted) %>%
  ggplot(aes(day, margin)) +
  geom_point(size = 3, alpha = .5, color = "grey") +
  geom_line(aes(day, smooth_1), color="red", lty = 2) +
  geom_line(aes(day, smooth_2), color="orange", lty = 1)
```

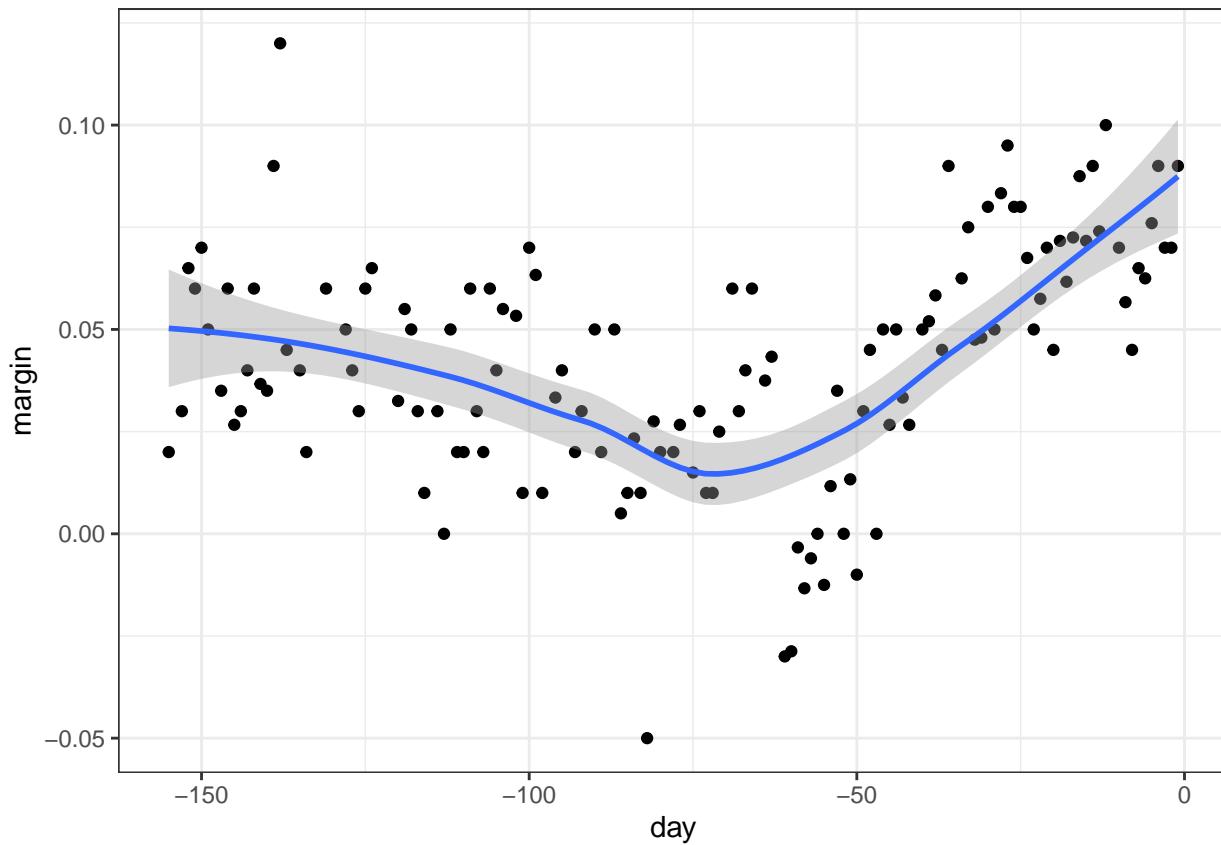


La gráfica anterior muestra la comparación, donde las líneas roja punteada representa líneas ajustadas y la naranja representa paráboles ajustadas (en general, el ajuste de líneas suele arrojar menos ruido que el de paráboles, aunque esto depende del contexto del problema).

**Nota:** Nótese que ggplot usa loess en la función “geom\_smooth()”, así, lo siguiente:

```
polls_2008 %>%
  ggplot(aes(day, margin)) +
  geom_point() +
  geom_smooth()

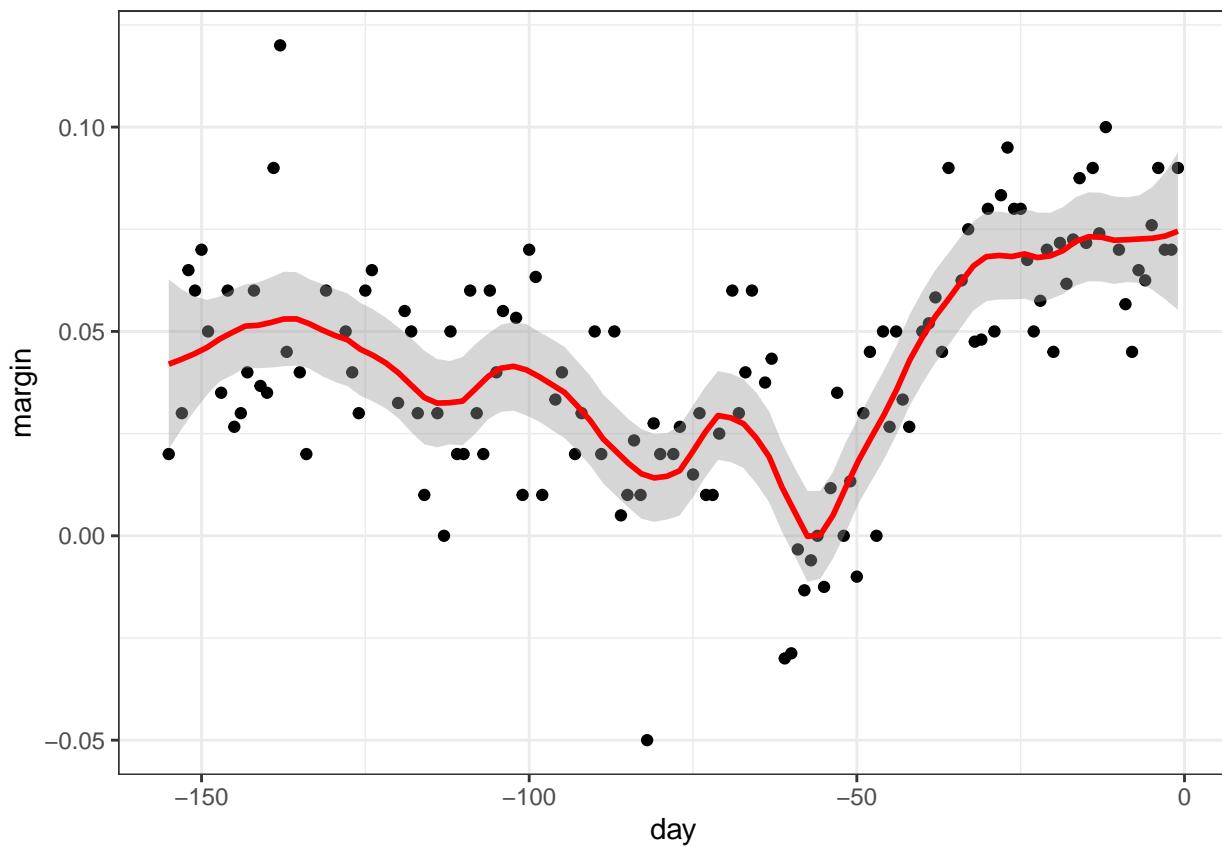
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Se obtiene una línea loess ajustada; sin embargo, el comportamiento mostrado es raramente el óptimo. Así, puede cambiarse el grado del ajuste para obtener una mejor estimación de la tendencia:

```
polls_2008 %>%
  ggplot(aes(day, margin)) +
  geom_point() +
  geom_smooth(color="red", span = 0.15, method = "loess", method.args = list(degree=1))

## `geom_smooth()` using formula 'y ~ x'
```



### 8.3.4. Assessment 22

In the Wrangling course of this series, PH125.6x, we used the following code to obtain mortality counts for Puerto Rico for 2015-2018:

```
library(tidyverse)
library(lubridate)
library(purrr)
library(pdftools)

fn <- system.file("extdata", "RD-Mortality-Report_2015-18-180531.pdf", package="dslabs")

dat <- map_df(str_split(pdf_text(fn), "\n"), function(s){
  s <- str_trim(s)
  header_index <- strwhich(s, "2015")[1]
  tmp <- str_split(s[header_index], "\\s+", simplify = TRUE)
  month <- tmp[1]
  header <- tmp[-1]
  tail_index <- strwhich(s, "Total")
  n <- str_count(s, "\\d+")
  out <- c(1:header_index, which(n==1), which(n>=28), tail_index:length(s))
  s[-out] %>%
    str_remove_all("[^\\d\\s]") %>%
    str_trim() %>%
    str_split_fixed("\\s+", n = 6) %>%
    .[,1:5] %>%
    as_data_frame() %>%
    setNames(c("day", header)) %>%
    mutate(month = month,
           day = as.numeric(day)) %>%
    gather(year, deaths, -c(day, month)) %>%
    mutate(deaths = as.numeric(deaths))
}) %>%
  mutate(month = recode(month, "JAN" = 1, "FEB" = 2, "MAR" = 3, "APR" = 4, "MAY" = 5, "JUN" = 6,
         "JUL" = 7, "AGO" = 8, "SEP" = 9, "OCT" = 10, "NOV" = 11, "DEC" = 12)) %>%
  mutate(date = make_date(year, month, day)) %>%
  dplyr::filter(date <= "2018-05-01")
```

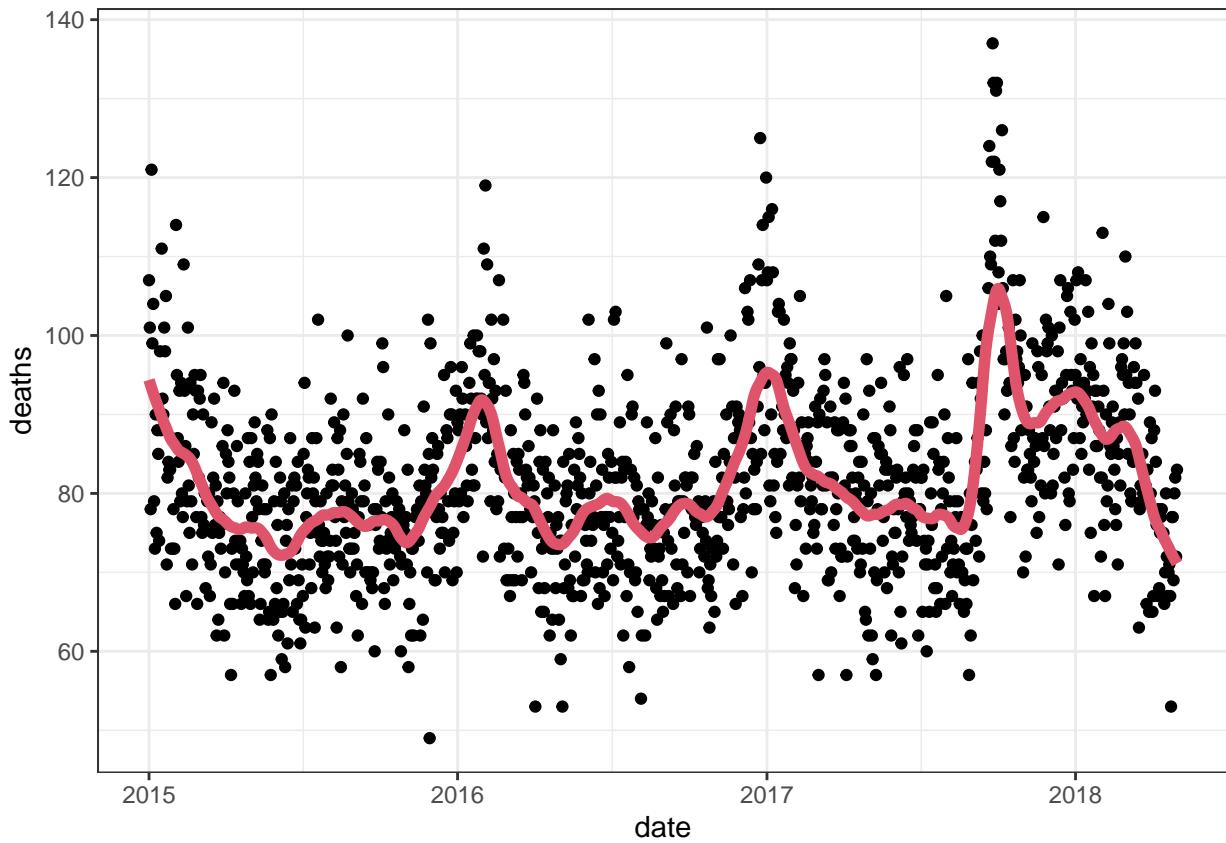
Use the loess() function to obtain a smooth estimate of the expected number of deaths as a function of date. Plot this resulting smooth function. Make the span about two months long and use degree = 1.

```
span <- 60/as.numeric(diff(range(dat$date)))

fit <- dat %>%
  mutate(x = as.numeric(date)) %>%
  loess(deaths ~ x, data =., span = span, degree = 1)

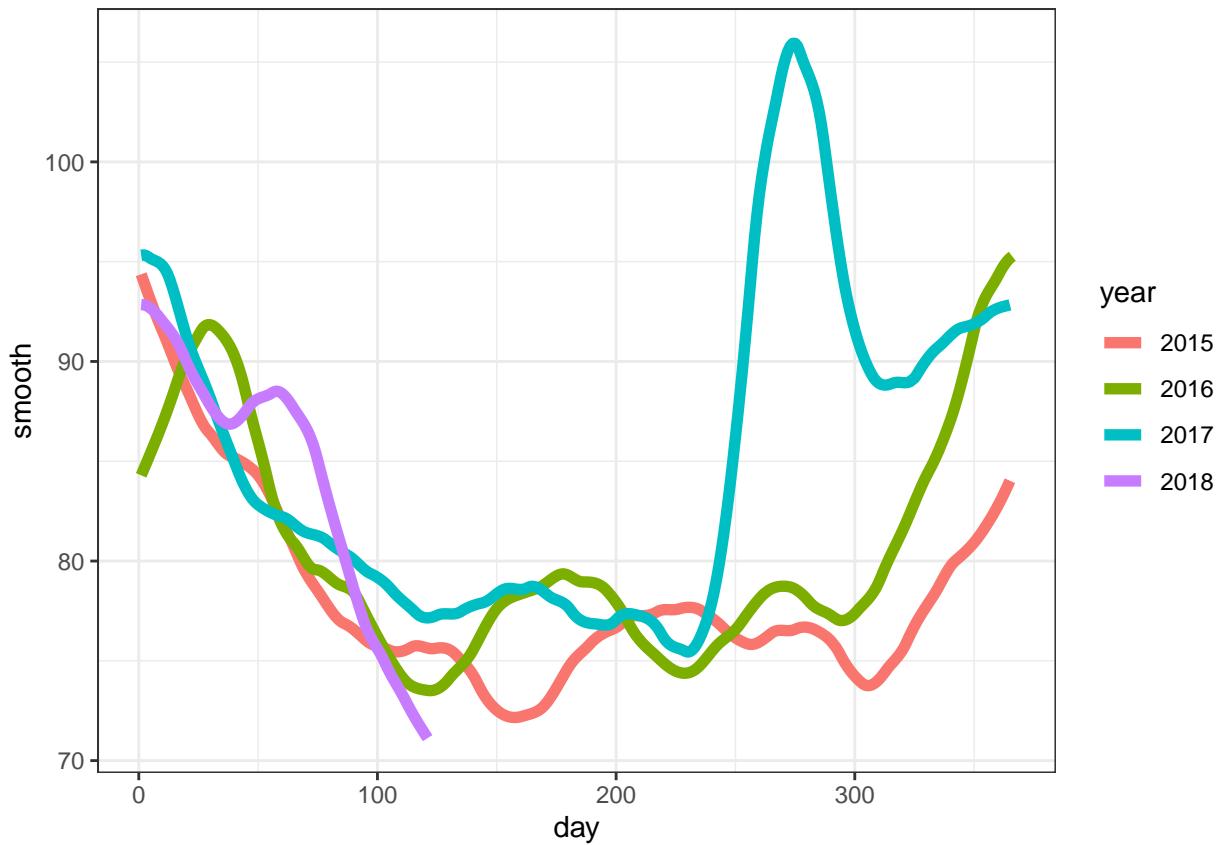
dat %>%
  mutate(smooth = predict(fit, as.numeric(date))) %>%
  ggplot() +
  geom_point(aes(date, deaths)) +
  geom_line(aes(date, smooth), lwd = 2, col =2)

## Warning: Removed 1 rows containing missing values (geom_point).
```



Work with the same data as in Q1 to plot smooth estimates against day of the year, all on the same plot, but with different colors for each year.

```
dat %>%
  mutate(smooth = predict(fit, as.numeric(date)),
        day = yday(date), year = as.character(year(date))) %>%
  ggplot(aes(day, smooth, col = year)) +
  geom_line(lwd = 2)
```



Suppose we want to predict 2s and 7s in the mnist\_27 dataset with just the second covariate. Can we do this? On first inspection it appears the data does not have much predictive power. In fact, if we fit a regular logistic regression the coefficient for x\_2 is not significant! This can be seen using this code:

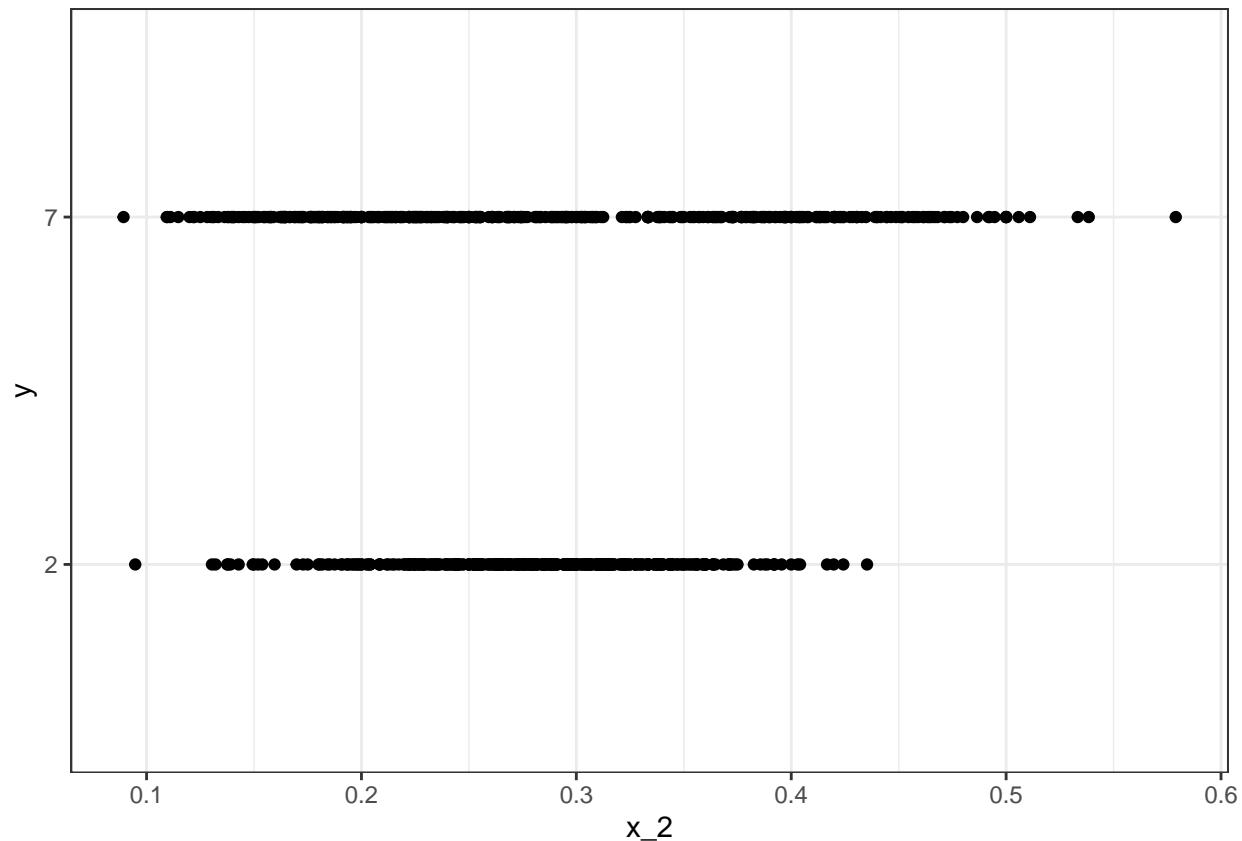
```
library(broom)

mnist_27$train %>%
  glm(y ~ x_2, family = "binomial", data = .) %>%
  tidy()

## # A tibble: 2 x 5
##   term      estimate std.error statistic p.value
##   <chr>      <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept) -0.0907    0.247    -0.368   0.713
## 2 x_2         0.685     0.827     0.829   0.407
```

Plotting a scatterplot here is not useful since y is binary:

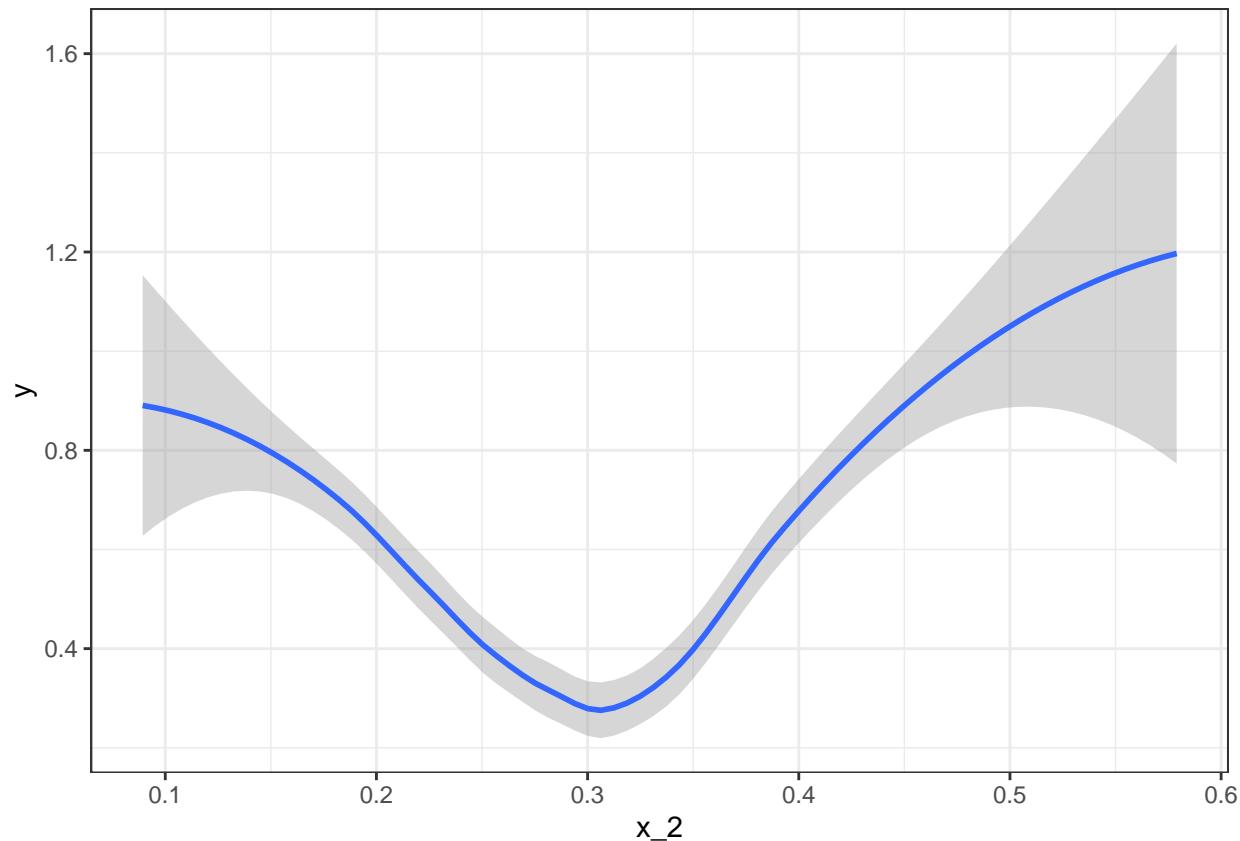
```
qplot(x_2, y, data = mnist_27$train)
```



Fit a loess line to the data above and plot the results.

```
mnist_27$train %>%
  mutate(y = ifelse(y=="7", 1, 0)) %>%
  ggplot(aes(x_2, y)) +
  geom_smooth(method = "loess")

## `geom_smooth()` using formula 'y ~ x'
```



### 8.3.5. Trabajando con matrices

En el aprendizaje automático, las situaciones donde todos los predictores son numéricos (o pueden ser convertidos a numéricos) son muy comunes. Una exemplificación es la base de datos de dígitos, donde cada pixel registra un número entre 0 y 255; cárguese esta base de datos:

```
library(tidyverse)
library(dslabs)

mnist <- read_mnist()
```

Puede resultar conveniente guardar a los predictores en una matriz y los resultados en un vector, en vez de usar un *data frame*.

```
class(mnist$train$images)
```

```
## [1] "matrix" "array"
```

Nótese que es una matriz sumamente grande; para fines ilustrativos, tomaremos los primeros 1 000 predictores y las primeras 1 000 etiquetas:

```
x <- mnist$train$images[1:1000,]

y <- mnist$train$labels[1:1000]
```

**La razón principal para utilizar matrices es que ciertas operaciones matemáticas necesarias para desarrollar un código eficiente pueden ser llevadas a cabo utilizando técnicas del álgebra lineal.**

Para motivar el uso de matrices, estableceremos cinco retos:

1. Estudiar la distribución de la oscuridad total de los pixeles y cómo varía por dígito.
2. Estudiar la variación de cada pixel y remover los predictores(columnas) asociados a pixeles que no cambian mucho y, por tanto, no proveen mucha información para la clasificación.
3. Poner a cero los valores bajos que probablemente hagan ruido. Primero, mirando a la distribución de todos los valores de los pixeles y usando esta elección para hacer un *cutoff* para definir espacios no escritos. Después, hacer cualquier cosa por debajo de ese *cutoff* cero.
4. Binarizar los datos. Primero, observando a la distribución de los valores de los pixeles, usando esta elección para elegir un *cutoff* y distinguir entre lo escrito y no escrito. Así, convertir todas las entradas en 1 o 0 respectivamente.
5. Escalar cada predictor en cada entrada para que tenga la misma media y desviación estándar.

**8.3.5.1. Notación matricial** En álgebra matricial, existen tres tipos de objetos: escalares (e.g.  $a = 1$ ), vectores (e.g.  $\mathbf{X} = \begin{pmatrix} x_1 & x_2 & \dots & x_n \end{pmatrix}$ ) y matrices:

$$\mathbf{X} = \begin{pmatrix} x_{11} & x_{22} & \dots & x_{1p} \\ x_{21} & x_{22} & \dots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{np} \end{pmatrix}_{(n \times p)}$$

En R, podemos extraer la dimensión de una matriz con la función “dim()”:

```
x_1 <- 1:5

x_2 <- 6:10
```

```
dim(cbind(x_1, x_2))

## [1] 5 2

dim(x)

## [1] 1000 784
```

Si bien teóricamente un vector es de dimensión  $nx1$  o  $1xn$ , en R no se reconoce:

```
dim(x_1)
```

```
## NULL
```

Sin embargo, podemos convertir explícitamente un vector a matriz mediante “as.matrix()”:

```
dim(as.matrix(x_1))
```

```
## [1] 5 1
```

También podemos convertir un vector a matriz con “matrix()”, la cual nos permite especificar el número de renglones y columnas:

```
my_vector <- 1:15
```

```
mat <- matrix(my_vector, 5, 3)
```

```
class(mat)
```

```
## [1] "matrix" "array"
```

Alternativamente, puede construirse la matriz por renglón en vez de por columna mediante el argumento “byrow”:

```
mat_t <- matrix(my_vector, 3, 5, byrow = TRUE)
```

```
mat_t
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]     1     2     3     4     5
## [2,]     6     7     8     9    10
## [3,]    11    12    13    14    15
```

Como puede verse, lo que realizamos fue una transposición matricial. R nos permite realizar esta operación directamente con la función “t()”:

```
identical(t(mat), mat_t)
```

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

**Nota:** Es muy importante recordar que “matrix()” recicla valores en el vector sin avisar. Esto es, si el producto de columnas y renglones no empareja con la longitud del vector, ocurre lo siguiente:

```
matrix(my_vector, 5, 5)
```

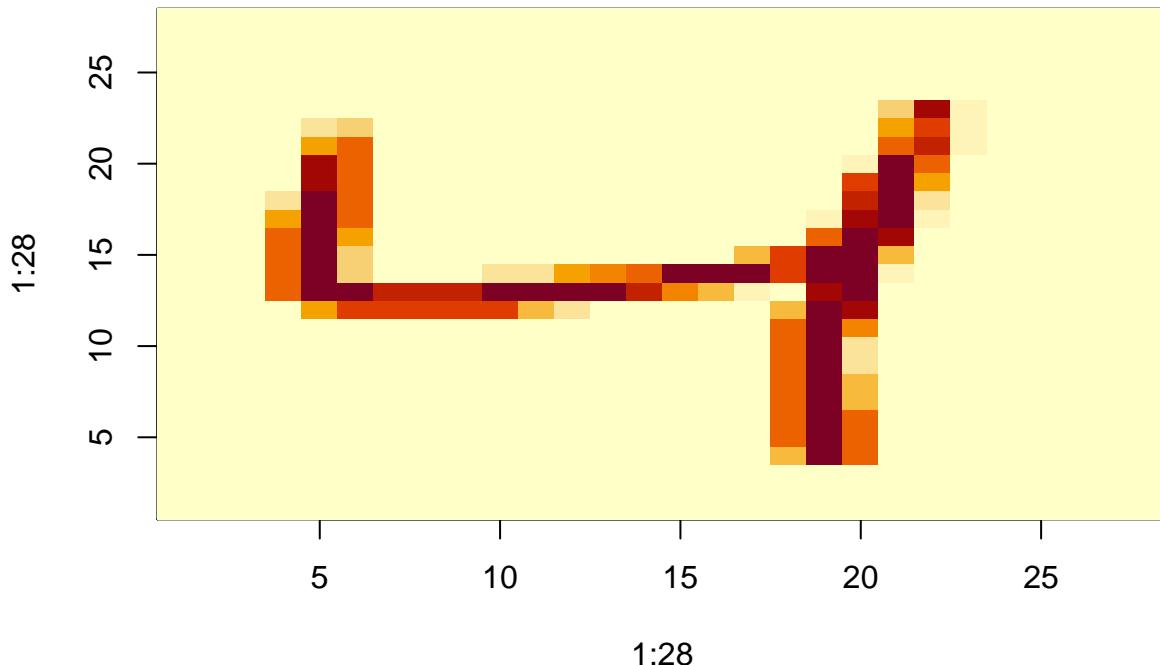
```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]     1     6    11     1     6
## [2,]     2     7    12     2     7
## [3,]     3     8    13     3     8
## [4,]     4     9    14     4     9
## [5,]     5    10    15     5    10
```

Retomando el ejemplo de los dígitos y pixeles, supongamos que queremos colocar la intensidad del pixel de la tercera entrada (sabemos que es un 4) en una parrilla, podemos usar lo siguiente:

```
grid <- matrix(x[3,], 28, 28)
```

Para confirmar que hemos realizado la tarea correctamente, podemos usar la función “image()”, la cual muestra una imagen del tercer argumento:

```
image(1:28, 1:28, grid[,28:1])
```



Retómese el primer reto que planteamos (relacionado a la oscuridad total de los pixeles). Respecto a esto, queremos resumir la suma de los valores de cada renglón y después visualizar cómo varían estos valores por dígito. La función “rowSums()” toma una matriz como entrada y computa los valores deseados:

```
sums <- rowSums(x)
```

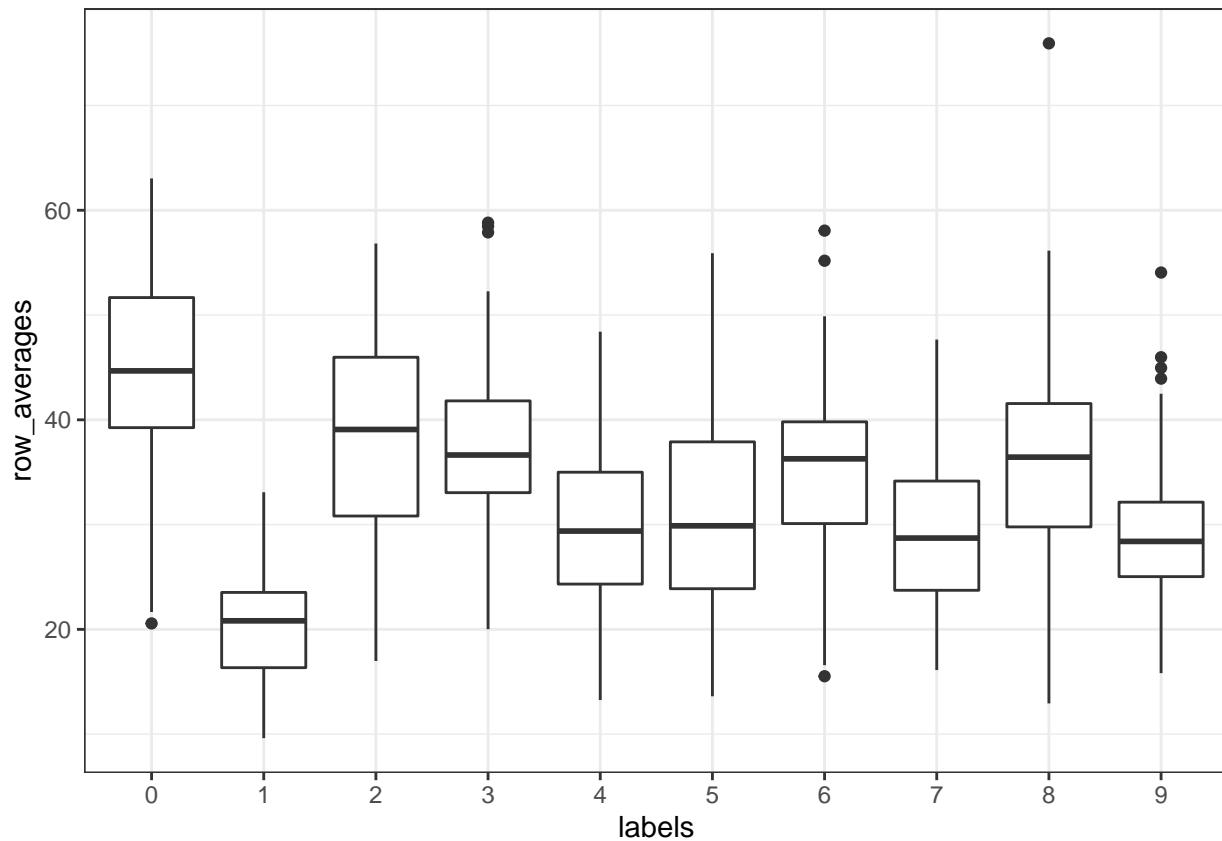
También podemos computar las medias con la función “rowMeans()”:

```
avg <- rowMeans(x)
```

Con esto, podemos generar un *boxplot* para observar cómo la intensidad media de pixeles cambia de un dígito a otro:

```
data_frame(labels = as.factor(y), row_averages = avg) %>%
  qplot(labels, row_averages, data = ., geom = "boxplot")
```

```
## Warning: `data_frame()` was deprecated in tibble 1.1.0.
## Please use `tibble()` instead.
```



Donde podemos ver que los 1s usan menos tinta que otros dígitos.

Nótese que también podemos computar las sumas y medias por columna con “`colSum()`” y `colMeans()`. El paquete `matrixStats` agrega funciones que realizan operaciones útiles en cada renglón o columna, incluyendo `rowSds()` y `colSds()` (desviaciones estándar). Nótese que estas funciones son similares a “`sapply()`” ya que permiten aplicar una función a una partición de nuestros datos.

Así, “`rowMeans()`” es equivalente a:

```
avgs <- apply(x, 1, mean)
```

Y las desviaciones estándar pueden calcularse mediante:

```
sds <- apply(x, 2, sd)
```

Pasemos ahora al segundo reto planteado (estudiar la variación de cada pixel y remover las columnas asociadas a pixeles que no cambian demasiado y no son informativos). Cuantificaremos esta variación con la desviación estándar de todas las entradas; como cada columna representa un pixel, usaremos la función “`colSds()`”:

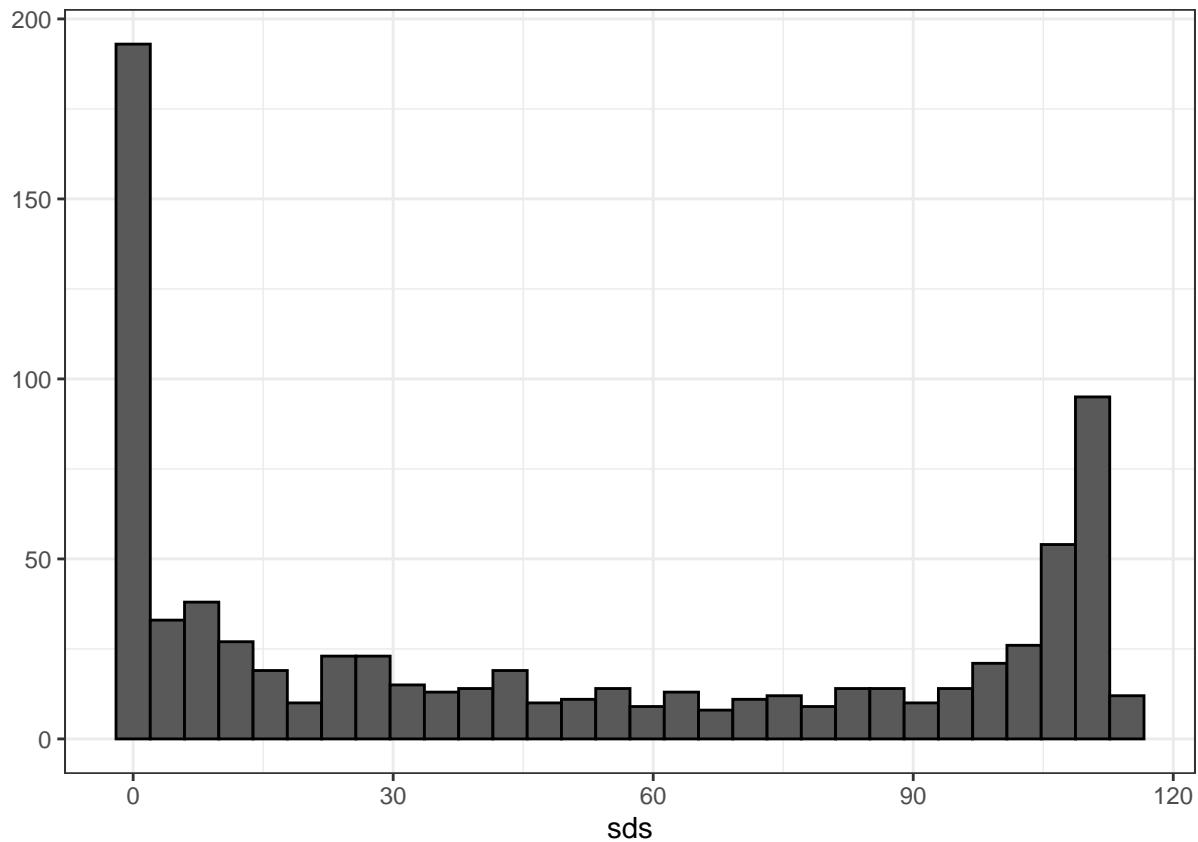
```
library(matrixStats)
```

```
## Warning: package 'matrixStats' was built under R version 4.0.5
##
## Attaching package: 'matrixStats'
##
## The following object is masked from 'package:dplyr':
##
##     count
```

```
sds <- colSds(x)
```

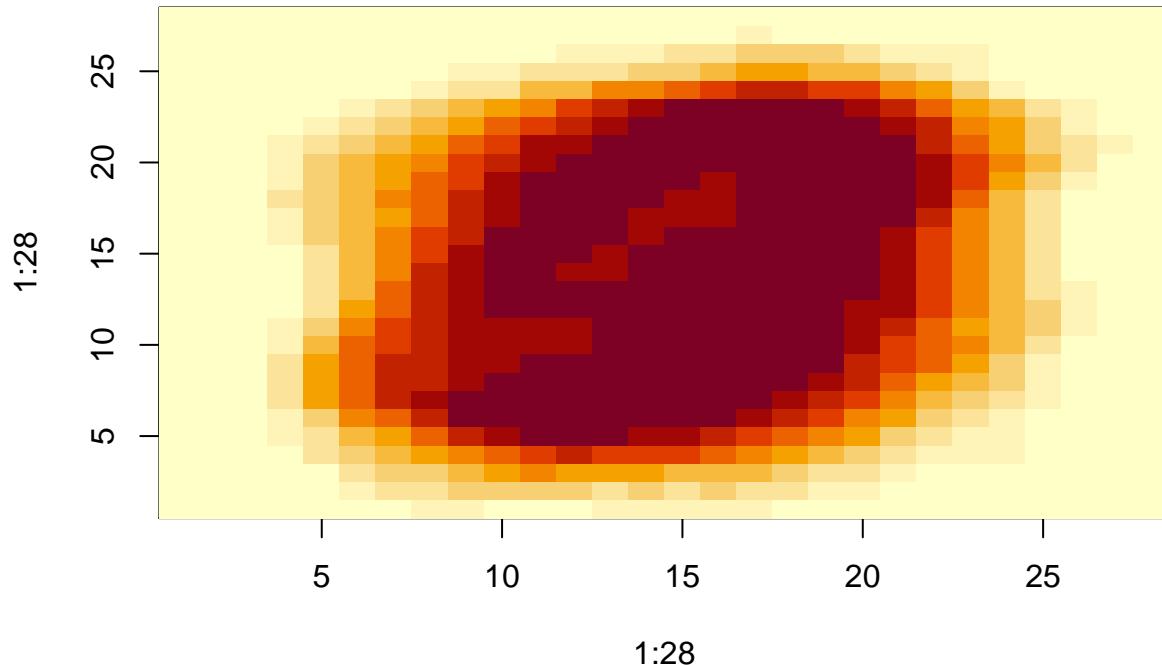
Una rápida inspección de la distribución de estos valores muestra que algunos pixeles tienen una variabilidad entrada por entrada muy baja, lo cual tiene sentido, ya que no se escribe en todas las partes de los cuadrantes.

```
qplot(sds, bins= "30", color = I("black"))
```



De hecho, la variación por locación es como se muestra a continuación:

```
image(1:28, 1:28, matrix(sds, 28, 28) [, 28:1])
```



Puede verse poca variación en las esquinas, debido a que escribimos los dígitos en el centro.

En la primera sección de este documento, *R Basics*, describimos cómo extraer columnas y renglones:

```
cols <- x[,c(351, 352)]
```

```
rows <- x[c(2,3),]
```

Así, si queremos remover predictores no informativos de nuestra matriz (conservando las columnas con una desviación estándar mayor a 60), podemos usar el siguiente código:

```
new_x <- x[, colSds(x) > 60]
```

```
dim(new_x)
```

```
## [1] 1000 314
```

**Nota:** Si seleccionamos una columna o renglón, el resultado se convertirá en un vector y dejará de ser una matriz:

```
class(x[,1])
```

```
## [1] "integer"
```

```
dim(x[1,])
```

```
## NULL
```

Sin embargo, podemos conservar la clase “matrix” con el argumento “drop”:

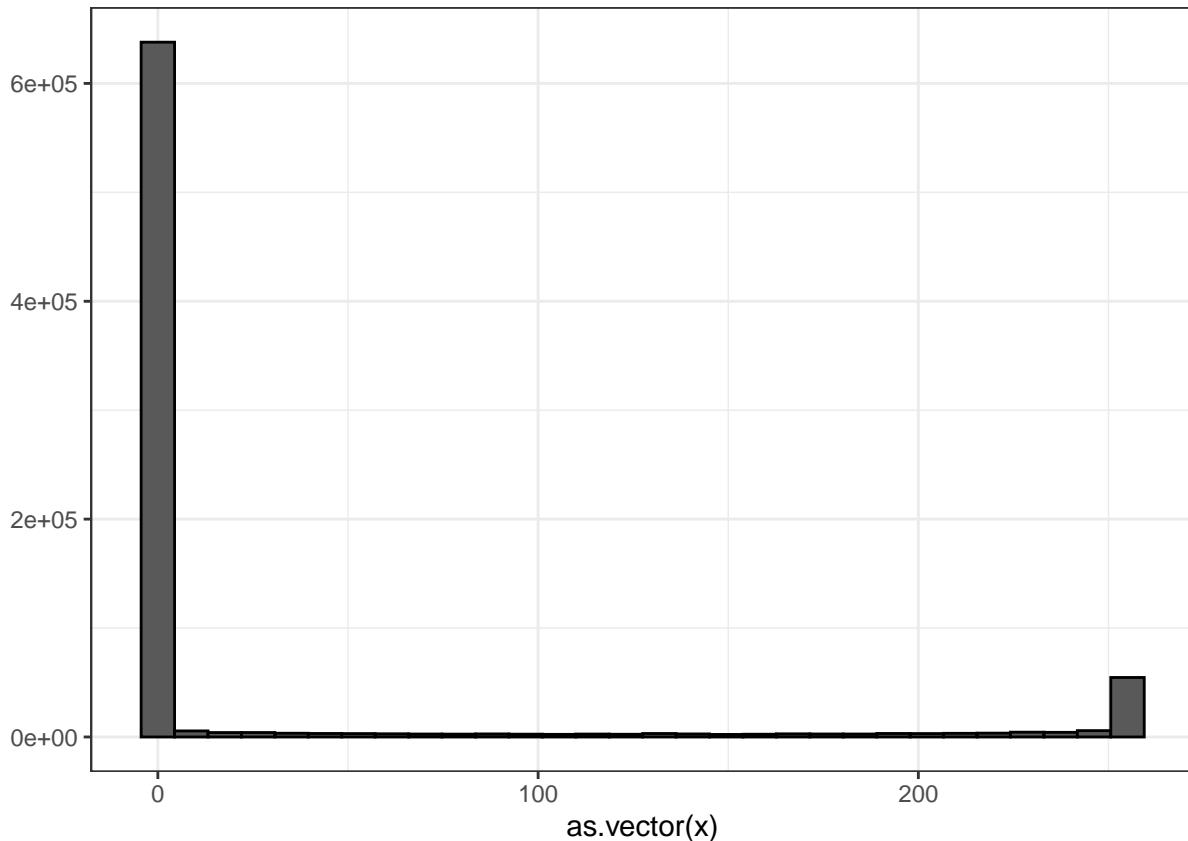
```
class(x[, 1, drop = FALSE])  
  
## [1] "matrix" "array"  
dim(x[, 1, drop = FALSE])  
  
## [1] 1000     1
```

Para nuestro siguiente reto, queremos visualizar un histograma de todos nuestros pixeles; ya hemos visto cómo convertir vectores a matrices, pero también podemos hacer lo opuesto con la función “as.vector()”:

```
mat <- matrix(1:15, 5, 3)  
  
mat  
  
##      [,1] [,2] [,3]  
## [1,]     1     6    11  
## [2,]     2     7    12  
## [3,]     3     8    13  
## [4,]     4     9    14  
## [5,]     5    10    15  
  
as.vector(mat)  
  
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
```

Así, para ver un histograma de todos nuestros predictores, úsese la siguiente línea de código:

```
qplot(as.vector(x), bins = 30, color = I("black"))
```



Donde vemos una clara dicotomía que puede ser explicada como partes con tinta y partes sin tinta. Supóngase que creemos que valores menores a 25 son manchas, entonces podemos hacerlos cero como se muestra enseguida:

```
new_x <- x

new_x[new_x < 50] <- 0
```

Para observar lo que realizamos, obsérvese una matriz más pequeña con un ejemplo más simple:

```
mat <- matrix(1:15, 5, 3)

mat[mat < 4] <- 0

mat

##      [,1] [,2] [,3]
## [1,]    0    6   11
## [2,]    0    7   12
## [3,]    0    8   13
## [4,]    4    9   14
## [5,]    5   10   15
```

También podemos realizar operaciones más complejas como la que sigue:

```
mat <- matrix(1:15, 5, 3)

mat[mat > 6 & mat < 12] <- 0

mat

##      [,1] [,2] [,3]
## [1,]    1    6    0
## [2,]    2    0   12
## [3,]    3    0   13
## [4,]    4    0   14
## [5,]    5    0   15
```

Para el siguiente reto, queremos binarizar los datos (derivado del último histograma, es sensato afirmar que los datos son mayormente binarios, ya que hay tinta o no la hay).

Con el código siguiente, cambiamos todos los valores menores a 255/2 a 0s y mayores a 255/2 a 1s.

```
bin_x <- x

bin_x[bin_x < 255/2] <- 0

bin_x[bin_x > 255/2] <- 1
```

Pero también podemos convertir lo anterior a una matriz usando operadores lógicos y coercionándolos a número:

```
bin_X <- (x > 255/2)*1
```

Para nuestro reto final, en el cual estandarizaremos renglones o columnas, usaremos vectorización. En R, si sustraemos un vector de una matriz, el primer elemento de cada vector es sustraído de la primera fila de la matriz; el segundo elemento del vector se sustrae del segundo renglón de la matriz, y así sucesivamente.

Utilizando notación matemática:

$$\begin{pmatrix} X_{11} & \dots & X_{1p} \\ X_{21} & \dots & X_{2p} \\ \vdots & \ddots & \vdots \\ X_{n1} & \dots & X_{np} \end{pmatrix} - \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} X_{11} - a_1 & \dots & X_{1p} - a_1 \\ X_{21} - a_2 & \dots & X_{2p} - a_2 \\ \vdots & \ddots & \vdots \\ X_{n1} - a_n & \dots & X_{np} - a_n \end{pmatrix}$$

Así, podemos escalar cada renglón de una matriz con el siguiente código:

```
tmp <- (x - rowMeans(x)) / rowSds(x)
```

Nótese que esto no funciona para columnas, en caso de querer realizar una operación así, tendremos que transponer la matriz:

```
tmp <- (t(x) - colMeans(x))
```

Para esta tarea, también podemos usar la función “sweep()”, que funciona de manera similar a “apply()”: toma cada entrada de un vector y lo sustrae de la fila o columna correspondiente. Por ejemplo, en el siguiente código, restamos la media a las columnas

```
x_mean_0 <- sweep(x, 2, colMeans(x))
```

“sweep()” tiene otro argumento que permite definir la operación aritmética (por default es una resta). Así, para dividir por la desviación estándar:

```
x_mean_0 <- sweep(x, 2, colMeans(x))
```

```
x_standarized <- sweep(x_mean_0, 2, colSds(x), FUN = "/")
```

De manera sucinta, se presentan otras operaciones matriciales relevantes a continuación:

- Multiplicación matricial: %\*%

```
mult <- t(x) %*% x
```

- Producto cruzado: crossprod()

```
mult <- crossprod(x)
```

- Matriz inversa: solve()

- Descomposición qr: qr()

### 8.3.6. Assessment 23

Which line of code correctly creates a 100 by 10 matrix of randomly generated normal numbers and assigns it to x?

```
x <- matrix(rnorm(100*10), 100, 10)
```

Write the line of code that would give you the specified information about the matrix x that you generated in q1. Do not include any spaces in your line of code.

**Dimension of x**

```
dim(x)
```

```
## [1] 100 10
```

**Number of rows of x**

```
nrow(x)
```

```
## [1] 100
```

**Number of columns of x**

```
ncol(x)
```

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

Which of the following lines of code would add the scalar 1 to row 1, the scalar 2 to row 2, and so on, for the matrix x?

```
x <- x + seq(nrow(x))
```

```
x <- sweep(x, 1, 1:nrow(x), FUN = "+")
```

Which of the following lines of code would add the scalar 1 to column 1, the scalar 2 to column 2, and so on, for the matrix x?

```
x <- sweep(x, 2, 1:ncol(x), FUN = "+")
```

Which code correctly computes the average of each row of x?

```
rowMeans(x)
```

```
## [1] 6.86 9.31 11.74 14.11 15.60 17.69 19.38 21.64 24.26 25.42
## [11] 27.62 29.30 31.33 33.17 35.91 37.42 39.18 41.40 43.77 45.64
## [21] 47.56 49.83 51.15 53.16 55.52 57.78 59.62 61.64 63.77 65.45
## [31] 67.43 69.91 71.17 73.12 75.09 77.44 79.70 81.51 83.00 85.08
## [41] 87.33 89.63 91.93 93.84 95.64 97.57 99.69 101.16 103.31 105.63
## [51] 107.52 109.51 111.63 113.97 115.09 117.92 119.49 121.41 124.01 126.27
## [61] 127.39 129.55 131.82 133.62 135.58 137.77 139.93 141.47 143.63 145.64
## [71] 147.91 150.16 151.00 153.80 155.47 157.97 159.72 161.38 163.32 165.33
## [81] 167.52 169.58 171.01 172.84 176.01 177.42 179.56 181.26 183.55 185.40
## [91] 187.51 189.95 191.52 193.35 195.59 197.33 199.38 201.68 203.89 206.03
```

Which code correctly computes the average of each column of x?

```
colMeans(x)
```

```
## [1] 102 103 104 105 106 107 108 109 110 111
```

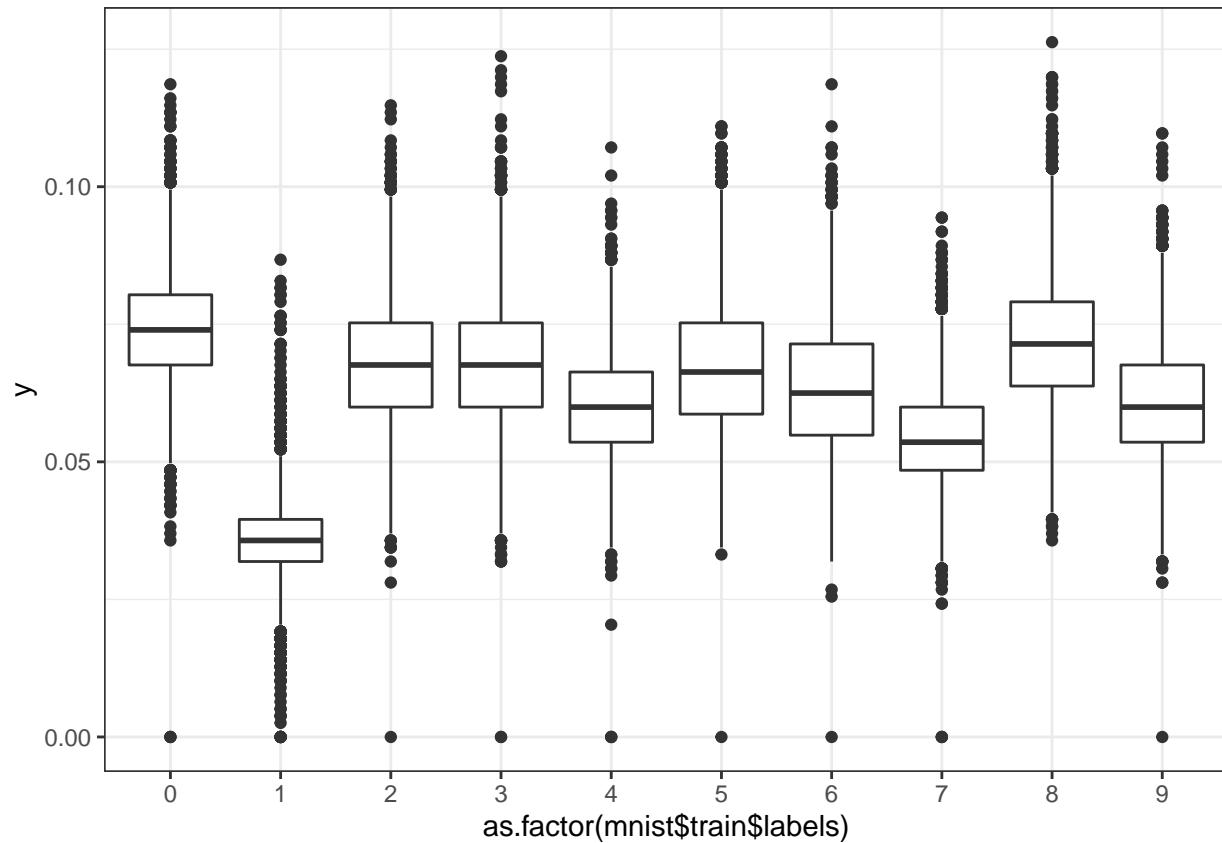
For each observation in the mnist training data, compute the proportion of pixels that are in the grey area, defined as values between 50 and 205 (but not including 50 and 205). (To visualize this, you can make a boxplot by digit class.) What proportion of the 60000x784 pixels

in the mnist training data are in the grey area overall, defined as values between 50 and 205? Report your answer to at least 3 significant digits.

```
mnist <- read_mnist()

y <- rowMeans(mnist$train$images > 50 & mnist$train$images < 205)

qplot(as.factor(mnist$train$labels), y, geom = "boxplot")
```



```
mean(y)

## [1] 0.0618
```

## 8.4. Distancia, kNN, validación cruzada y modelos generativos

### 8.4.1. Vecinos cercanos

Muchas técnicas del aprendizaje automático se basan en ser capaces de definir distancias entre observaciones utilizando características o predictores.

A manera de revisión, defínase la distancia entre dos puntos  $A$  y  $B$ ; la distancia euclídea  $AB$  está dada por la siguiente fórmula:

$$\text{dist}(A, B) = \sqrt{(A_x - B_x)^2 + (A_y - B_y)^2}$$

Si nuestros números son unidimensionales, la fórmula anterior es equivalente a:

$$\text{dist}(A, B) = \sqrt{(A - B)^2} = |A - B|$$

Anteriormente, introdujimos datos de práctica con una matriz de medidas para 784 características. Para fines ilustrativos, miramos a muestras aleatorias de 2s y 7s:

```
if(!exists("mnist")) mnist <- read_mnist()

set.seed(0, sample.kind = "Rounding")

## Warning in set.seed(0, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used

ind <- which(mnist$train$labels %in% c(2,7)) %>%
  sample(500)

x <- mnist$train$images[ind,] #predictores
y <- mnist$train$labels[ind] #etiquetas
```

Si queremos realizar tareas de suavización, nos interesa entonces describir las distancias entre observaciones (dígitos). Si queremos seleccionar características, podríamos encontrar pixeles que se comporten similarmente a través de muestras.

Cuando nos encontramos con datos de altas dimensiones, los puntos no se encuentran en el plano Cartesiano, por lo que no pueden visualizarse y hay que pensarlos de manera abstracta. Por ejemplo, en el caso de dígitos, un predictor  $X_i$  está definido como puntos en un espacio dimensional de orden 784:  $X_i = (X_{i1}, \dots, x_{i784})^T$ .

Una vez definidos nuestros puntos de esa manera, la distancia euclídea es similar al caso de dos dimensiones. Por ejemplo, para las observaciones 1 y 2, la distancia es:

$$\text{dist}(1, 2) = \sqrt{\sum_{j=1}^{784} (x_{1j} - x_{2j})^2}$$

Fijémonos ahora en las primeras cuatro observaciones:

```
y[1:3]
```

```
## [1] 7 7 2
```

El vector de predictores para cada una de estas observaciones serán guardados en tres objetos:

```
x_1 <- x[1,]
x_2 <- x[2,]
x_3 <- x[3,]
```

Ahora consideremos las distancias: esperamos que estas sean pequeñas cuando se trata del mismo número y mayores cuando sean diferentes:

```
sqrt(sum((x_1 - x_2)^2)) #iguales
## [1] 2080
sqrt(sum((x_1 - x_3)^2)) #diferentes
## [1] 2252
sqrt(sum((x_2 - x_3)^2)) #diferentes
## [1] 2643
```

Una manera más rápida de computar esto es mediante un producto cruzado:

```
sqrt(crossprod(x_1 - x_2)) #iguales
##      [,1]
## [1,] 2080
sqrt(crossprod(x_1 - x_3)) #diferentes
##      [,1]
## [1,] 2252
sqrt(crossprod(x_2 - x_3)) #diferentes
##      [,1]
## [1,] 2643
```

Así, podemos computar todas las distancias de todas las observaciones relativamente rápido con la función “dist()”. Esta función toma una matriz y calcula las distancias entre cada renglón y produce un objeto de clase dist:

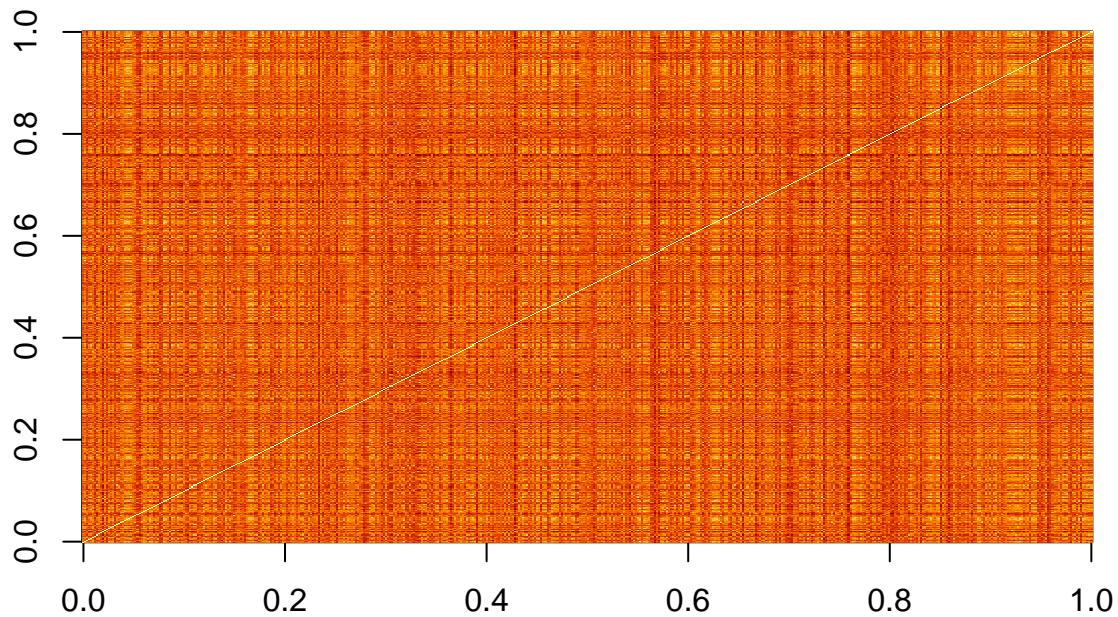
```
d <- dist(x)
class(d)
## [1] "dist"
```

Podemos coercionar este objeto a matriz como sigue:

```
as.matrix(d)[1:3, 1:3]
##      1    2    3
## 1    0 2080 2252
## 2 2080    0 2643
## 3 2252 2643    0
```

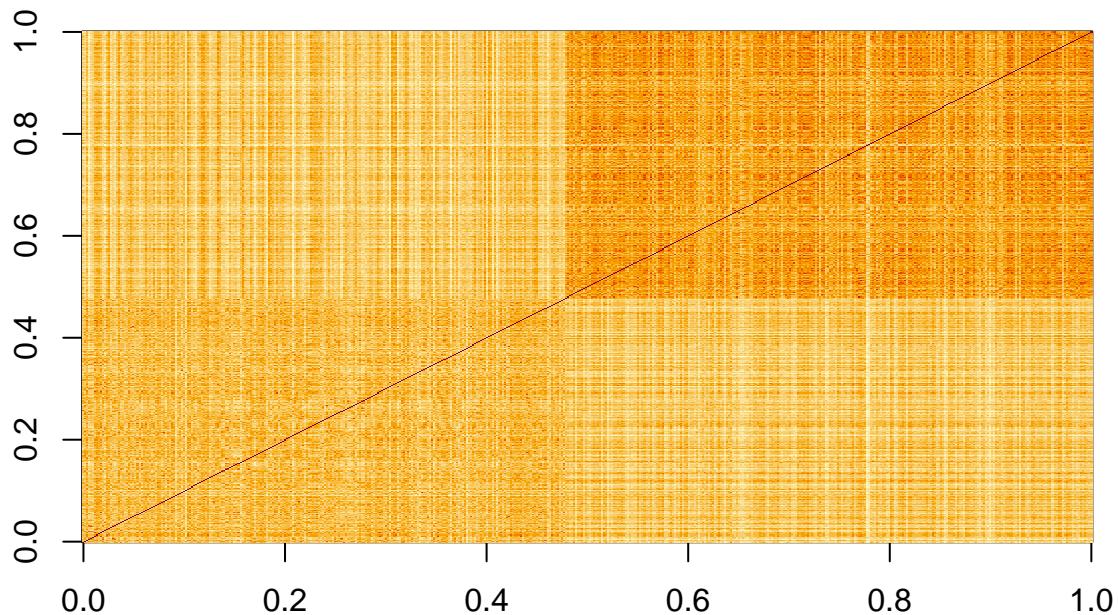
También podemos ver una imagen de estas distancias como se muestra a continuación:

```
image(as.matrix(d))
```



Si ordenamos las distancias por etiqueta, podemos ver que, en general, los 2s y 7s están más cerca unos de otros:

```
image(as.matrix(d)[order(y), order(y)], col = hcl.colors(12, "YlOrRd", rev=FALSE))
```



Donde podemos ver que si bien los 2s presentan una distancia pequeña, los 7 aún más. Así, para calcular la distancia entre todos los pares de los 784 predictores, hay que transponer la matriz:

```
d <- dist(t(x))
```

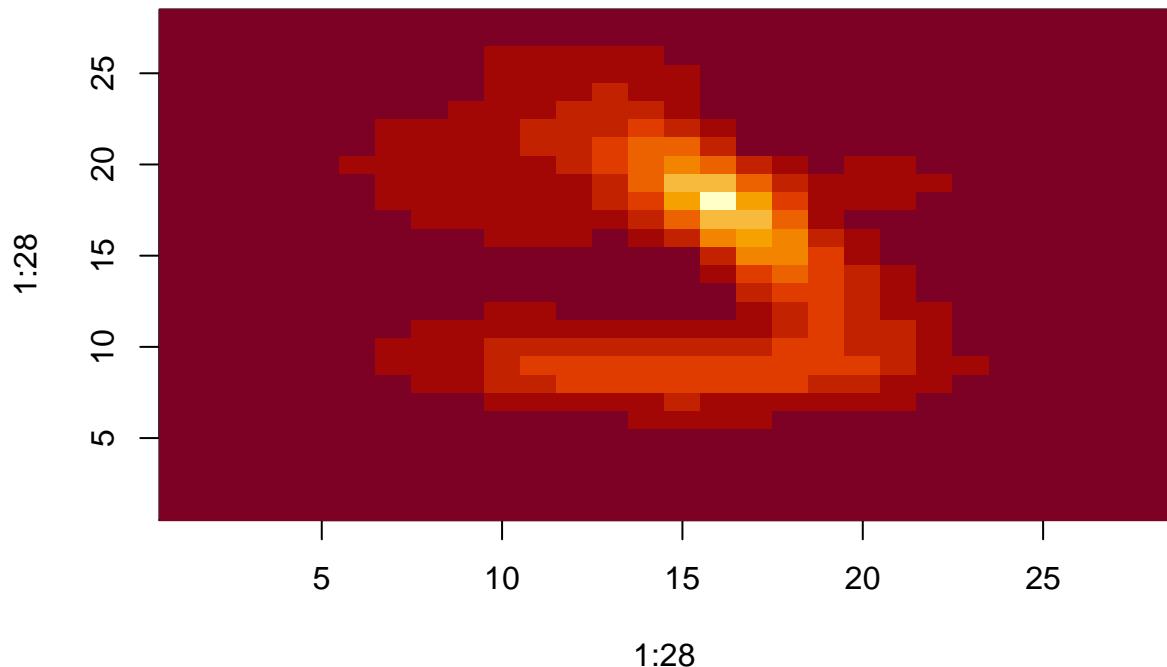
```
dim(as.matrix(d))
```

```
## [1] 784 784
```

Es interesante notar que si elegimos un predictor (pixel), podemos ver qué pixeles están más cerca, lo que significa que o ambos tienen tinta o no la tienen. Como ejemplo, considérese el pixel 492 y sus distancias con todos los demás:

```
d_492 <- as.matrix(d)[492,]
```

```
image(1:28, 1:28, matrix(d_492, 28, 28))
```



No es sorprendente que los pixeles cercanos físicamente también lo están matemáticamente.

#### 8.4.2. Assessment 24

Load the following dataset:

```
library(dslabs)
data(tissue_gene_expression)
```

This dataset includes a matrix x:

```
dim(tissue_gene_expression$x)
```

```
## [1] 189 500
```

This matrix has the gene expression levels of 500 genes from 189 biological samples representing seven different tissues. The tissue type is stored in y:

```
table(tissue_gene_expression$y)
```

```
##
##   cerebellum      colon endometrium hippocampus      kidney      liver
##           38          34          15          31          39          26
##   placenta
##           6
```

Which of the following lines of code computes the Euclidean distance between each observation and stores it in the object d?

```
d <- dist(tissue_gene_expression$x)
```

Using the dataset from Q1, compare the distances between observations 1 and 2 (both cerebellum), observations 39 and 40 (both colon), and observations 73 and 74 (both endometrium). Distance-wise, are samples from tissues of the same type closer to each other than tissues of different type?

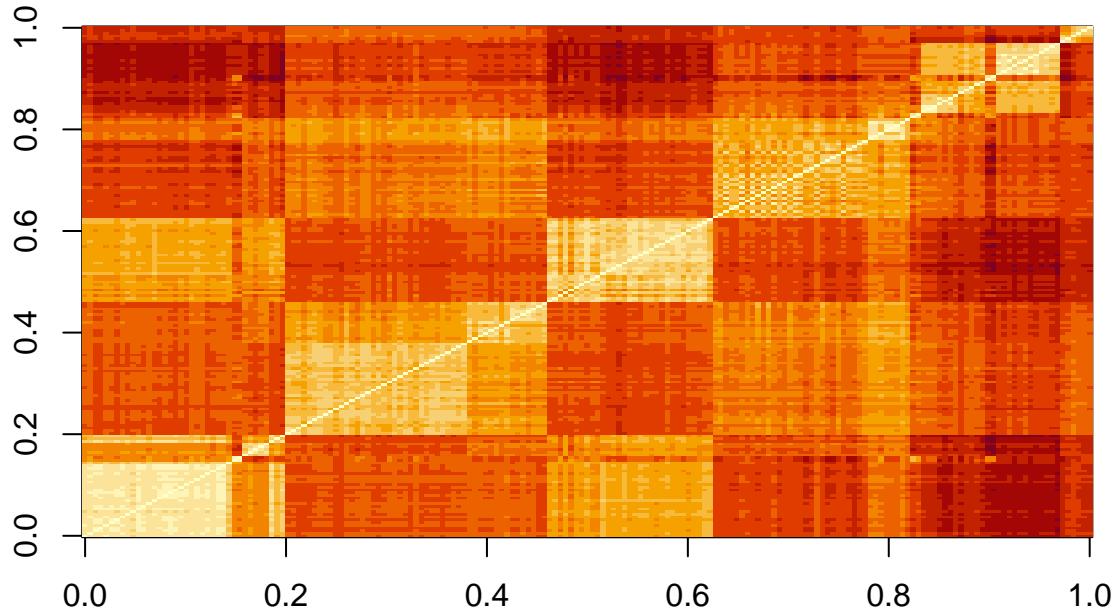
```
index <- c(1, 2, 39, 40, 73, 74)
```

```
as.matrix(d)[index, index]
```

```
##          cerebellum_1 cerebellum_2 colon_1 colon_2 endometrium_1
## cerebellum_1      0.00     7.01 22.69 22.70      21.1
## cerebellum_2      7.01     0.00 22.38 22.07      20.9
## colon_1           22.69    22.38 0.00  8.19      15.0
## colon_2           22.70    22.07 8.19  0.00      14.8
## endometrium_1      21.13   20.88 15.00 14.80      0.0
## endometrium_2      21.78   20.67 18.09 17.00      14.3
##          endometrium_2
## cerebellum_1      21.8
## cerebellum_2      20.7
## colon_1            18.1
## colon_2            17.0
## endometrium_1      14.3
## endometrium_2      0.0
```

Make a plot of all the distances using the `image()` function to see if the pattern you observed in Q2 is general.

```
image(as.matrix(d))
```



#### 8.4.3. kNN

A continuación, aprenderemos nuestro primer algoritmo de aprendizaje automático, el *k-nearest neighbors algorithm*; para ello, usaremos el ejemplo de dígitos con dos predictores utilizado anteriormente.

Este algoritmo está relacionado con la suavización y la probabilidad condicional de que un dígito sea 7 ( $Y = 1$ ); i.e.  $p(x_1, x_2) = Pr(Y = 1|X_1 = x_1, X_2 = x_2)$ . Esto se debe a que los 0s y 1s que observamos son ruidosos; es decir, que algunas regiones de la probabilidad condicional no son cercanas ni a 0 ni a 1.

kNN estima la probabilidad condicional de manera similar a la suavización de barras; sin embargo, kNN es fácilmente adaptable para trabajar con dimensiones múltiples.

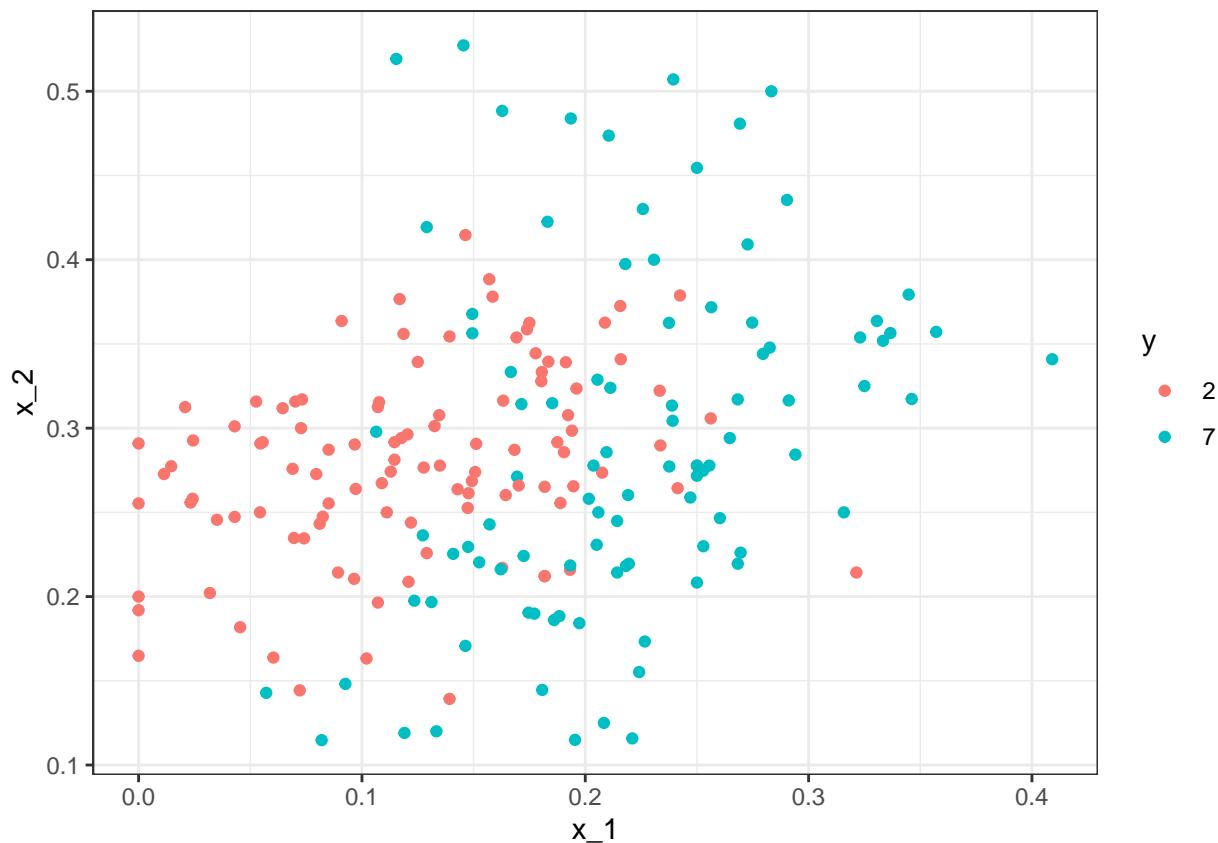
Primero, definiremos la distancia entre las observaciones basándonos en características. Básicamente, para cualquier punto que queramos estimar la probabilidad condicional, nos fijaremos en los k-puntos más cercanos y los promediaremos (este conjunto se conoce como vecindad). En otras palabras, obtendremos una probabilidad condicional estimada,  $\hat{p}(x_1, x_2)$ .

Además, podemos controlar la flexibilidad de nuestro estimado mediante  $k$ : una  $k$  más grande suavizará el estimado más; una  $k$  pequeña resultará en estimados más ondulados y flexibles.

Primero, computemos las predicciones glm del modelo logístico que habíamos utilizado anteriormente:

```
library(tidyverse)
library(dslabs)
data("mnist_27")

mnist_27$test %>%
  ggplot(aes(x_1, x_2, color = y)) +
  geom_point()
```



```
library(caret)

fit_glm <- glm(y~x_1+x_2, data=mnist_27$train, family="binomial")

p_hat_logistic <- predict(fit_glm, mnist_27$test)

y_hat_logistic <- factor(ifelse(p_hat_logistic > 0.5, 7, 2))

confusionMatrix(data = y_hat_logistic, reference = mnist_27$test$y)$overall[1]

## Accuracy
##      0.76
```

Nótese que tenemos una precisión de 0.76. Ahora, comparemos este resultado con kNN mediante el uso de la función “knn3()” del paquete caret. Esta función puede usarse de varias formas: (1) especificando la fórmula y un *data frame* donde la primera tiene la forma “outcome ~ predictor\_1 + predictor\_2 + predictor\_3” o (2) si vamos a usar todos los predictores, usamos un *shortcut* “outcome ~ .”:

```
knn_fit <- knn3(y ~., data = mnist_27$train)
```

Otra forma de usar esta función es (3) cuando el primer argumento es una matriz de predictores y el segundo un vector de resultados:

```
x <- as.matrix(mnist_27$train[,2:3]) #matriz de predictores

y <- mnist_27$train$y #matriz de resultados

knn_fit <- knn3(x, y)
```

Esta función también requiere que escojamos un parámetro (número de vecindades para incluir). Comencemos con  $k = 5$  (el cual es el default):

```
knn_fit <- knn3(y ~ ., data=mnist_27$train, k = 5)
```

Dado que esta base de datos está balanceada y nos importa igualmente la sensibilidad y la especificidad, usaremos precisión para evaluar nuestro rendimiento.

La función “predict()” para “knn3()” produce o una probabilidad para cada clase o el resultado que maximiza la probabilidad:

```
y_hat_knn <- predict(knn_fit, mnist_27$test, type = "class") # class arroja los resultados
```

Una vez con esto, podemos calcular la precisión mediante la matriz de confusión:

```
confusionMatrix(data = y_hat_knn, reference = mnist_27$test$y)$overall["Accuracy"]
```

```
## Accuracy
## 0.815
```

Sin embargo, nótese qué ocurre cuando predecimos en conjunto de práctica y en el de prueba:

```
y_hat_knn <- predict(knn_fit, mnist_27$train, type = "class")
confusionMatrix(data = y_hat_knn, reference = mnist_27$train$y)$overall["Accuracy"]
```

```
## Accuracy
## 0.882
y_hat_knn <- predict(knn_fit, mnist_27$test, type = "class")
confusionMatrix(data = y_hat_knn, reference = mnist_27$test$y)$overall["Accuracy"]
```

```
## Accuracy
## 0.815
```

El *overtraining* es una razón por la cual podemos tener una precisión mayor en el conjunto de práctica comparado con el conjunto de prueba. El peor caso de este fenómeno es cuando  $k = 1$ , donde el estimado de cada par  $(x_1, x_2)$  en el conjunto de práctica es obtenido con solo la  $y$  correspondiente a ese punto (obtendríamos precisión casi perfecta, como en las siguientes líneas de código):

```
knn_fit_1 <- knn3(y ~ ., data = mnist_27$train, k = 1)
y_hat_knn_1 <- predict(knn_fit_1, mnist_27$train, type = "class")
confusionMatrix(data=y_hat_knn_1, reference=mnist_27$train$y)$overall[["Accuracy"]]
```

```
## [1] 0.995
```

```
y_hat_knn_1 <- predict(knn_fit_1, mnist_27$test, type = "class")
confusionMatrix(data=y_hat_knn_1, reference=mnist_27$test$y)$overall[["Accuracy"]]
```

```
## [1] 0.74
```

Probemos un ejemplo con una  $k$  mucho mayor, sea  $k = 401$ :

```
knn_fit_401 <- knn3(y ~ ., data = mnist_27$train, k = 401)
```

```
y_hat_knn_401 <- predict(knn_fit_401, mnist_27$test, type = "class")
```

```
confusionMatrix(data=y_hat_knn_401, reference=mnist_27$test$y)$overall["Accuracy"]
```

```
## Accuracy
## 0.79
```

Donde la precisión no es muy buena, de hecho, es similar a la de una regresión logística. Al contrario del primer ejemplo, esto se conoce como *oversmoothing*.

Así, para encontrar la  $k$  óptima, repitamos el ejercicio para diferentes valores de la vecindad; probemos todos los números impares de 3 a 251:

```
ks <- seq(3, 251, 2)

library(purrr)

accuracy <- map_df(ks, function(k){
  fit <- knn3(y ~ ., data = mnist_27$train, k = k)

  y_hat <- predict(fit, mnist_27$train, type = "class")
  cm_train <- confusionMatrix(data = y_hat, reference = mnist_27$train$y)
  train_error <- cm_train$overall["Accuracy"]

  y_hat <- predict(fit, mnist_27$test, type = "class")
  cm_test <- confusionMatrix(data = y_hat, reference = mnist_27$test$y)
  test_error <- cm_test$overall["Accuracy"]

  tibble(train = train_error, test = test_error)
})
```

Ahora, podemos escoger la  $kk$  que maximiza nuestra precisión:

```
ks[which.max(accuracy$test)]

## [1] 41
max(accuracy$test)

## [1] 0.86
```

#### 8.4.4. Assessment 25

Previously, we used logistic regression to predict sex based on height. Now we are going to use knn to do the same. Set the seed to 1, then use the caret package to partition the dslabs heights data into a training and test set of equal size. Use the sapply() function to perform knn with  $k$  values of seq(1, 101, 3) and calculate F1 scores with the F\_meas() function using the default value of the relevant argument.

What is the max value of F\_1?

```
library(dslabs)
library(tidyverse)
library(caret)
data("heights")

set.seed(1)

test_index <- createDataPartition(heights$sex, times = 1, p = 0.5, list = FALSE)

test_set <- heights[test_index,]

train_set <- heights[-test_index,]
```

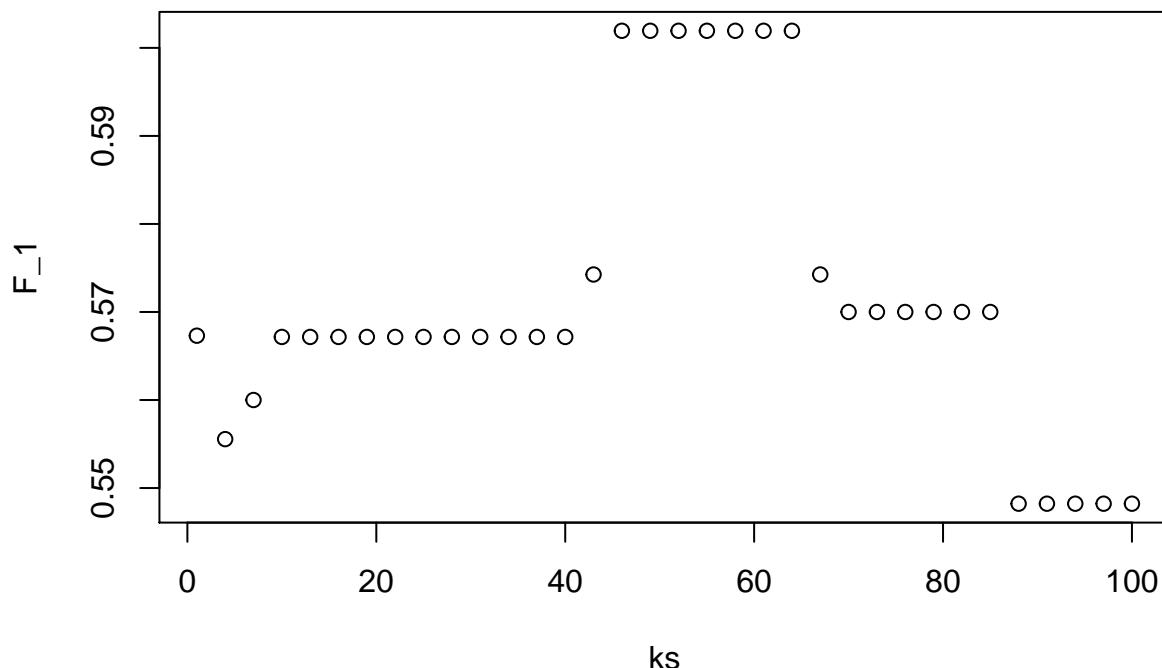
```

ks <- seq(1, 101, 3)

F_1 <- sapply(ks, function(k){
  fit <- knn3(sex ~ height, data = train_set, k = k)
  y_hat <- predict(fit, test_set, type = "class") %>%
    factor(levels = levels(train_set$sex))
  F_meas(data = y_hat, reference = test_set$sex)
})

plot(ks, F_1)

```



```

max(F_1)

## [1] 0.602

At what value of k does the max occur?

ks[which.max(F_1)]

```

```
## [1] 46
```

\*Next we will use the same gene expression example used in the Comprehension Check: Distance exercises. You can load it like this:\*\*

```

library(dslabs)
library(caret)
data("tissue_gene_expression")

```

First, set the seed to 1 and split the data into training and test sets with  $p = 0.5$ . Then, report

the accuracy you obtain from predicting tissue type using KNN with  $k = \text{seq}(1, 11, 2)$  using `sapply()` or `map_df()`. Note: use the `createDataPartition()` function outside of `sapply()` or `map_df()`.

$k=1, 3, 5, 7, 9, 11$

```
set.seed(1, sample.kind = "Rounding")
y <- tissue_gene_expression$y
x <- tissue_gene_expression$x
tissue_df <- data.frame(tissue_gene_expression)

test_index <- createDataPartition(tissue_df$y, times = 1, p = 0.5, list = FALSE)
train_set <- tissue_df[-test_index,]
test_set <- tissue_df[test_index,]
ks = seq(1, 11, 2)

accuracy <- map_df(ks, function(k){
  fit <- knn3(y ~ ., data = train_set, k = k)
  y_hat <- predict(fit, test_set, type = "class") %>% factor(levels = levels(test_set$y))
  confusionMatrix(y_hat, test_set$y)$overall["Accuracy"]
})

accuracy

## # A tibble: 6 x 1
##   Accuracy
##   <dbl>
## 1 0.990
## 2 0.969
## 3 0.948
## 4 0.917
## 5 0.917
## 6 0.906
```

#### 8.4.5. Validación cruzada

**8.4.5.1. k-fold** Cuando todo lo que tenemos a nuestra disposición es una base de datos, podemos estimar el error cuadrado medio (MSE) con el error cuadrado medio observado como sigue:

$$\hat{MSE} = \frac{1}{N} \sum_{i=1}^N (\hat{y}_i - y_i)^2$$

Estas dos cantidades son normalmente conocidas como **error real** y **error aparente**, donde el último es una variable aleatoria dado que nuestros datos son aleatorios y, al utilizar el algoritmo en el conjunto de datos de donde lo obtuvimos, puede llevar a *overtraining*.

La **validación cruzada** es una técnica que permite aliviar estos problemas. Para entender este concepto, conviene pensar al error real, una cantidad teórica, como el promedio de muchos errores aparentes obtenidos al aplicar el algoritmo a nuevas muestras aleatorias de los datos:

$$\frac{1}{B} \sum_{b=1}^B \frac{1}{N} \sum_{i=1}^N (\hat{y}_i^b - y_i^b)^2$$

La idea de la validación cruzada es imitar esta configuración teórica como mejor sea posible y con los datos disponibles. Para ello, deberemos generar una serie de muestras aleatorias diferentes. Así, estas generarán aleatoriamente conjuntos de datos no usados para práctica, sino para estimar el error real. El primer enfoque para esto es la **validación cruzada k-fold**.

Recuérdese que un algoritmo de *machine learning* comienza con una base de datos a partir de la cual crearemos un algoritmo que será usado en una base de datos completamente independiente de la primera. Dado que no tenemos acceso a la segunda, realizamos una partición de nuestra primera base de datos para practicar y probar el algoritmo.

Los valores típicos para dividir nuestro conjunto de datos son: 10 %-20 % para el conjunto de prueba y 80 %-90 % para el conjunto de práctica (recuérdese una vez más que es indispensable no usar el conjunto de prueba cuando practicamos nuestro algoritmo).

El siguiente paso es escoger parámetros, sea  $\lambda$  el conjunto de parámetros de elección. Así, tenemos que optimizar  $\lambda$  sin usar nuestro conjunto de prueba y evitar el *overtraining*.

Para evitar esto, haremos uso de *k\_fold*: para cada conjunto de parámetros del algoritmo que se considere, estimaremos el MSE, para después escoger los parámetros con el menor MSE.

Antes de comenzar el procedimiento de validación cruzada, es necesario fijar todos los parámetros, dado que computaremos el MSE para un parámetro dado. Usaremos la notación  $\hat{y}_i(\lambda)$  para denotar la predicción obtenida al utilizar el parámetro  $\lambda$  en la observación  $i$ . Esto implica:

$$MSE(\lambda) = \frac{1}{B} \sum_{b=1}^B \frac{1}{N} \sum_{i=1}^N (\hat{y}_i^b(\lambda) - y_i^b)^2$$

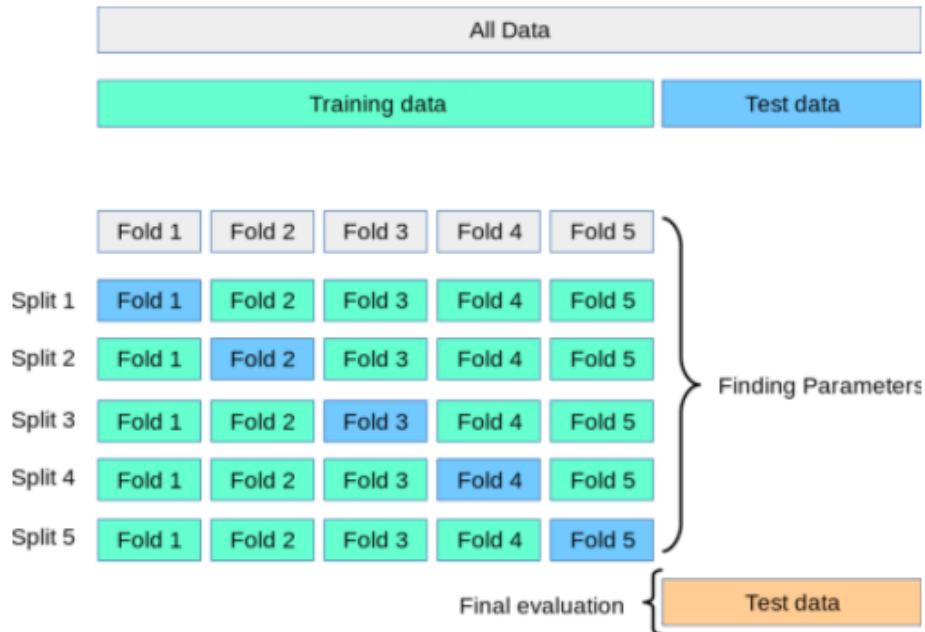
Esto es, el promedio de  $B$  muestras que hemos tomado de los MSE obtenidos de los datos separados como conjunto de prueba. Con esta fórmula, queremos considerar conjuntos de datos que puedan ser pensados como muestras aleatorias independientes. Realizaremos esta acción  $k$  veces.

Para construir, por ejemplo, la primera muestra, escogemos  $M = N/K$  observaciones (redondeadas al entero más cercano) y asumimos que es una muestra aleatoria  $y_1^b, \dots, y_M^b$ ; en este primer ejemplo,  $b = 1$  (se trata del primer “pliegue”).

Así, ahora tenemos tres conjuntos de datos: el de prueba, separado originalmente, y ahora el de práctica y uno nuevo (derivado de nuestra muestra aleatoria) llamado de **validación**. Ahora, podemos estimar nuestro modelo en el conjunto de práctica y computar el error aparente en el conjunto de validación como sigue:

$$\hat{MSE}_b(\lambda) = \frac{1}{M} \sum_{i=1}^M (\hat{y}_i^b(\lambda) - y_i^b)^2$$

Si solo tomamos una muestra, el error estimado será bastante ruidoso: por este motivo es que se toman  $k$  muestras que no se superpongan. La siguiente imagen resume este proceso:

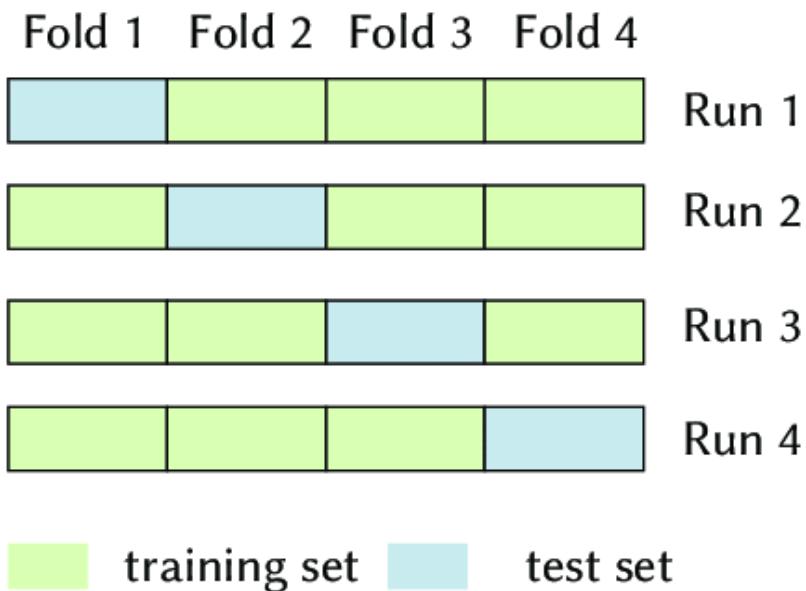


Así, este proceso se repite para cada conjunto  $b = 1, \dots, K$  de tal forma que obtengamos  $k$  estimadores MSE,  $\hat{MSE}_1(\lambda), \dots, \hat{MSE}_K(\lambda)$ . En nuestro estimado final, calcularemos el promedio como sigue:

$$\hat{MSE}(\lambda) = \frac{1}{B} \sum_{b=1}^K \hat{MSE}_b(\lambda)$$

El cual es un estimador de nuestra pérdida. El último paso será escoger  $\lambda$  que minimicen el MSE.

Así, ya podremos usar nuestras estimaciones realizadas en el conjunto de práctica para evaluar nuestro conjunto de prueba. De hecho, también podemos realizar una validación cruzada a la hora de evaluar:



No obstante lo anterior, es común utilizar solo un conjunto de prueba.

Ahora, ¿cómo escoger  $k$ ? Valores altos de pliegues son preferibles porque los datos de práctica imitarán mejor a los originales; la desventaja es que mientras más alta  $k$ , más tiempo tomará para calcularse. Así, las elecciones populares suelen ser  $k = 5$  y  $k = 10$ .

#### 8.4.6. Assessment 26

Generate a set of random predictors and outcomes using the following code:

```
library(tidyverse)
library(caret)

set.seed(1996, sample.kind="Rounding")

n <- 1000

p <- 10000

x <- matrix(rnorm(n*p), n, p)

colnames(x) <- paste("x", 1:ncol(x), sep = "_")

y <- rbinom(n, 1, 0.5) %>%
  factor()

x_subset <- x[,sample(p, 100)]
```

Because  $x$  and  $y$  are completely independent, you should not be able to predict  $y$  using  $x$  with accuracy greater than 0.5. Confirm this by running cross-validation using logistic regression to fit the model. Because we have so many predictors, we selected a random sample  $x_{\text{subset}}$ .

Use the subset when training the model.

```
fit <- train(x_subset, y)

## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :
## non-uniform 'Rounding' sampler used
fit$results

##   mtry Accuracy Kappa AccuracySD KappaSD
## 1     2    0.505 0.00304    0.0146  0.0296
## 2    51    0.505 0.00575    0.0197  0.0367
## 3   100    0.505 0.00649    0.0230  0.0451
```

Now, instead of using a random selection of predictors, we are going to search for those that are most predictive of the outcome. We can do this by comparing the values for the group to those in the group, for each predictor, using a t-test. You can do perform this step like this:

```
install.packages("BiocManager")
```

```
## package 'BiocManager' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##   C:\Users\marco\AppData\Local\Temp\RtmpOursnv\downloaded_packages
BiocManager::install("genefilter")
library(genefilter)

tt <- colttests(x, y)
```

Which of the following lines of code correctly creates a vector of the p-values called pvals?

```
pvals <- tt$p.value
```

Create an index ind with the column numbers of the predictors that were “statistically significantly” associated with y. Use a p-value cutoff of 0.01 to define “statistically significantly.” How many predictors survive this cutoff?

```
ind <- which(pvals <= 0.01)

length(ind)
```

```
## [1] 108
```

Now re-run the cross-validation after redefining x\_subset to be the subset of x defined by the columns showing “statistically significant” association with y. What is the accuracy now?

```
x_subset <- x[,ind]
```

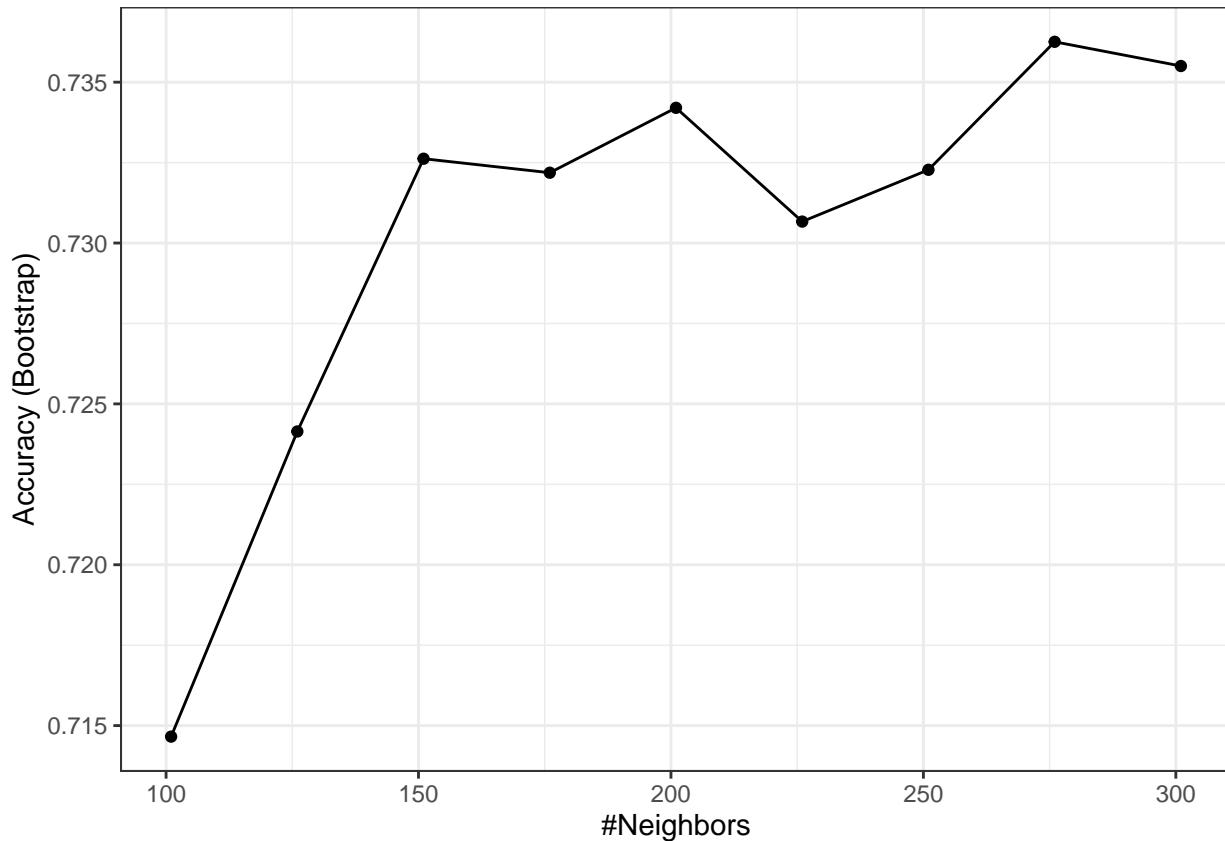
```
fit <- train(x_subset, y, method = "glm")
```

```
## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :
## non-uniform 'Rounding' sampler used
fit$results
```

```
##   parameter Accuracy Kappa AccuracySD KappaSD
## 1       none    0.757 0.512    0.0273  0.0553
```

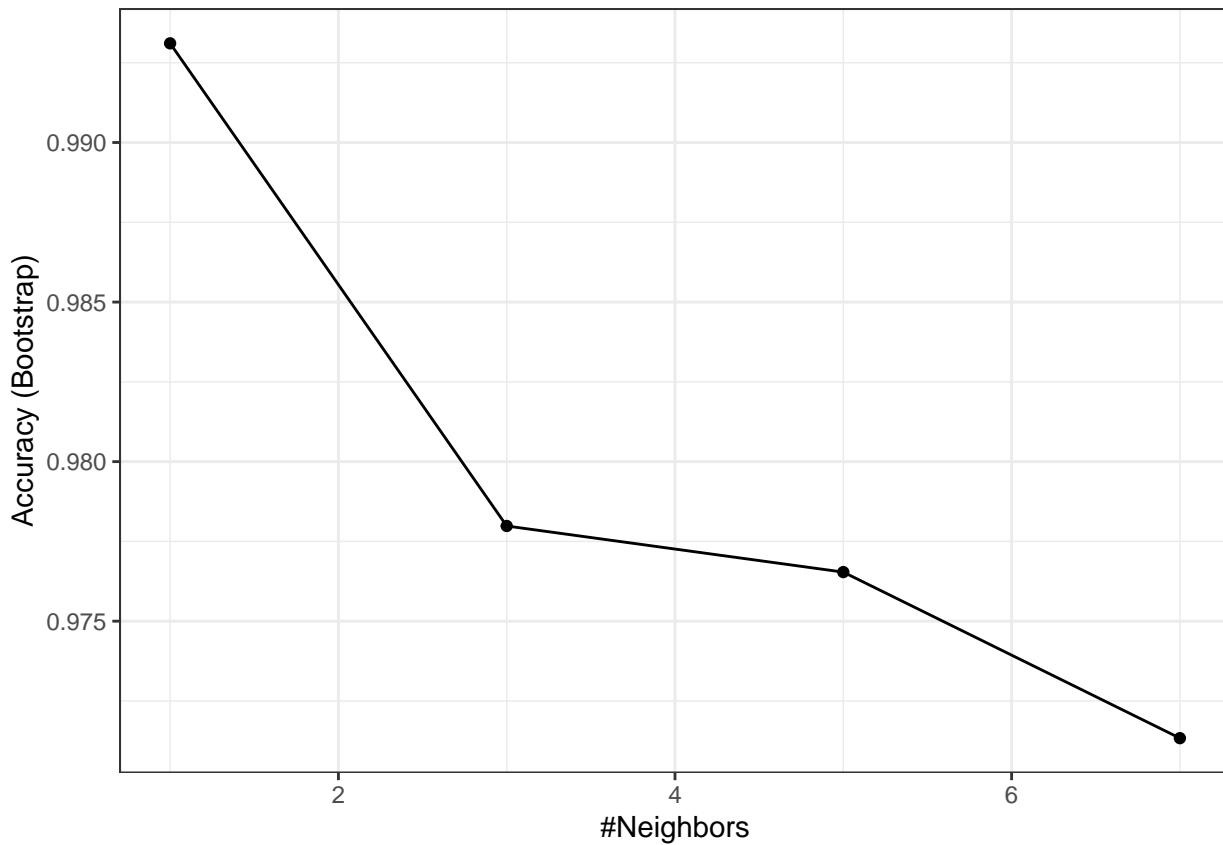
Re-run the cross-validation again, but this time using kNN. Try out the following grid k = seq(101, 301, 25) of tuning parameters. Make a plot of the resulting accuracies.

```
fit <- train(x_subset, y, method = "knn", tuneGrid = data.frame(k = seq(101, 301, 25)))  
  
## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :  
## non-uniform 'Rounding' sampler used  
ggplot(fit)
```



Use the `train()` function with kNN to select the best k for predicting tissue from gene expression on the `tissue_gene_expression` dataset from `dslabs`. Try `k = seq(1,7,2)` for tuning parameters. For this question, do not split the data into test and train sets (understand this can lead to overfitting, but ignore this for now). What value of k results in the highest accuracy?

```
library(dslabs)  
data(tissue_gene_expression)  
  
fit <- with(tissue_gene_expression, train(x, y, method = "knn",  
                                         tuneGrid = data.frame( k = seq(1, 7, 2))))  
  
## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :  
## non-uniform 'Rounding' sampler used  
ggplot(fit)
```



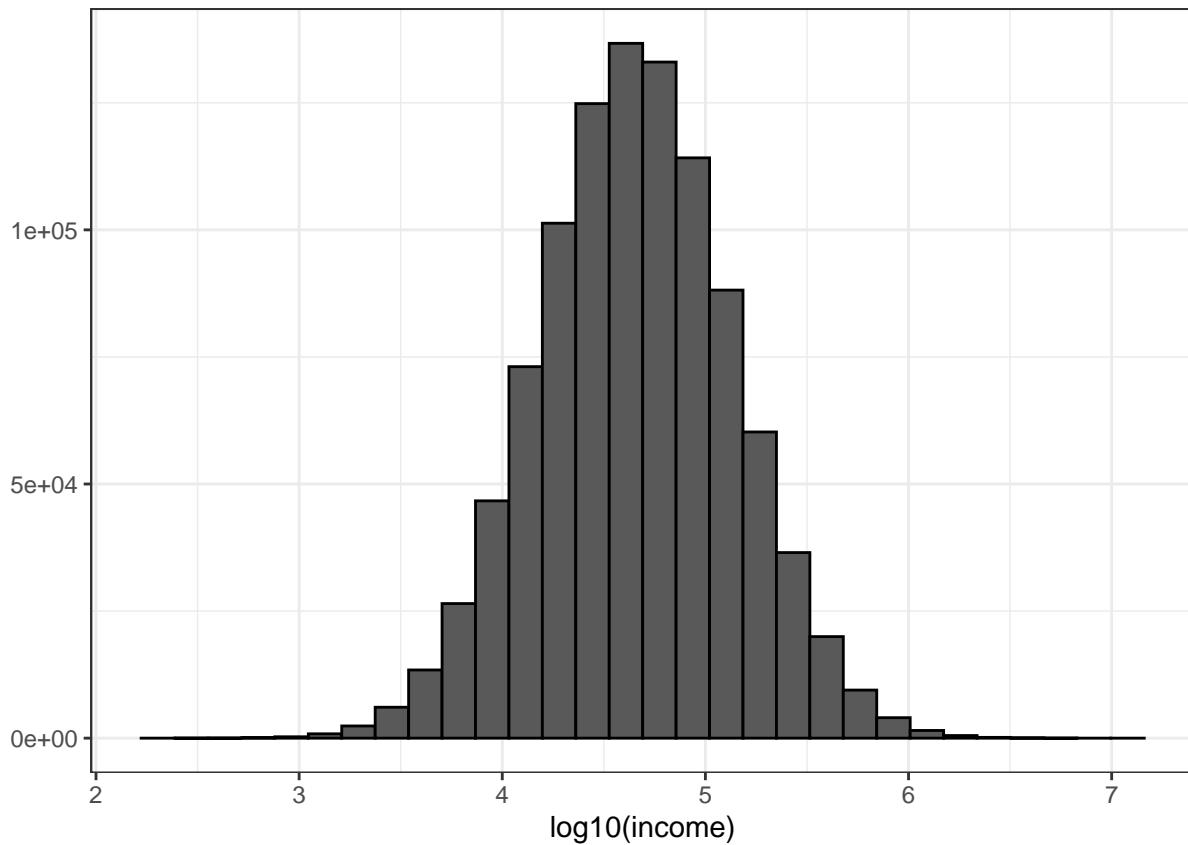
```
fit$results
```

```
##   k Accuracy Kappa AccuracySD KappaSD
## 1 1     0.993 0.992      0.0118  0.0145
## 2 3     0.978 0.973      0.0235  0.0288
## 3 5     0.977 0.971      0.0287  0.0352
## 4 7     0.971 0.965      0.0310  0.0380
```

**8.4.6.1. Bootstrap** Cuando no tenemos acceso a la población entera, podemos usar el método *bootstrap* para estimar la mediana  $m$ .

Como ejemplo, supóngase que la distribución del ingreso de una población es como sigue:

```
n <- 10^6
income <- rnorm(n, log10(45000), log10(3))
qplot(log10(income), bins = 30, color = I("black"))
```



Donde la mediana es:

```
m <- median(income)
```

```
m
```

```
## [1] 44958
```

Supóngase que no tenemos acceso a la población total y queremos estimar la mediana. Tomemos una muestra de 250:

```
set.seed(1, sample.kind = "Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
N <- 250
```

```
X <- sample(income, N)
```

```
M <- median(X)
```

```
M
```

```
## [1] 42481
```

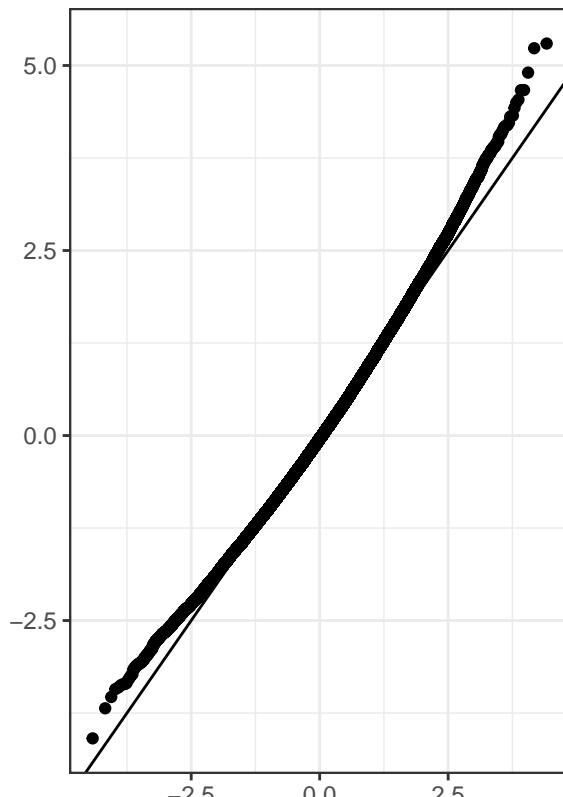
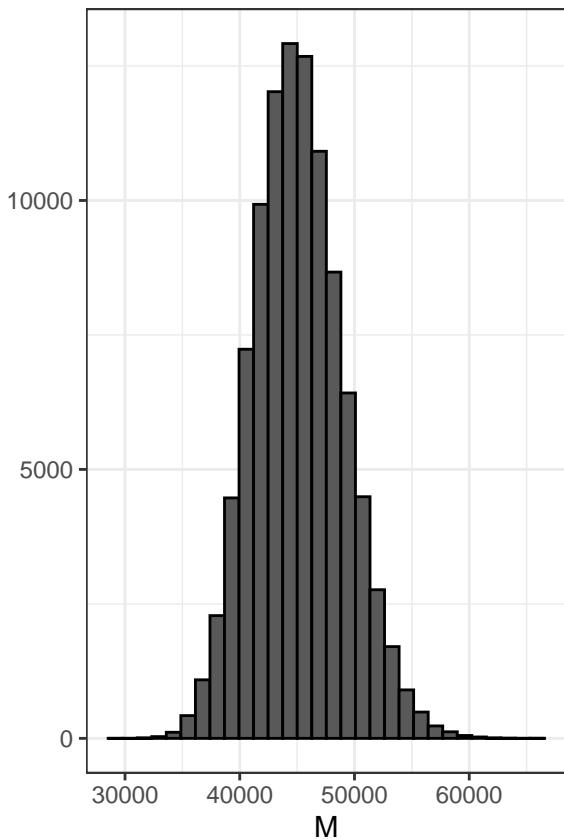
¿Puede construirse un intervalo de confianza? ¿Cuál es la distribución de la mediana muestral? A partir de una simulación Monte Carlo, vemos que esta distribución es aproximadamente normal con el siguiente valor esperado:

```
library(gridExtra)

B <- 10^5

M <- replicate(B, {
  X <- sample(income, N)
  median(X)
})

p1 <- qplot(M, bins = 30, color = I("black"))
p2 <- qplot(sample = scale(M)) + geom_abline()
grid.arrange(p1, p2, ncol = 2)
```



```
mean(M)
## [1] 45138
sd(M)
## [1] 3916
```

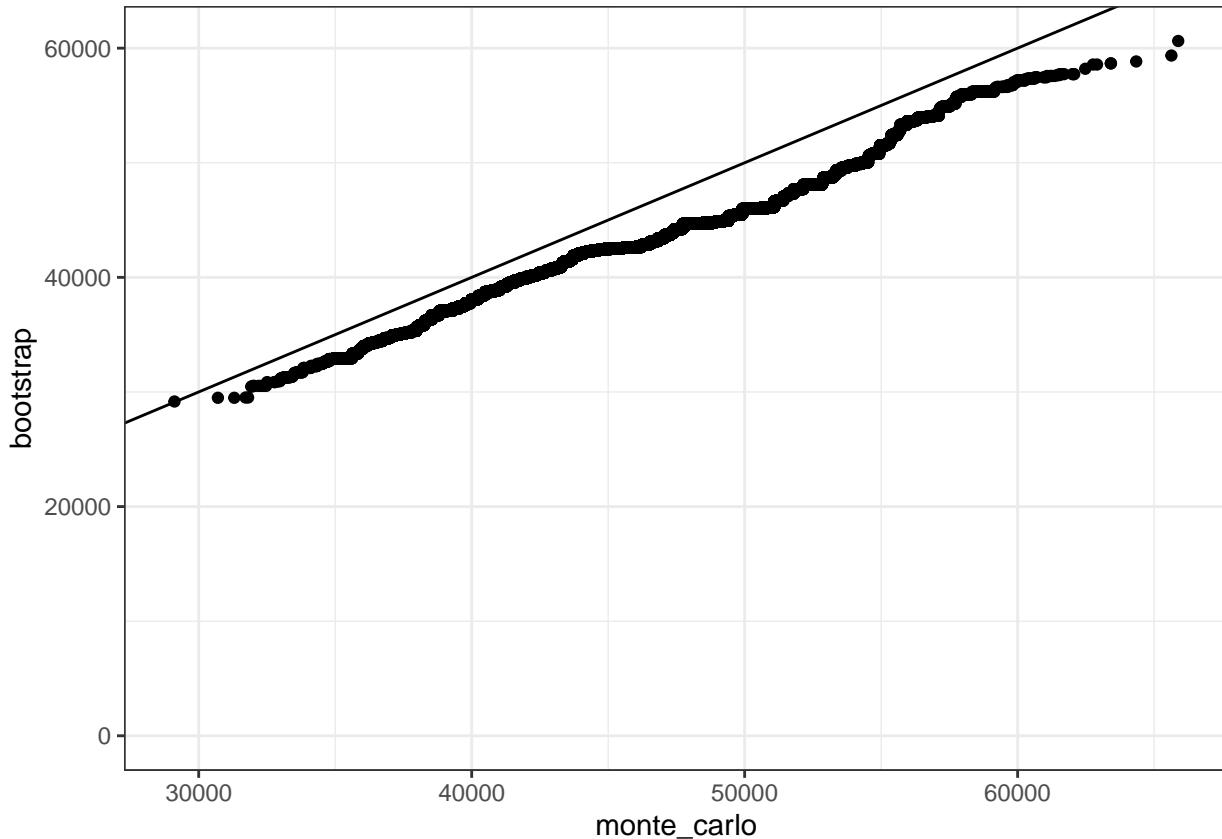
En el pasado, cuando no conocemos la distribución, hemos utilizado el TLC; sin embargo, este aplica para medias, y ahora nos interesa la mediana. Así, *bootstrap* nos permite aproximar una simulación Monte Carlo sin tener acceso a la distribución total: supondremos que la muestra es la población entera y realizaremos las muestras con reemplazo.

Así, calculamos las estadísticas descriptivas, llamadas muestra *bootstrap*:

```
B <- 10^5
M_star <- replicate(B, {
  X_star <- sample(X, N, replace = TRUE)
  median(X_star)
})
```

Ahora podemos revisar qué tan cerca estamos de la distribución real:

```
tibble(monte_carlo = sort(M), bootstrap = sort(M_star)) %>%
  qplot(monte_carlo, bootstrap, data = .) +
  geom_abline()
```



Obsérvese que, si bien no es perfecta, es una aproximación decente. En particular, véanse los cuantiles necesarios para formar un intervalo de confianza al 95 %:

```
quantile(M, c(0.05, 0.95))
```

```
##      5%    95%
## 39053 51864
```

```
quantile(M_star, c(0.05, 0.95))
```

```
##      5%    95%
## 37076 47697
```

Esto es un mejor resultado que si usáramos arbitrariamente el TLC:

```
median(X) + 1.96 * sd(X) / sqrt(N) * c(-1, 1)
```

```
## [1] 27441 57522
```

Si sabemos que la distribución es normal, podemos usar *bootstrap* para estimar la media, el error estándar y luego formar un intervalo de confianza:

```
mean(M) + 1.96 * sd(M) * c(-1,1)
## [1] 37462 52814
mean(M_star) + 1.96 * sd(M_star) * c(-1, 1)
## [1] 35972 48569
```

#### 8.4.7. Assessment 27

The `createResample()` function can be used to create bootstrap samples. For example, we can create the indexes for 10 bootstrap samples for the `mnist_27` dataset like this:

```
library(dslabs)
library(caret)
data(mnist_27)

set.seed(1995, sample.kind="Rounding")

## Warning in set.seed(1995, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
indexes <- createResample(mnist_27$train$y, 10)
```

\*How many times do 3, 4, and 7 appear in the first resampled index?\*\*

```
sum(indexes[[1]] == 3)
## [1] 1
sum(indexes[[1]] == 4)
## [1] 4
sum(indexes[[1]] == 7)
## [1] 0
```

We see that some numbers appear more than once and others appear no times. This has to be this way for each dataset to be independent. Repeat the exercise for all the resampled indexes. What is the total number of times that 3 appears in all of the resampled indexes?

```
x <- sapply(indexes, function(ind){
  sum(ind == 3)
})

sum(x)
```

```
## [1] 11
```

A random dataset can be generated with the following code:

```
y <- rnorm(100, 0, 1)
```

Estimate the 75th quantile, which we know is `qnorm(0.75)`, with the sample quantile: `quantile(y, 0.75)`. Now, set the seed to 1 and perform a Monte Carlo simulation with 10,000 repetitions, generating the random dataset and estimating the 75th quantile each time. What is the expected value and standard error of the 75th quantile? Report all answers to at least 3 decimal digits.

Expected value

```
set.seed(1, sample.kind = "Rounding")

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
B <- 10000

q_75 <- replicate(B, {
  y <- rnorm(100, 0, 1)
  quantile(y, 0.75)
})

mean(q_75)

## [1] 0.666

Standard error
sd(q_75)

## [1] 0.135
```

In practice, we can't run a Monte Carlo simulation. Use the sample:

```
set.seed(1, sample.kind = "Rounding")

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
y <- rnorm(100, 0, 1)
```

Set the seed to 1 again after generating y and use 10 bootstrap samples to estimate the expected value and standard error of the 75th quantile.

**Expected value**

```
set.seed(1, sample.kind = "Rounding")

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
indexes <- createResample(y, 10)

q_75_star <- sapply(indexes, function(ind){
  y_star <- y[ind]
  quantile(y_star, 0.75)
})

mean(q_75_star)

## [1] 0.731
```

**Standard error**

```
sd(q_75_star)
```

```
## [1] 0.0742
```

Repeat the exercise from Q4 but with 10,000 bootstrap samples instead of 10. Set the seed to 1 first.

\*\*\*\*Expected value and standard error\*\*

```
set.seed(1, sample.kind = "Rounding")

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used

indexes <- createResample(y, 10000)

q_75_star <- sapply(indexes, function(ind){
  y_star <- y[ind]
  quantile(y_star, 0.75)
})

mean(q_75_star)

## [1] 0.674
sd(q_75_star)

## [1] 0.0931
```

#### 8.4.8. Modelos generativos

Hasta ahora, hemos descrito cómo, al usar pérdida cuadrada, las probabilidades y esperanzas condicionales proveen el mejor enfoque para desarrollar una regla de decisión. En un caso binario, la mejor opción es usar la llamada **Regla de Bayes**, que es la regla de decisión basada en la probabilidad condicional real:

$$p(x) = \Pr(Y = 1|X = x)$$

Nótese que hemos estimado la probabilidad condicional directamente sin tomar en cuenta la distribución de los predictores; o sea, hemos seguido un enfoque discriminatorio.

No obstante, el Teorema de Bayes afirma que, conociendo la distribución de los predictores  $X$  puede ser útil. Los modelos que modelan una distribución conjunta de  $Y$  y los predictores  $X$  se conocen como **modelos generativos**.

Comenzaremos con el más sencillo de ellos, *Naive Bayes*. Recuérdese que el Teorema de Bayes demuestra que podemos escribir una probabilidad condicional de la siguiente manera:

$$p(x) = \Pr(Y = 1|X = x) = \frac{f_{X|Y=1}(x)\Pr(Y = 1)}{f_{X|Y=0}(x)\Pr(Y = 0) + f_{X|Y=1}(x)\Pr(Y = 1)}$$

Donde  $f$  son las funciones de distribución de los predictores para las dos clases,  $Y = 1$  y  $Y = 0$ . La fórmula anterior implica que si podemos estimar estas probabilidades condicionales, los predictores, podemos desarrollar un conjunto de decisiones robusto. El reto mayor está en estimar estas probabilidades condicionales.

Comenzaremos con un ejemplo simple pero ilustrativo: predecir el género a partir de la altura. Obtengamos los datos y generemos los conjuntos de práctica y prueba como sigue:

```
library(caret)
data("heights")

y <- heights$height

set.seed(2, sample.kind = "Rounding")

## Warning in set.seed(2, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used

test_index <- createDataPartition(y, times = 1, p = 0.5, list = FALSE)

train_set <- heights %>%
  slice(-test_index)

test_set <- heights %>%
  slice(test_index)
```

En este caso, *naive Bayes* es particularmente apropiado, dado que sabemos que la distribución normal es una buena aproximación de las distribuciones condicionales de la altura, dado el sexo, para hombres y mujeres. Esto implica que podemos aproximar las distribuciones condicionales  $f_{X|Y=1}$  y  $f_{X|Y=0}$  simplemente con una estimación de sus medias y desviaciones estándar a partir de nuestros datos:

```
params <- train_set %>%
  group_by(sex) %>%
  summarize(avg = mean(height), sd = sd(height))

params
```

```
## # A tibble: 2 x 3
##   sex     avg     sd
##   <fct>  <dbl>  <dbl>
## 1 Female  64.5  4.02
## 2 Male    69.3  3.52
```

La prevalencia, sea  $\pi = Pr(Y = 1)$  (proporción de mujeres) también puede ser estimada de los datos:

```
pi <- train_set %>%
  summarize(pi = mean(sex == "Female")) %>%
  .$pi

pi
```

```
## [1] 0.229
```

Así, con nuestros estimadores de media y desviación estándar podemos derivar la regla de decisión real:

```
x <- test_set$height

f0 <- dnorm(x, params$avg[2], params$sd[2])

f1 <- dnorm(x, params$avg[1], params$sd[1])

p_hat_bayes <- f1*pi / (f1*pi + f0*(1-pi))
```

Puede mostrarse matemáticamente que esta estimación de *naive Bayes* es sumamente similar a una regresión logística para este caso particular.

**8.4.8.1. Controlando prevalencia** El enfoque de *naive Bayes* toma en cuenta un parámetro para incluir las diferencias en prevalencia  $\pi = Pr(Y = 1)$ . Utilizando nuestra muestra, estimamos las probabilidades condicionales y la prevalencia.

Utilizando gorros para denotar a los estimadores, podemos reescribir la estimación de la probabilidad condicional como sigue:

$$\hat{p}(x) = Pr(Y = 1|X = x) = \frac{\hat{f}_{X|Y=1}(x)\hat{\pi}}{\hat{f}_{X|Y=0}(x)(1 - \hat{\pi}) + \hat{f}_{X|Y=1}(x)\hat{\pi}}$$

Como discutimos, nuestra muestra tiene una prevalencia mucho menor que la población general. Así, si usamos la regla de que la probabilidad condicional tiene que ser mayor a 0.5 para predecir “mujeres”, nuestra precisión se verá afectada por la baja sensibilidad:

```
y_hat_bayes <- ifelse(p_hat_bayes > 0.5, "Female", "Male")

sensitivity(data = factor(y_hat_bayes), reference = factor(test_set$sex))

## [1] 0.263
```

Esto ocurre porque el algoritmo pondera con mayor fuerza a la especificidad, dada la baja prevalencia:

```
specificity(data = factor(y_hat_bayes), reference = factor(test_set$sex))

## [1] 0.953
```

Por tanto, si quisieramos extrapolar esto a la población general, la precisión general se vería muy afectada por la baja sensibilidad. *Naive Bayes* nos proporciona una forma directa para corregir esto, dado que podemos forzar que  $\hat{\pi}$  sea diferente.

Así, para balancear sensibilidad y especificidad, y en lugar de cambiar el *cutoff* de la regla de decisión, modificaremos  $\hat{\pi} = 0.5$ :

```
p_hat_bayes_unbiased <- f1*0.5 / (f1*0.5 + f0*(1-0.5))

y_hat_bayes_unbiased <- ifelse(p_hat_bayes_unbiased > 0.5, "Female", "Male")
```

Nótese la diferencia en la sensibilidad y especificidad:

```
sensitivity(data = factor(y_hat_bayes_unbiased), reference = factor(test_set$sex))

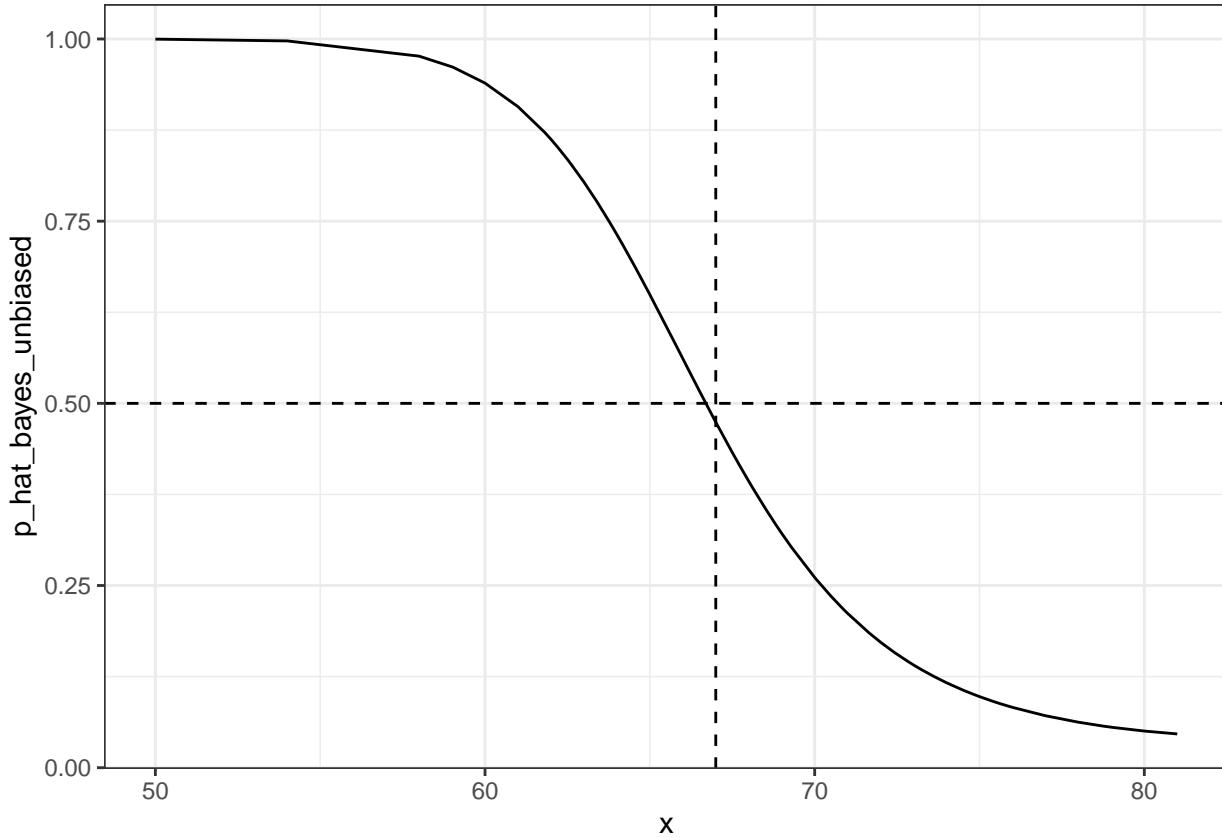
## [1] 0.712

specificity(data = factor(y_hat_bayes_unbiased), reference = factor(test_set$sex))

## [1] 0.821
```

El siguiente gráfico muestra que la nueva regla nos da un *cutoff* muy intuitivo entre 66 y 67:

```
qplot(x, p_hat_bayes_unbiased, geom = "line") +
  geom_hline(yintercept = 0.5, lty = 2) +
  geom_vline(xintercept = 67, lty = 2)
```



**8.4.8.2. qda y lda** El **Análisis Discriminatorio Cuadrático (QDA)** es una versión de *naive Bayes* en donde asumimos que las probabilidades condicionales para los predictores son normales multitudinarias.

Como ejemplo, retómese el caso de predicción de dígitos con 2s y 7s:

```
data("mnist_27")
```

En este caso, tenemos dos predictores, por lo que asumiremos que la distribución condicional es normal bivariada. Por tanto, tendremos que estimar dos medias, dos desviaciones estándar y dos correlaciones para cada caso. Una vez obtenido esto, podemos aproximar las distribuciones condicionales  $f_{X_1, X_2|Y=1}$  y  $f_{X_1, X_2|Y=0}$ :

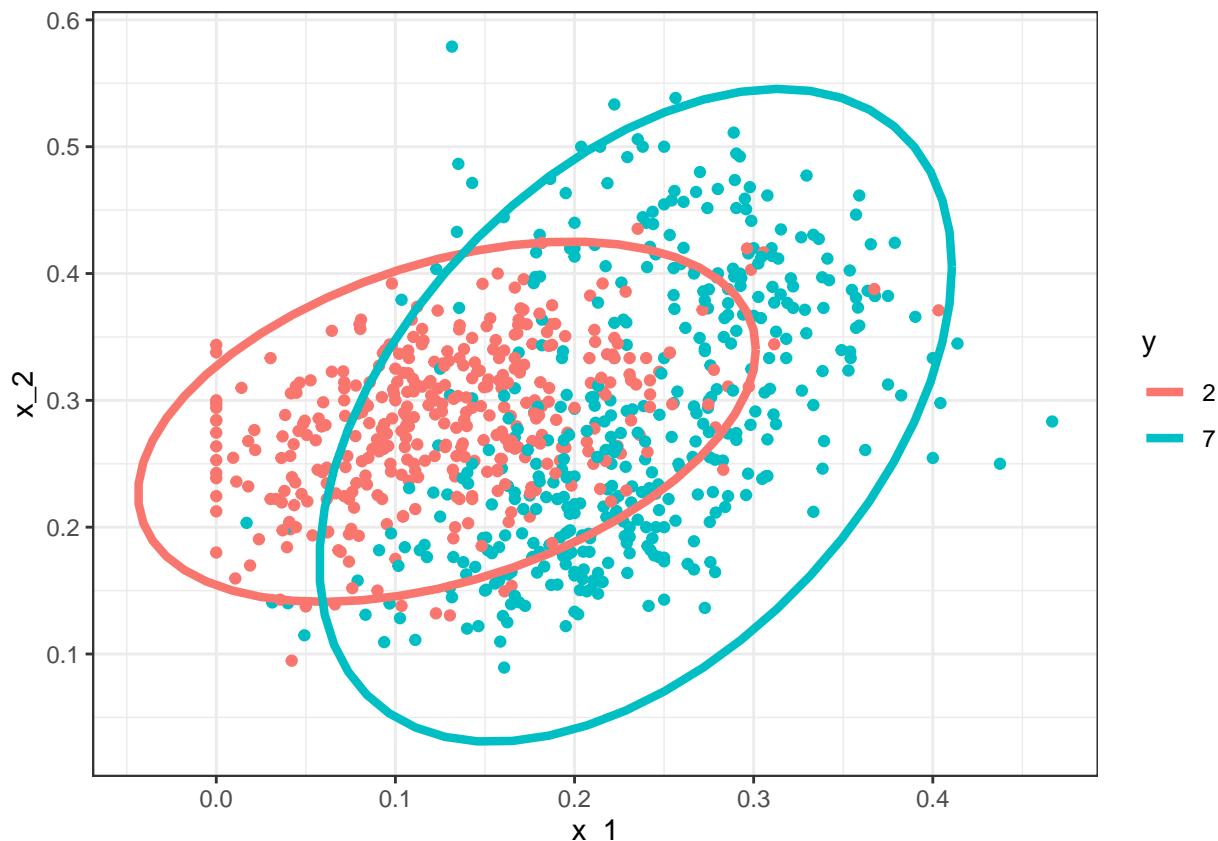
```
params <- mnist_27$train %>%
  group_by(y) %>%
  summarize(avg_1 = mean(x_1), avg_2 = mean(x_2),
            sd_1 = sd(x_1), sd_2 = sd(x_2), r = cor(x_1, x_2))

params

## # A tibble: 2 x 6
##   y     avg_1   avg_2   sd_1   sd_2      r
##   <fct> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 2     0.129  0.283  0.0702 0.0578  0.401
## 2 7     0.234  0.288  0.0719 0.105   0.455
```

De manera visual, podemos ver cómo lucen las densidades normales estimadas (las regiones circuladas incluyen al 95 % de los puntos):

```
mnist_27$train %>%
  mutate(y = factor(y)) %>%
  ggplot(aes(x_1, x_2, fill = y, color = y)) +
  geom_point(show.legend = FALSE) +
  stat_ellipse(type="norm", lwd = 1.5)
```



Podemos estimar el modelo mediante el paquete caret:

```
library(caret)

train_qda <- train(y ~ .,
                     method = "qda",
                     data = mnist_27$train)

## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :
## non-uniform 'Rounding' sampler used

Obsérvese que obtenemos una precisión relativamente alta:
```

```
y_hat <- predict(train_qda, mnist_27$test)

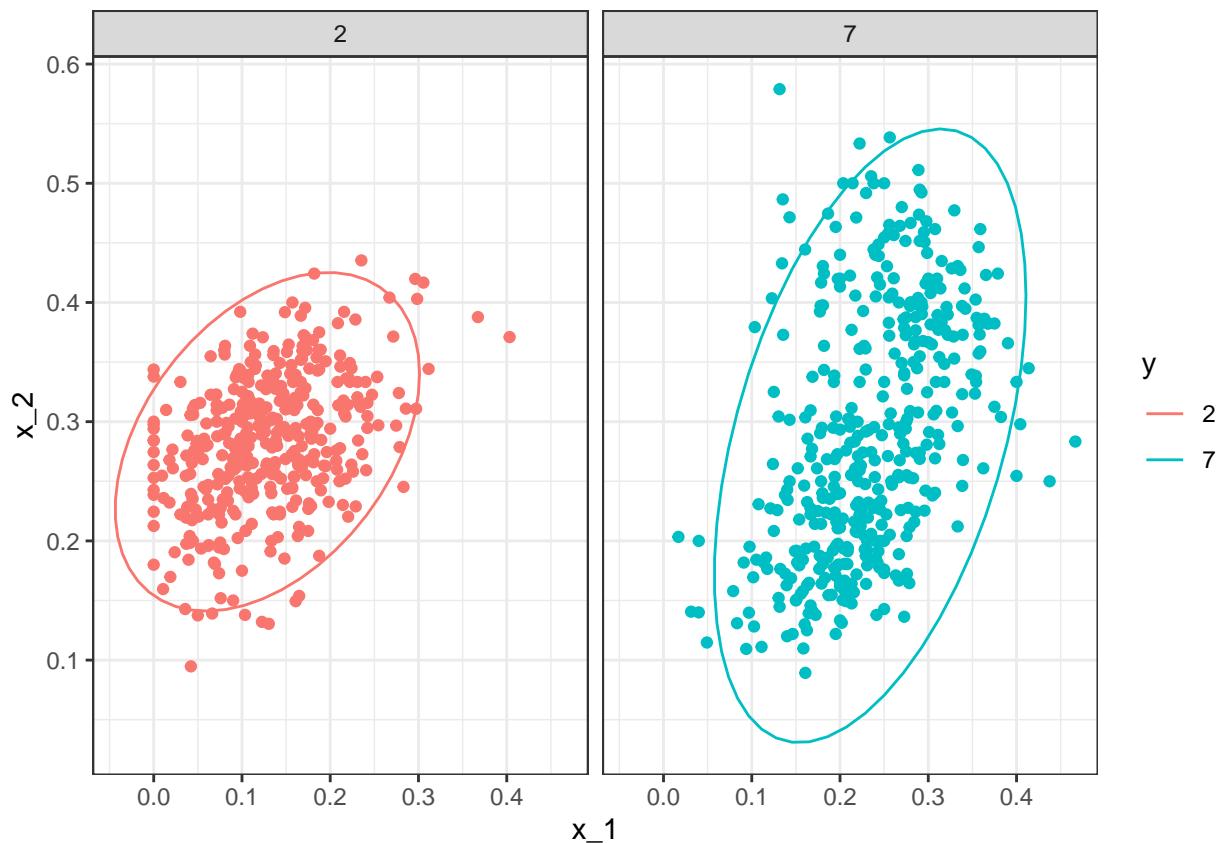
confusionMatrix(data = y_hat, reference = mnist_27$test$y)$overall["Accuracy"]

## Accuracy
##      0.82
```

Sin embargo, la estimación no es tan buena como la realizada con kernels: esto se debe a que puede mostrarse matemáticamente que el límite debe ser una función cuadrática de la forma  $x_2 = ax_1^2 + bx_1 + c$ . Otra razón por la que QDA no funciona también como con kernel es que los supuestos de normalidad no se cumplan adecuadamente.

A pesar de que para el caso de los 2s, una aproximación normal bivariada parece razonable, no es el caso para los 7s (nótese una ligera curvatura):

```
mnist_27$train %>% mutate(y = factor(y)) %>%
  ggplot(aes(x_1, x_2, fill = y, color = y)) +
  geom_point(show.legend = FALSE) +
  stat_ellipse(type="norm") +
  facet_wrap(~y)
```



A medida que el número de predictores se incrementa, QDA es exponencialmente más difícil de usar. Si tuviéramos 10 predictores, deberíamos computar 45 correlaciones para cada clase. En general, la siguiente fórmula nos dice cuántos parámetros tendremos que estimar:

$$K * (2p + p * (p - 1)/2)$$

Una solución relativamente simple para el problema de tener muchos parámetros es asumir que la estructura de la correlación es la misma para todas las clases, lo cual reduce el número de parámetros a estimar. Aplicando esto al ejemplo anterior:

```
params <- mnist_27$train %>%
  group_by(y) %>%
  summarize(avg_1 = mean(x_1), avg_2 = mean(x_2),
            sd_1 = sd(x_1), sd_2 = sd(x_2), r = cor(x_1, x_2))

params <- params %>%
  mutate(sd_1 = mean(sd_1), sd_2 = mean(sd_2),
        r=mean(r))

params

## # A tibble: 2 x 6
##   y     avg_1 avg_2   sd_1   sd_2      r
##   <fct> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 2      0.129  0.283  0.0710 0.0813 0.428
## 2 7      0.234  0.288  0.0710 0.0813 0.428
```

Dado que suponemos que las estimaciones tienen las mismas desviaciones estándar y correlaciones, podemos mostrar matemáticamente que el límite está alineado, como ocurre con la regresión logística. Por tal motivo, llamamos a este método **Análisis Discriminatorio Lineal (LDA)**. En este ejemplo, la falta de flexibilidad no nos permite obtener un buen estimador, como puede verse:

```
train_lda <- train(y ~., method = "lda", data = mnist_27$train)

## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :
## non-uniform 'Rounding' sampler used

y_hat <- predict(train_lda, mnist_27$test)

confusionMatrix(data = y_hat, reference = mnist_27$test$y)$overall["Accuracy"]

## Accuracy
##      0.75
```

**8.4.8.3. Más de tres clases** A continuación, revisaremos un ejemplo más complicado, uno con tres clases en vez de dos. Considerese el ejemplo de detección de dígitos, pero ahora con 1s, 2s y 7s:

```
if(!exists("mnist"))mnist <- read_mnist()

set.seed(3456, sample.kind="Rounding")

## Warning in set.seed(3456, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

El código para generar nuestros datos es bastante complejo y se muestra a continuación:

```
index_127 <- sample(which(mnist$train$labels %in% c(1,2,7)), 2000)

y <- mnist$train$labels[index_127]

x <- mnist$train$images[index_127,]

index_train <- createDataPartition(y, p=0.8, list = FALSE)

# obtener los cuadrantes

# objeto temporal para ayudar a la creación de cuadrantes

row_column <- expand.grid(row=1:28, col=1:28)
upper_left_ind <- which(row_column$col <= 14 & row_column$row <= 14)
lower_right_ind <- which(row_column$col > 14 & row_column$row > 14)

# binarizar los valores. Arriba de 200 hay tinta, abajo de 200 no hay tinta

x <- x > 200

# cbind proporción de pixeles en el cuadrante superior derecho y en el inferior izquierdo

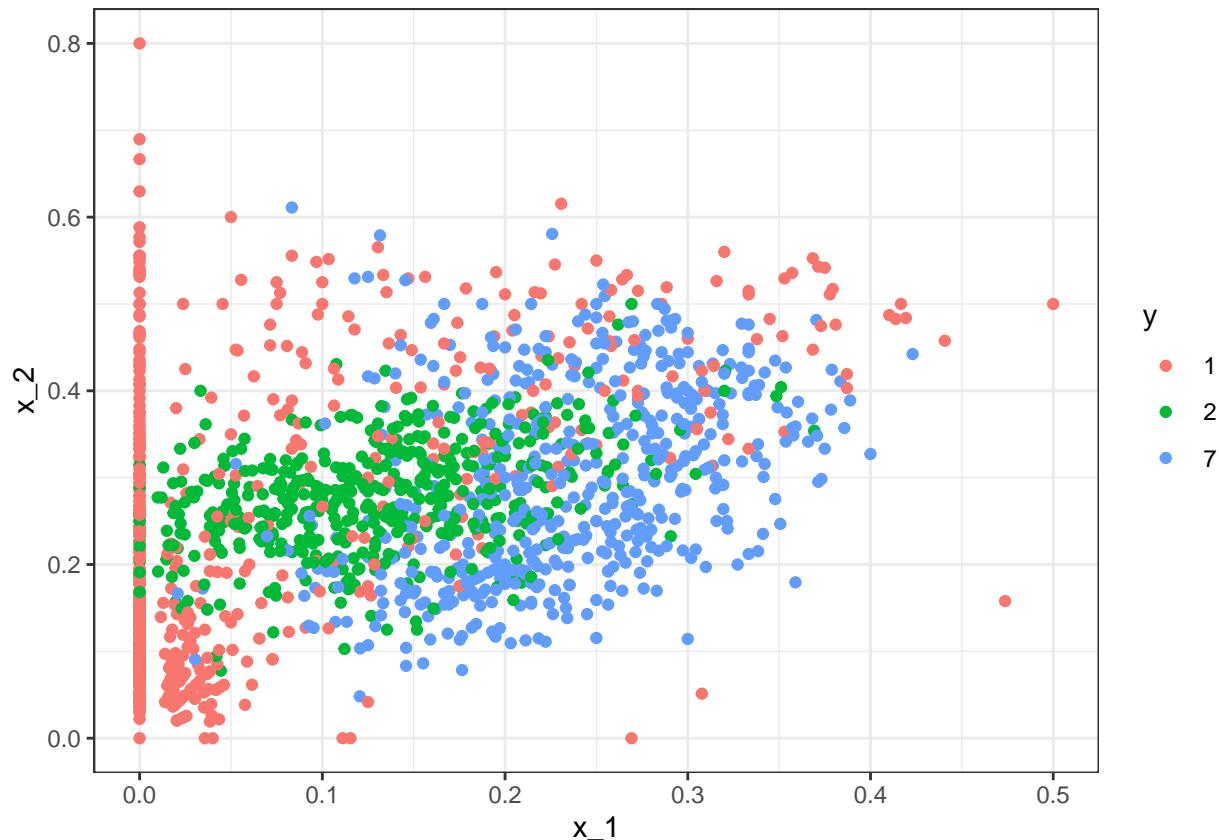
x <- cbind(rowSums(x[,upper_left_ind])/rowSums(x),
            rowSums(x[,lower_right_ind])/rowSums(x))

train_set <- data.frame(y = factor(y[index_train]),
                         x_1 = x[index_train,1],
                         x_2 = x[index_train,2])
```

```
test_set <- data.frame(y = factor(y[-index_train]),
                        x_1 = x[-index_train,1],
                        x_2 = x[-index_train,2])
```

A continuación se muestran los datos del conjunto de práctica:

```
train_set %>%
  ggplot(aes(x_1, x_2, color=y)) +
  geom_point()
```



Como ejemplo, ajustaremos un modelo QDA:

```
train_qda <- train(y ~.,
                     method = "qda",
                     data = train_set)
```

```
## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :
## non-uniform 'Rounding' sampler used
```

Entonces, ¿cómo varían las cosas? Primero, nótese que estamos estimando tres probabilidades condicionales. Así, con “predict()” de tipo “prob”, obtenemos una matriz de tres columnas: probabilidades para los 1s, 2s y 7s para cada predicción (renglones):

```
predict(train_qda, test_set, type = "prob") %>%
  head()

##      1     2     7
## 1 0.2223 0.660 0.1180
```

```
## 2 0.1926 0.454 0.3539
## 3 0.6275 0.322 0.0505
## 4 0.0462 0.101 0.8529
## 5 0.2167 0.623 0.1604
## 6 0.1267 0.335 0.5383
```

Así, si usamos “predict()” con su configuración por default, obtenemos las predicciones *per se*:

```
predict(train_qda, test_set)
```

```
## [1] 2 2 1 7 2 7 2 1 7 1 7 7 2 7 7 1 1 7 1 7 7 1 7 7 7 7 1 1 1 2 1 7 2 7 1 7 2
## [38] 7 2 7 7 7 7 1 7 7 1 2 2 7 1 7 2 2 1 7 7 2 2 1 7 1 1 1 2 2 1 1 1 2 7 1 7 7
## [75] 2 1 7 1 7 2 7 1 2 2 1 2 2 2 1 1 1 7 1 2 1 7 2 2 2 7 1 1 1 7 1 7 7 7 2 7 1
## [112] 7 7 2 1 2 1 1 2 1 2 2 1 2 1 7 1 1 2 1 7 2 7 1 2 1 2 1 2 1 1 7 2 2 7 1 1
## [149] 2 2 2 7 7 1 7 1 7 2 1 2 7 7 1 7 7 7 1 7 7 2 2 1 2 2 1 2 2 2 1 2 7 2 7 7
## [186] 7 1 7 7 2 2 1 2 1 2 7 1 7 7 1 1 1 7 1 1 7 2 1 7 7 2 2 1 7 2 2 2 2 7 2 2 1
## [223] 2 2 1 1 1 7 1 7 1 7 7 7 2 7 1 1 2 2 1 7 1 7 7 7 1 1 7 1 1 7 2 1 2 1 2 1 7
## [260] 1 7 1 2 7 7 7 7 7 2 1 7 1 7 2 1 7 7 1 7 7 7 2 7 1 2 7 2 2 7 2 2 7 2 1 2 1
## [297] 1 1 7 1 1 7 7 1 7 2 1 7 7 7 7 1 2 2 7 7 1 1 7 1 2 1 2 1 7 7 1 1 1 7 1 2 7
## [334] 7 1 1 7 2 7 7 7 1 7 7 7 7 7 2 2 1 7 7 2 1 2 2 7 7 1 7 7 1 1 7 7 1 1 1 2 1
## [371] 7 2 7 2 7 1 2 2 1 1 7 2 7 2 1 2 7 7 7 2 7 1 1 7 1 7 2 7 7
## Levels: 1 2 7
```

Nuestra matriz de confusión es una tabla de 3x3, dado que podemos tener dos tipos de errores para cada clase:

```
confusionMatrix(predict(train_qda, test_set), test_set$y)
```

```
## Confusion Matrix and Statistics
##
##          Reference
## Prediction 1 2 7
##           1 111 17 7
##           2 14 80 17
##           7 19 25 109
##
## Overall Statistics
##
##          Accuracy : 0.752
##             95% CI : (0.706, 0.794)
##    No Information Rate : 0.361
##    P-Value [Acc > NIR] : <2e-16
##
##          Kappa : 0.627
##
## McNemar's Test P-Value : 0.0615
##
## Statistics by Class:
##
##          Class: 1 Class: 2 Class: 7
## Sensitivity          0.771   0.656   0.820
## Specificity          0.906   0.888   0.835
## Pos Pred Value       0.822   0.721   0.712
## Neg Pred Value       0.875   0.854   0.902
## Prevalence           0.361   0.306   0.333
## Detection Rate       0.278   0.201   0.273
```

```
## Detection Prevalence    0.338    0.278    0.383
## Balanced Accuracy      0.838    0.772    0.827
```

Nótese que para sensibilidad y especificidad tenemos un par de valores para cada clase; esto se debe que para definir estos términos, necesitamos un resultado binario.

Para el caso de LDA, ajustemos el modelo como se muestra enseguida:

```
train_lda <- train(y ~ .,
                     method = "lda",
                     data = train_set)

## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :
## non-uniform 'Rounding' sampler used
confusionMatrix(predict(train_lda, test_set), test_set$y)$overall[["Accuracy"]]

## Accuracy
##     0.664
```

Donde la precisión es sustancialmente menor, principalmente debido a que nuestras regiones límites ahora son delimitadas por líneas y no curvas.

Los resultados para knn son, de hecho, mucho mejores

```
train_knn <- train(y ~ .,
                     method = "knn",
                     tuneGrid = data.frame(k = seq(15, 51, 2)),
                     data = train_set)

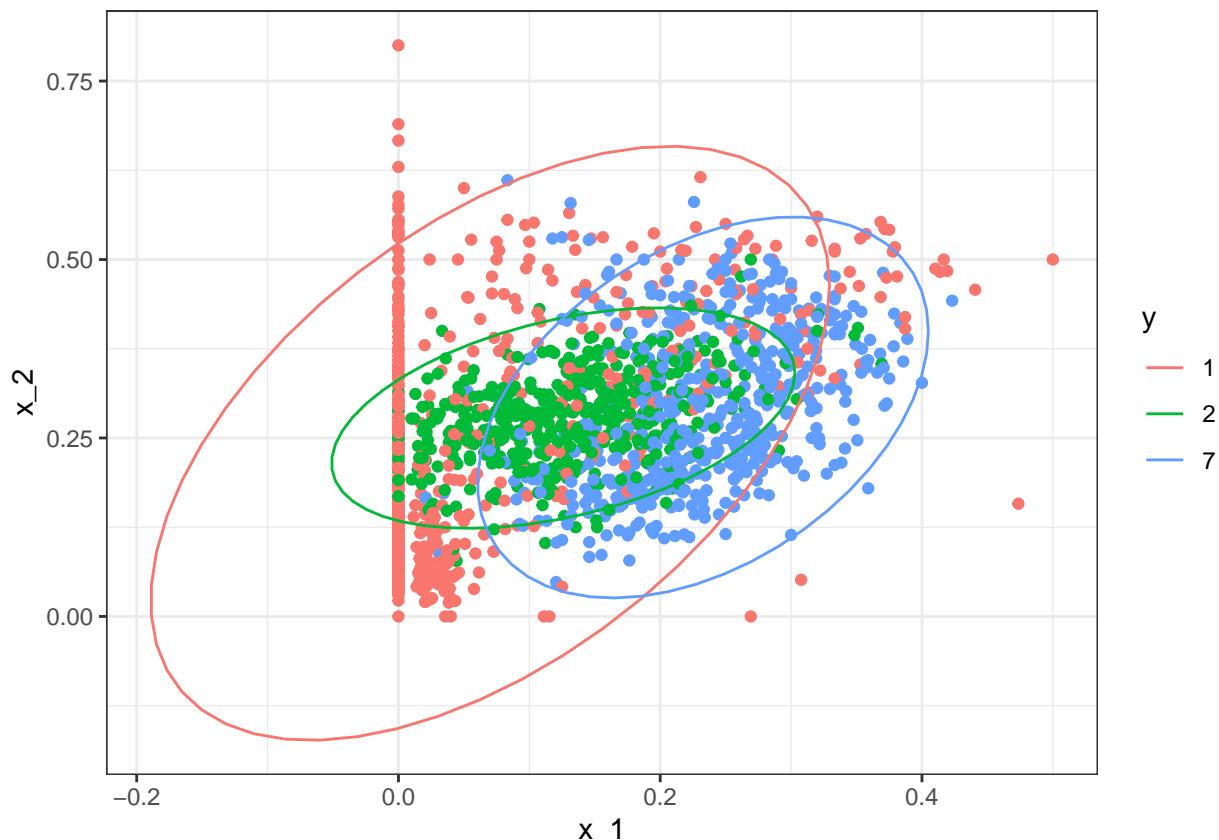
## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :
## non-uniform 'Rounding' sampler used
confusionMatrix(predict(train_knn, test_set), test_set$y)$overall[["Accuracy"]]

## Accuracy
##     0.772
```

Como hemos visto, lo anterior sucede porque las vecindades que se definen permiten mucho mayor flexibilidad para la estimación de las probabilidades condicionales.

Así, QDA y LDA no funcionan tan bien debido a la falta de ajuste de estos modelos. Podemos ver esto graficando los datos y notando que, al menos los 1s, definitivamente no son normalmente distribuidos bivariados:

```
train_set %>%
  mutate(y = factor(y)) %>%
  ggplot(aes(x_1, x_2, fill = y, color=y)) +
  geom_point(show.legend = FALSE) + stat_ellipse(type="norm")
```



Así, en resumen, los modelos generativos pueden ser sumamente poderosos, pero solo si podemos aproximar exitosamente la distribución conjunta de los predictores para cada clase.

#### 8.4.9. Assessment 28

In the following exercises, we are going to apply LDA and QDA to the tissue\_gene\_expression dataset from dslabs. We will start with simple examples based on this dataset and then develop a realistic example.

Create a dataset of samples from just cerebellum and hippocampus, two parts of the brain, and a predictor matrix with 10 randomly selected columns using the following code:

```
library(dslabs)
library(caret)
library(tidyverse)
data("tissue_gene_expression")

set.seed(1993, sample.kind="Rounding")

## Warning in set.seed(1993, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used

ind <- which(tissue_gene_expression$y %in% c("cerebellum", "hippocampus"))

y <- droplevels(tissue_gene_expression$y[ind])

x <- tissue_gene_expression$x[ind, ]

x <- x[, sample(ncol(x), 10)]
```

Use the train() function to estimate the accuracy of LDA. For this question, use the version of x and y created with the code above: do not split them or tissue\_gene\_expression into training and test sets (understand this can lead to overfitting). Report the accuracy from the train() results (do not make predictions). What is the accuracy? Enter your answer as a percentage or decimal (eg “50 %” or “0.50”) to at least the thousandths place.

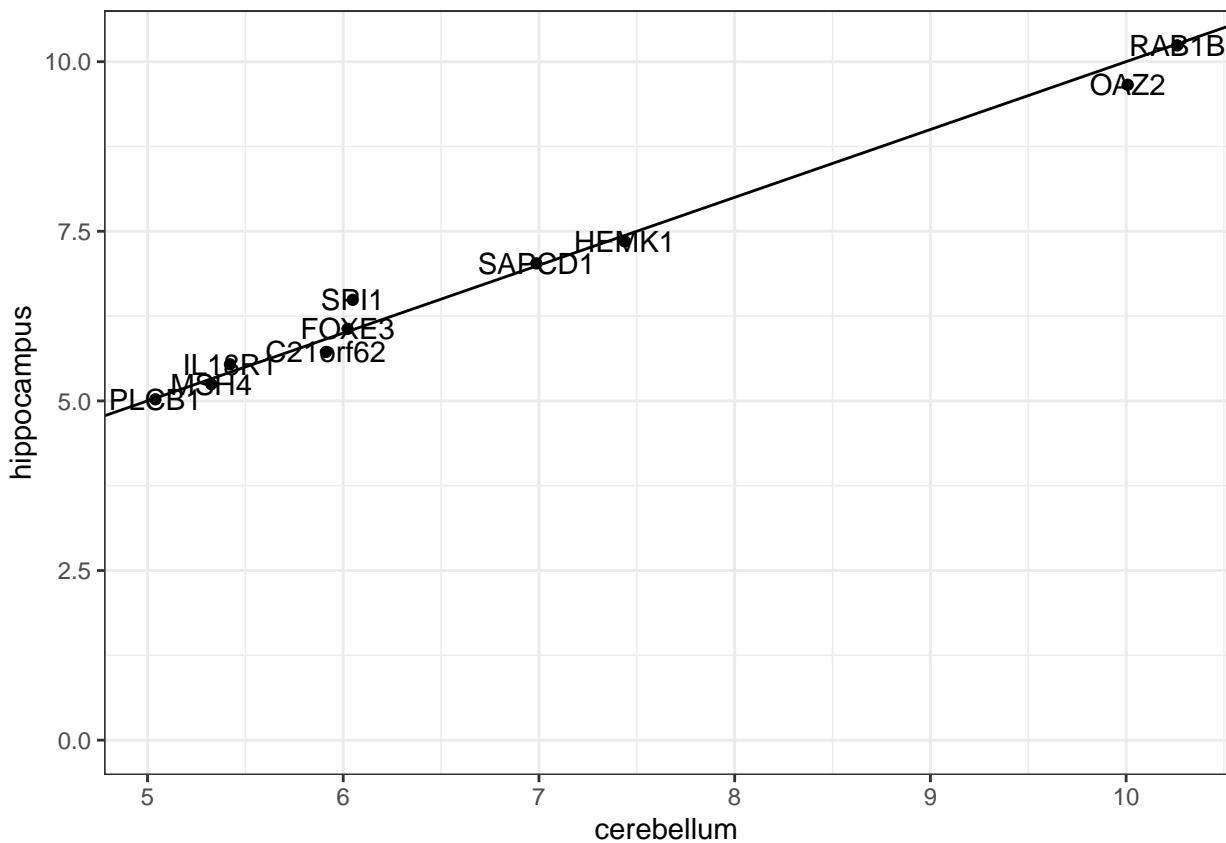
```
fit_lda <- train(x, y, method = "lda")

## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :
## non-uniform 'Rounding' sampler used
fit_lda$results["Accuracy"]

##   Accuracy
## 1     0.867
```

In this case, LDA fits two 10-dimensional normal distributions. Look at the fitted model by looking at the finalModel component of the result of train(). Notice there is a component called means that includes the estimated means of both distributions. Plot the mean vectors against each other and determine which predictors (genes) appear to be driving the algorithm. Which TWO genes appear to be driving the algorithm (i.e. the two genes with the highest means)?

```
t(fit_lda$finalModel$means) %>%
  data.frame() %>%
  mutate(predictor_name = rownames(.)) %>%
  ggplot(aes(cerebellum, hippocampus, label = predictor_name)) +
  geom_point() +
  geom_text() +
  geom_abline()
```



Repeat the exercise in Q1 with QDA. Create a dataset of samples from just cerebellum and hippocampus, two parts of the brain, and a predictor matrix with 10 randomly selected columns using the following code:

```
library(dslabs)
library(caret)
data("tissue_gene_expression")

set.seed(1993, sample.kind="Rounding")

## Warning in set.seed(1993, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
ind <- which(tissue_gene_expression$y %in% c("cerebellum", "hippocampus"))

y <- droplevels(tissue_gene_expression$y[ind])

x <- tissue_gene_expression$x[ind, ]

x <- x[, sample(ncol(x), 10)]
```

Use the `train()` function to estimate the accuracy of QDA. For this question, use the version of `x` and `y` created above instead of the default from `tissue_gene_expression`. Do not split them into training and test sets (understand this can lead to overfitting). What is the accuracy?

```
fit_qda <- train(x, y, method = "qda")

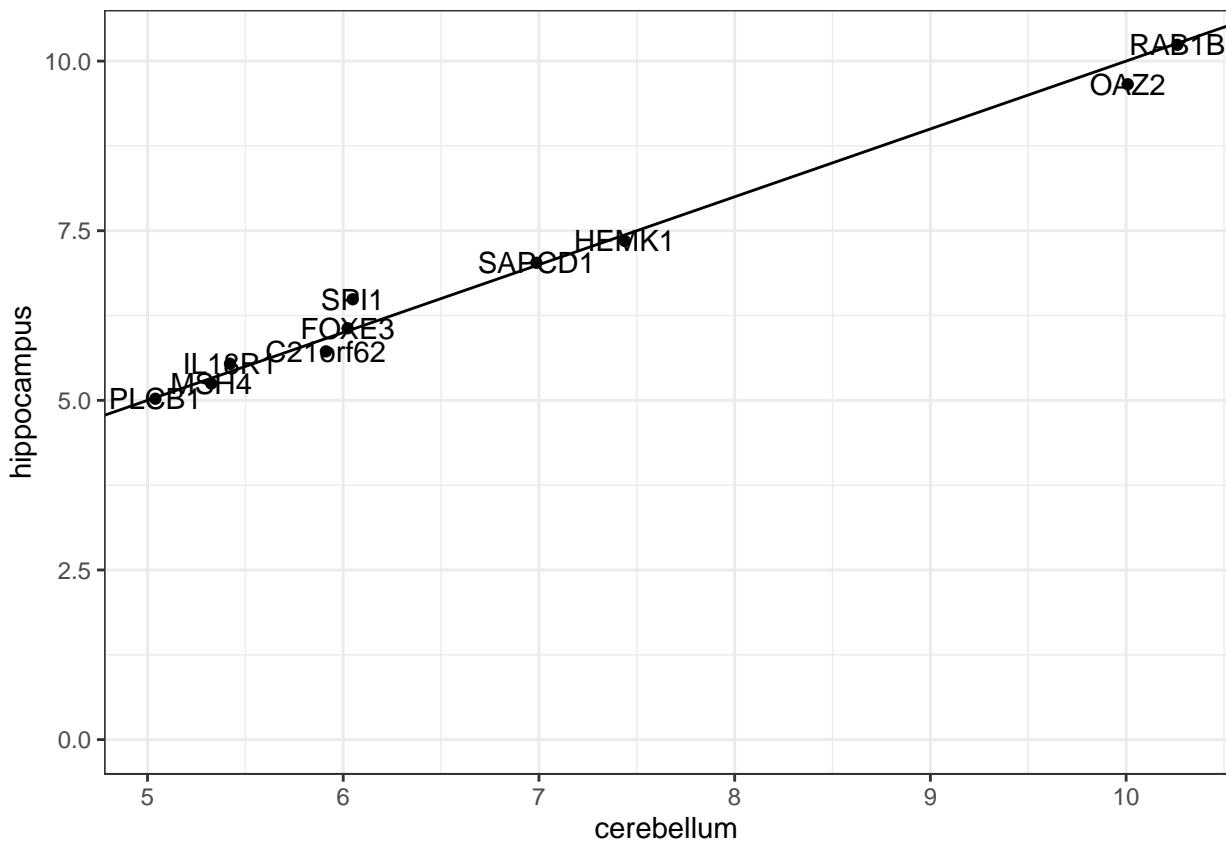
## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :
```

```
## non-uniform 'Rounding' sampler used
fit_qda$results["Accuracy"]
```

```
## Accuracy
## 1 0.805
```

Which TWO genes drive the algorithm when using QDA instead of LDA (i.e. the two genes with the highest means)?

```
t(fit_qda$finalModel$means) %>% data.frame() %>%
  mutate(predictor_name = rownames(.)) %>%
  ggplot(aes(cerebellum, hippocampus, label = predictor_name)) +
  geom_point() +
  geom_text() +
  geom_abline()
```



One thing we saw in the previous plots is that the values of the predictors correlate in both groups: some predictors are low in both groups and others high in both groups. The mean value of each predictor found in colMeans(x) is not informative or useful for prediction and often for purposes of interpretation, it is useful to center or scale each column. This can be achieved with the preProcess argument in train(). Re-run LDA with preProcess = "center". Note that accuracy does not change, but it is now easier to identify the predictors that differ more between groups than based on the plot made in Q2. Which TWO genes drive the algorithm after performing the scaling?

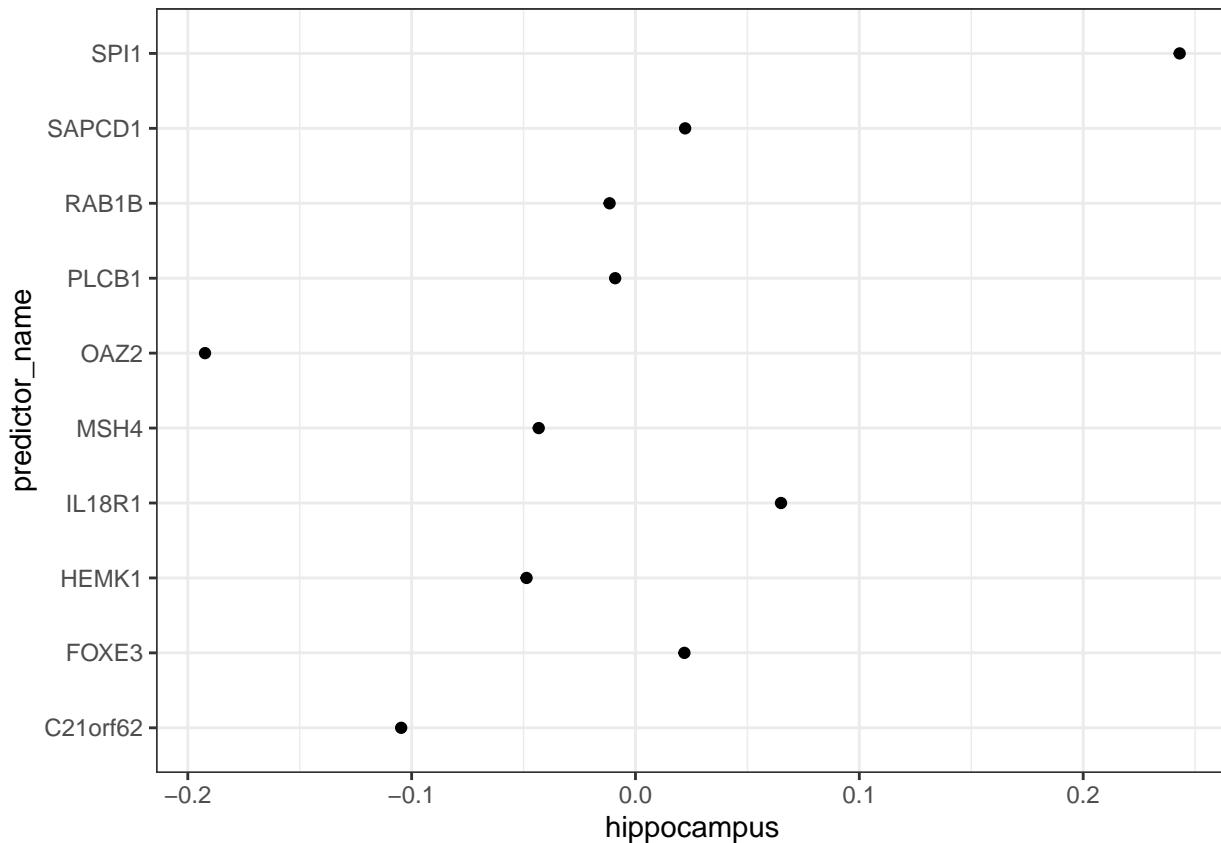
```
fit_lda <- train(x, y, method = "lda", preProcess = "center")
```

```
## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :
```

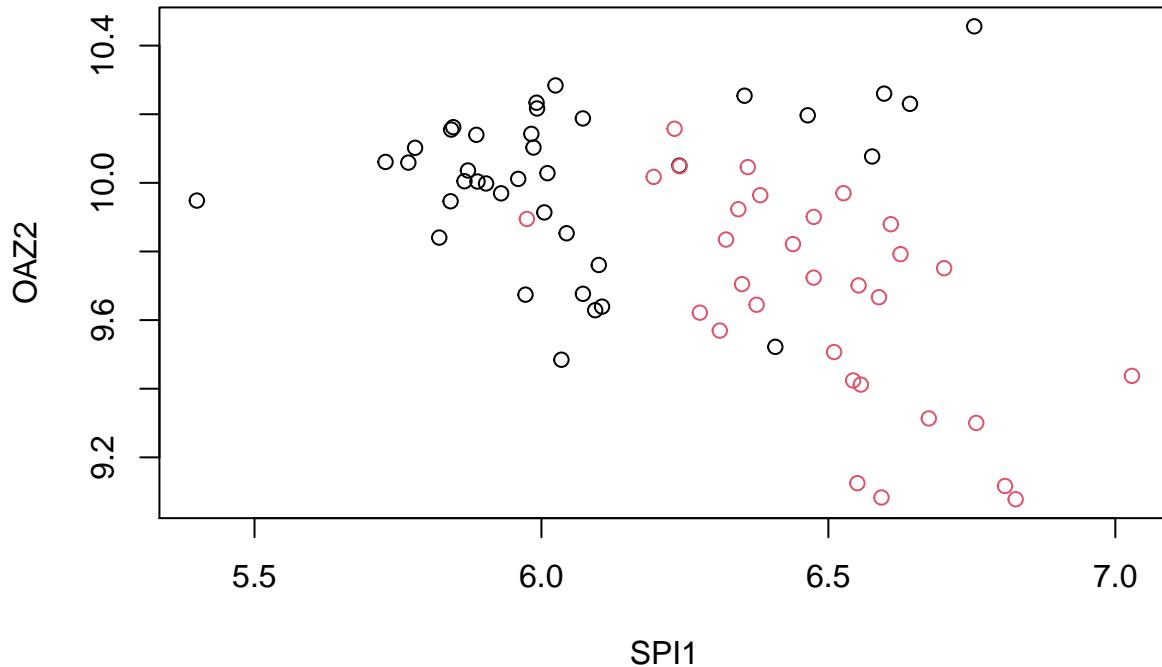
```
## non-uniform 'Rounding' sampler used
fit_lda$results["Accuracy"]

## Accuracy
## 1 0.865

t(fit_lda$finalModel$means) %>% data.frame() %>%
  mutate(predictor_name = rownames(.)) %>%
  ggplot(aes(predictor_name, hippocampus)) +
  geom_point() +
  coord_flip()
```



```
d <- apply(fit_lda$finalModel$means, 2, diff)
ind <- order(abs(d), decreasing = TRUE)[1:2]
plot(x[, ind], col = y)
```



Now we are going to increase the complexity of the challenge slightly. Repeat the LDA analysis from Q5 but using all tissue types. Use the following code to create your dataset:

```
library(dslabs)
library(caret)
data("tissue_gene_expression")

set.seed(1993, sample.kind="Rounding")

## Warning in set.seed(1993, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used

y <- tissue_gene_expression$y
x <- tissue_gene_expression$x
x <- x[, sample(ncol(x), 10)]
```

What is the accuracy using LDA?

```
fit_lda <- train(x, y, method = "lda", preProcess = c("center"))

## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :
## non-uniform 'Rounding' sampler used
fit_lda$results["Accuracy"]

##   Accuracy
## 1      0.798
```

## 8.5. Clasificación con más de dos clases y paquete Caret

### 8.5.1. Clasificación con más de dos clases

Hasta ahora, hemos descrito cómo métodos como QDA y LDA no son adecuados para usarse con bases de datos con muchos predictores, dado que el número de parámetros por estimar es demasiado grande. Por su parte, métodos como la regresión local o kernel, si bien no tienen parámetros por estimar, también enfrentan un reto en casos con múltiples predictores debido a la llamada *curse of dimensionality*.

Este fenómeno ocurre cuando, al existir múltiples predictores, la ventana o vecindad utilizada para incluir un porcentaje dado de los datos se vuelve muy grande. Con vecindades más grandes, el método pierde flexibilidad. La dimensión en este contexto se refiere al hecho que cuando tenemos  $p$  predictores, la distancia entre dos observaciones se calcula en un espacio  $p$ -dimensional.

Para introducir a los *Classification and Regression Trees (CART)*, usaremos una nueva base de datos relacionada con la composición del aceite de oliva en ocho ácidos grasos:

```
library(dslabs)
library(tidyverse)

data("olive")

head(olive)

##          region      area palmitic palmitoleic stearic oleic linoleic
## 1 Southern Italy North-Apulia    10.75     0.75    2.26  78.2   6.72
## 2 Southern Italy North-Apulia    10.88     0.73    2.24  77.1   7.81
## 3 Southern Italy North-Apulia     9.11     0.54    2.46  81.1   5.49
## 4 Southern Italy North-Apulia     9.66     0.57    2.40  79.5   6.19
## 5 Southern Italy North-Apulia    10.51     0.67    2.59  77.7   6.72
## 6 Southern Italy North-Apulia     9.11     0.49    2.68  79.2   6.78
##      linolenic arachidic eicosenoic
## 1       0.36     0.60     0.29
## 2       0.31     0.61     0.29
## 3       0.31     0.63     0.29
## 4       0.50     0.78     0.35
## 5       0.50     0.80     0.46
## 6       0.51     0.70     0.44
```

Para efectos ilustrativos, trataremos de predecir la región a partir de los valores de composición de ácidos grasos como predictores. Sean tres regiones:

```
table(olive$region)

##
## Northern Italy           Sardinia Southern Italy
##             151                  98                 323
```

Removeremos la columna de área, dado que no la usaremos como predictor:

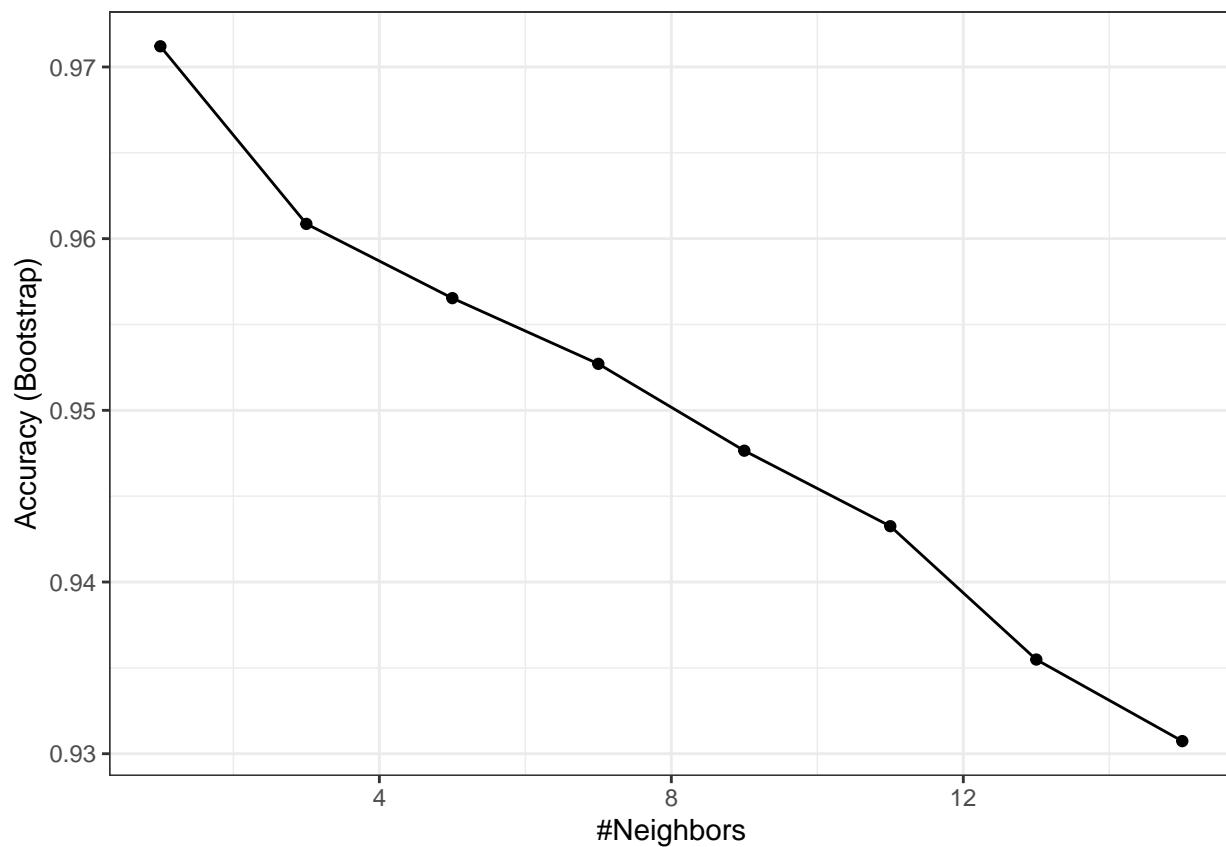
```
olive <- select(olive, -area)
```

Observemos cuál sería el resultado de usar k-vecinos :

```
library(caret)

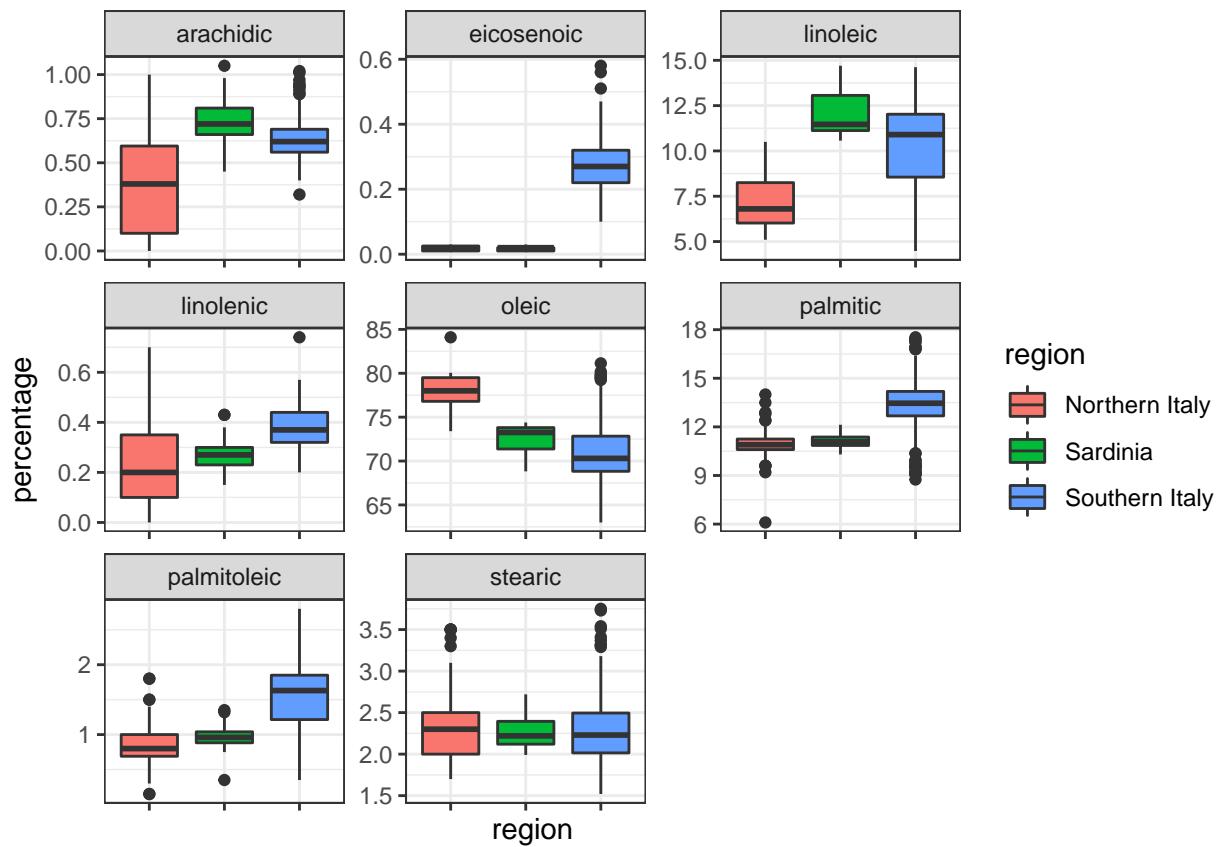
fit <- train(region ~.,
              method = "knn",
              tuneGrid = data.frame(k = seq(1, 15, 2)), data = olive)
```

```
ggplot(fit)
```



Si bien podemos hallar una precisión bastante alta, una inspección a los datos sugiere que los resultados podrían ser aún mejores. Por ejemplo, considérese la distribución de cada predictor estratificada por región:

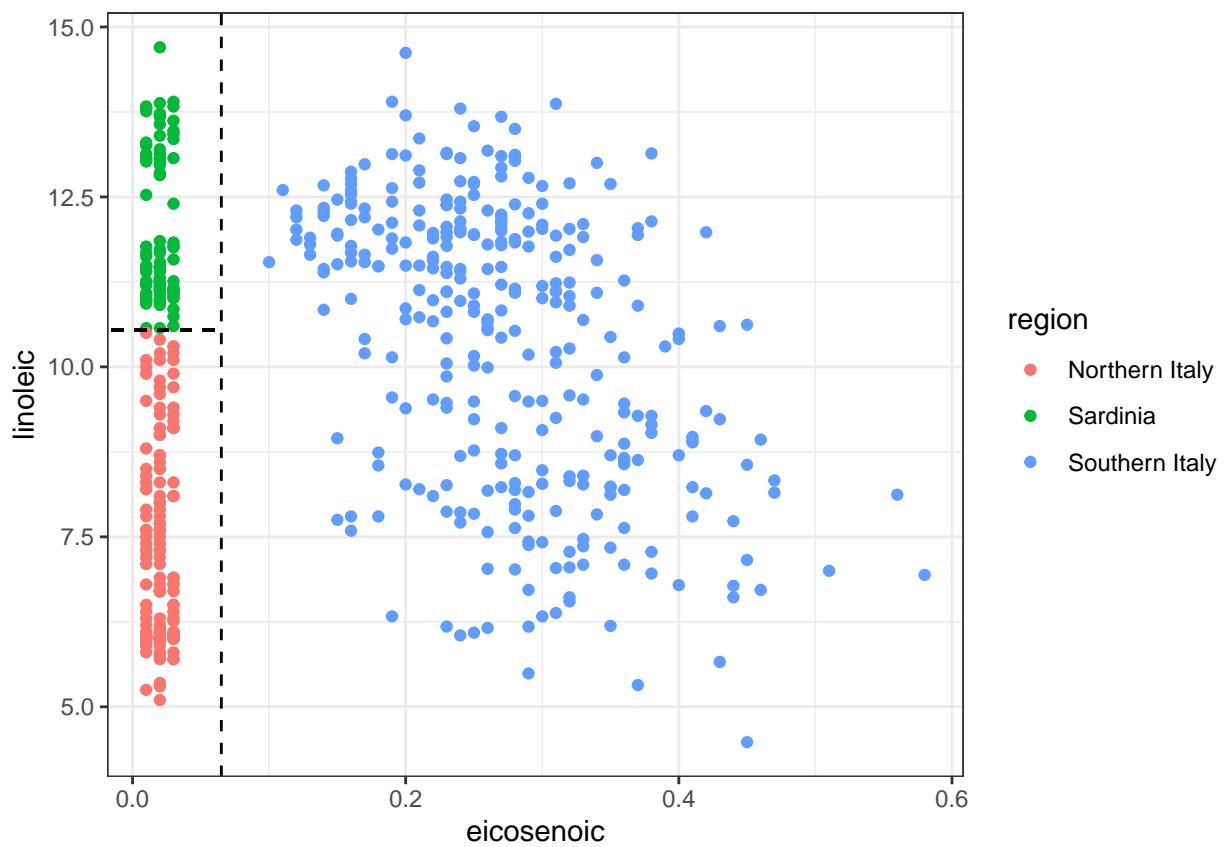
```
olive %>%
  gather(fatty_acid, percentage, -region) %>%
  ggplot(aes(region, percentage, fill = region)) +
  geom_boxplot() +
  facet_wrap(~fatty_acid, scales = "free") +
  theme(axis.text.x = element_blank())
```



Puede verse lo siguiente: uno de los ácidos grados solo está presente en el sur de Italia (eicosenoic) y otro (linoleic) separa al sur y Sardinia. Esto implica que podríamos construir un algoritmo más preciso. Esto también puede verse al graficar los valores de estos dos ácidos grasos:

```
p <- olive %>%
  ggplot(aes(eicosenoic, linoleic, color = region)) +
  geom_point()

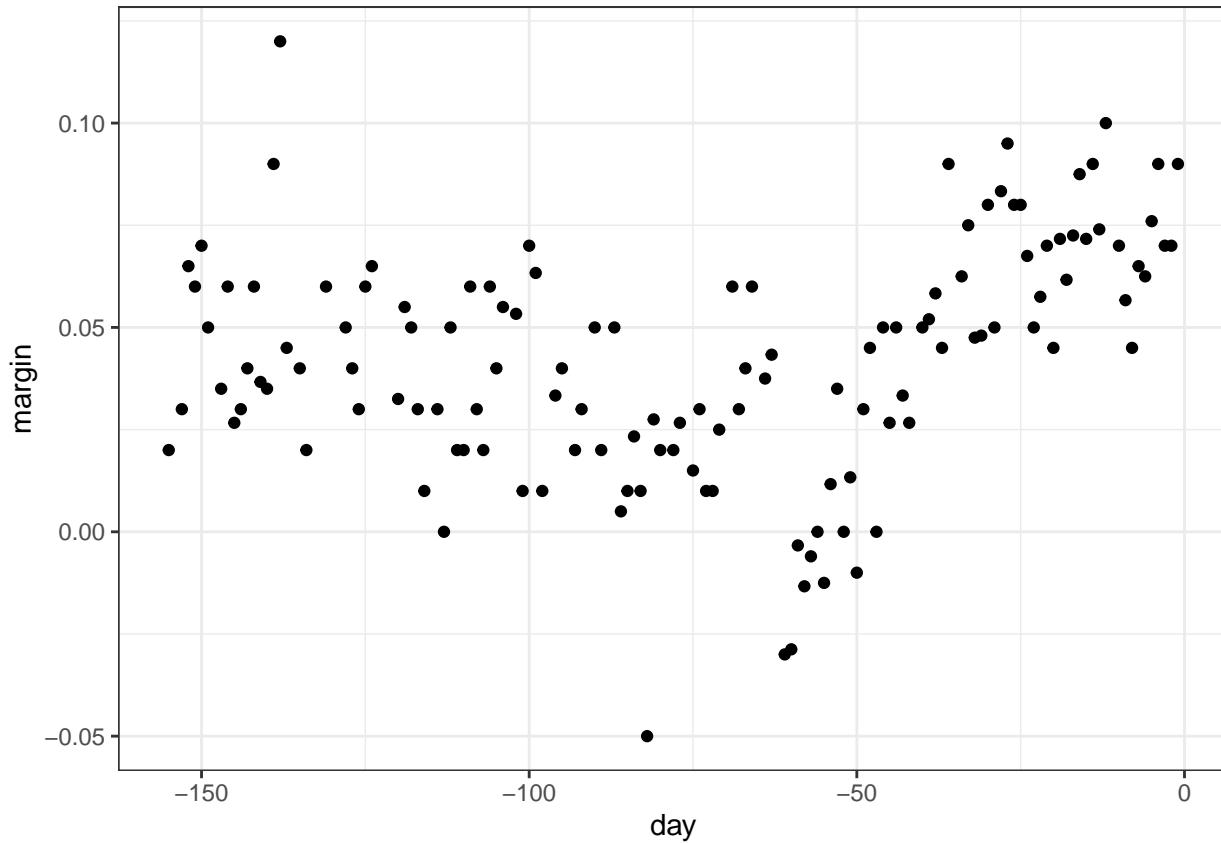
p + geom_vline(xintercept = 0.065, lty = 2) +
  geom_segment(x = -0.2, y = 10.54, xend = 0.065, yend = 10.54, color = "black", lty = 2)
```



Es evidente que, incluso visualmente, podemos construir una regla de decisión como sigue: si el primer predictor es mayor a 0.065, predeciremos Southern Italy; si no, nos fijaremos en el segundo predictor, donde si es mayor a 10.535, predeciremos Sardinia. Esta regla de decisión implica un **árbol de decisión**. Cuando los resultados son continuos, llamamos a los resultados **árboles de regresión**, donde intentaremos estimar la esperanza condicional  $f(x) = E[Y|X = x]$ .

Considérese el ejemplo de las encuestas electorales de 2008:

```
data("polls_2008")
qplot(day, margin, data = polls_2008)
```



La idea general será construir un árbol de decisión, donde al final de cada nodo tendremos una diferente predicción  $\hat{Y}$ . Primero, haremos una partición del espacio en  $J$  regiones no traslapadas,  $R_1, \dots, R_J$ . Para cada observación que esté dentro de la región  $x_i \in R_j$ , predeciremos  $\hat{Y}$  con el promedio de las observaciones de práctica  $Y_i$  en la región  $x_i \in R_j$ .

¿Cómo decidiremos las particiones  $R_1, \dots, R_J$ ? Los árboles de decisión crean particiones de manera recursiva: supóngase que ya tenemos una partición. Así, tenemos que decidir qué predictor  $j$  usaremos en la siguiente partición y dónde hacerlo dentro de ese predictor.

Si ya tenemos una partición de tal manera que cada observación  $i$  esté en exactamente una de estas particiones. Para cada una de estas, dividiremos aún más mediante el siguiente algoritmo: hay que encontrar un predictor  $j$  y un valor  $s$  que definan dos nuevas particiones  $R_1$  y  $R_2$ , las cuales dividirán nuestras observaciones en los siguientes conjuntos:  $R_1(j, s) = \{X | X_j < s\}$  y  $R_2(j, s) = \{X | X_j \geq s\}$ .

Luego, en cada conjunto definiremos promedios  $\bar{y}_{R_1}$  y  $\bar{y}_{R_2}$  y los usaremos como predicciones. Por tanto, podemos hacer esto para muchas Js y Ss; sin embargo, escogeremos la combinación que minimice la suma cuadrada de residuales definida por la siguiente fórmula

$$\sum_{i: x_i \in R_1(j, s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i: x_i \in R_2(j, s)} (y_i - \hat{y}_{R_2})^2$$

Esto se hará de manera recursiva, donde seguiremos encontrando nuevas regiones para dividir en dos. Una vez que terminemos la partición del espacio de predictores en regiones, podremos hacer predicciones por región usando las observaciones de cada una.

Para el ejemplo de las encuestas, usaremos la función “rpart()” del paquete rpart:

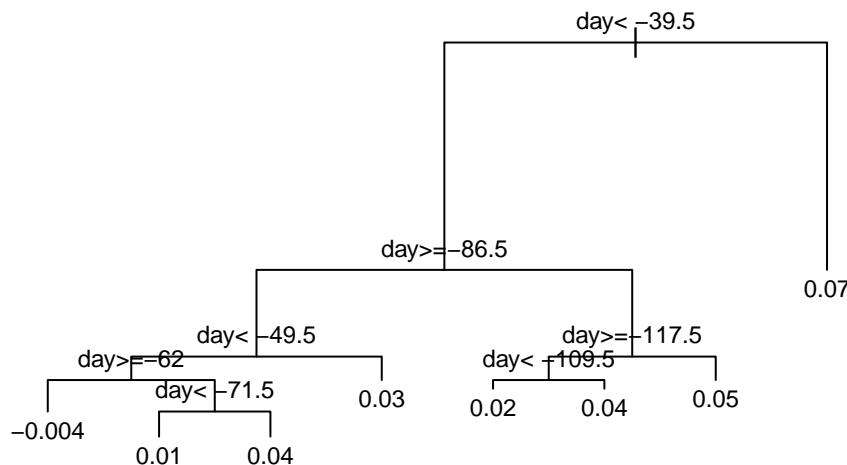
```
library(rpart)

fit <- rpart(margin ~., data = polls_2008)
```

Como solo hay un predictor, no tenemos que decidir cuál predictor  $j$  usar para dividir, pero sí tenemos que definir el valor  $s$ . Visualmente, puede verse dónde se hicieron las divisiones con el siguiente código:

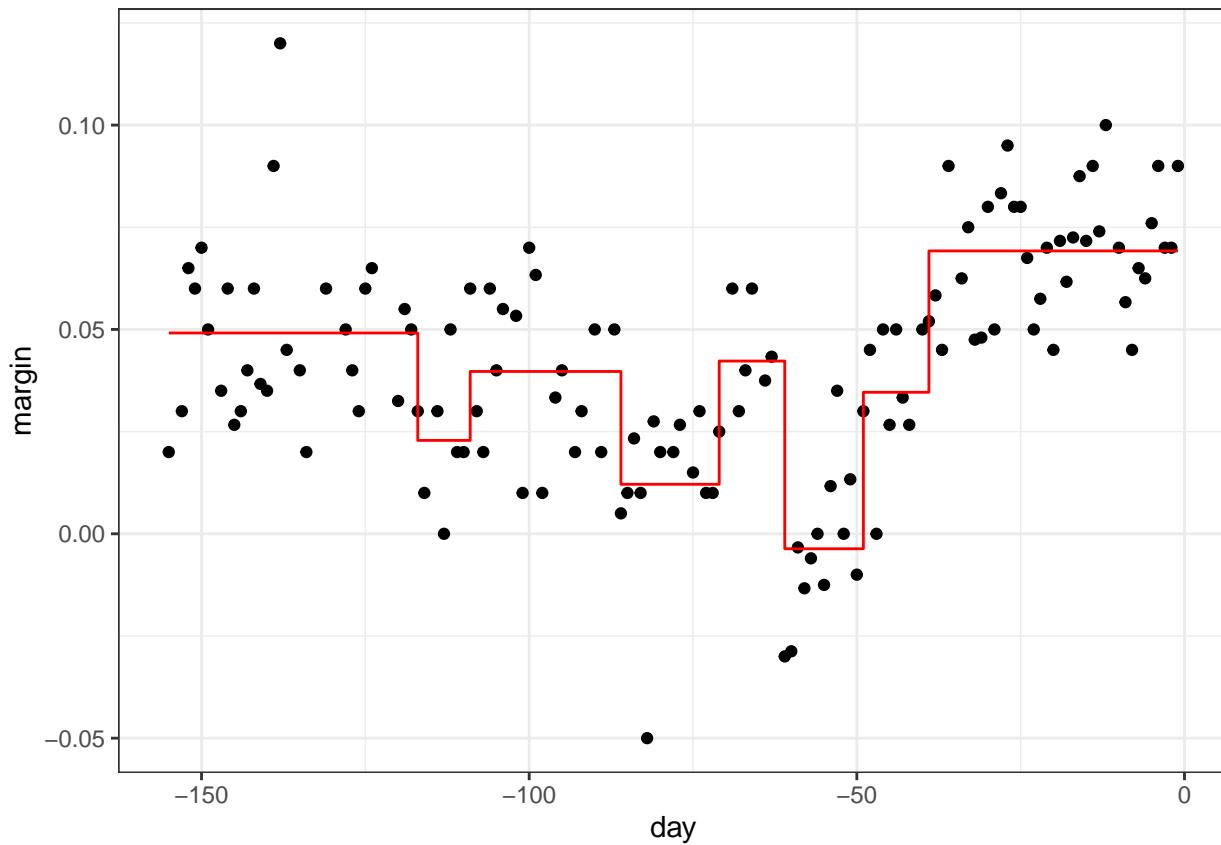
```
plot(fit, margin = 0.1)

text(fit, cex = 0.75)
```



Donde vemos que terminamos con 8 particiones. Así, el estimado final  $f(\hat{x})$  luce como sigue:

```
polls_2008 %>%
  mutate(y_hat = predict(fit)) %>%
  ggplot() +
  geom_point(aes(day, margin)) +
  geom_step(aes(day, y_hat), col="red")
```



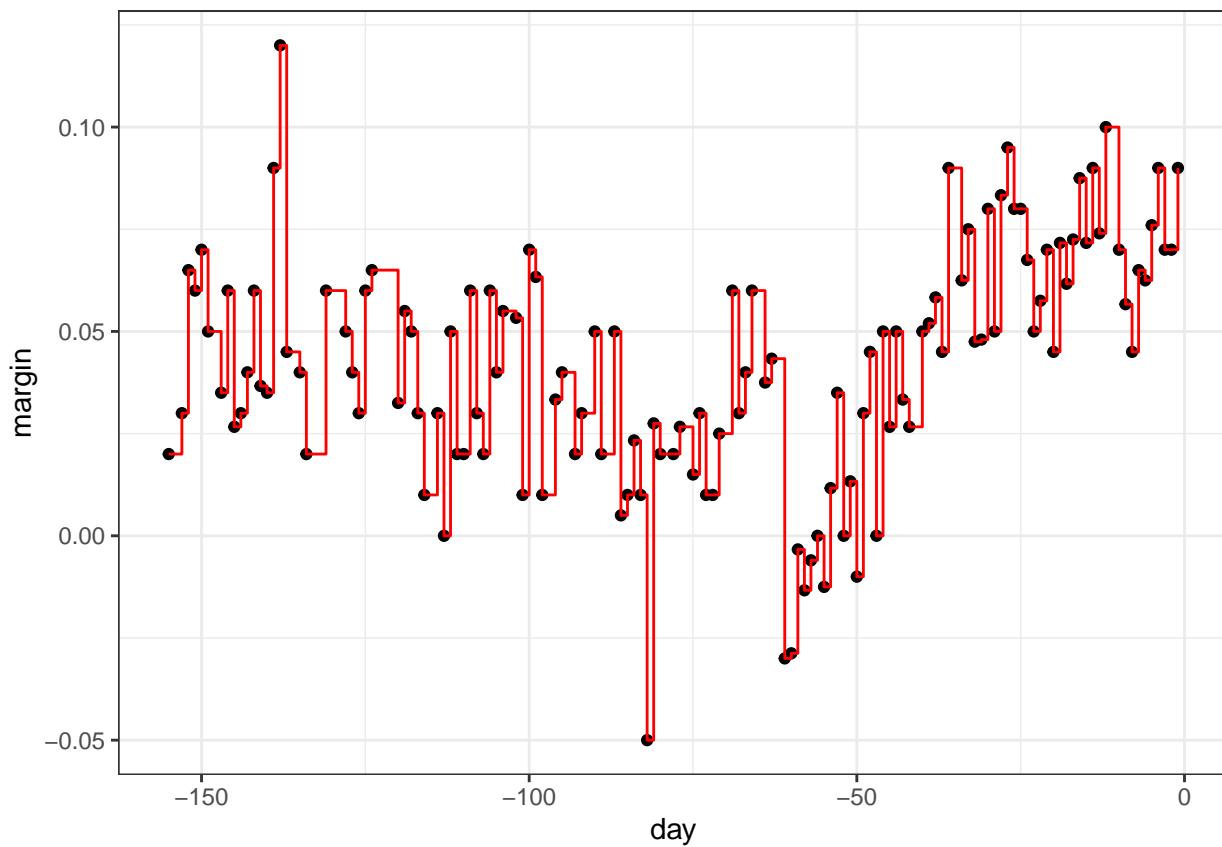
Pero, ¿por qué el algoritmo se detuvo en ocho particiones?

Nótese que cada vez que dividimos y definimos dos nuevas particiones, la suma de cuadrados residuales del conjunto de práctica decrece. Esto se debe a que, a mayores particiones, nuestro modelo tiene mayor flexibilidad para adaptarse a los datos de práctica. Sin embargo, para evitar un *overtraining*, el algoritmo define un mínimo para que la suma cuadrada de residuales “mejore” con una partición adicional. Este parámetro se conoce como **parámetro de complejidad (CP)**: así, la suma cuadrada de residuales debe mejorar por un factor CP para que se agregue una nueva partición.

Otro aspecto importante, es que el algoritmo define un mínimo de observaciones para ser divididas. Esto se define mediante el argumento “minsplit” de la función “rpart()”, donde el default es 20. Además, también existe un mínimo para cada partición, a través del argumento “minbucket”, cuyo default es “round(minsplit/3)”.

¿Qué ocurriría si fijamos  $cp = 0$  y  $minsplit = 2$ ? Nuestra predicción serían los datos originales, dado que el árbol se seguirá dividiendo hasta que el RSS sea minimizado en 0:

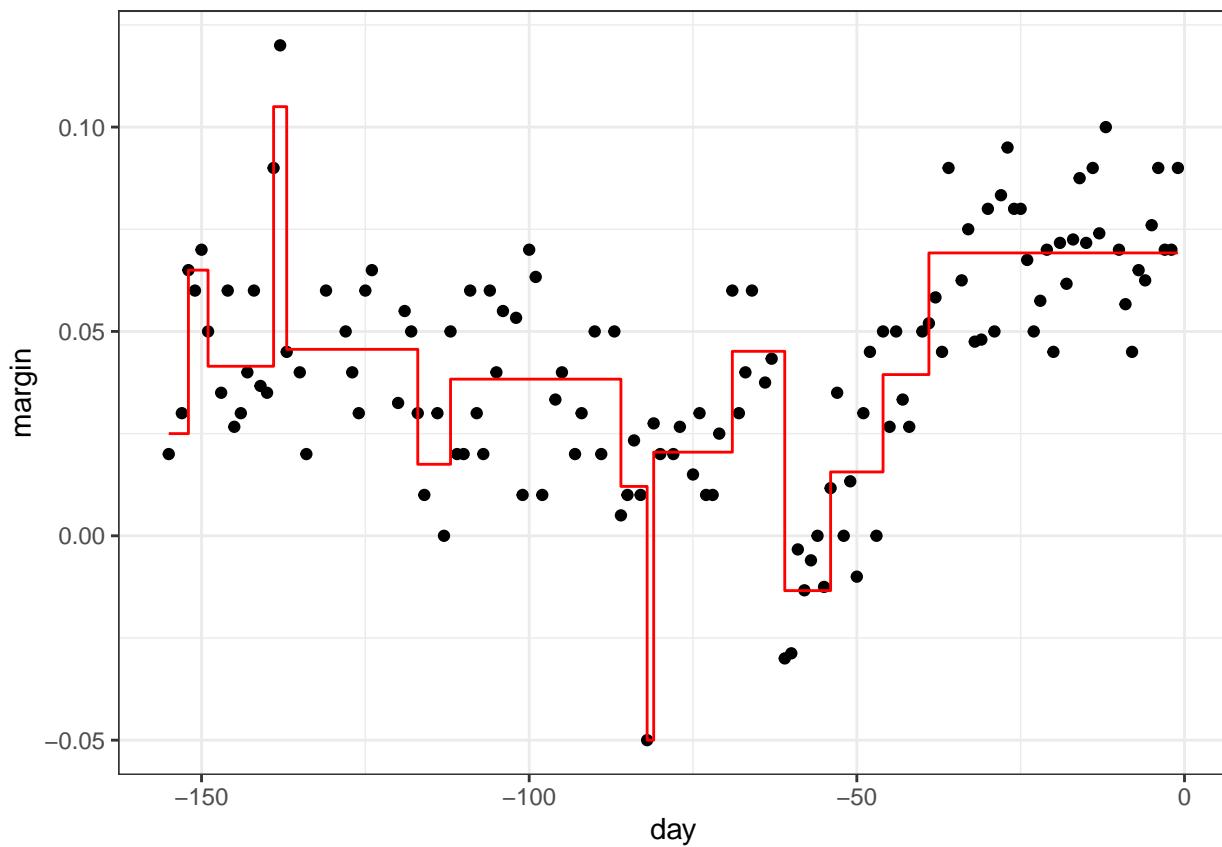
```
fit <- rpart(margin ~ ., data = polls_2008, control = rpart.control(cp = 0, minsplit = 2))
polls_2008 %>%
  mutate(y_hat = predict(fit)) %>%
  ggplot() +
  geom_point(aes(day, margin)) +
  geom_step(aes(day, y_hat), col="red")
```



Nótese que en esta metodología también podemos “podar” los árboles al quitar particiones que no cumplan un criterio de CP:

```
pruned_fit <- prune(fit, cp = 0.01)

polls_2008 %>%
  mutate(y_hat = predict(pruned_fit)) %>%
  ggplot() +
  geom_point(aes(day, margin)) +
  geom_step(aes(day, y_hat), col="red")
```

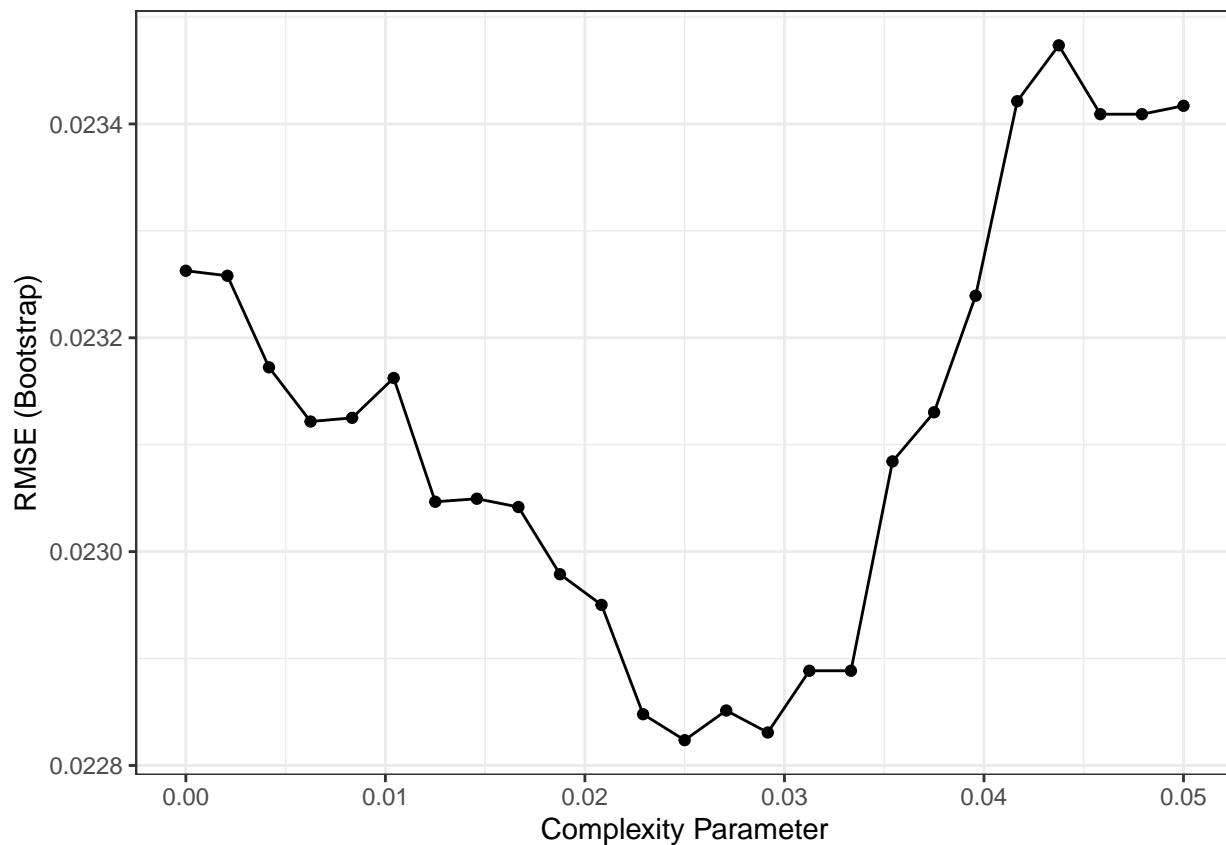


¿Es el valor por default de CP el mejor? Para hacer una elección óptima, podemos usar validación cruzada a través de la función “train()” del paquete caret, por ejemplo:

```
library(caret)

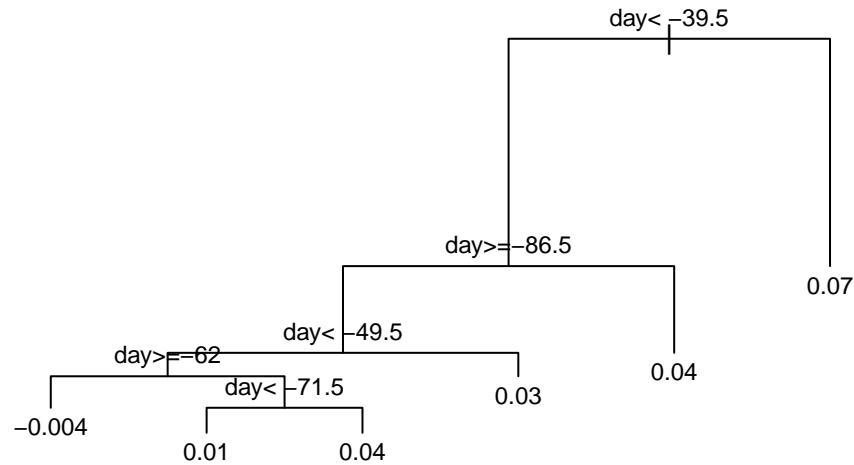
train_rpart <- train(margin~.,
                      method = "rpart",
                      tuneGrid = data.frame(cp = seq(0, 0.05, len = 25)),
                      data = polls_2008)

## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :
## non-uniform 'Rounding' sampler used
ggplot(train_rpart)
```



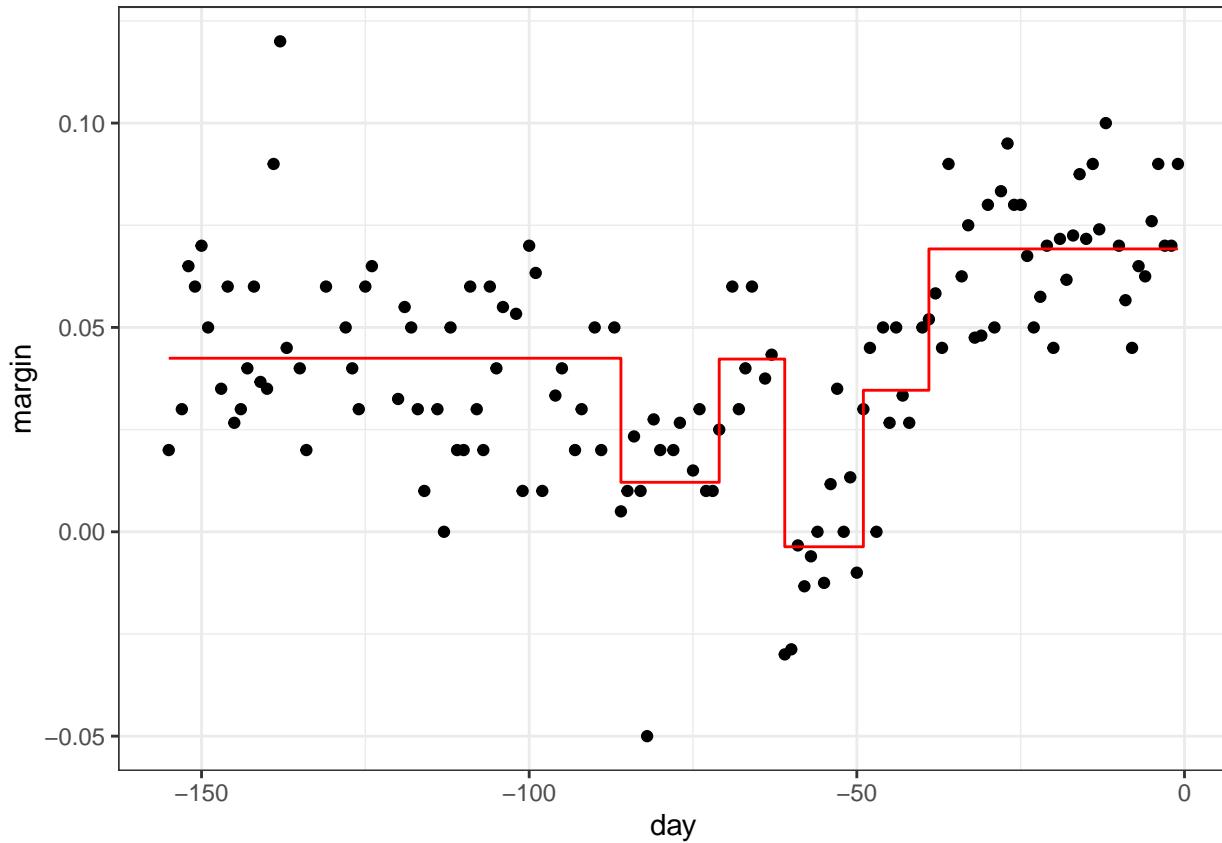
Para ver el árbol final que minimiza el error, úsese el siguiente código:

```
plot(train_rpart$finalModel, margin = 0.1)  
text(train_rpart$finalModel, cex = 0.75)
```



Y, dado que solo tenemos un predictor, podemos graficar  $f(\hat{x})$ :

```
polls_2008 %>%
  mutate(y_hat = predict(train_rpart)) %>%
  ggplot() +
  geom_point(aes(day, margin)) +
  geom_step(aes(day, y_hat), col="red")
```



**8.5.1.1. Árboles de clasificación** Cuando los resultados son categóricos, nos referimos a estos métodos como árboles de clasificación , donde usamos los mismos principios que en el caso continuo pero con algunas ligeras diferencias a tomar en cuenta:

1. En vez de tomar el promedio al final de cada nodo, ahora predeciremos con la clase que tenga la mayoría del voto en cada nodo. Esto es, predeciremos cuál es la clase que aparece más en un nodo.
2. Ya no podemos usar RSS para decidir nuestras particiones. En su lugar, podemos usar métricas más sofisticadas, como el Índice Gini o *Entropy*.

Si definimos  $\hat{p}_{m,k}$  proporción de observaciones en la partición  $m$  que son de clase  $k$ , entonces el Índice Gini se define:

$$\sum_{k=1}^K \hat{p}_{m,k}(1 - \hat{p}_{m,k})$$

Mientras que *Entropy* es:

$$-\sum_{k=1}^K \hat{p}_{m,k} \log(\hat{p}_{m,k}) \quad \text{con } 0 * \log(0) \text{ definido como 0}$$

Nótese que si una partición  $m$  tiene solo una clase, entonces se cumple que  $\hat{p}_{m,1} = 1, \hat{p}_{m,2} = 0, \dots, \hat{p}_{m,K} = 0$ , lo que implica que tanto el Índice Gini como *Entropy* son 0, su valor más pequeño.

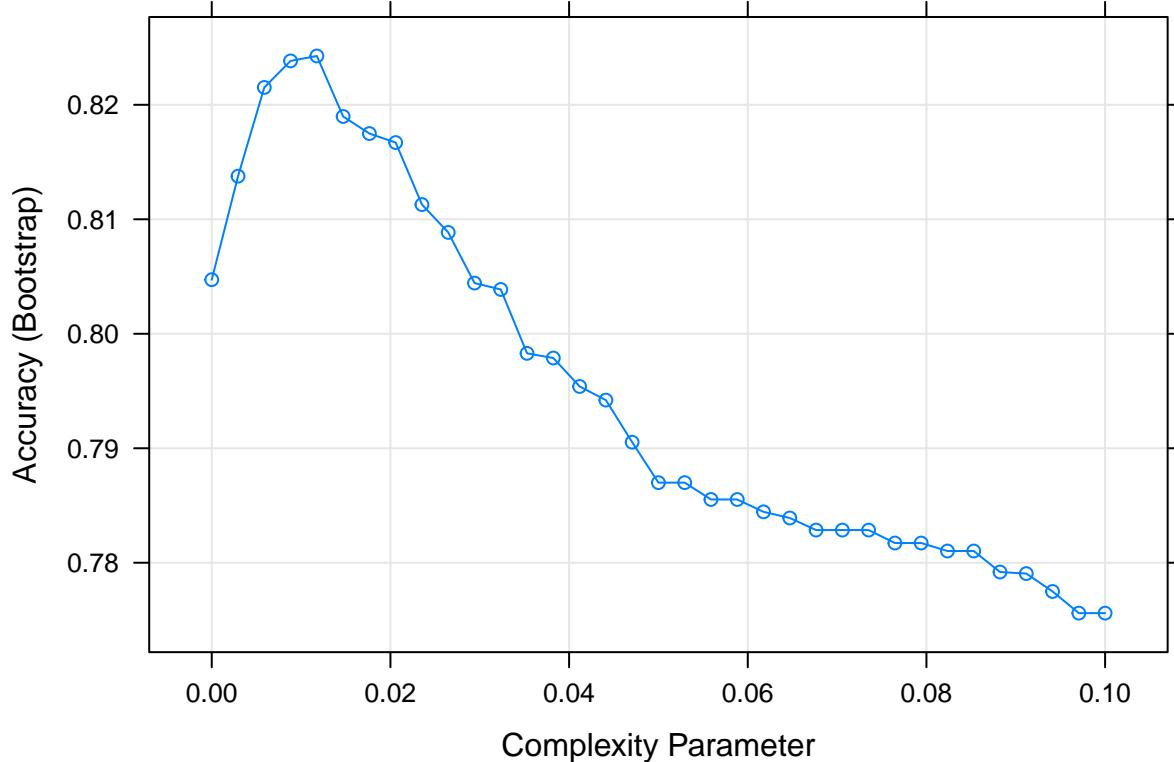
Veamos un ejemplo: observemos cómo los árboles de clasificación se comportan en el caso de 2s y 7s:

```

train_rpart <- train(y~.,
                      method = "rpart",
                      tuneGrid = data.frame(cp = seq(0.0, 0.1, len = 35)),
                      data = mnist_27$train)

## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :
## non-uniform 'Rounding' sampler used
plot(train_rpart)

```



A partir de lo anterior, podemos escoger el parámetro de complejidad más adecuado.

```

confusionMatrix(predict(train_rpart,
                        mnist_27$test),
                mnist_27$test$y)$overall["Accuracy"]

## Accuracy
## 0.805

```

Nótese que esto es mejor que una regresión logística, pero no tan bueno como los métodos kernel, dado que las cotas para estimar las probabilidades condicionales no pueden ser suavizadas. No obstante, este tipo de árboles tienen algunas ventajas a considerar:

- Ventajas: Son altamente interpretables y fáciles de visualizar. Pueden modelar procesos de decisión humanos y no requieren el uso de predictores *dummy* para variables categóricas.
- Desventajas: Su enfoque vía particiones recursivas puede presentar fácilmente *overtraining* y, por tanto, se dificulta su utilización para la práctica. Además, en términos de precisión, son raramente el mejor método dado que no son muy flexibles y son altamente inestables a cambios en los datos de práctica.

**8.5.1.2. Bosques aleatorios** Los *random forests* son un enfoque muy popular que minimizan las desventajas de los árboles de decisión. Su objetivo es mejorar el rendimiento de predicción y reducir la inestabilidad mediante el promedio de múltiples árboles.

Para lograr esto, se hace uso de la siguiente característica:

- Agregación *bootstrap*: primero, se construyen varios árboles de decisión  $T_1, \dots, T_B$ ; segundo, para cada observación  $j$  en el conjunto de prueba, se forma una predicción  $\hat{y}_j$  con el árbol  $T_j$ ; finalmente, para obtener una predicción final, se combinan las predicciones de cada árbol en resultados continuos o categóricos. Para el primero:

$$\hat{y} = \frac{1}{B} \sum_{j=1}^B \hat{y}_j$$

Para el caso categórico, predecimos  $\hat{y}$  con el voto mayoritario (clase más frecuente entre  $\hat{y}_1, \dots, \hat{y}_T$ ).

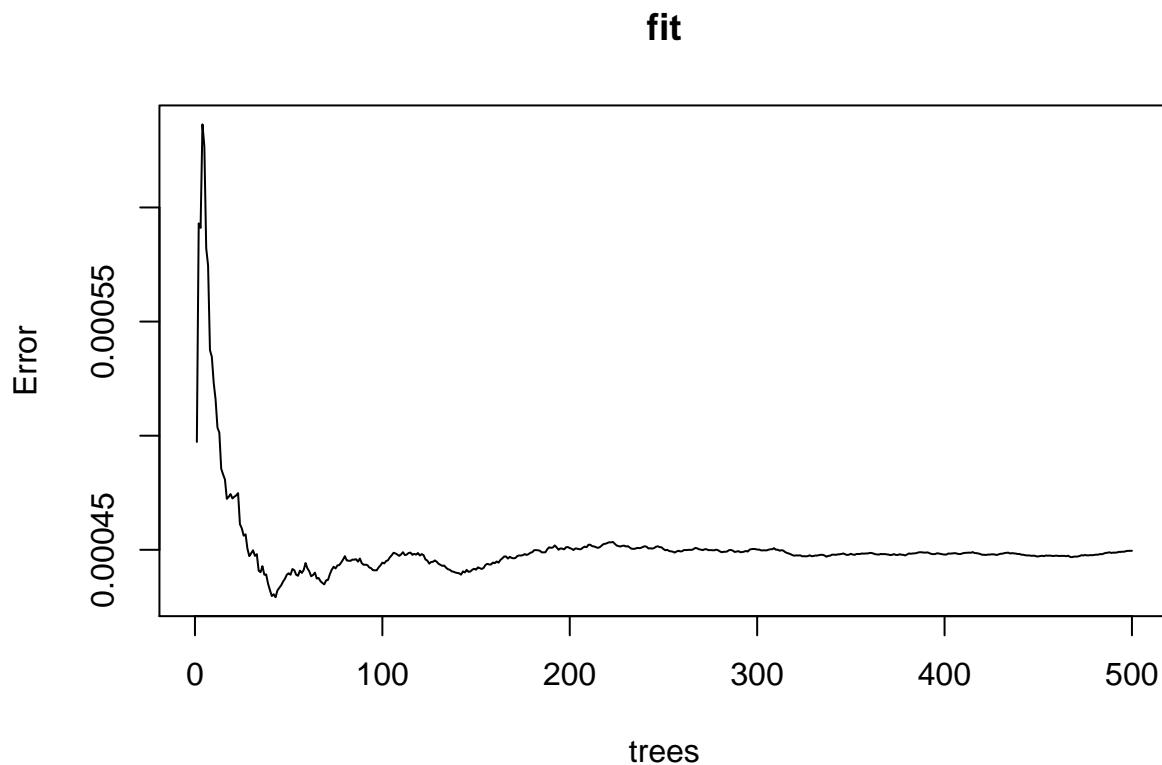
Así, para obtener varios árboles de decisión de un solo conjunto de práctica, usamos *bootstrap*, por lo que para crear, digamos,  $B$  árboles: para el árbol  $T_j$ ,  $j = 1, \dots, B$  del conjunto de práctica de tamaño  $N$ , construimos un conjunto de práctica *bootstrap* muestreando  $N$  observaciones del primer conjunto de práctica con reemplazo:

```
library(randomForest)

fit <- randomForest(margin ~., data = polls_2008)
```

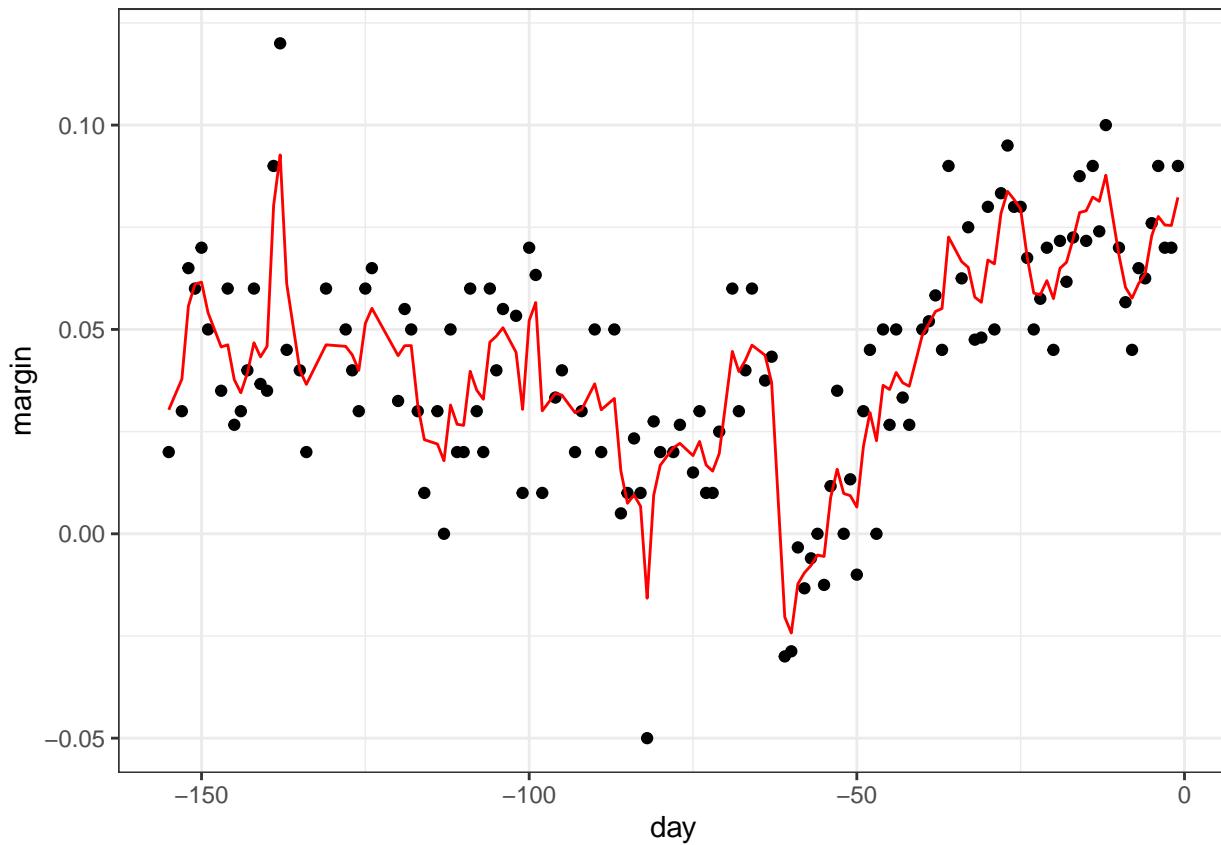
Donde podemos ver que el algoritmo mejora a medida que agregamos más árboles:

```
plot(fit)
```



Es decir, alrededor del árbol 200, el algoritmo se estabiliza y minimiza el error.

```
polls_2008 %>%
  mutate(y_hat = predict(fit, newdata = polls_2008)) %>%
  ggplot() +
  geom_point(aes(day, margin)) +
  geom_line(aes(day, y_hat), col="red")
```



Puede verse que el resultado es relativamente suave.

Cambiemos ahora al ejemplo de 2s y 7s:

```
train_rf <- randomForest(y ~ ., data = mnist_27$train)

confusionMatrix(predict(train_rf, mnist_27$test),
                mnist_27$test$y)$overall["Accuracy"]

## Accuracy
## 0.785
```

Pero no hemos optimizado los parámetros en ninguna medida: usemos caret para esto (en forma de comentarios dado que paquete “Rborist” presenta problemas de compilación con R Markdown):

```
#library(Rborist)

#fit <- train(y~.,
#              method = "Rborist",
#              tuneGrid = data.frame(predFixed = 2,
#                                    minNode = seq(3, 50)),
#              data = mnist_27$train)

#confusionMatrix(predict(train_rf_2, mnist_27$test), mnist_27$test$y)$overall["Accuracy"]
```

Así, podemos controlar la suavidad del bosque aleatorio en varias maneras: (1) limitando el tamaño de cada nodo (2) utilización de una selección aleatoria de características para dividir las particiones, con el argumento “mtry”.

Una desventaja de los bosques aleatorios es la pérdida de interpretabilidad. No obstante, la “importancia de la variable” nos ayuda con esto: nos dice cuánto influye cada predictor en la predicción final.

### 8.5.2. Assessment 29

Create a simple dataset where the outcome grows 0.75 units on average for every increase in a predictor, using this code:

```
library(rpart)

n <- 1000
sigma <- 0.25

set.seed(1, sample.kind = "Rounding")

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used

x <- rnorm(n, 0, 1)
y <- 0.75 * x + rnorm(n, 0, sigma)
dat <- data.frame(x = x, y = y)
```

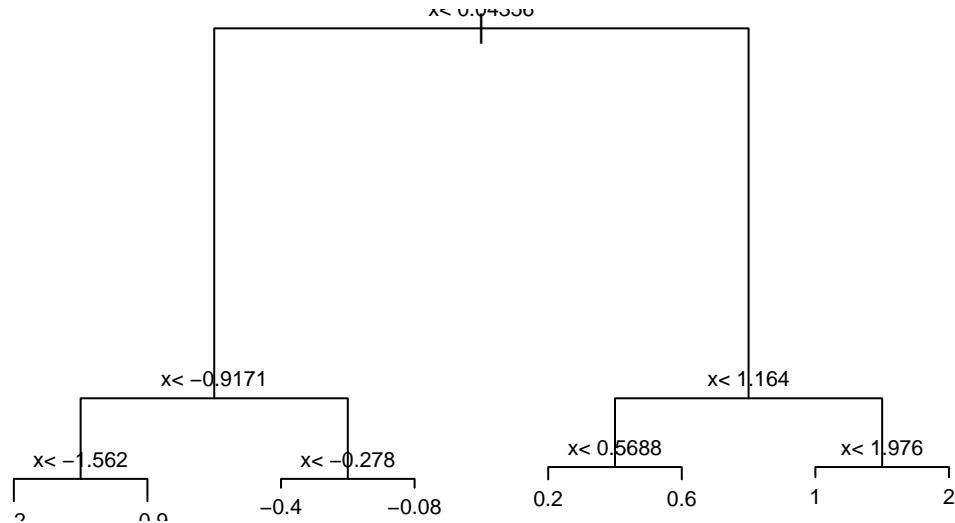
Which code correctly uses rpart() to fit a regression tree and saves the result to fit?

```
fit <- rpart(y ~ ., data = dat)
```

Which of the following plots has the same tree shape obtained in Q1?

```
plot(fit)

text(fit, cex = 0.65)
```

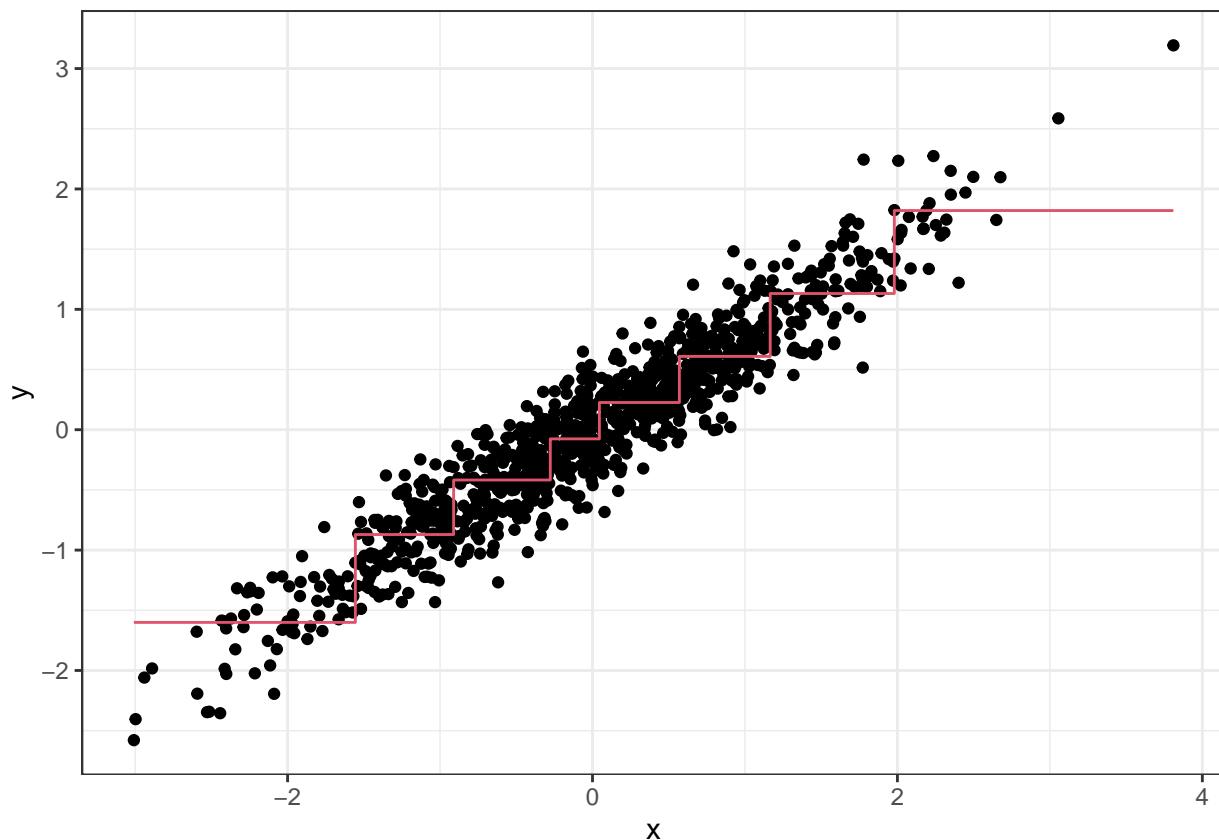


Below is most of the code to make a scatter plot of y versus x along with the predicted values based on the fit.

```

library(tidyverse)

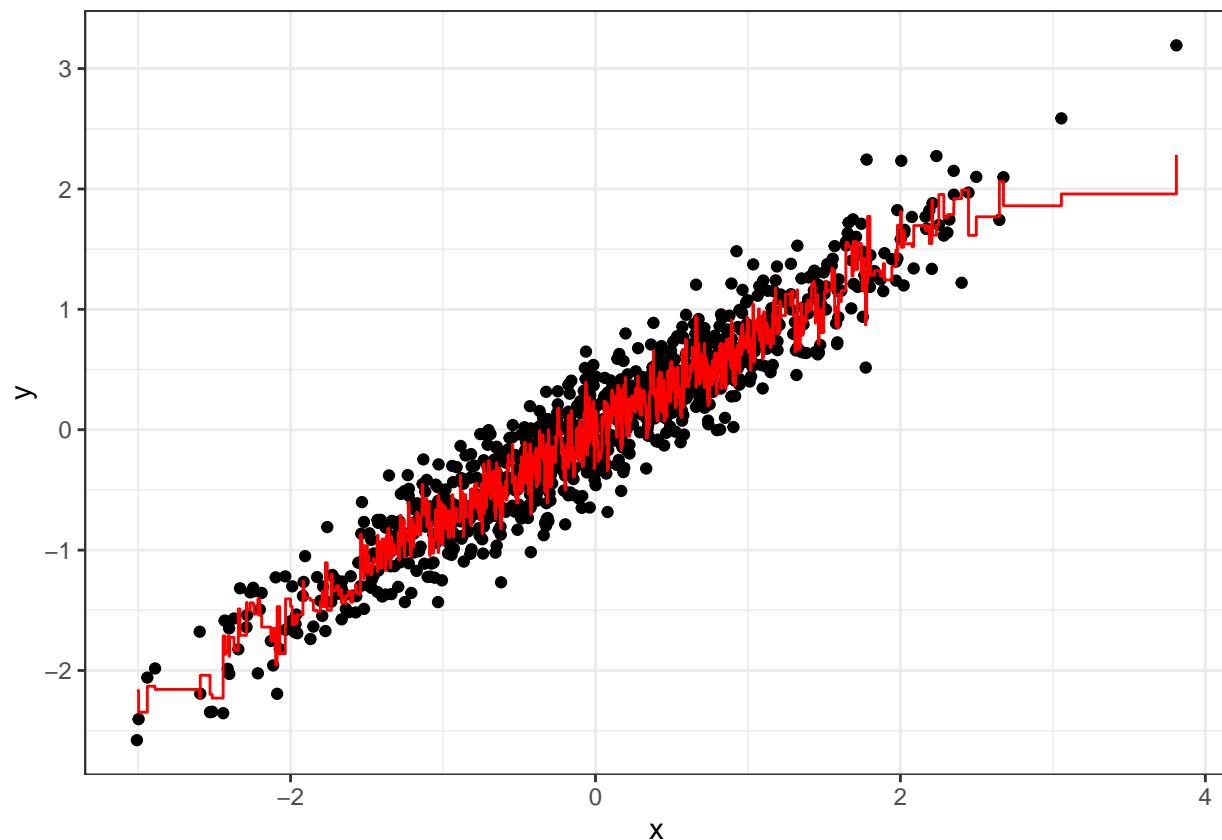
dat %>%
  mutate(y_hat = predict(fit)) %>%
  ggplot() +
  geom_point(aes(x, y)) +
  geom_step(aes(x, y_hat), col=2)
  
```



Now run Random Forests instead of a regression tree using `randomForest()` from the `randomForest` package, and remake the scatterplot with the prediction line. Part of the code is provided for you below.

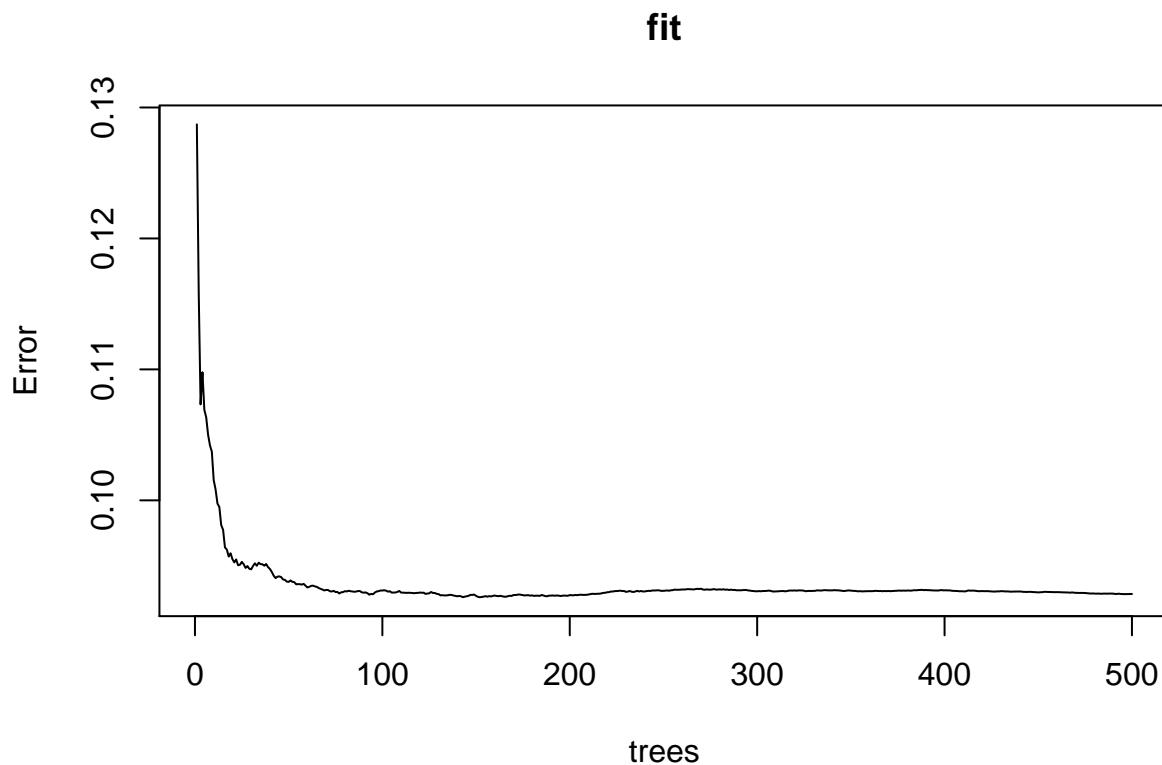
```
library(randomForest)

fit <- randomForest(y ~ x, data = dat)
dat %>%
  mutate(y_hat = predict(fit)) %>%
  ggplot() +
  geom_point(aes(x, y)) +
  geom_step(aes(x, y_hat), col = "red")
```



Use the `plot()` function to see if the Random Forest from Q4 has converged or if we need more trees.

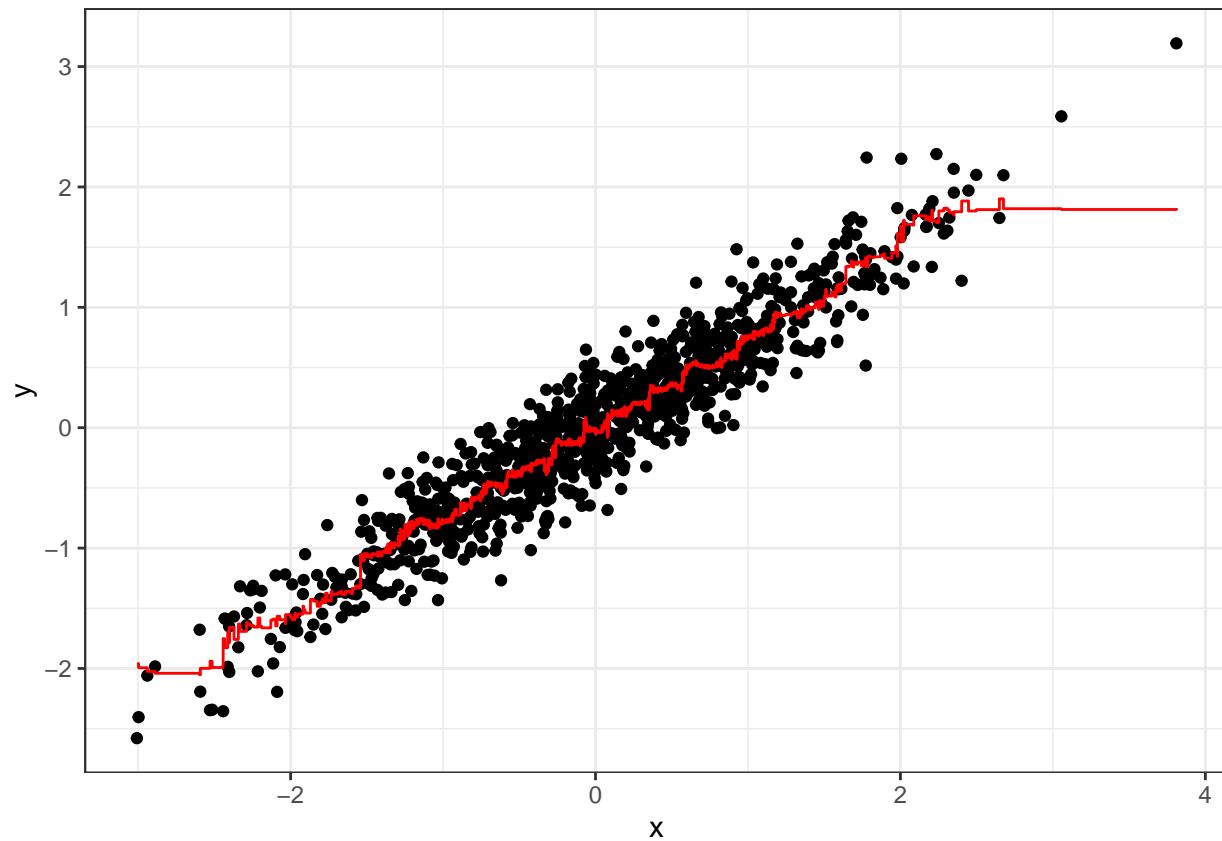
```
plot(fit)
```



It seems that the default values for the Random Forest result in an estimate that is too flexible (unsmooth). Re-run the Random Forest but this time with a node size of 50 and a maximum of 25 nodes. Remake the plot.

```
library(randomForest)

fit <- randomForest(y ~ x, data = dat, nodesize = 50, maxnodes = 25)
dat %>%
  mutate(y_hat = predict(fit)) %>%
  ggplot() +
  geom_point(aes(x, y)) +
  geom_step(aes(x, y_hat), col = "red")
```



### 8.5.3. Paquete Caret

Hemos aprendido sobre regresiones, regresiones logísticas, *k-nearest neighbors* como algoritmos de aprendizaje automático. Muchos de estos algoritmos, y más, están implementados en R, aunque a través de diferentes paquetes, autores y sintaxis.

El paquete caret consolida estas diferencias y provee de consistencia: actualmente, cuenta con 238 métodos que pueden ser visitados en este [sitio](#).

Nótese que caret no instala automáticamente los paquetes para usar estos métodos, por lo que para usar alguno, es necesario instalar el paquete.

Además, caret incluye una función para hacer validación cruzada. Como ejemplo, considérese el caso de 2s y 7s:

```
library(dslabs)

data(mnist_27)
```

La función “train()” nos permite practicar varios logaritmos con una sintaxis similar:

```
library(caret)

train_glm <- train(y~., method = "glm", data = mnist_27$train)

train_knn <- train(y~., method = "knn", data = mnist_27$train)
```

Para hacer predicciones, podemos ver los resultados de esta función sin tener que ver los específicos de “predict.glm” o “predict.knn”; en su lugar, usaremos la función “predict.train()”:

```
y_hat_glm <- predict(train_glm, mnist_27$test, type = "raw")

y_hat_knn <- predict(train_knn, mnist_27$test, type = "raw")
```

También, rápidamente podemos comparar las matrices de confusión:

```
confusionMatrix(y_hat_glm, mnist_27$test$y)$overall[[ "Accuracy"]]

## [1] 0.75

confusionMatrix(y_hat_knn, mnist_27$test$y)$overall[[ "Accuracy"]]

## [1] 0.84
```

**8.5.3.1. Parámetros de afinación** Cuando un algoritmo incluye parámetros de afinación, “train()” automáticamente usa validación cruzada para decidir entre valores default. Puede verse un resumen del modelo mediante “modelLookup()”, así como los parámetros que se están optimizando con cada uno:

```
library(caret)

library(dslabs)

data("mnist_27")

modelLookup("knn")

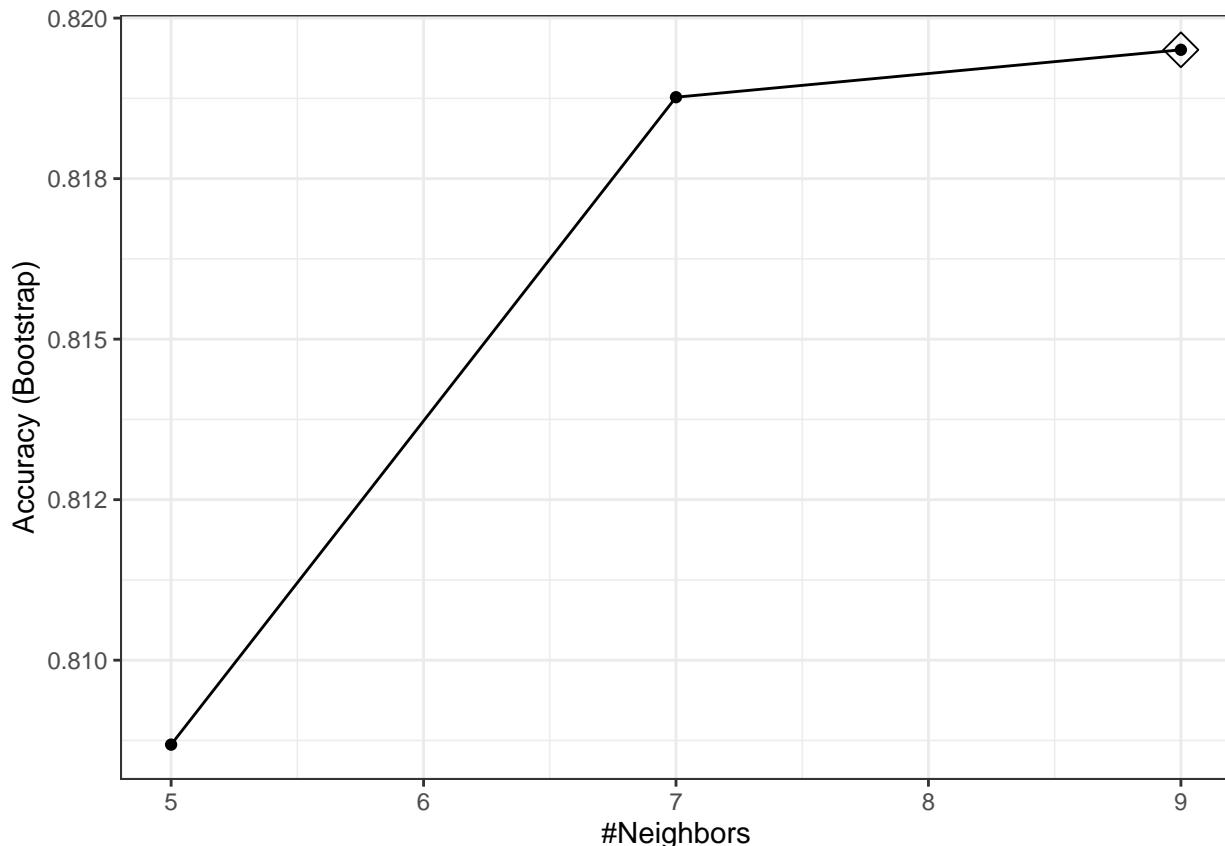
##   model parameter      label forReg forClass probModel
## 1   knn          k #Neighbors    TRUE     TRUE     TRUE
```

Así, la función “train()” con los argumentos por default, podemos ver los resultados de la validación cruzada con ggplot:

```
train_knn <- train(y~., method = "knn", data = mnist_27$train)

## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :
## non-uniform 'Rounding' sampler used

ggplot(train_knn, highlight = TRUE)
```



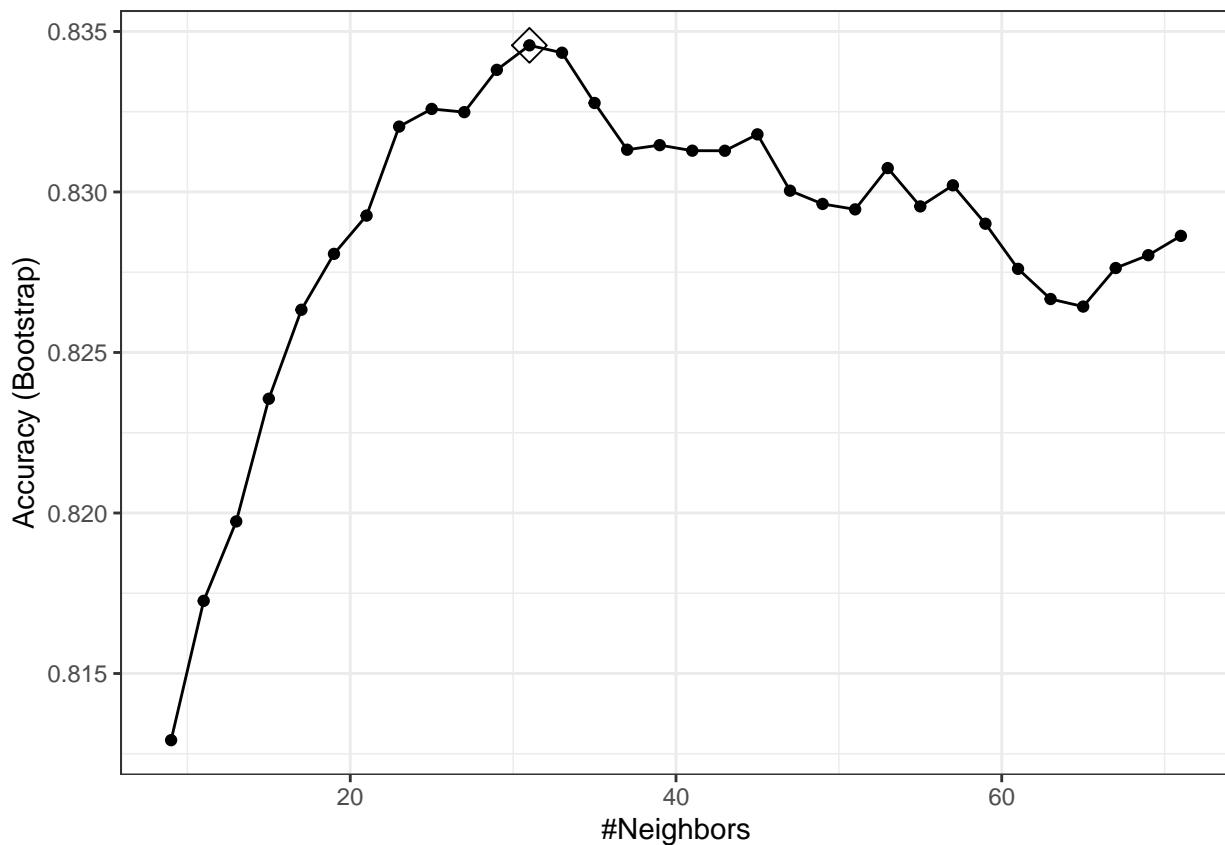
Por default, la validación cruzada se realiza probando 25 muestras *bootstrap* comprimidas del 25% de las observaciones. En la gráfica anterior, vemos que se probó y optimizó con 5, 7 y 9, siendo el último el mejor. Pero podría existir una  $k$  mejor; para cambiar esto, usamos el argumento “tuneGrid”, el cual selecciona la rejilla de valores a ser comparada.

Considérese un ejemplo probando 30 valores entre 9 y 67 para ver la  $k$  que optimiza la precisión:

```
train_knn <- train(y~., method = "knn",
                     data = mnist_27$train,
                     tuneGrid = data.frame(k = seq(9, 71, 2)))
```

```
## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :
## non-uniform 'Rounding' sampler used

ggplot(train_knn, highlight = TRUE)
```



```
train_knn$bestTune
```

```
##      k
## 12 31
```

También podemos accesar al modelo con el mejor rendimiento de la siguiente manera:

```
train_knn$finalModel
```

```
## 31-nearest neighbor model
## Training set outcome distribution:
##
##    2    7
## 379 421
```

Si aplicamos “predict()” al resultado de “train()”, usará este mejor modelo para hacer las predicciones. Ahora, podemos probar este modelo en nuestro conjunto de prueba como sigue:

```
confusionMatrix(predict(train_knn, mnist_27$test, type = "raw"),
                 mnist_27$test$y)$overall["Accuracy"]
```

```
## Accuracy
## 0.845
```

Por otra parte, el argumento “trControl” en la función “trainControl()” puede ser usado para cambiar el modo en que la validación cruzada se realiza. Así, podemos hacer más rápido el código anterior al usar una validación de 10 pliegues (10 muestras de validación cada una con 10 % de los datos):

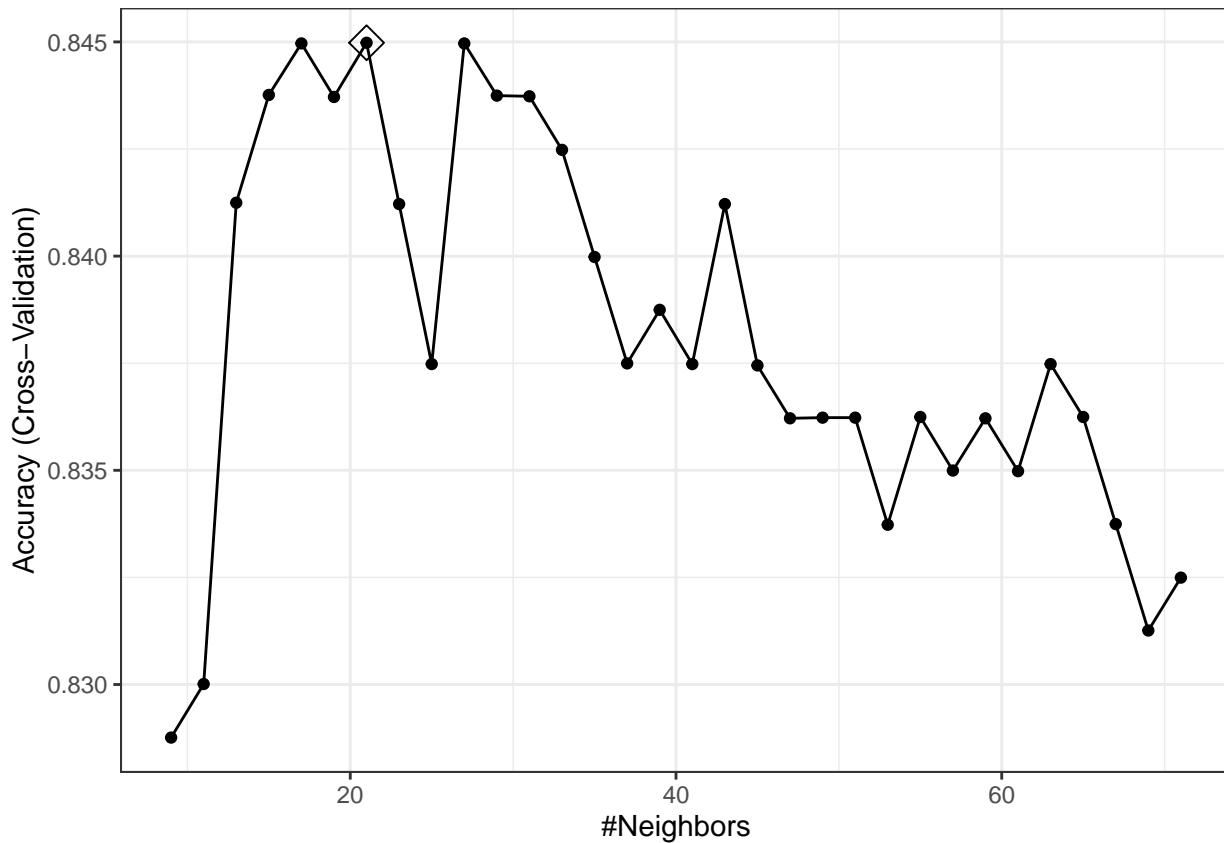
```
control <- trainControl(method = "cv", number = 10, p = .9)
```

```

train_knn_cv <- train(y ~ ., method = "knn",
                      data = mnist_27$train,
                      tuneGrid = data.frame(k = seq(9, 71, 2)),
                      trControl = control)

## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :
## non-uniform 'Rounding' sampler used
ggplot(train_knn_cv, highlight = TRUE)

```



Puede verse que esta gráfica presenta mayor variabilidad del nivel de precisión: esto es esperable dado que tomamos muestras para estimarla.

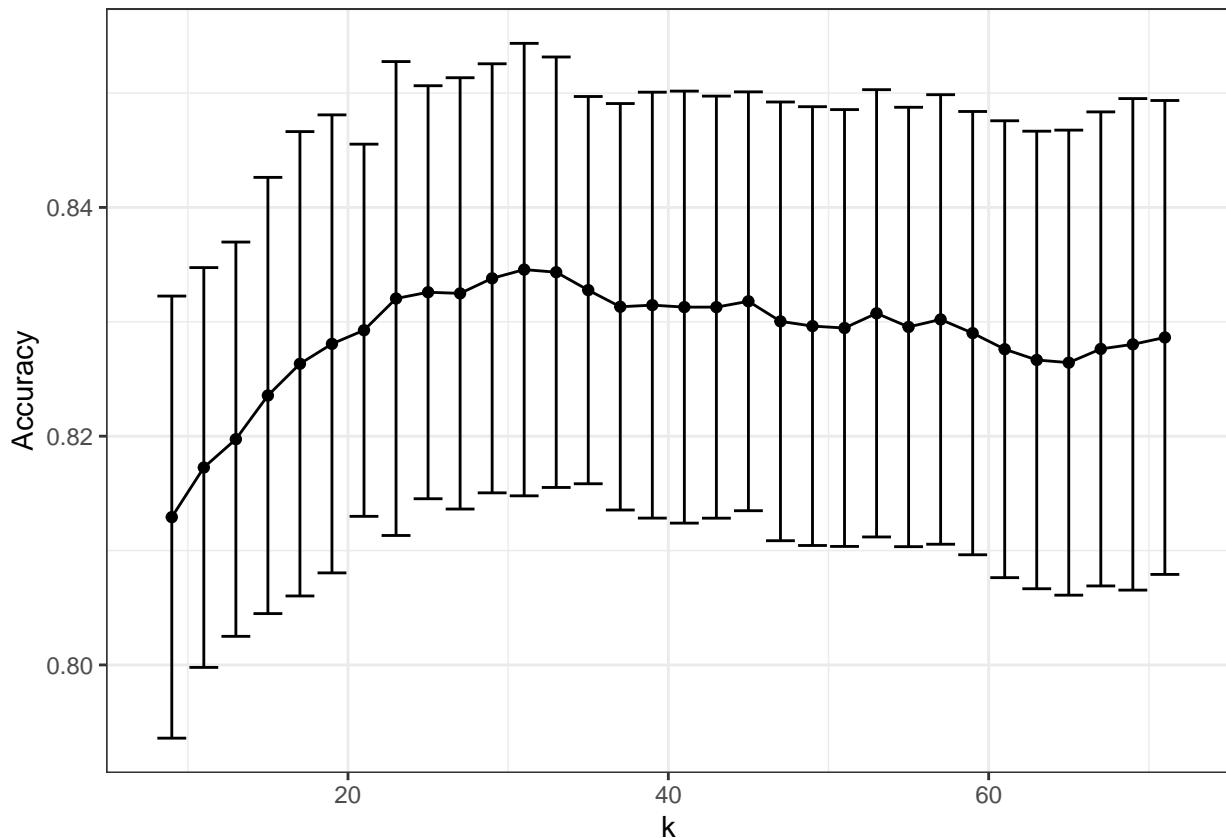
Nótese que la función “train()” también provee los valores de desviaciones estándar para cada parámetro que se probó:

```

library(tidyverse)

train_knn$results %>%
  ggplot(aes(x = k, y = Accuracy)) +
  geom_line() +
  geom_point() +
  geom_errorbar(aes(x = k,
                    ymin = Accuracy - AccuracySD,
                    ymax = Accuracy + AccuracySD))

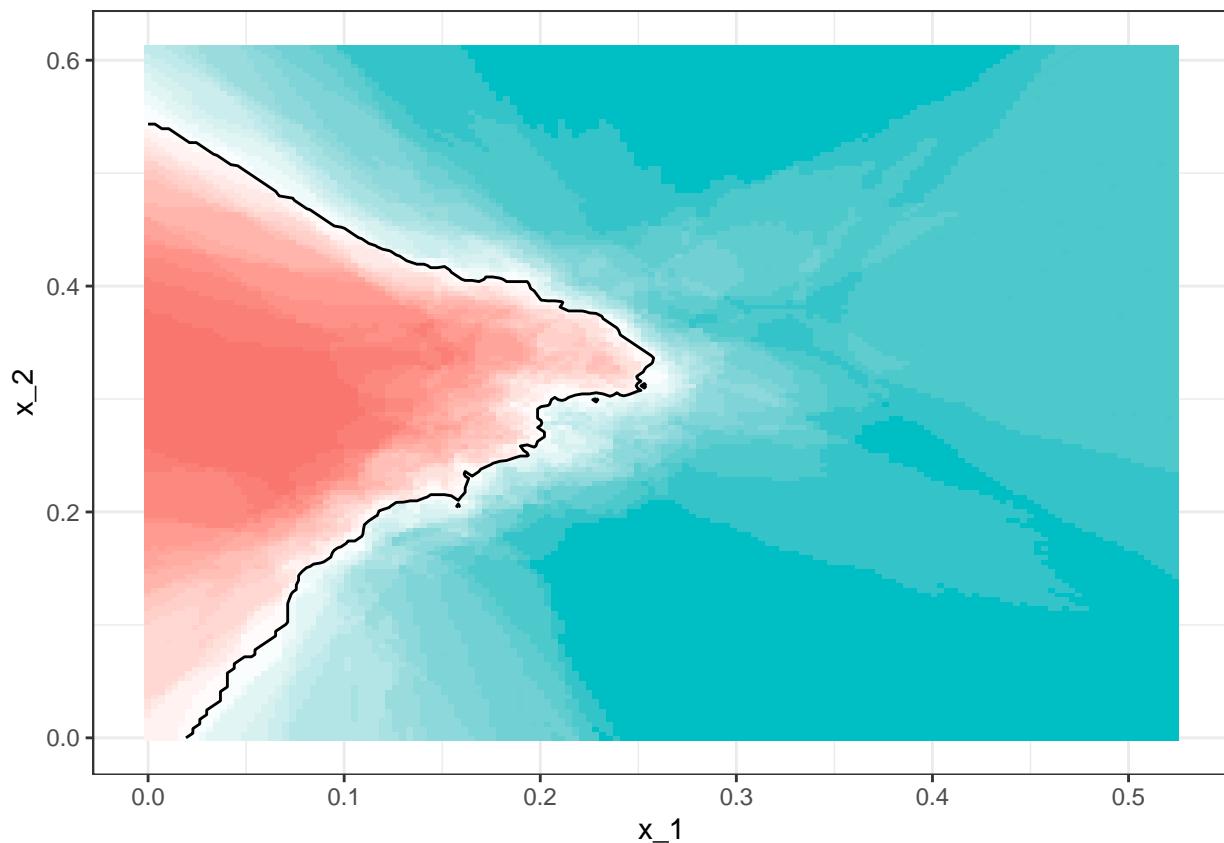
```



Para finalizar este ejemplo, nótese que el mejor modelo knn se aproxima a la probabilidad condicional bastante bien, aunque nuestra frontera es bastante ondulada:

```
plot_cond_prob <- function(p_hat=NULL){
  tmp <- mnist_27$true_p
  if(!is.null(p_hat)){
    tmp <- mutate(tmp, p=p_hat)
  }
  tmp %>% ggplot(aes(x_1, x_2, z=p, fill=p)) +
    geom_raster(show.legend = FALSE) +
    scale_fill_gradientn(colors=c("#F8766D", "white", "#00BFC4")) +
    stat_contour(breaks=c(0.5), color="black")
}

plot_cond_prob(predict(train_knn, mnist_27$true_p, type = "prob")[,2])
```



Para corregir esto, podemos usar loess, específicamente, el método gamLoess:

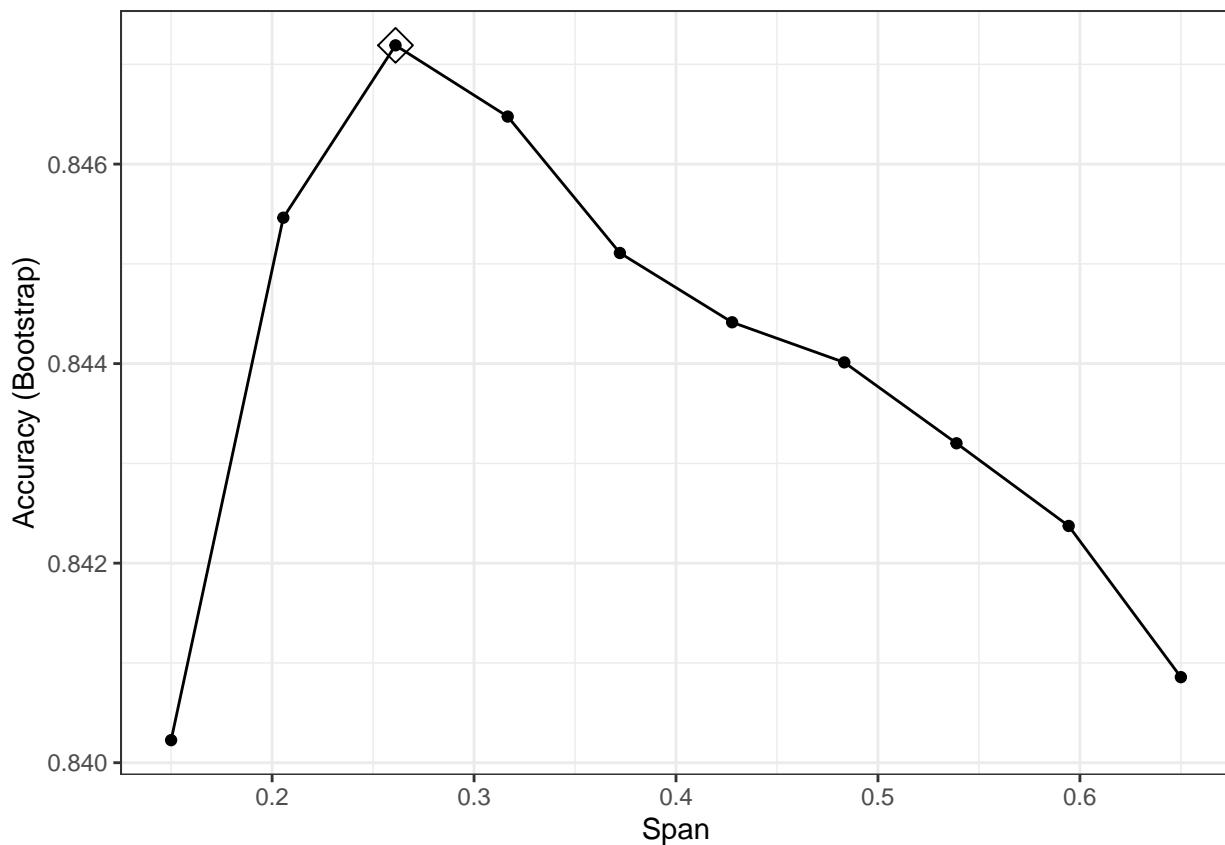
```
library(gam)
modelLookup("gamLoess")
##      model parameter  label forReg forClass probModel
## 1  gamLoess      span   Span    TRUE     TRUE    TRUE
## 2  gamLoess      degree Degree   TRUE     TRUE    TRUE
```

Donde vemos que con este modelo hay que optimizar dos parámetros, *span* y el grado. Para el siguiente ejemplo, fíjese el grado = 1. Para probar diferentes valores del *span*, tenemos que incluir una columna en la tabla con el grado:

```
grid <- expand.grid(span = seq(0.15, 0.65, len = 10), degree =1)

train_loess <- train(y~.,
                      method = "gamLoess",
                      tuneGrid = grid,
                      data = mnist_27$train)

ggplot(train_loess, highlight = TRUE)
```



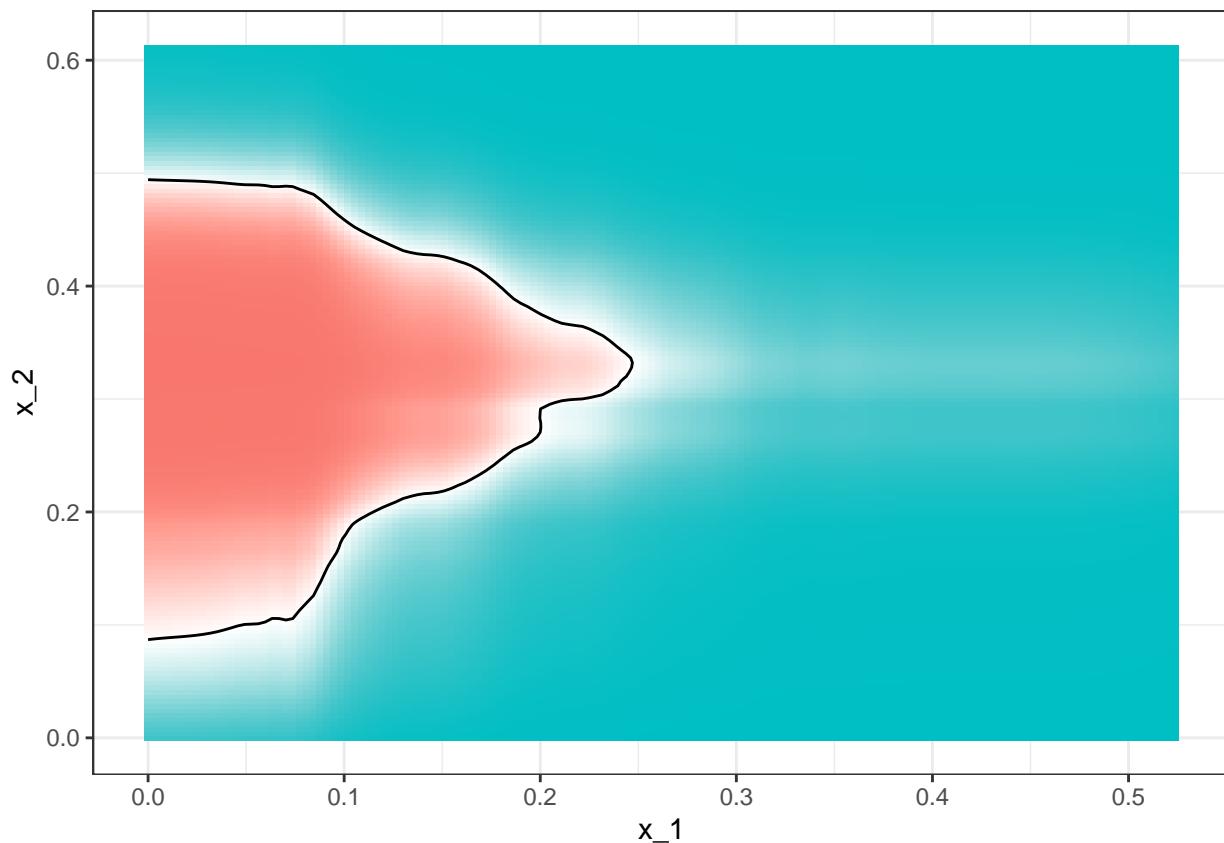
```
confusionMatrix(data = predict(train_loess, mnist_27$test),  
                 reference = mnist_27$test$y)$overall["Accuracy"]
```

```
## Accuracy  
##      0.84
```

Podemos ver que la probabilidad condicional estimada es bastante más suave que la anterior:

```
p1 <- plot_cond_prob(predict(train_loess, mnist_27$true_p, type = "prob")[,2])
```

```
p1
```



**Nota:** No todos los parámetros del aprendizaje automático se afinan. Por ejemplo, en modelos de regresión o LDA, ajustamos nuestro modelo usando estimadores cuadrados o de máxima verosimilitud.

#### 8.5.4. Ejercicios del Titanic

Use the titanic\_train data frame from the titanic library as the starting point for this project.

```
library(titanic)
library(caret)
library(tidyverse)
library(rpart)

options(digits = 3)

titanic_clean <- titanic_train %>%
  mutate(Survived = factor(Survived),
         Embarked = factor(Embarked),
         Age = ifelse(is.na(Age), median(Age, na.rm = TRUE), Age), # reemplazo de NAs
         FamilySize = SibSp + Parch + 1) %>%      # cuenta de miembros por familia
  select(Survived, Sex, Pclass, Age, Fare, SibSp, Parch, FamilySize, Embarked)
```

Split titanic\_clean into test and training sets - after running the setup code, it should have 891 rows and 9 variables. Set the seed to 42, then use the caret package to create a 20% data partition based on the Survived column. Assign the 20% partition to test\_set and the remaining 80% partition to train\_set. How many observations are in the training set?

```
set.seed(42, sample.kind = 'Rounding')

test_index <- createDataPartition(titanic_clean$Survived, times = 1, p = 0.2, list = FALSE)

train_set <- titanic_clean[-test_index,]

test_set <- titanic_clean[test_index,]

nrow(train_set)
```

## [1] 712

How many observations are in the test set?

```
set.seed(42, sample.kind = 'Rounding')

## Warning in set.seed(42, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used

nrow(test_set)
```

## [1] 179

What proportion of individuals in the training set survived?

```
set.seed(42, sample.kind = 'Rounding')

## Warning in set.seed(42, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used

mean(train_set$Survived==1)

## [1] 0.383
```

The simplest prediction method is randomly guessing the outcome without using additional predictors. These methods will help us determine whether our machine learning algorithm

performs better than chance. How accurate are two methods of guessing Titanic passenger survival? Set the seed to 3. For each individual in the test set, randomly guess whether that person survived or not by sampling from the vector c(0,1) (Note: use the default argument setting of prob from the sample function). What is the accuracy of this guessing method?

```
set.seed(3, sample.kind = 'Rounding')

guess <- sample(c(0,1), nrow(test_set), replace = TRUE)

test_set %>%
  filter(Survived == guess) %>%
  summarize(n() / nrow(test_set))

##   n()/nrow(test_set)
## 1           0.475
```

Use the training set to determine whether members of a given sex were more likely to survive or die. Apply this insight to generate survival predictions on the test set. What proportion of training set females/males survived?

```
train_set %>%
  group_by(Sex) %>%
  summarize(Survived= mean(Survived==1))

## # A tibble: 2 x 2
##   Sex     Survived
##   <chr>    <dbl>
## 1 female    0.731
## 2 male      0.197
```

Predict survival using sex on the test set: if the survival rate for a sex is over 0.5, predict survival for all individuals of that sex, and predict death if the survival rate for a sex is under 0.5. What is the accuracy of this sex-based prediction method on the test set?

```
test_set %>%
  summarize((sum(Sex == "female" & Survived == 1) + sum(Sex == "male" & Survived == 0)) / n())

##   `/~(...
## 1     0.821
```

In the training set, which class(es) (Pclass) were passengers more likely to survive than die?

```
survival_class <- titanic_clean %>%
  group_by(Pclass) %>%
  summarize(Prediction= ifelse(mean(Survived == 1) >= 0.5, 1, 0))

survival_class

## # A tibble: 3 x 2
##   Pclass Prediction
##   <int>     <dbl>
## 1     1         1
## 2     2         0
## 3     3         0
```

Predict survival using passenger class on the test set: predict survival if the survival rate for a class is over 0.5, otherwise predict death. What is the accuracy of this class-based prediction method on the test set?

```
test_set %>%
  inner_join(survival_class, by='Pclass') %>%
  summarize(Accuracy = mean(Survived == Prediction))
```

```
##   Accuracy
## 1      0.704
```

Use the training set to group passengers by both sex and passenger class. Which sex and class combinations were more likely to survive than die (i.e. >50% survival)?

```
survival_class <- titanic_clean %>%
  group_by(Sex, Pclass) %>%
  summarize(Prediction = ifelse(mean(Survived == 1) > 0.5, 1, 0))
```

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

```
## # A tibble: 6 x 3
## # Groups:   Sex [2]
##   Sex     Pclass Prediction
##   <chr>   <int>     <dbl>
## 1 female     1         1
## 2 female     2         1
## 3 female     3         0
## 4 male       1         0
## 5 male       2         0
## 6 male       3         0
```

Predict survival using both sex and passenger class on the test set. Predict survival if the survival rate for a sex/class combination is over 0.5, otherwise predict death. What is the accuracy of this sex- and class-based prediction method on the test set?

```
test_set %>%
  inner_join(survival_class, by = c("Sex", "Pclass")) %>%
  summarize(Prediction = mean(Survived == Prediction))
```

```
##   Prediction
## 1      0.821
```

Use the confusionMatrix() function to create confusion matrices for the sex model, class model, and combined sex and class model. You will need to convert predictions and survival status to factors to use this function.

```
# sex model

sex_model <- train_set %>%
  group_by(Sex) %>%
  summarize(Survived_predict = ifelse(mean(Survived == 1) > 0.5, 1, 0))
test_set1 <- test_set %>%
  inner_join(sex_model, by = 'Sex')
cm1 <- confusionMatrix(data = factor(test_set1$Survived_predict),
                        reference = factor(test_set1$Survived))
cm1 %>%
  tidy() %>%
  filter(term == 'sensitivity') %>%
  .\$estimate

## [1] 0.873
```

```
cm1 %>%
  tidy() %>%
  filter(term == 'specificity') %>%
  .$estimate

## [1] 0.739

cm1 %>%
  tidy() %>%
  filter(term == 'balanced_accuracy') %>%
  .$estimate

## [1] 0.806

cm1

## Confusion Matrix and Statistics
##
##             Reference
## Prediction 0 1
##           0 96 18
##           1 14 51
##
##           Accuracy : 0.821
##           95% CI : (0.757, 0.874)
##   No Information Rate : 0.615
##   P-Value [Acc > NIR] : 1.72e-09
##
##           Kappa : 0.619
##
##   Mcnemar's Test P-Value : 0.596
##
##           Sensitivity : 0.873
##           Specificity : 0.739
##   Pos Pred Value : 0.842
##   Neg Pred Value : 0.785
##           Prevalence : 0.615
##   Detection Rate : 0.536
##   Detection Prevalence : 0.637
##   Balanced Accuracy : 0.806
##
##   'Positive' Class : 0
##

# class model

class_model <- train_set %>%
  group_by(Pclass) %>%
  summarize(Survived_predict = ifelse(mean(Survived == 1) > 0.5, 1, 0))
test_set2 <- test_set %>%
  inner_join(class_model, by = 'Pclass')
cm2 <- confusionMatrix(data = factor(test_set2$Survived_predict),
                        reference = factor(test_set2$Survived))
cm2 %>%
  tidy() %>%
  filter(term == 'sensitivity') %>%
```

```
.$estimate

## [1] 0.855
cm2 %>%
  tidy() %>%
  filter(term == 'specificity') %>%
  .$estimate

## [1] 0.464
cm2 %>%
  tidy() %>%
  filter(term == 'balanced_accuracy') %>%
  .$estimate

## [1] 0.659
cm2

## Confusion Matrix and Statistics
##
##           Reference
## Prediction 0 1
##          0 94 37
##          1 16 32
##
##           Accuracy : 0.704
##                 95% CI : (0.631, 0.77)
##      No Information Rate : 0.615
##      P-Value [Acc > NIR] : 0.00788
##
##           Kappa : 0.337
##
## McNemar's Test P-Value : 0.00601
##
##           Sensitivity : 0.855
##           Specificity : 0.464
##      Pos Pred Value : 0.718
##      Neg Pred Value : 0.667
##           Prevalence : 0.615
##      Detection Rate : 0.525
## Detection Prevalence : 0.732
##    Balanced Accuracy : 0.659
##
##    'Positive' Class : 0
##
# sex and class model

sex_class_model <- train_set %>%
  group_by(Sex, Pclass) %>%
  summarize(Survived_predict = ifelse(mean(Survived == 1) > 0.5, 1, 0))

## `summarise()` has grouped output by 'Sex'. You can override using the `.`groups` argument.
test_set3 <- test_set %>%
  inner_join(sex_class_model, by=c('Sex', 'Pclass'))
```

```

cm3 <- confusionMatrix(data = factor(test_set3$Survived_predict),
                        reference = factor(test_set3$Survived))
cm3 %>%
  tidy() %>%
  filter(term == 'sensitivity') %>%
  .\$estimate

## [1] 0.991

cm3 %>%
  tidy() %>%
  filter(term == 'specificity') %>%
  .\$estimate

## [1] 0.551

cm3 %>%
  tidy() %>%
  filter(term == 'balanced_accuracy') %>%
  .\$estimate

## [1] 0.771

cm3

## Confusion Matrix and Statistics
##
##             Reference
## Prediction    0     1
##           0 109   31
##           1     1   38
##
##             Accuracy : 0.821
##                 95% CI : (0.757, 0.874)
##      No Information Rate : 0.615
##      P-Value [Acc > NIR] : 1.72e-09
##
##             Kappa : 0.589
##
## McNemar's Test P-Value : 2.95e-07
##
##             Sensitivity : 0.991
##             Specificity  : 0.551
##      Pos Pred Value : 0.779
##      Neg Pred Value : 0.974
##          Prevalence : 0.615
##      Detection Rate : 0.609
## Detection Prevalence : 0.782
##      Balanced Accuracy : 0.771
##
##      'Positive' Class : 0
##

```

Use the `F_meas()` function to calculate scores for the sex model, class model, and combined sex and class model. You will need to convert predictions to factors to use this function. Which model has the highest  $F_1$  score?

```
F_meas(data=factor(test_set1$Survived), reference = factor(test_set1$Survived_predict))
## [1] 0.857
F_meas(data=factor(test_set2$Survived), reference = factor(test_set2$Survived_predict))
## [1] 0.78
F_meas(data=factor(test_set3$Survived), reference = factor(test_set3$Survived_predict))
## [1] 0.872
```

Set the seed to 1. Train a model using linear discriminant analysis (LDA) with the caret lda method using fare as the only predictor. What is the accuracy on the test set for the LDA model?

```
fit_lda <- train(Survived ~ Fare, data = train_set, method = 'lda')

Survived_hat <- predict(fit_lda, test_set)

mean(test_set$Survived == Survived_hat)
## [1] 0.693
```

Set the seed to 1. Train a model using quadratic discriminant analysis (QDA) with the caret qda method using fare as the only predictor. What is the accuracy on the test set for the QDA model?

```
fit_qda <- train(Survived ~ Fare, data = train_set, method = 'qda')

Survived_hat <- predict(fit_qda, test_set)

mean(test_set$Survived == Survived_hat)
## [1] 0.693
```

Set the seed to 1. Train a logistic regression model with the caret glm method using age as the only predictor. What is the accuracy of your model (using age as the only predictor) on the test set ?

```
fit_logreg_a <- glm(Survived ~ Age, data = train_set, family = 'binomial')

survived_hat_a <- ifelse(predict(fit_logreg_a, test_set) >= 0, 1, 0)

mean(survived_hat_a == test_set$Survived)
## [1] 0.615
```

Set the seed to 1. Train a logistic regression model with the caret glm method using four predictors: sex, class, fare, and age. What is the accuracy of your model (using these four predictors) on the test set ?

```
fit_logreg_b <- glm(Survived ~ Sex + Pclass + Fare + Age,
                      data = train_set, family = 'binomial')

survived_hat_b <- ifelse(predict(fit_logreg_b, test_set) >= 0, 1, 0)

mean(survived_hat_b == test_set$Survived)
## [1] 0.849
```

Set the seed to 1. Train a logistic regression model with the caret `glm` method using all predictors. Ignore warnings about rank-deficient fit. What is the accuracy of your model (using all predictors) on the test set ?

```
str(train_set)

## 'data.frame':    712 obs. of  9 variables:
## $ Survived : Factor w/ 2 levels "0","1": 2 2 1 2 2 2 2 1 1 1 ...
## $ Sex      : chr  "female" "female" "male" "female" ...
## $ Pclass   : int  3 1 3 3 2 3 1 3 3 3 ...
## $ Age     : num  26 35 2 27 14 4 58 20 39 14 ...
## $ Fare    : num  7.92 53.1 21.07 11.13 30.07 ...
## $ SibSp   : int  0 1 3 0 1 1 0 0 1 0 ...
## $ Parch   : int  0 0 1 2 0 1 0 0 5 0 ...
## $ FamilySize: num  1 2 5 3 2 3 1 1 7 1 ...
## $ Embarked : Factor w/ 4 levels "", "C", "Q", "S": 4 4 4 4 2 4 4 4 4 4 ...
fit_logreg_c <- glm(Survived ~ ., data = train_set, family = 'binomial')

survived_hat_c <- ifelse(predict(fit_logreg_c, test_set) >= 0, 1, 0)

mean(survived_hat_c == test_set$Survived)

## [1] 0.849
```

Set the seed to 6. Train a kNN model on the training set using the caret `train` function. Try tuning with `k = seq(3, 51, 2)`. What is the optimal value of the number of neighbors `k`?

```
set.seed(6, sample.kind = "Rounding")

## Warning in set.seed(6, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
k <- seq(3, 51, 2)

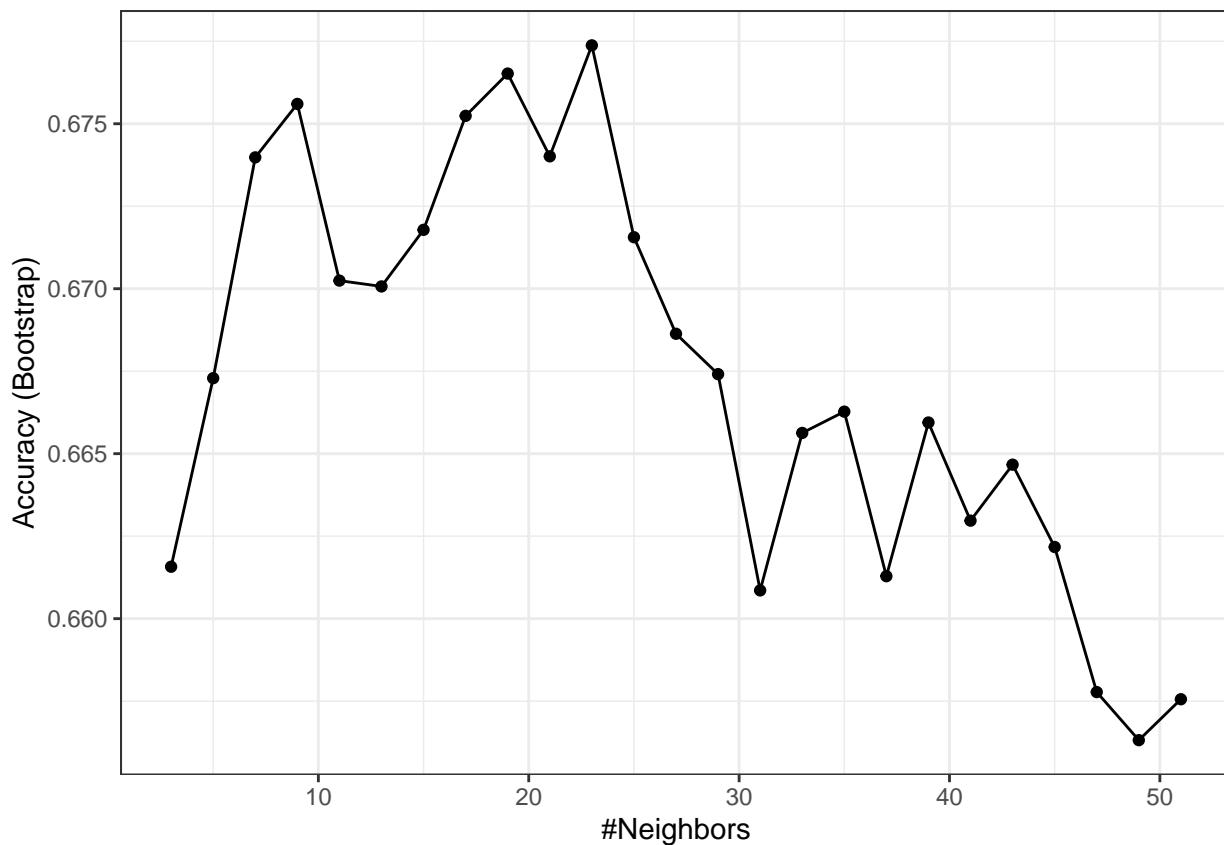
fit_knn <- train(Survived~.,
                  data = train_set,
                  method = "knn",
                  tuneGrid = data.frame(k))

## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :
## non-uniform 'Rounding' sampler used
fit_knn$bestTune

##      k
## 11 23
```

lot the kNN model to investigate the relationship between the number of neighbors and accuracy on the training set. Of these values of `k`, which yields the highest accuracy?

```
ggplot(fit_knn)
```



What is the accuracy of the kNN model on the test set?

```
survived_hat <- predict(fit_knn, test_set) %>%
  factor(levels = levels(test_set$Survived))

cm_test <- confusionMatrix(data = survived_hat, reference = test_set$Survived)

cm_test$overall["Accuracy"]

## Accuracy
## 0.726
```

Set the seed to 8 and train a new kNN model. Instead of the default training control, use 10-fold cross-validation where each partition consists of 10% of the total. Try tuning with  $k = \text{seq}(3, 51, 2)$ . What is the optimal value of  $k$  using cross-validation?

```
set.seed(8, sample.kind = "Rounding")

fit_knn <- train(Survived ~ .,
  data=train_set,
  method = "knn",
  tuneGrid = data.frame(k = seq(3, 51, 2)),
  trControl = trainControl(method = "cv", number=10, p=0.9))

survived_hat <- predict(fit_knn, test_set)

cm_test <- confusionMatrix(data = survived_hat, reference = test_set$Survived)
```

```
fit_knn$bestTune
```

```
##   k
## 2 5
```

What is the accuracy on the test set using the cross-validated kNN model?

```
cm_test$overall["Accuracy"]
```

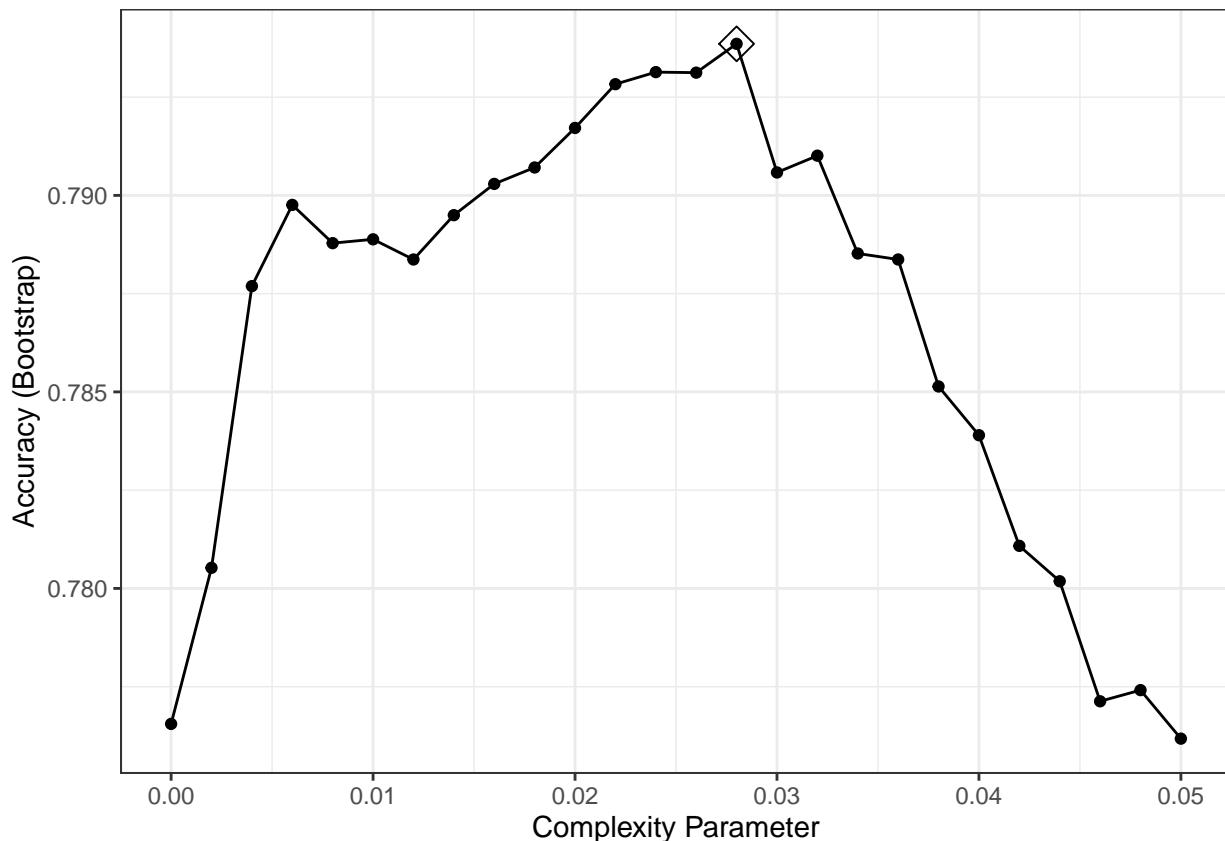
```
## Accuracy
## 0.648
```

Set the seed to 10. Use caret to train a decision tree with the rpart method. Tune the complexity parameter with `cp = seq(0, 0.05, 0.002)`. What is the optimal value of the complexity parameter (`cp`)?

```
set.seed(10, sample.kind = 'Rounding')

fit_rpart <- train(Survived ~.,
                     data=train_set,
                     method = "rpart",
                     tuneGrid = data.frame(cp = seq(0, 0.05, 0.002)))

ggplot(fit_rpart, highlight = TRUE)
```



```
survived_hat <- predict(fit_rpart, test_set)
```

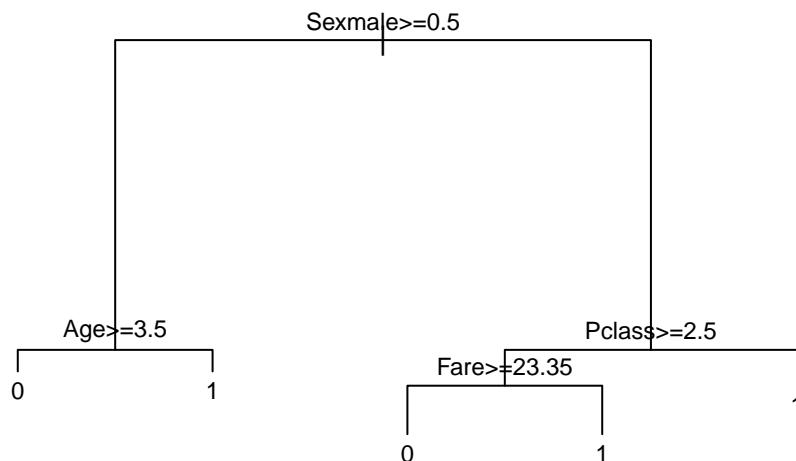
```
cm_test <- confusionMatrix(data = survived_hat, reference = test_set$Survived)
```

```
cm_test$overall["Accuracy"]

## Accuracy
##      0.838

inspect the final model and plot the decision tree. Which variables are used in the decision tree?
plot(fit_rpart$finalModel, margin=0.1)

text(fit_rpart$finalModel, cex = 0.75)
```



Set the seed to 14. Use the caret train() function with the rf method to train a random forest. Test values of mtry = seq(1:7). Set ntree to 100.

What mtry value maximizes accuracy? What is the accuracy of the random forest model on the test set? Use varImp() on the random forest model object to determine the importance of various predictors to the random forest model. What is the most important variable?

```
set.seed(14, sample.kind = 'Rounding')

## Warning in set.seed(14, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used

fit_rf <- train(Survived ~.,
                  data = train_set,
                  method = "rf",
                  tuneGrid = data.frame(mtry = seq(1, 7)),
```

```
ntree = 100)

## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :
## non-uniform 'Rounding' sampler used
fit_rf$bestTune

## mtry
## 3    3
survived_hat <- predict(fit_rf, test_set)

mean(survived_hat == test_set$Survived)

## [1] 0.86
varImp(fit_rf)

## rf variable importance
##
##          Overall
## Sexmale     100.000
## Fare        74.434
## Age         59.531
## Pclass      27.605
## FamilySize  19.358
## SibSp       9.435
## Parch       5.889
## EmbarkedS   4.288
## EmbarkedC   0.623
## EmbarkedQ   0.000
```

## 8.6. Ajuste de modelos y sistemas de recomendación

### 8.6.1. Caso de estudio: MNIST

A continuación, usaremos lo aprendido de aprendizaje automático en una base de datos real, la *Modified National Institute of Standards and Technology database (MNIST)*:

```
library(dslabs)
```

```
mnist <- read_mnist()
```

Esta base de datos ya incluye dos componentes importantes: el conjunto de práctica y el de prueba:

```
names(mnist)
```

```
## [1] "train" "test"
```

Para fines prácticos, consideraremos un subconjunto de datos del total: tomaremos una muestra de 10 000 renglones aleatorios para el conjunto de práctica y 1 000 para el de prueba:

```
set.seed(123)
```

```
index <- sample(nrow(mnist$train$images), 10000)
```

```
x <- mnist$train$images[index, ]
```

```
y <- factor(mnist$train$labels[index])
```

```
index <- sample(nrow(mnist$test$images), 1000)
```

```
x_test <- mnist$test$images[index, ]
```

```
y_test <- factor(mnist$test$labels[index])
```

Los pasos pre-procesamiento comunes incluyen:

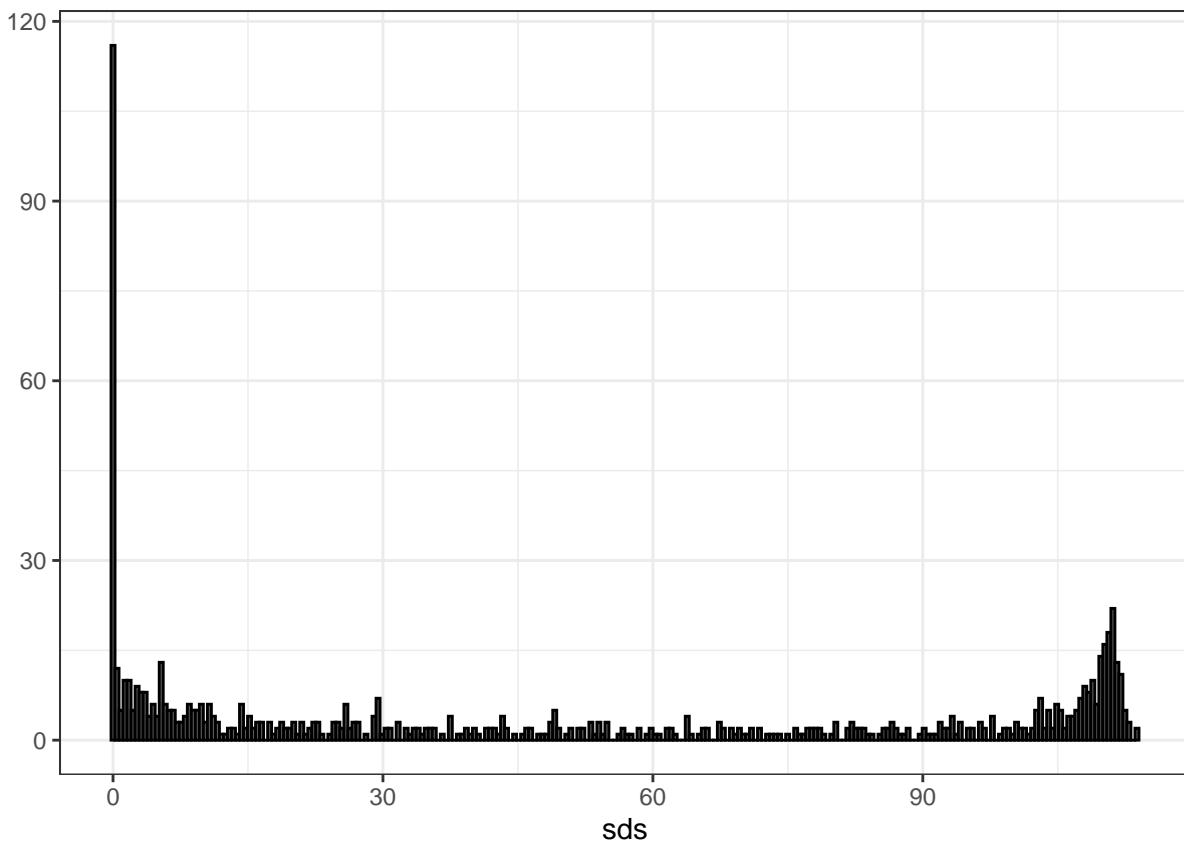
1. Estandarizar o transformar predictores; y
2. Remover predictores que no son útiles, altamente correlacionados, con pocos valores únicos o con varianza cercana a 0.

Considérese el caso con una varianza muy pequeña: podemos ver que hay muchas características con variabilidad cero:

```
library(matrixStats)
library(tidyverse)
```

```
sds <- colSds(x)
```

```
qplot(sds, bins = 256, color = I("black"))
```



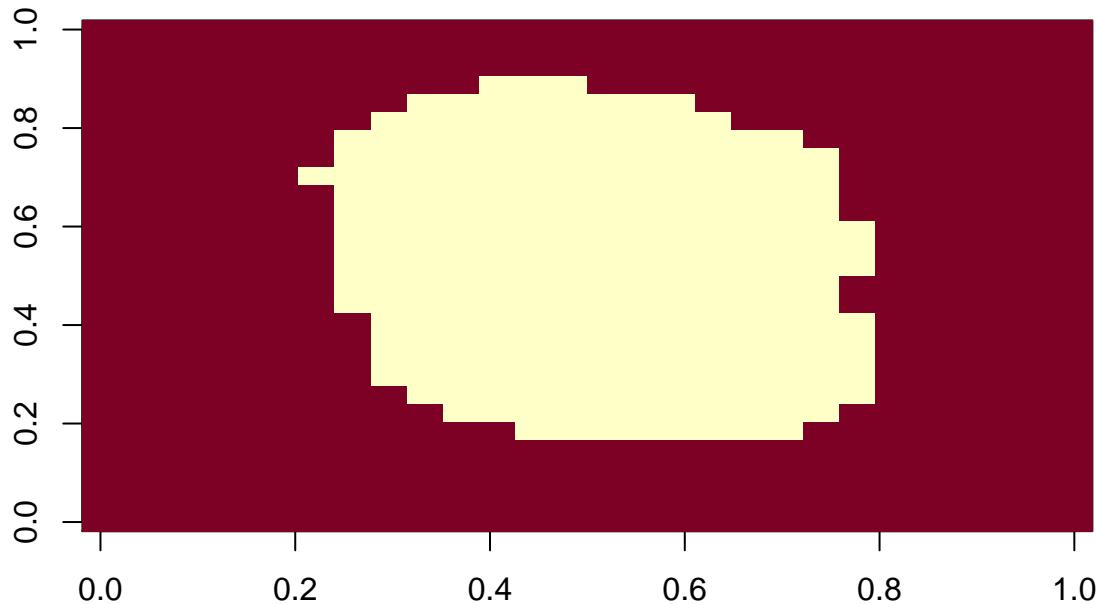
La librería caret incluye una función que recomienda características a remover dada su varianza cercana a 0:

```
library(caret)
```

```
nzv <- nearZeroVar(x)
```

Podemos ver las columnas que se remueven como sigue:

```
image(matrix(1:784 %in% nzv, 28, 28))
```



Una vez removidas, conservamos el siguiente número de columnas:

```
col_index <- setdiff(1:ncol(x), nzv)
```

```
length(col_index)
```

```
## [1] 252
```

Antes de comenzar el proceso de ajuste de modelos, debemos agregar nombres de columnas a las matrices, dado que es un requisito del paquete caret:

```
colnames(x) <- 1:ncol(mnist$train$images)
```

```
colnames(x_test) <- colnames(mnist$train$images)
```

**8.6.1.1. kNN** Para comenzar, debemos optimizar el número de vecindades. Tómese en cuenta que, al momento de correr el algoritmo, deberemos computar la distancia entre cada observación del conjunto de práctica y de prueba (lo cual implican muchas operaciones). Por tanto, usaremos una validación cruzada de k pliegues para mejorar la velocidad del cálculo (nótese que el siguiente código tomará bastante tiempo para correr):

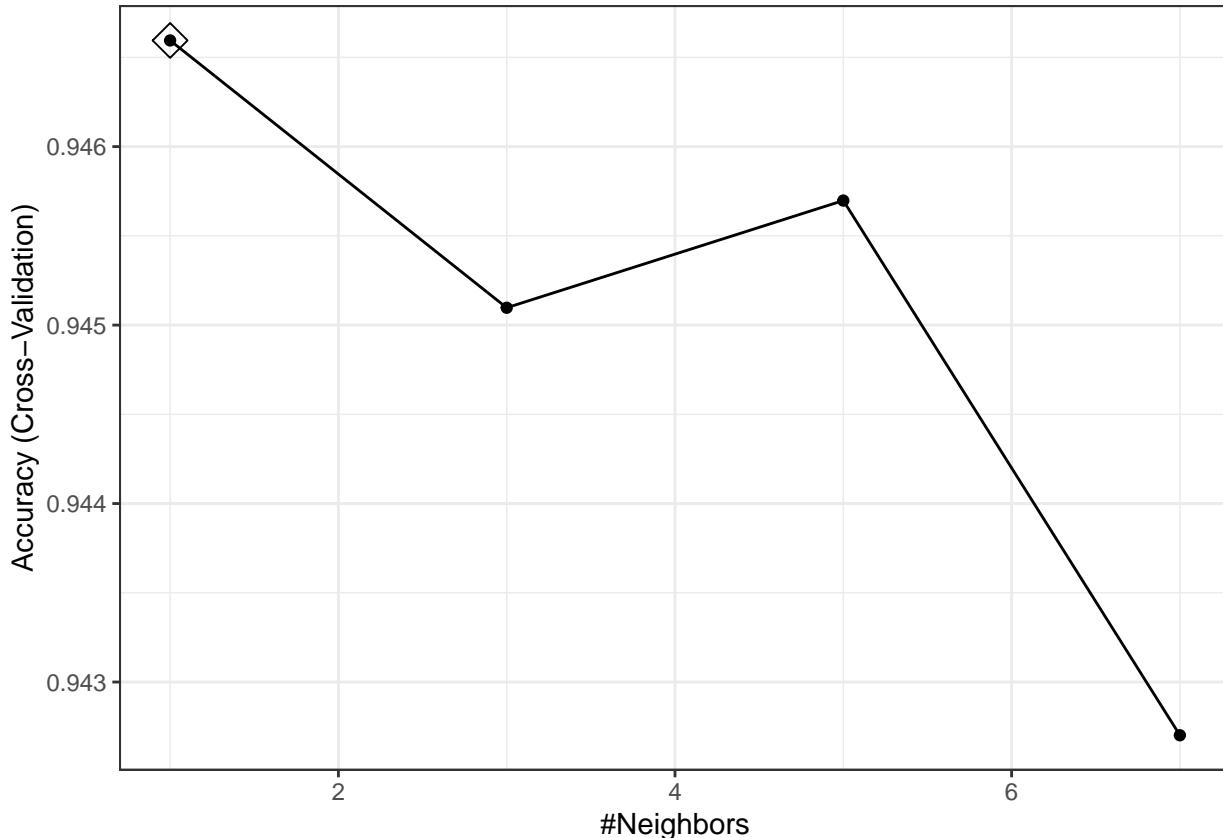
```
control <- trainControl(method = "cv", number = 10, p = .9)
```

```
train_knn <- train(x[,col_index], y,
                     method = "knn",
                     tuneGrid = data.frame(k = c(1,3,5,7)),
                     trControl = control)
```

```
## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :
## non-uniform 'Rounding' sampler used
```

Así, podemos visualizar el modelo (número de vecindades) que maximizan la precisión:

```
ggplot(train_knn, highlight = TRUE)
```



Como alternativa, puede hacerse el mismo análisis para una muestra más pequeña y con menos pliegues:

```
n <- 1000

b <- 2

index <- sample(nrow(x), n)

control <- trainControl(method = "cv", number = b, p = 0.9)

train_knn <- train(x[index, col_index], y[index],
                     method = "knn",
                     tuneGrid = data.frame(k=c(2,5,7)),
                     trControl = control)

## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :
## non-uniform 'Rounding' sampler used
```

```
ggplot(train_knn, highlight = TRUE)
```



Es recomendable usar el código anterior e ir probando diferentes valores para  $n$  y  $b$ , de tal manera que nos podamos hacer una idea de cuánto llevará correr el código en todos los datos.

Así, una vez que optimizamos nuestro algoritmo, podemos probarlo en la base de datos completa:

```
fit_knn <- knn3(x[,col_index], y, k =3)
```

Donde vemos que nuestra precisión es bastante alta:

```
y_hat_knn <- predict(fit_knn,
                      x_test[,col_index],
                      type = "class")

cm <- confusionMatrix(y_hat_knn, factor(y_test))

## Confusion Matrix and Statistics
##          Reference
## Prediction  0   1   2   3   4   5   6   7   8   9
##           0 102   0   1   0   0   0   0   0   0   1
##           1   0 121   1   0   1   1   1   3   0   0
##           2   0   0 101   1   0   0   0   0   0   0
##           3   0   0   0  77   0   1   0   0   4   1
##           4   0   0   0   0 102   0   0   1   3   0
```

```

##      5   0   0   0   2   0   68   0   0   6   0
##      6   0   0   1   0   1   0 101   0   0   0
##      7   0   0   2   2   0   0   0 104   0   1
##      8   0   0   0   2   0   0   0   0   77   0
##      9   0   0   0   0   5   0   0   2   1 102
##
## Overall Statistics
##
##          Accuracy : 0.955
## 95% CI : (0.94, 0.967)
## No Information Rate : 0.121
## P-Value [Acc > NIR] : <2e-16
##
##          Kappa : 0.95
##
## McNemar's Test P-Value : NA
##
## Statistics by Class:
##
##          Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity      1.000      1.000      0.953      0.917      0.936      0.971
## Specificity      0.998      0.992      0.999      0.993      0.996      0.991
## Pos Pred Value    0.981      0.945      0.990      0.928      0.962      0.895
## Neg Pred Value    1.000      1.000      0.994      0.992      0.992      0.998
## Prevalence        0.102      0.121      0.106      0.084      0.109      0.070
## Detection Rate    0.102      0.121      0.101      0.077      0.102      0.068
## Detection Prevalence  0.104      0.128      0.102      0.083      0.106      0.076
## Balanced Accuracy   0.999      0.996      0.976      0.955      0.966      0.981
##
##          Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity      0.990      0.945      0.846      0.971
## Specificity      0.998      0.994      0.998      0.991
## Pos Pred Value    0.981      0.954      0.975      0.927
## Neg Pred Value    0.999      0.993      0.985      0.997
## Prevalence        0.102      0.110      0.091      0.105
## Detection Rate    0.101      0.104      0.077      0.102
## Detection Prevalence  0.103      0.109      0.079      0.110
## Balanced Accuracy   0.994      0.970      0.922      0.981

```

De la especificidad y sensibilidad mostradas, podemos ver que los 8s son los más difíciles de detectar, mientras que el dígito detectado incorrectamente más veces son los 7s. Veamos si podemos hacerlo mejor con bosques aleatorios; bajo esta metodología, el tiempo de cómputo de R se incrementa aún más, dado que construiremos muchos árboles y podemos afinar muchos parámetros.

En comparación, el paquete Rborist tiene menos utilidades pero es más rápido que randomForest. Para este ejemplo, usaremos una validación de 5 pliegues con el primer paquete; también reduciremos el número de árboles que ajustemos y tomaremos un subconjunto de observaciones para hacer cada árbol:

```

#control <- trainControl(method = "cv", number = 5, p = 0.8)

#grid <- expand.grid(minNode = c(1,5), predFixed = c(10, 15, 25, 35, 50))

#train_rf <- train(x[,col_index],
#  # y,
#  # method = "Rborist",
#  # nTree = 50,

```

```
# trControl = control,
# tuneGrid = grid,
# nSamp = 5000)
```

Podemos ver el resultado final con ggplot:

```
#ggplot(train_rf)
```

Y podemos elegir los mejores parámetros:

```
#train_rf$bestTune
```

Ahora, estamos listos para optimizar nuestro árbol final:

```
#fit_rf <- Rborist(x[, col_index], y,
#      nTree = 1000,
#      minNode = train_rf$bestTune$minNode,
#      predFixed = train_rf$bestTune$predFixed)
```

Una vez que se completó la evaluación del código anterior, podemos ver nuestra precisión a través de la matriz de confusión, la cual demuestra que la mejoramos sobre el método de vecindades cercanas:

```
#y_hat_rf <- factor(levels(y)[predict(fit_rf, x_test[, col_index])$yPred])

#cm <- confusionMatrix(y_hat_rf, y_test)

#cm$overall["Accuracy"]
```

Como comentamos, una limitación de los bosques aleatorios es que no son muy intepretables; sin embargo, el concepto de **importancia de la variable** ayuda un poco.

Para ver esto, usaremos el paquete randomForest:

```
library(randomForest)

x <- mnist$train$images[,]

y <- factor(mnist$train$labels[,])

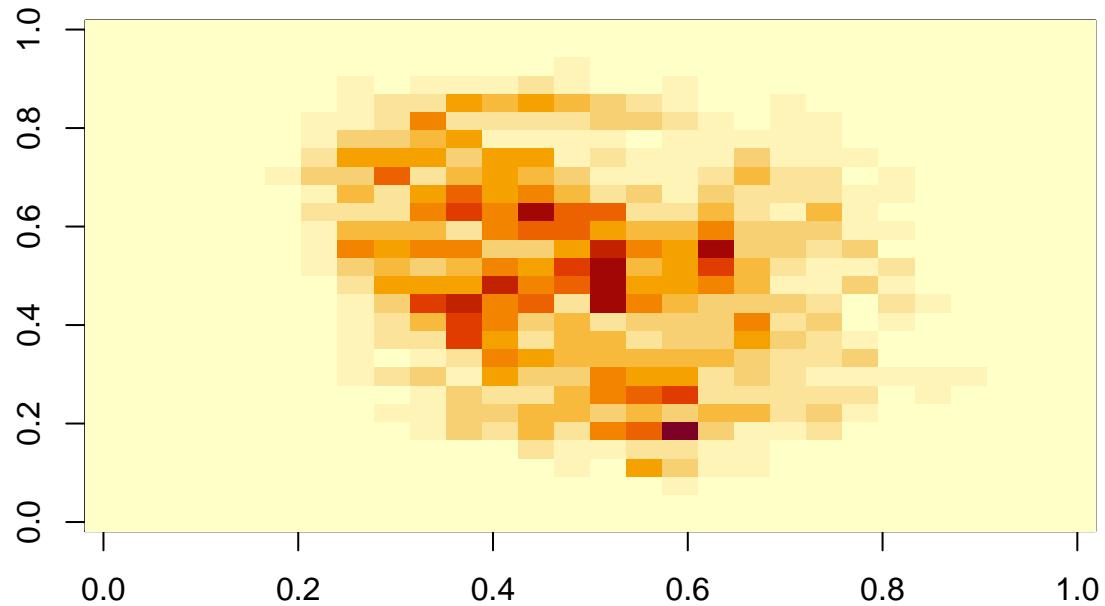
rf <- randomForest(x, y, ntree = 50)
```

Así, podemos calcular la importancia de cada característica:

```
imp <- importance(rf)
```

En este ejemplo en particular, es útil explorar la importancia mediante una imagen. En la siguiente, cada característica está graficada en el lugar de la imagen de donde :

```
image(matrix(imp, 28, 28))
```



Como vimos, incluso con una precisión alta cometimos errores. Podemos comparar los resultados de las vecindades cercanas y los de bosques aleatorios

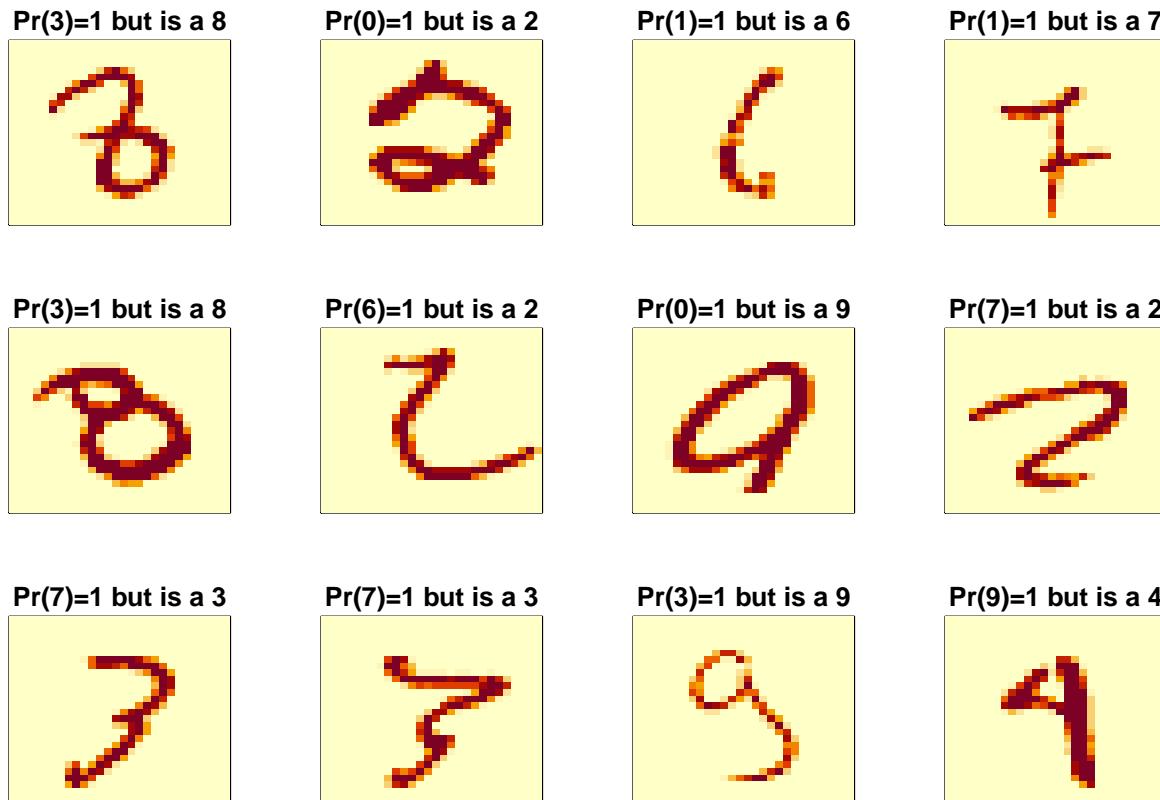
```
p_max <- predict(fit_knn, x_test[,col_index])

p_max <- apply(p_max, 1, max)

ind <- which(y_hat_knn != y_test)

ind <- ind[order(p_max[ind], decreasing = TRUE)]

rafalib::mypar(3,4)
for(i in ind[1:12]){
  image(matrix(x_test[i,], 28, 28)[, 28:1],
        main = paste0("Pr(",y_hat_knn[i],"=",round(p_max[i], 2),
                      " but is a ",y_test[i]),
        xaxt="n", yaxt="n")
}
```



En el caso de bosques aleatorios:

```
#p_max <- predict(fit_rf, x_test[,col_index])$census

#p_max <- p_max / rowSums(p_max)

#p_max <- apply(p_max, 1, max)

#ind <- which(y_hat_rf != y_test)

#ind <- ind[order(p_max[ind], decreasing = TRUE)]

#rafalib::mypad(3,4)
#for(i in ind[1:12]){
#  image(matrix(x_test[i,], 28, 28)[, 28:1],
#        main = paste0("Pr(",y_hat_rf[i],"=",round(p_max[i], 2),
#                     " but is a ",y_test[i]),
#        xaxt="n", yaxt="n")
#}
```

Una idea muy poderosa en el aprendizaje automático es la de **ensamblar** diferentes algoritmos en un solo con el objetivo de mejorar las predicciones. Como ejemplo, combinemos los bosques aleatorios y las vecindades cercanas:

```
#p_rf <- predict(fit_rf, x_test[,col_index])$census

#p_rf <- p_rf / rowSums(p_rf)
```

```
#p_knn <- predict(fit_knn, x_test[,col_index])  
  
#p <- (p_rf + p_knn)/2  
  
#y_pred <- factor(apply(p, 1, which.max)-1)  
  
#confusionMatrix(y_pred, y_test)
```

En la práctica, se pueden ensamblar decenas o cientos de diferentes métodos, lo cual puede llevar a beneficios y mejoras sustanciales en nuestras estimaciones.

### 8.6.2. Assessment 30

Use the training set to build a model with several of the models available from the caret package. We will test out 10 of the most common machine learning models in this exercise:

```
models <- c("glm", "lda", "naive_bayes", "svmLinear", "knn", "gamLoess", "multinom", "qda", "rf")
```

Apply all of these models using train() with all the default parameters. You may need to install some packages. Keep in mind that you will probably get some warnings. Also, it will probably take a while to train all of the models - be patient! Run the following code to train the various models:

Now that you have all the trained models in a list, use sapply() or map() to create a matrix of predictions for the test set. You should end up with a matrix with length(mnist\_27test\$y) rows and length(models) columns. What are the dimensions of the matrix of predictions?

```
models_y_hat <- sapply(fits, function(fit_model){  
  predict(fit_model, mnist_27$test)  
})
```

```
dim(models_y_hat)
```

```
## [1] 200 9
```

Report the mean accuracy across all models.

```
model_accuracies <- colMeans(models_y_hat == mnist_27$test$y)
```

```
model_accuracies
```

	glm	lda	naive_bayes	svmLinear	knn	gamLoess
##	0.750	0.750	0.795	0.755	0.840	0.845
##	multinom	qda	rf			
##	0.750	0.820	0.780			

```
mean(model_accuracies)
```

```
## [1] 0.787
```

Next, build an ensemble prediction by majority vote and compute the accuracy of the ensemble. Vote 7 if more than 50% of the models are predicting a 7, and 2 otherwise. What is the accuracy of the ensemble?

```
df<-as.data.frame(table(models_y_hat[,]), stringsAsFactors =TRUE)
```

```
df$Var1
```

```
## [1] 2  
## Levels: 2
```

```
y_hat_maj <- sapply(seq(1,nrow(models_y_hat)), function(index_line){  
  df <- as.data.frame(table(models_y_hat[index_line,]))  
  df[which.max(df$Freq),]$Var1  
})
```

```
mean(y_hat_maj == mnist_27$test$y)
```

```
## [1] 0.805
```

How many of the individual methods do better than the ensemble?

```
model_indices <- model_accuracies > mean(models_y_hat == mnist_27$test$y)
sum(model_indices)

## [1] 4
models[model_indices]

## [1] "naive_bayes" "knn"          "gamLoess"     "qda"
```

It is tempting to remove the methods that do not perform well and re-do the ensemble. The problem with this approach is that we are using the test data to make a decision. However, we could use the minimum accuracy estimates obtained from cross validation with the training data for each model from `fit$results$Accuracy`. Obtain these estimates and save them in an object. Report the mean of these training set accuracy estimates. What is the mean of these training set accuracy estimates?

```
acc_hat <- sapply(fits, function(fit) min(fit$results$Accuracy))

mean(acc_hat)

## [1] 0.807
```

Now let's only consider the methods with a minimum accuracy estimate of greater than or equal to 0.8 when constructing the ensemble. Vote 7 if 50% or more of those models are predicting a 7, and 2 otherwise. What is the accuracy of the ensemble now?

```
ind <- acc_hat >= 0.8

votes <- rowMeans(models_y_hat[,ind] == "7")

y_hat <- ifelse(votes>=0.5, 7, 2)

mean(y_hat == mnist_27$test$y)

## [1] 0.83
```

### 8.6.3. Sistemas de recomendación

Un **sistema de recomendación** usa *ratings* que los usuarios proporcionan con el objetivo de hacer recomendaciones. Como ejemplo, considérese un sistema de recomendación de películas:

```
library(dslabs)
```

```
library(tidyverse)
```

```
data("movielens")
```

Podemos ver que los datos están en formato *tidy*:

```
head(movielens)
```

```
##   movieId                      title    year
## 1      31             Dangerous Minds 1995
## 2     1029                      Dumbo 1941
## 3     1061                     Sleepers 1996
## 4     1129        Escape from New York 1981
## 5    1172 Cinema Paradiso (Nuovo cinema Paradiso) 1989
## 6     1263           Deer Hunter, The 1978
##   genres userId rating timestamp
## 1   Drama     1    2.5 1260759144
## 2 Animation|Children|Drama|Musical     1    3.0 1260759179
## 3          Thriller     1    3.0 1260759182
## 4 Action|Adventure|Sci-Fi|Thriller     1    2.0 1260759185
## 5          Drama     1    4.0 1260759205
## 6    Drama|War     1    2.0 1260759151
```

Donde cada renglón representa una calificación otorgada por un usuario a una película:

```
nrow(movielens)
```

```
## [1] 100004
```

Podemos ver el número de usuarios únicos y para cuántas películas únicas hicieron una calificación:

```
movielens %>%
  summarize(n_users = n_distinct(userId),
            n_movies = n_distinct(movieId))
```

```
##   n_users n_movies
## 1     671     9066
```

Esto implica un total de  $671 * 9066 = 6'043'026$  calificaciones. Dado que nuestra base de datos tiene  $\approx 100'000$  usuarios, significa que no todos ellos calificaron todas las películas. Por tanto, piénsese en una matriz muy grande donde los usuarios son los renglones y las películas las columnas, dejando muchas celdas en blanco.

Usando la función “gather()” podríamos hacer esto, sin embargo, el tamaño es tal que R no lo soporta. En consecuencia, usaremos una muestra más pequeña de nuestro *data frame*. Como ejemplificación, a continuación se muestra el formato deseado para cinco usuarios y 4 películas:

```
keep <- movielens %>%
  dplyr::count(movieId) %>%
  top_n(5) %>%
  pull(movieId)
```

```
## Selecting by n
```

```
tab <- movielens %>%
  filter(userId %in% c(13:17)) %>%
  filter(movieId %in% keep) %>%
  select(userId, title, rating) %>%
  spread(title, rating)
```

```
tab %>%
  knitr::kable()
```

userId	Forrest Gump	Pulp Fiction	Shawshank Redemption, The	Silence of the Lambs, The	Star Wars: Episode I
13	5.0	3.5	4.5	NA	
15	1.0	5.0	2.0	5.0	
16	NA	NA	4.0	NA	
17	2.5	5.0	5.0	4.5	

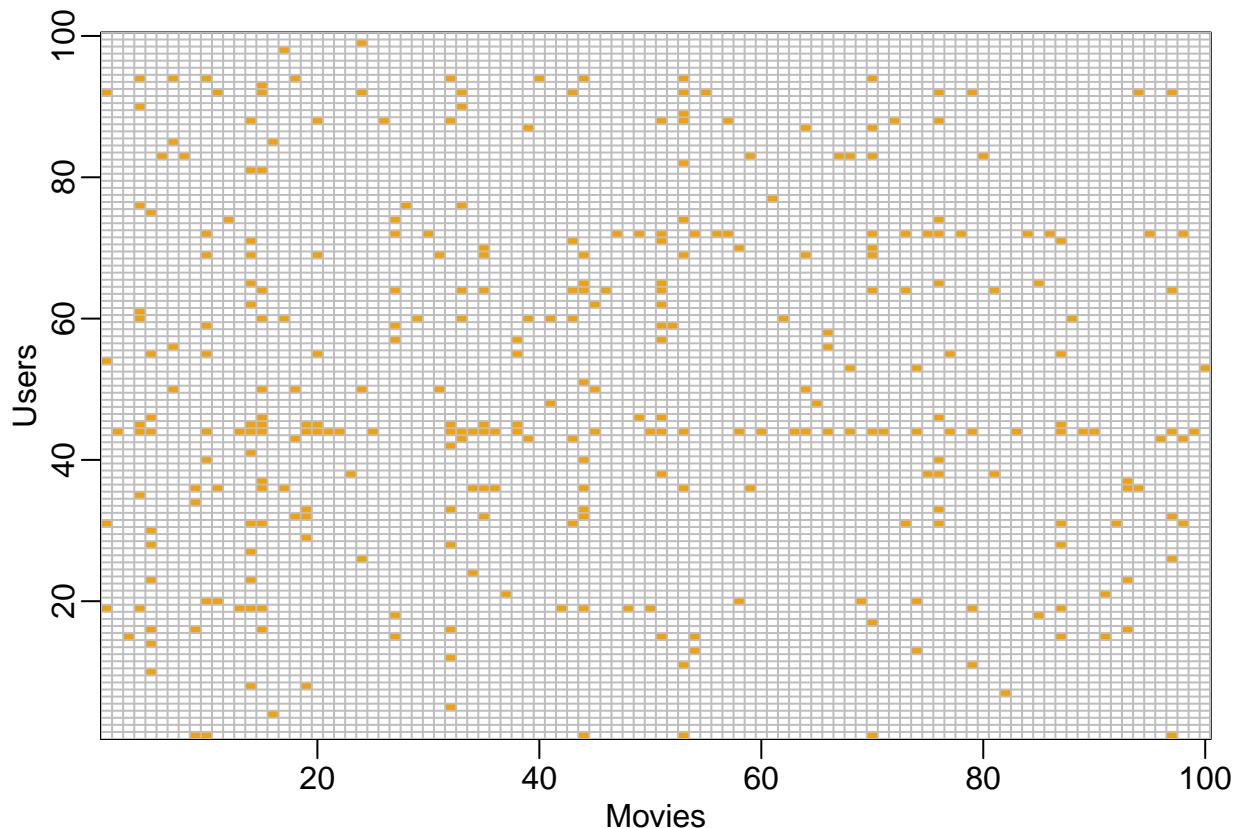
Así, puede pensarse al sistema de recomendación como la predicción de los NAs. Para visualizar qué tan escasa es la matriz, úsese el siguiente código para muestras 100 personas y 100 películas, donde el color amarillo muestra una calificación que sí existe:

```
users <- sample(unique(movielens$userId), 100)

rafalib::mypar()

movielens %>%
  filter(userId %in% users) %>%
  select(userId, movieId, rating) %>%
  mutate(rating = 1) %>%
  spread(movieId, rating) %>%
  select(sample(ncol(.), 100)) %>%
  as.matrix() %>% t(.) %>%
  image(1:100, 1:100,. , xlab="Movies", ylab="Users")

abline(h=0:100+0.5, v=0:100+0.5, col = "grey")
```

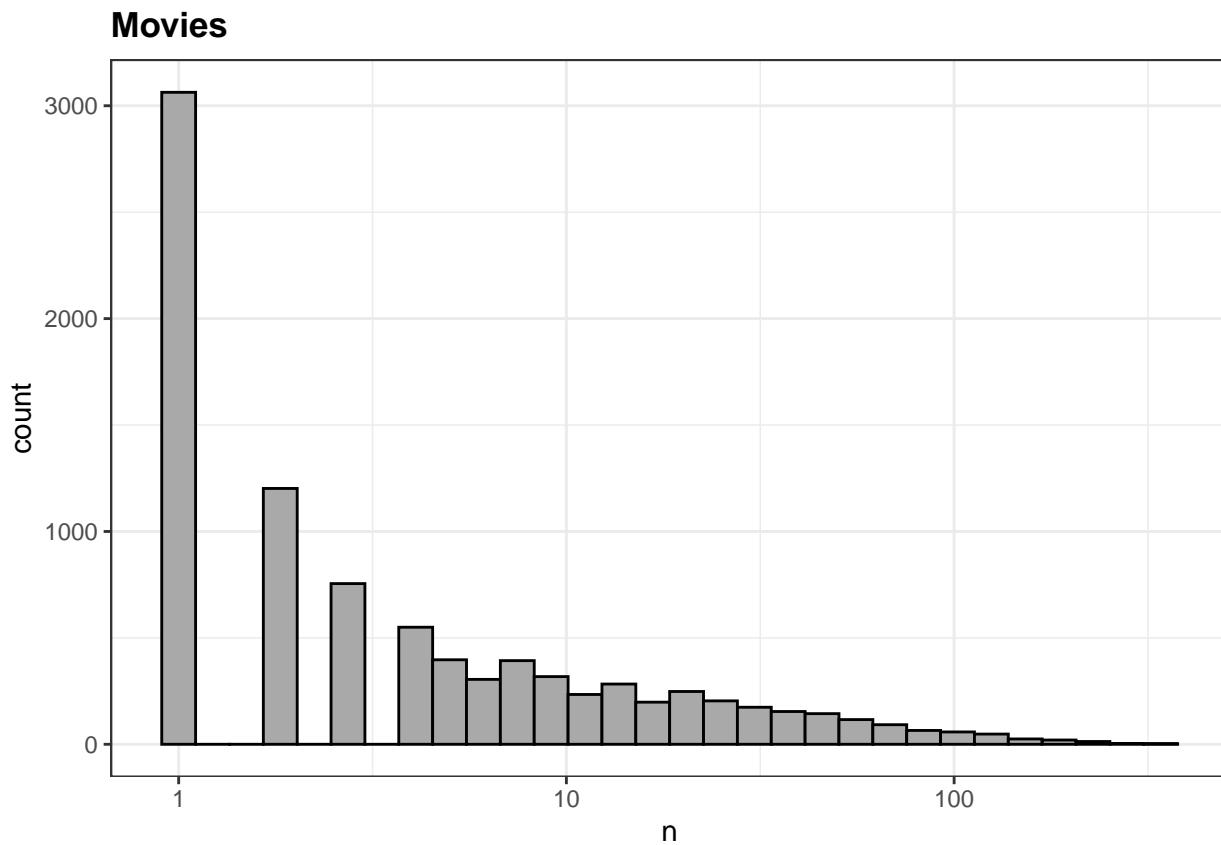


El reto de aprendizaje automático es sustancialmente más complicado en este ejemplo, dado que cada resultado  $y$  tiene un conjunto de predictores diferente: nótese que si queremos predecir la calificación de la película  $i$  por el usuario  $u$ , todas las calificaciones hechas para esa película, así como todas las calificaciones hechas por ese usuario, son predictores.

Además, podríamos incluir información sobre películas que consideramos similares a la película  $i$ , o bien, usuarios similares a  $u$ . Es decir, en esencia, la matriz completa podría ser usada como predictor de cada celda.

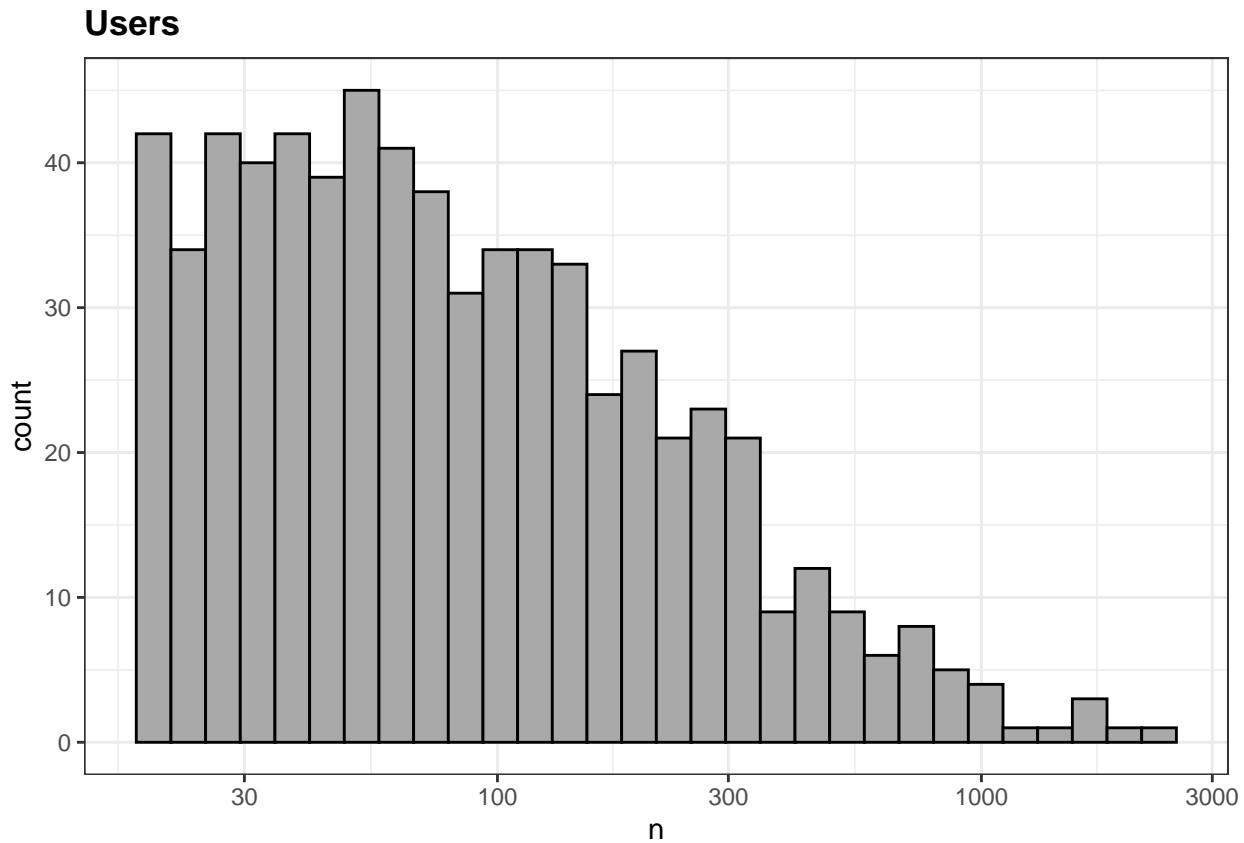
Primero, obsérvense algunas propiedades de los datos: la primera es que algunas películas son más calificadas que otras:

```
movielens %>%
  dplyr::count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black", fill = "darkgray") +
  scale_x_log10() +
  ggtitle("Movies")
```



Una segunda observación es que algunos usuarios son más activos que otros al calificar películas:

```
movielens %>%
  dplyr::count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black", fill = "darkgray") +
  scale_x_log10() +
  ggtitle("Users")
```



A continuación, crearemos el conjunto de prueba para nuestro algoritmo:

```
library(caret)

set.seed(755, sample.kind = "Rounding")

test_index <- createDataPartition(movielens$rating, times = 1,
                                  p = 0.2, list = FALSE)

train_set <- movielens[-test_index,]
test_set <- movielens[test_index,]
```

Para estar seguros que no estamos incluyendo a usuarios y películas en el conjunto de prueba que no aparecen en el de práctica, los removemos mediante “semi\_join()”:

```
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

Antes de comenzar a comparar modelos y los subsecuentes análisis, debemos establecer una marca sobre una “predicción o rendimiento aceptables”; esto se hace definiendo una función de pérdida.

En este ejemplo, usaremos la Raíz del Error Cuadrático Medio (RMSE). Si  $y_{u,i}$  es la calificación de la película  $i$  por el usuario  $u$  y  $\hat{y}_{u,i}$  es nuestra predicción, entonces el RMSE es:

$$\text{RMSE} = \sqrt{\frac{1}{N} \sum_{u,i} (\hat{y}_{u,i} - y_{u,i})^2}$$

Si  $RMSE > 1$ , significa que nuestra predicción es errónea por 1 o más estrellas, por lo que no es buena. A continuación se define una función para calcular el RMSE para un vector de calificaciones:

```
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings-predicted_ratings)^2))
}
```

Comencemos por construir un sistema de recomendación extremadamente simple: predecir la misma calificación para todas las películas, sin importar el usuario y la calificación. Esto implicaría un modelo como el que sigue:

$$Y_{u,i} = \mu + \varepsilon_{u,i}$$

Donde  $\mu$  es la calificación real obtenida por las películas y  $\varepsilon$  la variación contemplada con media cero. Por otro lado, sabemos que el estimador que minimiza el RMSE es el MCO de  $\mu$ , el cual solo es un promedio de las calificaciones:

```
mu_hat <- mean(train_set$rating)

mu_hat

## [1] 3.54
```

Esto es, el promedio de calificación de todas las películas por todos los usuarios. Ahora, calculemos el RMSE en el conjunto de prueba:

```
naive_rmse <- RMSE(test_set$rating, mu_hat)

naive_rmse

## [1] 1.05
```

El cual es bastante alto. Nótese que si probamos cualquier otro número, el RMSE; esto es obvio, dado que el RMSE se minimiza con nuestro estimador:

```
predictions <- rep(2.5, nrow(test_set))

RMSE(test_set$rating, predictions)

## [1] 1.49
```

Antes de continuar en el mejoramiento de nuestras estimaciones, construiremos una tabla para depositar los valores que vayamos obteniendo:

```
rmse_results <- tibble(method = "Just the average", RMSE = naive_rmse)
```

El modelo anterior puede mejorarse agregando un factor que represente el promedio de calificación para cada película  $i$ :

$$Y_{u,i} = \mu + b_i + \varepsilon_{u,i}$$

Para estimar este parámetro, úsese el siguiente procedimiento (dado que son miles de películas, “lm()” resulta muy lento):

```
# fit <- lm(rating ~ as.factor(userID), data = movielens)
```

Sin embargo, en esta situación particular, sabemos que el estimador MCO  $\hat{b}_i$  es solo el promedio de  $y_{u,i}$  menos el promedio general de cada película  $i$ :

```

mu <- mean(train_set$rating)

movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

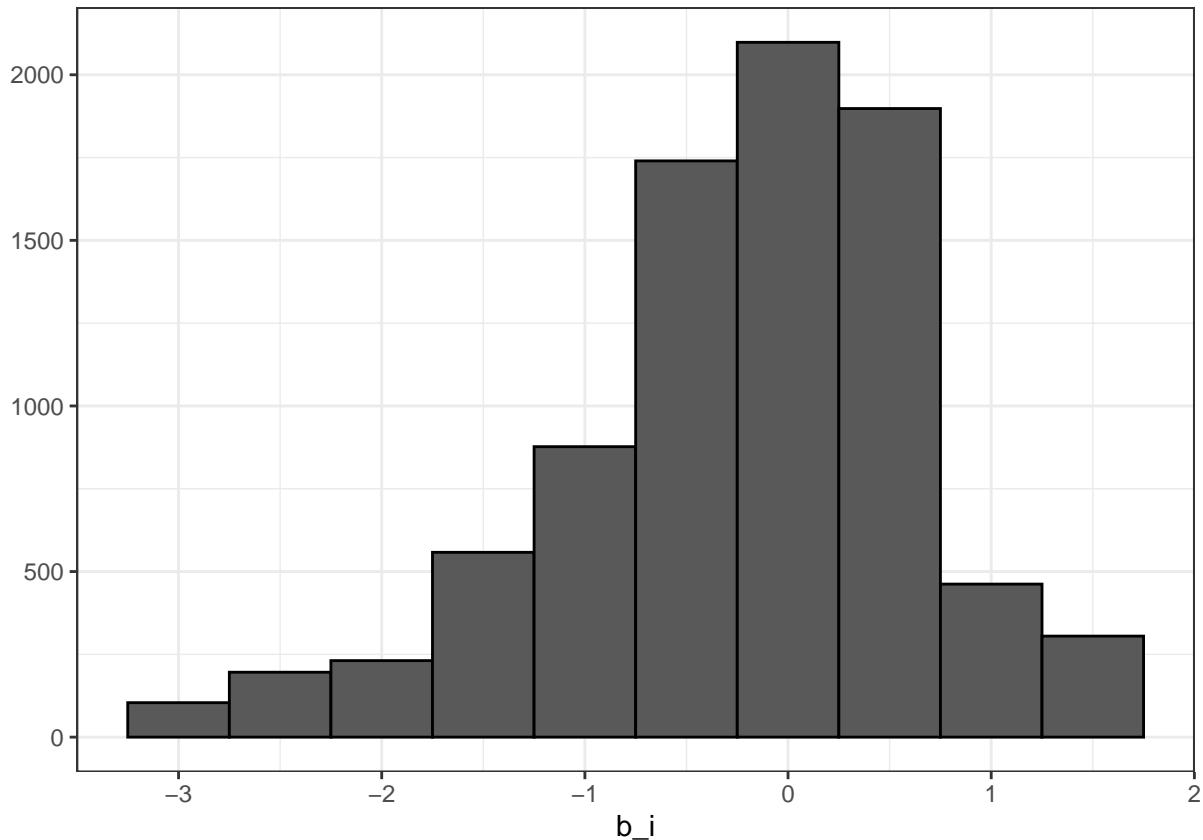
```

Podemos ver que estos estimadores varían sustantivamente:

```

movie_avgs %>%
  qplot(b_i, geom = "histogram", bins = 10, data = ., color = I("black"))

```



¿Qué tanto mejora nuestra predicción?

```

predicted_ratings <- mu + test_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  .$b_i

model_1_rmse <- RMSE(predicted_ratings, test_set$rating)

rmse_results <- bind_rows(rmse_results,
                           tibble(method = "Movie Effect Model",
                                  RMSE = model_1_rmse))

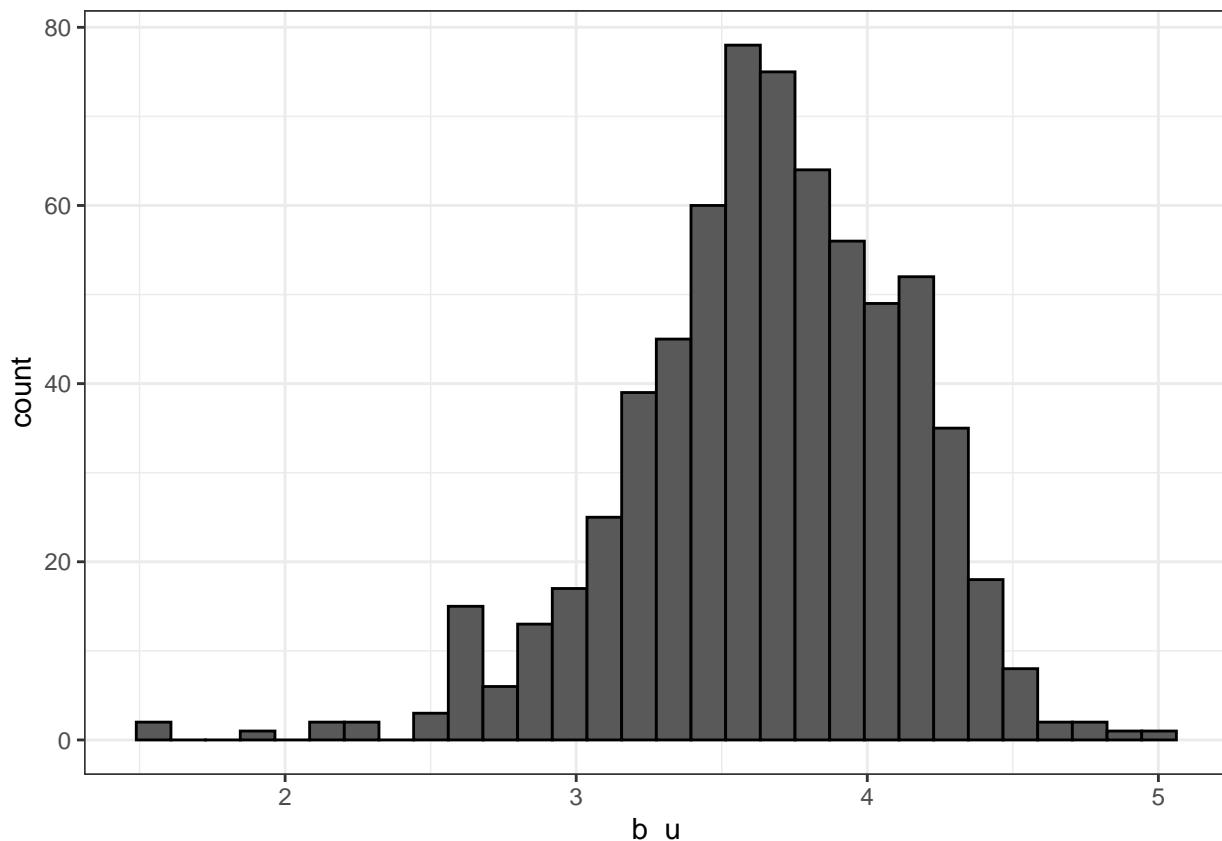
rmse_results %>%
  knitr::kable()

```

method	RMSE
Just the average	1.048
Movie Effect Model	0.986

Es decir, el RMSE cayó, aunque muy ligeramente. Intuitivamente, podemos realizar la misma tarea pero ahora para usuarios. Para ello, calculemos el promedio de calificación por usuario para aquellos que han calificado más de 100 películas:

```
train_set %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating)) %>%
  filter(n() >= 100) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "black")
```



Así, nuestro siguiente modelo es:

$$Y_{u,i} = \mu + b_i + b_u + \varepsilon_{u,i}$$

Así, para el ajuste del modelo, calcularemos la media general  $\hat{\mu}$ , los efectos por película  $\hat{b}_i$ , y luego estimaremos el efecto por usuario  $\hat{b}_u$ , restando el promedio de los residuales obtenidos después de remover el promedio general y el efecto por película de las calificaciones  $y_{u,i}$

```
user_avgs <- test_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
```

¿Qué tan bien lo hicimos?

```
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred

model_2_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                           data_frame(method="Movie + User Effects Model",
                                      RMSE = model_2_rmse ))
rmse_results %>%
  knitr::kable()
```

method	RMSE
Just the average	1.048
Movie Effect Model	0.986
Movie + User Effects Model	0.885

Puede verse que el último modelo es el que presenta el RMSE más pequeño, por tanto, el mejor que pudimos realizar.

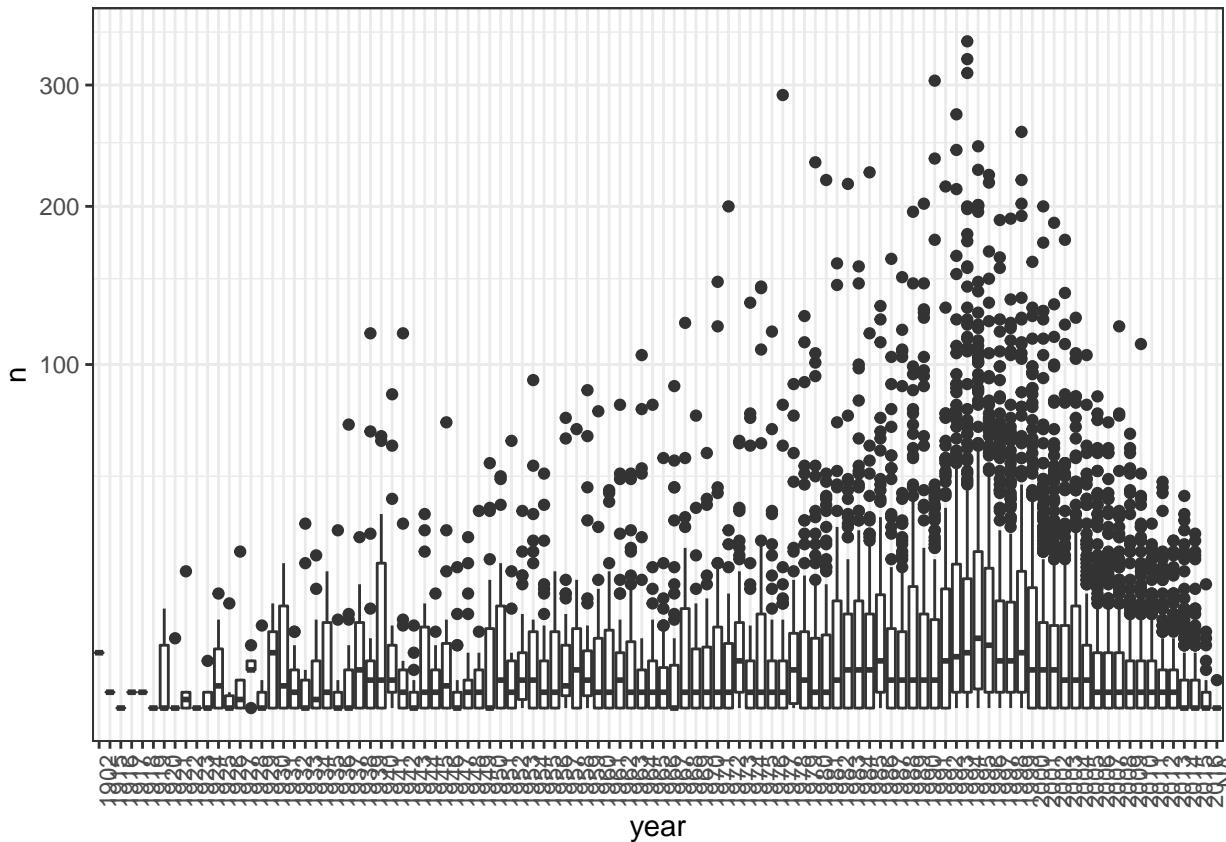
#### 8.6.4. Assessment 31

The following exercises all work with the movielens data, which can be loaded using the following code:

```
library(tidyverse)
library(lubridate)
library(dslabs)
data("movielens")
```

Compute the number of ratings for each movie and then plot it against the year the movie came out using a boxplot for each year. Use the square root transformation on the y-axis (number of ratings) when creating your plot. What year has the highest median number of ratings?

```
movielens %>%
  group_by(movieId) %>%
  summarize(n = n(), year = as.character(first(year))) %>%
  qplot(year, n, data = ., geom = "boxplot") +
  coord_trans(y = "sqrt") +
  theme (axis.text.x = element_text(angle = 90, hjust = 1))
```



We see that, on average, movies that came out after 1993 get more ratings. We also see that with newer movies, starting in 1993, the number of ratings decreases with year: the more recent a movie is, the less time users have had to rate it. Among movies that came out in 1993 or later, select the top 25 movies with the highest average number of ratings per year ( $n/year$ ), and calculate the average rating of each of them. To calculate number of ratings per year, use 2018 as the end year.

What is the average rating for the movie The Shawshank Redemption? What is the average number of ratings per year for the movie Forrest Gump?

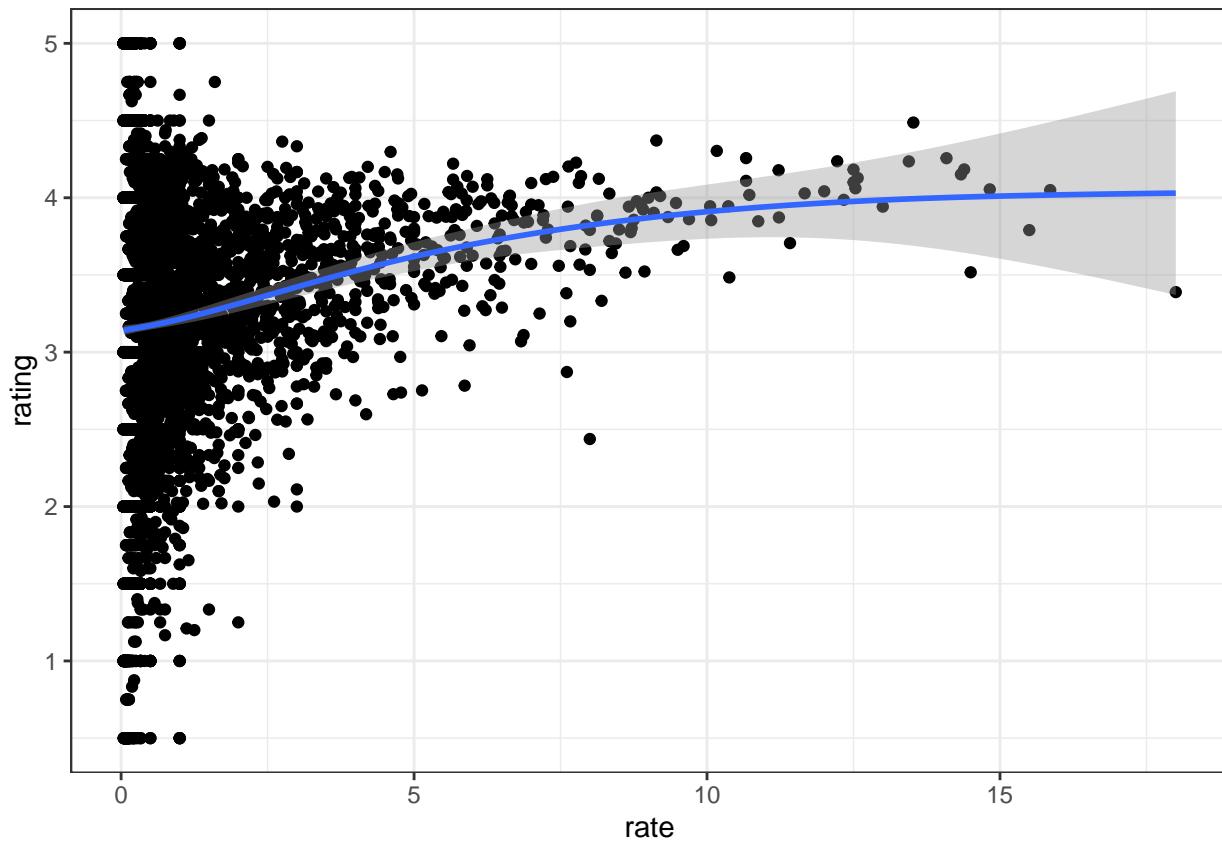
```
movielens %>%
  filter(year >= 1993) %>%
  group_by(movieId) %>%
  summarize(n = n(), years = 2018 - first(year),
            title = title[1],
            rating = mean(rating)) %>%
  mutate(rate = n/years) %>%
  top_n(25, rate) %>%
  arrange(desc(rate))

## # A tibble: 25 x 6
##   movieId     n  years title          rating    rate
##       <int> <int> <dbl> <chr>        <dbl>    <dbl>
## 1     356     341    24 Forrest Gump  4.05    14.2
## 2    79132     111     8 Inception  4.05    13.9
## 3    2571      259    19 Matrix, The  4.18    13.6
## 4     296      324    24 Pulp Fiction 4.26    13.5
## 5     318      311    24 Shawshank Redemption, The 4.49    13.0
## 6    58559     121    10 Dark Knight, The 4.24    12.1
## 7    4993      200    17 Lord of the Rings: The Fellowship of the Ri~ 4.18    11.8
## 8    5952      188    16 Lord of the Rings: The Two Towers, The 4.06    11.8
## 9    7153      176    15 Lord of the Rings: The Return of the King, ~ 4.13    11.7
## 10   2858      220    19 American Beauty 4.24    11.6
## # ... with 15 more rows
```

From the table constructed in Q2, we can see that the most frequently rated movies tend to have above average ratings. This is not surprising: more people watch popular movies. To confirm this, stratify the post-1993 movies by ratings per year and compute their average ratings. To calculate number of ratings per year, use 2018 as the end year. Make a plot of average rating versus ratings per year and show an estimate of the trend.

```
movielens %>%
  filter(year >= 1993) %>%
  group_by(movieId) %>%
  summarize(n = n(), years = 2017 - first(year),
            title = title[1],
            rating = mean(rating)) %>%
  mutate(rate = n/years) %>%
  ggplot(aes(rate, rating)) +
  geom_point()+
  geom_smooth()

## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



The movielens dataset also includes a time stamp. This variable represents the time and date in which the rating was provided. The units are seconds since January 1, 1970. Create a new column date with the date.

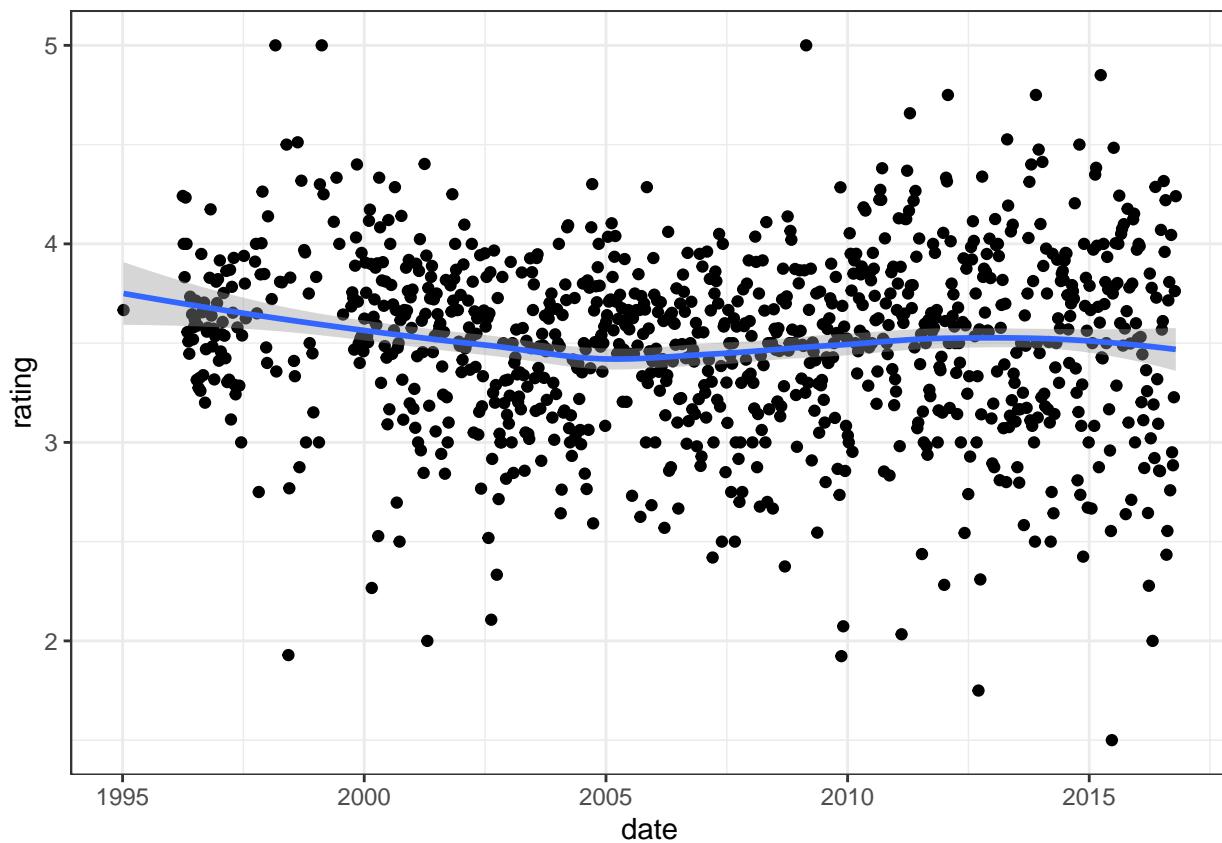
```
library(lubridate)

movielens <- mutate(movielens, date = as_datetime(timestamp))
```

Compute the average rating for each week and plot this average against date. Hint: use the round\_date() function before you group\_by().

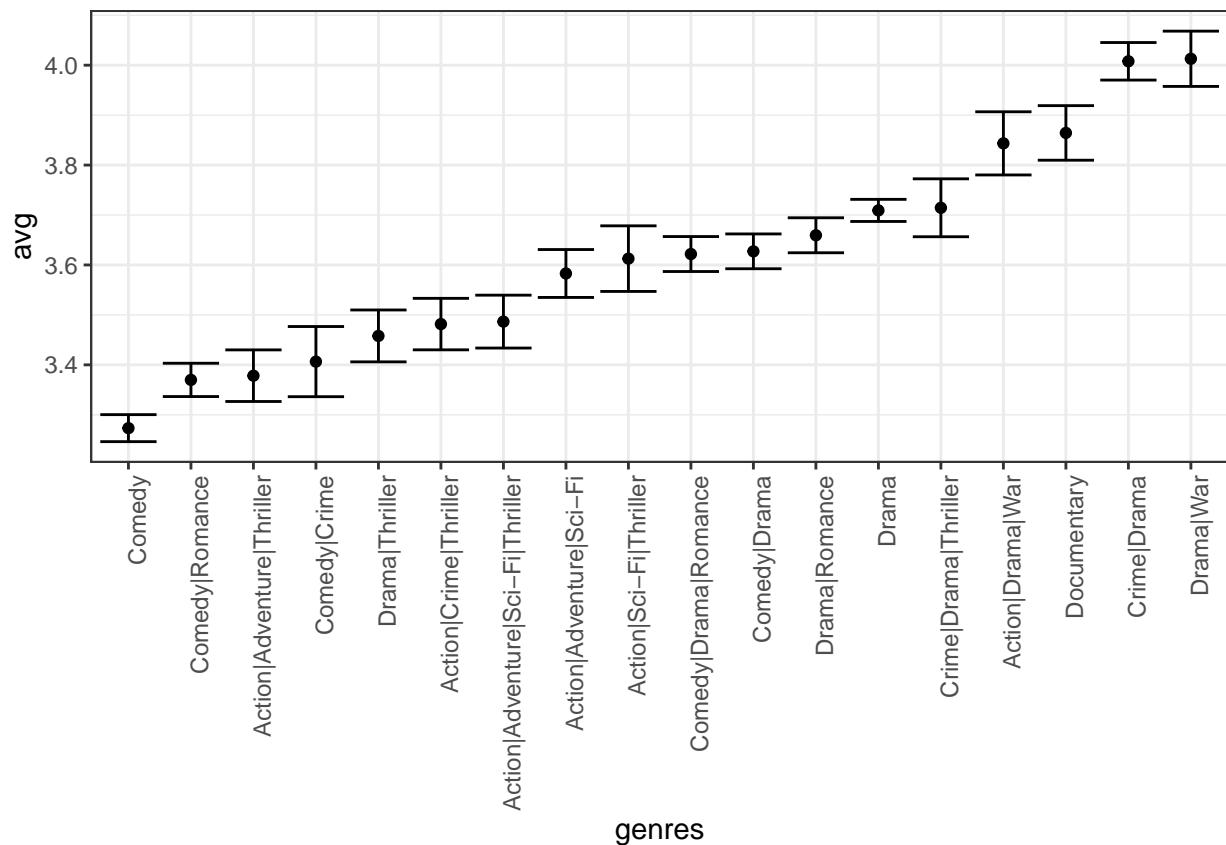
```
movielens %>%
  mutate(date = round_date(date, unit = "week")) %>%
  group_by(date) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(date, rating)) +
  geom_point() +
  geom_smooth()

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



The movielens data also has a genres column. This column includes every genre that applies to the movie. Some movies fall under several genres. Define a category as whatever combination appears in this column. Keep only categories with more than 1,000 ratings. Then compute the average and standard error for each category. Plot these as error bar plots. Which genre has the lowest average rating?

```
movielens %>%
  group_by(genres) %>%
  summarize(n = n(), avg = mean(rating), se = sd(rating)/sqrt(n())) %>%
  filter(n >= 1000) %>%
  mutate(genres = reorder(genres, avg)) %>%
  ggplot(aes(x = genres, y = avg, ymin = avg-2*se, ymax = avg+2*se)) +
  geom_point() +
  geom_errorbar() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



### 8.6.5. Regularización

En esta sección, revisaremos el concepto de regularización y cómo podemos usarla para mejorar nuestros resultados en el sistema de recomendación de películas. Nótese que, aunque hemos mejorado la precisión de nuestro algoritmo al minimizar el RMSE, el cambio no ha sido demasiado grande, ¿por qué?

```
library(dslabs)
library(tidyverse)
library(caret)
data("movielens")
set.seed(755)
test_index <- createDataPartition(y = movielens$rating, times = 1,
                                  p = 0.2, list = FALSE)
train_set <- movielens[-test_index,]
test_set <- movielens[test_index,]
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
mu_hat <- mean(train_set$rating)
naive_rmse <- RMSE(test_set$rating, mu_hat)
rmse_results <- data_frame(method = "Just the average", RMSE = naive_rmse)
mu <- mean(train_set$rating)
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
predicted_ratings <- mu + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_i
model_1_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                           data_frame(method="Movie Effect Model",
                                      RMSE = model_1_rmse ))
user_avgs <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred
model_2_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                           data_frame(method="Movie + User Effects Model",
                                      RMSE = model_2_rmse ))
```

Retómese el modelo donde solo utilizamos películas como predictores; a continuación se muestran 10 grandes errores que se cometieron:

```
test_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  mutate(residual = rating - (mu + b_i)) %>%
  arrange(desc(abs(residual))) %>%
```

```
select(title, residual) %>%
  slice(1:10) %>%
  knitr::kable()
```

title	residual
Day of the Beast, The (Día de la Bestia, El)	4.50
Horror Express	-4.00
No Holds Barred	4.00
Dear Zachary: A Letter to a Son About His Father	-4.00
Faust	-4.00
Hear My Song	-4.00
Confessions of a Shopaholic	-4.00
Twilight Saga: Breaking Dawn - Part 1, The	-4.00
Taxi Driver	-3.81
Taxi Driver	-3.81

Ahora, veamos las mejores 10 y peores 10 calificaciones basadas en nuestras estimaciones del efecto película

```
movie_titles <- movielens %>%
  select(movieId, title) %>%
  distinct()

movie_avgs %>%
  left_join(movie_titles, by = "movieId") %>%
  arrange(desc(b_i)) %>%
  select(title, b_i) %>%
  slice(1:10) %>%
  knitr::kable()
```

title	b_i
Lamerica	1.46
Love & Human Remains	1.46
Enfer, L'	1.46
Picture Bride (Bijo photo)	1.46
Red Firecracker, Green Firecracker (Pao Da Shuang Deng)	1.46
Faces	1.46
Maya Lin: A Strong Clear Vision	1.46
Heavy	1.46
Gate of Heavenly Peace, The	1.46
Death in the Garden (Mort en ce jardin, La)	1.46

```
movie_avgs %>%
  left_join(movie_titles, by = "movieId") %>%
  arrange(b_i) %>%
  select(title, b_i) %>%
  slice(1:10) %>%
  knitr::kable()
```

title	b_i
Santa with Muscles	-3.04
B*A*P*S	-3.04
3 Ninjas: High Noon On Mega Mountain	-3.04
Barney's Great Adventure	-3.04
Merry War, A	-3.04
Day of the Beast, The (Día de la Bestia, El)	-3.04
Children of the Corn III	-3.04
Whiteboyz	-3.04
Catfish in Black Bean Sauce	-3.04
Watcher, The	-3.04

Todas ellas tienen algo en común: son relativamente poco conocidas. Veamos qué tanto se calificaron:

```
train_set %>%
  dplyr::count(movieId) %>%
  left_join(movie_avgs) %>%
  left_join(movie_titles, by = "movieId") %>%
  arrange(desc(b_i)) %>%
  select(title, b_i, n) %>%
  slice(1:10) %>%
  knitr::kable()
```

## Joining, by = "movieId"

title	b_i	n
Lamerica	1.46	1
Love & Human Remains	1.46	3
Enfer, L'	1.46	1
Picture Bride (Bijo photo)	1.46	1
Red Firecracker, Green Firecracker (Pao Da Shuang Deng)	1.46	3
Faces	1.46	1
Maya Lin: A Strong Clear Vision	1.46	2
Heavy	1.46	1
Gate of Heavenly Peace, The	1.46	1
Death in the Garden (Mort en ce jardin, La)	1.46	1

```
train_set %>%
  dplyr::count(movieId) %>%
  left_join(movie_avgs) %>%
  left_join(movie_titles, by = "movieId") %>%
  arrange(b_i) %>%
  select(title, b_i, n) %>%
  slice(1:10) %>%
  knitr::kable()
```

## Joining, by = "movieId"

title	b_i	n
Santa with Muscles	-3.04	1
B*A*P*S	-3.04	1
3 Ninjas: High Noon On Mega Mountain	-3.04	1
Barney's Great Adventure	-3.04	1
Merry War, A	-3.04	1
Day of the Beast, The (Día de la Bestia, El)	-3.04	1
Children of the Corn III	-3.04	1
Whiteboyz	-3.04	1
Catfish in Black Bean Sauce	-3.04	1
Watcher, The	-3.04	1

Puede verse que las películas mejor y peor calificadas recibieron muy pocas opiniones, lo que incrementa la incertidumbre. Así, las calificaciones muy bajas o altas son más probables de aparecer cuando pocos usuarios calificaron esa película. Esto es, son estimaciones ruidosas y poco confiables.

Como no podemos construir intervalos de confianza como predicciones (dado que necesitamos una calificación en número y no en intervalo), haremos uso de la **regularización**. Esta metodología nos permite penalizar a estimaciones muy grandes, mediante la restricción de la variabilidad total del tamaño del efecto.

Así, para estimar nuestros parámetros  $b$ , ahora minimizaremos la siguiente ecuación, la cual incluye la penalización:

$$\frac{1}{N} \sum_{u,i} (u_{u,i} - \mu - b_i)^2 + \lambda \sum_i b_i^2$$

Donde el primer término es el MSE y el segundo la penalización a los términos que sean mayores cuando  $b$  crece. Por tanto, los valores  $b$  que minimizan la ecuación anterior son:

$$\hat{b}_i(\lambda) = \frac{1}{\lambda + n_i} \sum_{u=1}^{n_i} (Y_{u,i} - \hat{\mu})$$

Nótese que este procedimiento resultará en el efecto deseado: cuando  $n_i$  sea muy grande, lo que implicará una estimación estable, entonces  $\lambda$  es prácticamente ignorado, dado que  $n_i + \lambda \approx n_i$ . Sin embargo, si  $n_i$  es muy pequeño,  $\hat{b}_i \rightarrow 0$ .

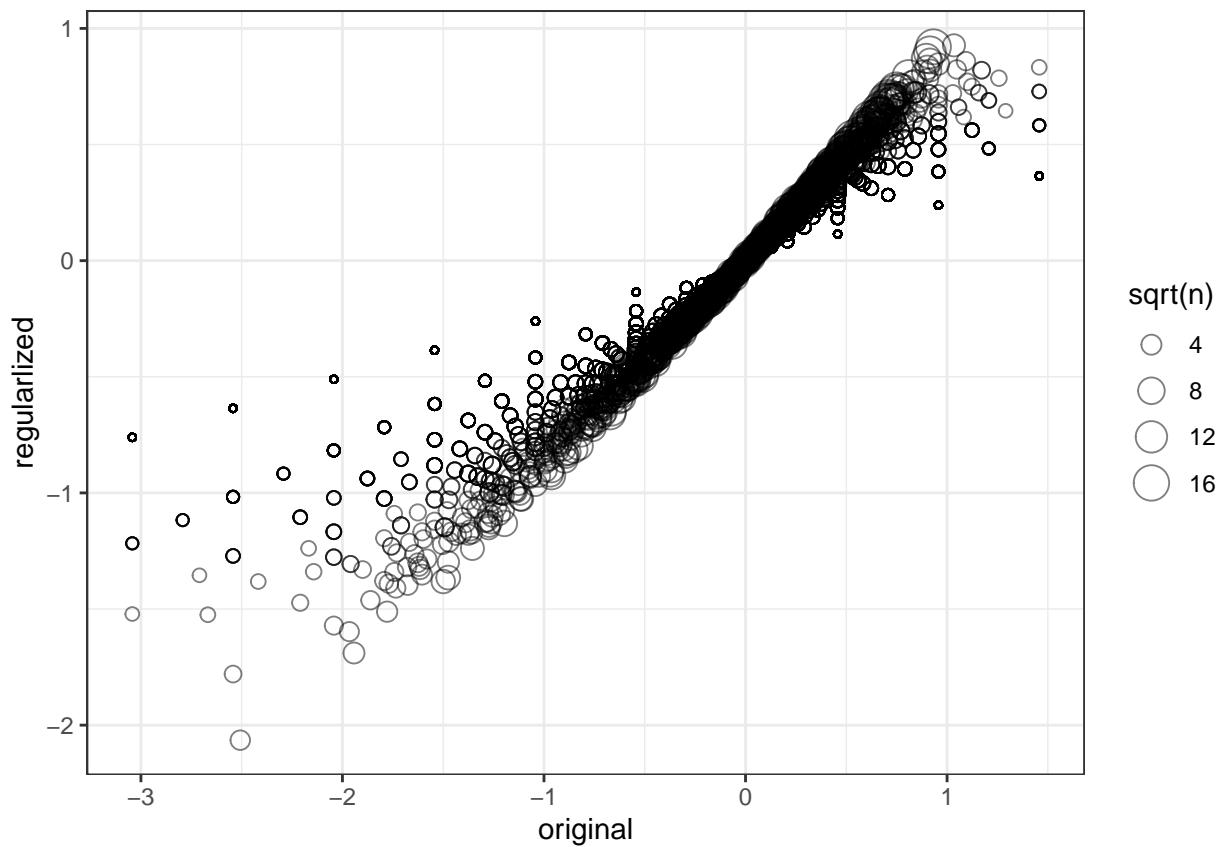
```
lambda <- 3

mu <- mean(train_set$rating)

movie_reg_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n() + lambda), n_i= n())
```

Para ver cómo disminuye el estimador, grafiquemos el regularizado contra el MSE:

```
data_frame(original = movie_avgs$b_i,
           regularized = movie_reg_avgs$b_i,
           n = movie_reg_avgs$n_i) %>%
  ggplot(aes(original, regularized, size=sqrt(n))) +
  geom_point(shape=1, alpha=0.5)
```



Donde vemos que, cuando  $n$  es pequeña, los valores se aproximan a . Volvamos a observar el top 10 de películas pero regularizándolas:

```
train_set %>%
  dplyr::count(movieId) %>%
  left_join(movie_reg_avgs) %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  select(title, b_i, n) %>%
  slice(1:10) %>%
  knitr::kable()
```

```
## Joining, by = "movieId"
```

title	b_i	n
All About Eve	0.927	26
Shawshank Redemption, The	0.921	240
Godfather, The	0.897	153
Godfather: Part II, The	0.871	100
Maltese Falcon, The	0.860	47
Best Years of Our Lives, The	0.859	11
On the Waterfront	0.847	23
Face in the Crowd, A	0.833	4
African Queen, The	0.832	36
All Quiet on the Western Front	0.824	11

Si nos fijamos en las películas, tienen más sentido. Lo mismo ocurre con las peores:

```
train_set %>%
  dplyr::count(movieId) %>%
  left_join(movie_reg_avgs) %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(b_i) %>%
  select(title, b_i, n) %>%
  slice(1:10) %>%
  knitr::kable()
```

## Joining, by = "movieId"

title	b_i	n
Battlefield Earth	-2.06	14
Joe's Apartment	-1.78	7
Speed 2: Cruise Control	-1.69	20
Super Mario Bros.	-1.60	13
Police Academy 6: City Under Siege	-1.57	10
After Earth	-1.52	4
Disaster Movie	-1.52	3
Little Nicky	-1.51	17
Cats & Dogs	-1.47	6
Blade: Trinity	-1.46	11

Pero, ¿mejoramos nuestros resultados?

```
predicted_ratings <- test_set %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  mutate(pred = mu + b_i) %>%
  .$pred

model_3_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                           data_frame(method="Regularized Movie Effect Model",
                                      RMSE = model_3_rmse ))
rmse_results %>%
  knitr::kable()
```

method	RMSE
Just the average	1.048
Movie Effect Model	0.986
Movie + User Effects Model	0.885
Regularized Movie Effect Model	0.965

Véase que mejoramos nuestro RMSE hasta. También nótese que  $\lambda$  es un parámetro de afinación, por lo que podemos usar validación cruzada para elegirlo (donde se demuestra por qué 3 fue una buena elección):

```
lambdas <- seq(0, 10, 0.25)

mu <- mean(train_set$rating)

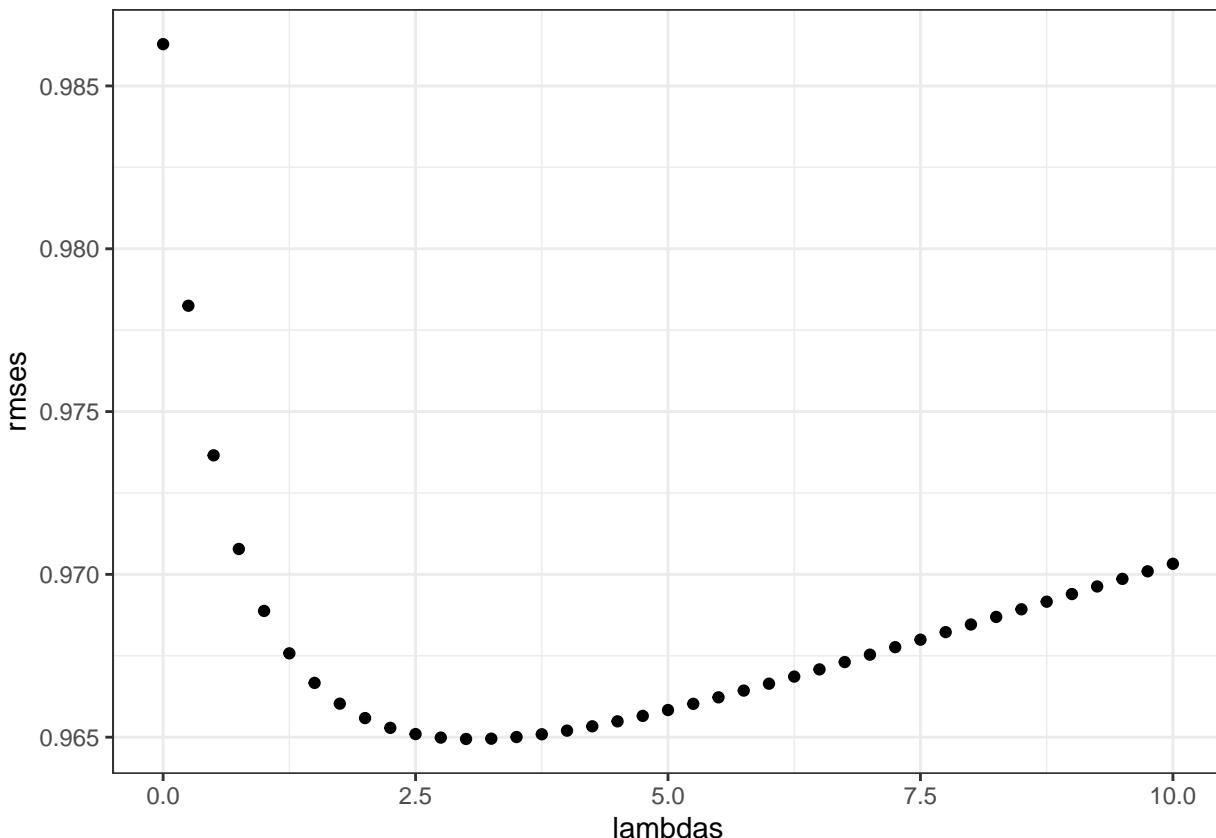
just_the_sum <- train_set %>%
  group_by(movieId) %>%
  summarize(s = sum(rating - mu), n_i = n())
rmses <- sapply(lambdas, function(l){
  predicted_ratings <- test_set %>%
    left_join(just_the_sum, by='movieId') %>%
```

```

    mutate(b_i = s/(n_i+1)) %>%
    mutate(pred = mu + b_i) %>%
    .$pred
  return(RMSE(predicted_ratings, test_set$rating))
}

qplot(lambdas, rmses)

```



```
lambdas[which.min(rmses)]
```

```
## [1] 3
```

También podemos usar la regularización para el efecto de los usuarios, donde ahora minimizaremos:

$$\frac{1}{N} \sum_{u,i} (y_{i,u} - \mu - b_i - b_u)^2 + \lambda (\sum_i b_i^2 + \sum_u b_u^2)$$

```
lambdas <- seq(0, 10, 0.25)
```

```

rmses <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+1))
  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%

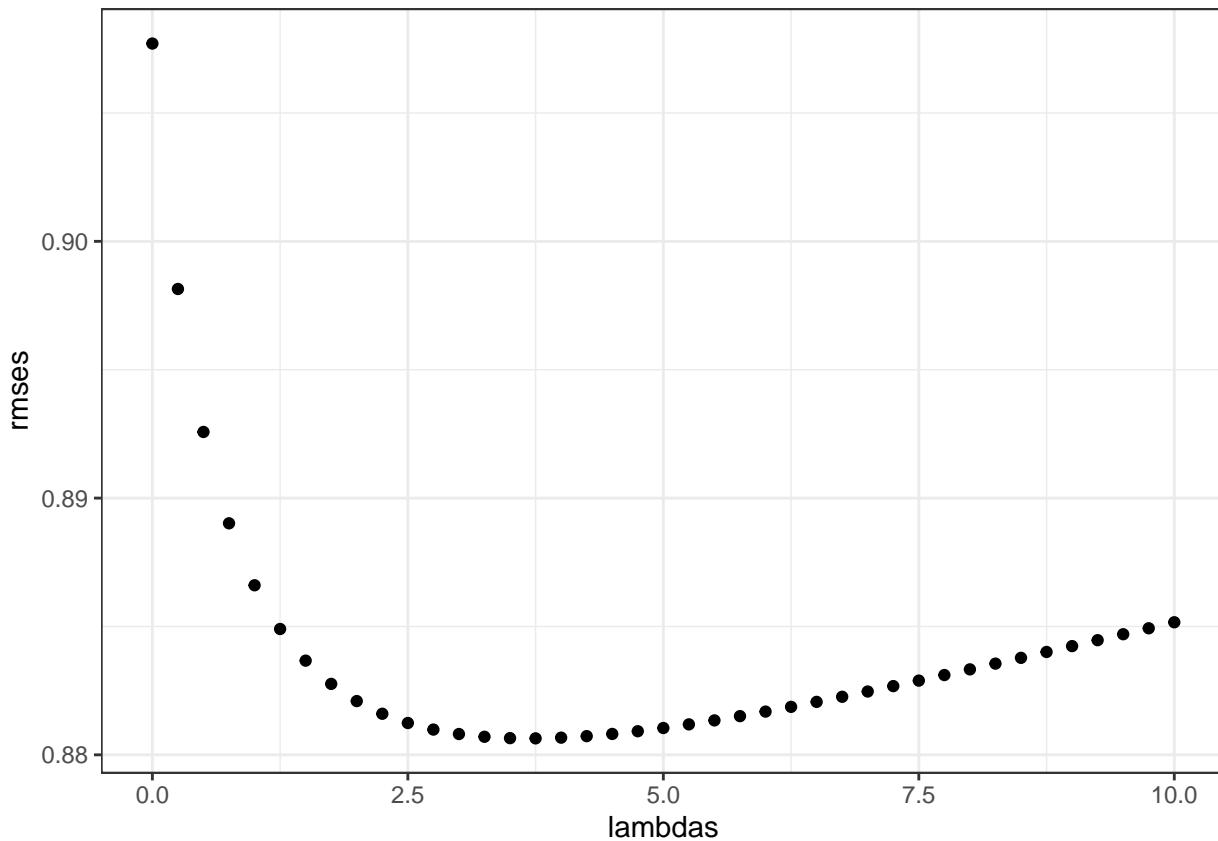
```

```

group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+1))
predicted_ratings <-
  test_set %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred
return(RMSE(predicted_ratings, test_set$rating))
}

qplot(lambdas, rmses)

```



```

lambda <- lambdas[which.min(rmses)]
lambda
## [1] 3.75
rmse_results <- bind_rows(rmse_results,
                           data_frame(method="Regularized Movie + User Effect Model",
                                      RMSE = min(rmses)))
rmse_results %>%
  knitr::kable()

```

method	RMSE
Just the average	1.048
Movie Effect Model	0.986
Movie + User Effects Model	0.885
Regularized Movie Effect Model	0.965
Regularized Movie + User Effect Model	0.881

### 8.6.6. Assessment 32

The exercises in Q1-Q8 work with a simulated dataset for 1000 schools. This pre-exercise setup walks you through the code needed to simulate the dataset.

If you have not done so already since the Titanic Exercises, please restart R or reset the number of digits that are printed with options(digits=7).

An education expert is advocating for smaller schools. The expert bases this recommendation on the fact that among the best performing schools, many are small schools. Let's simulate a dataset for 1000 schools. First, let's simulate the number of students in each school, using the following code:

```
library(tidyverse)

set.seed(1986, sample.kind="Rounding")

n <- round(2^rnorm(1000, 8, 1))
```

Now let's assign a true quality for each school that is completely independent from size. This is the parameter we want to estimate in our analysis. The true quality can be assigned using the following code:

```
set.seed(1, sample.kind="Rounding")

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used

mu <- round(80 + 2*rt(1000, 5))

range(mu)

## [1] 67 94

schools <- data.frame(id = paste("PS", 1:1000),
                       size = n,
                       quality = mu,
                       rank = rank(-mu))
```

We can see the top 10 schools using this code:

```
schools %>%
  top_n(10, quality) %>%
  arrange(desc(quality))

##      id size quality rank
## 1  PS 191   1036     94  1.0
## 2  PS 567    121     93  2.0
## 3  PS  95    235     91  3.0
## 4  PS 430     61     90  4.0
## 5  PS 343     78     89  5.0
## 6  PS 981    293     88  6.0
```

```

## 7 PS 558 196      87 7.0
## 8 PS 79 105      86 13.5
## 9 PS 113 653     86 13.5
## 10 PS 163 300    86 13.5
## 11 PS 266 2369   86 13.5
## 12 PS 400 550    86 13.5
## 13 PS 451 217    86 13.5
## 14 PS 477 341    86 13.5
## 15 PS 484 967    86 13.5
## 16 PS 561 723    86 13.5
## 17 PS 563 828    86 13.5
## 18 PS 865 586    86 13.5
## 19 PS 963 208    86 13.5

```

Now let's have the students in the school take a test. There is random variability in test taking, so we will simulate the test scores as normally distributed with the average determined by the school quality with a standard deviation of 30 percentage points. This code will simulate the test scores:

```

set.seed(1, sample.kind="Rounding")

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used

mu <- round(80 + 2*rt(1000, 5))

scores <- sapply(1:nrow(schools), function(i){
  scores <- rnorm(schools$size[i], schools$quality[i], 30)
  scores
})

schools <- schools %>%
  mutate(score = sapply(scores, mean))

```

What are the top schools based on the average score? Show just the ID, size, and the average score. Report the ID of the top school and average score of the 10th school. What is the ID of the top school? What is the average score of the 10th school (after sorting from highest to lowest average score)?

```

schools %>%
  top_n(10, score) %>%
  arrange(desc(score)) %>%
  select(id, size, score)

##      id size score
## 1 PS 567 121 95.8
## 2 PS 191 1036 93.5
## 3 PS 330 162 91.0
## 4 PS 701 83 90.5
## 5 PS 591 213 89.7
## 6 PS 205 172 89.3
## 7 PS 574 199 89.2
## 8 PS 963 208 89.0
## 9 PS 430 61 88.7
## 10 PS 756 245 88.0

```

Compare the median school size to the median school size of the top 10 schools based on the

score. What is the median school size overall? What is the median school size of the top 10 schools based on the score?

```
median(schools$size)

## [1] 261

schools %>%
  top_n(10, score) %>%
  .$size %>%
  median()

## [1] 186
```

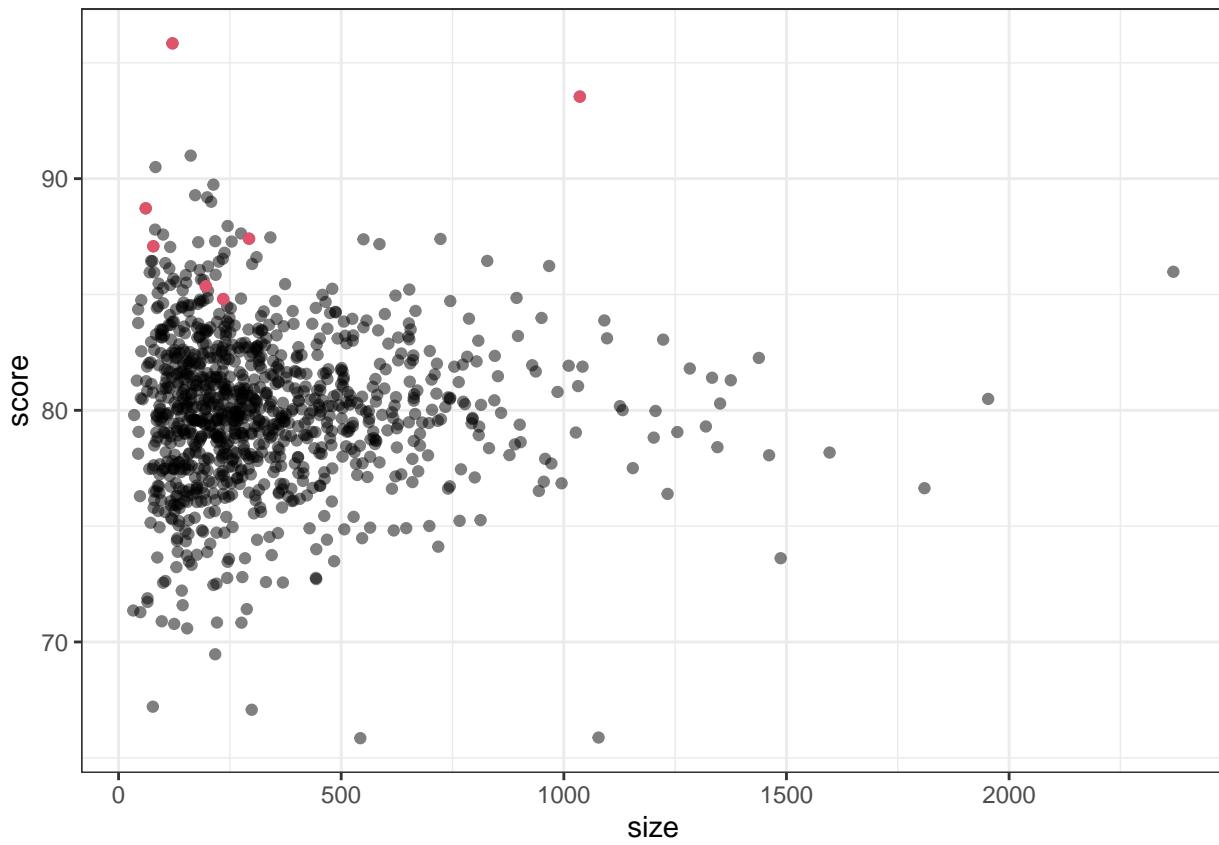
According to this analysis, it appears that small schools produce better test scores than large schools. Four out of the top 10 schools have 100 or fewer students. But how can this be? We constructed the simulation so that quality and size were independent. Repeat the exercise for the worst 10 schools. What is the median school size of the bottom 10 schools based on the score?

```
schools %>%
  top_n(-10, score) %>%
  .$size %>%
  median()

## [1] 219
```

From this analysis, we see that the worst schools are also small. Plot the average score versus school size to see what's going on. Highlight the top 10 schools based on the true quality.

```
schools %>% ggplot(aes(size, score)) +
  geom_point(alpha = 0.5) +
  geom_point(data = filter(schools, rank<=10), col = 2)
```



Let's use regularization to pick the best schools. Remember regularization shrinks deviations from the average towards 0. To apply regularization here, we first need to define the overall average for all schools, using the following code:

```
overall <- mean(sapply(scores, mean))
```

Then, we need to define, for each school, how it deviates from that average.

Write code that estimates the score above the average for each school but dividing by  $n + \alpha$  instead of  $n$ , with  $n$  the school size and  $\alpha$  a regularization parameter. Try  $\alpha = 25$ .

What is the ID of the top school with regularization?

```
alpha <- 25

score_reg <- sapply(scores, function(x) overall + sum(x-overall)/(length(x)+alpha))

schools %>%
  mutate(score_reg = score_reg) %>%
  top_n(10, score_reg) %>%
  arrange(desc(score_reg))

##      id size quality rank score score_reg
## 1  PS 191  1036     94   1.0  93.5    93.2
## 2  PS 567   121     93   2.0  95.8    93.1
## 3  PS 330   162     84  53.5  91.0    89.5
## 4  PS 591   213     83 104.5  89.7    88.7
## 5  PS 574   199     84  53.5  89.2    88.2
## 6  PS 205   172     85  28.5  89.3    88.1
```

```
## 7 PS 701 83      83 104.5 90.5      88.1
## 8 PS 963 208    86 13.5 89.0      88.0
## 9 PS 756 245    83 104.5 88.0      87.2
## 10 PS 561 723   86 13.5 87.4      87.2
```

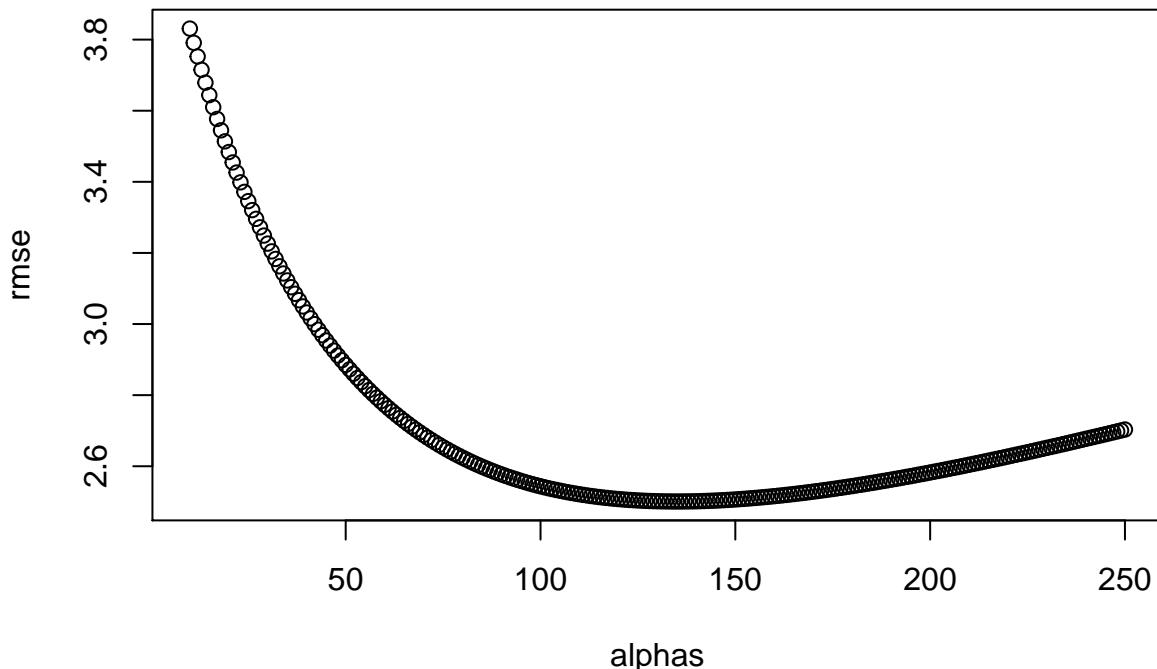
Notice that this improves things a bit. The number of small schools that are not highly ranked is now lower. Is there a better  $\alpha$ ? Using values of  $\alpha$  from 10 to 250, find the  $\alpha$  that minimizes the RMSE.

$$\text{RMSE} = \sqrt{\frac{1}{1000} \sum_{i=1}^{1000} (\text{quality} - \text{estimate})^2}$$

```
alphas <- seq(10,250)

rmse <- sapply(alphas, function(alpha){
  score_reg <- sapply(scores, function(x) overall+sum(x-overall)/(length(x)+alpha))
  mean((score_reg - schools$quality)^2)
})

plot(alphas, rmse)
```



```
alphas[which.min(rmse)]
```

```
## [1] 135
```

Rank the schools based on the average obtained with the best from Q6. Note that no small school is incorrectly included. What is the ID of the top school now? What is the regularized

average score of the 10th school now?

```
alpha <- alphas[which.min(rmse)]
score_reg <- sapply(scores, function(x)
  overall+sum(x-overall)/(length(x)+alpha))
schools %>% mutate(score_reg = score_reg) %>%
  top_n(10, score_reg) %>% arrange(desc(score_reg))

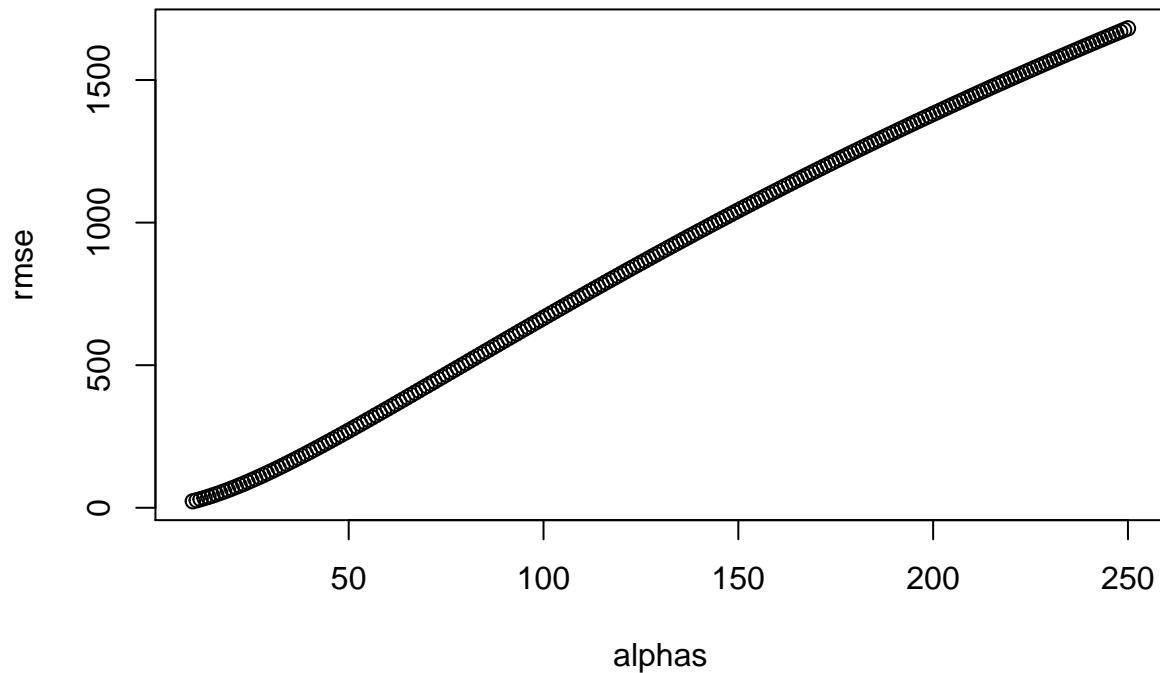
##      id size quality  rank score score_reg
## 1  PS 191 1036     94  1.0  93.5    92.0
## 2  PS 567  121     93  2.0  95.8    87.5
## 3  PS 561  723     86 13.5  87.4    86.2
## 4  PS 330  162     84 53.5  91.0    86.0
## 5  PS 591  213     83 104.5  89.7    86.0
## 6  PS 400  550     86 13.5  87.4    85.9
## 7  PS 865  586     86 13.5  87.2    85.8
## 8  PS 266 2369     86 13.5  86.0    85.7
## 9  PS 563  828     86 13.5  86.5    85.5
## 10 PS 574  199     84 53.5  89.2    85.5
```

A common mistake made when using regularization is shrinking values towards 0 that are not centered around 0. For example, if we don't subtract the overall average before shrinking, we actually obtain a very similar result. Confirm this by re-running the code from the exercise in Q6 but without removing the overall mean. What value of  $\alpha$  gives the minimum RMSE here?

```
alphas <- seq(10,250)

rmse <- sapply(alphas, function(alpha){
  score_reg <- sapply(scores, function(x) sum(x)/(length(x)+alpha))
  mean((score_reg - schools$quality)^2)
})

plot(alphas, rmse)
```



```
alphas[which.min(rmse)]
```

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

**8.6.6.1. Factorización de matrices** La factorización de matrices es un concepto ampliamente usado en el aprendizaje automático y está relacionado con el análisis de factores, la descomposición de valores singulares (SVD) y el análisis de componentes principales (PCA).

Previamente, describimos el siguiente modelo, que toma en cuenta las películas y las diferencias entre ellas mediante los parámetros  $b_i$ , así como a los usuarios y las diferencias a través de  $b_u$ :

$$Y_{u,i} = \mu + b_i + b_u + \varepsilon_{u,i}$$

Sin embargo, este modelo no toma en cuenta una fuente importante de variación relacionada con el hecho de que grupos de películas o usuarios presentan patrones calificaciones similares. Estudiaremos estos patrones mediante los resiguales obtenidos después de ajustar nuestro modelo:

$$r_{u,i} = y_{u,i} - \hat{b}_i - \hat{b}_u$$

Convertiremos los datos en una matriz de tal manera que cada usuario sea un renglón y cada película una columna. Para fines ilustrativos, consideraremos solo un subconjunto de datos de películas y usuarios con muchas calificaciones:

```
library(tidyverse)
library(dslabs)
data(movielens)
```

```

train_small <- movielens %>%
  group_by(movieId) %>%
  filter(n() >= 50 | movieId == 3252) %>% # 3252 es una película de ejemplo
  ungroup() %>%
  group_by(userId) %>%
  filter(n() >= 50) %>%
  ungroup()

y <- train_small %>%
  select(userId, movieId, rating) %>%
  spread(movieId, rating) %>%
  as.matrix()

```

Para facilitar la exploración de datos, agregamos nombres de renglones y columnas:

```

rownames(y) <- y[,1]

y <- y[,-1]

colnames(y) <- with(movie_titles, title[match(colnames(y), movieId)])

```

Lo cual puede convertirse en residuales sustrayendo las medias por columna y renglón:

```

y <- sweep(y, 1, rowMeans(y, na.rm = TRUE))

y <- sweep(y, 2, colMeans(y, na.rm = TRUE))

```

Ahora, si el modelo que hemos usado describe todas las señales válidas, mientras que lo demás es ruido, entonces el residual para diferentes películas debe ser independiente entre ellas; pero no lo son, como muestra el siguiente ejemplo:

```

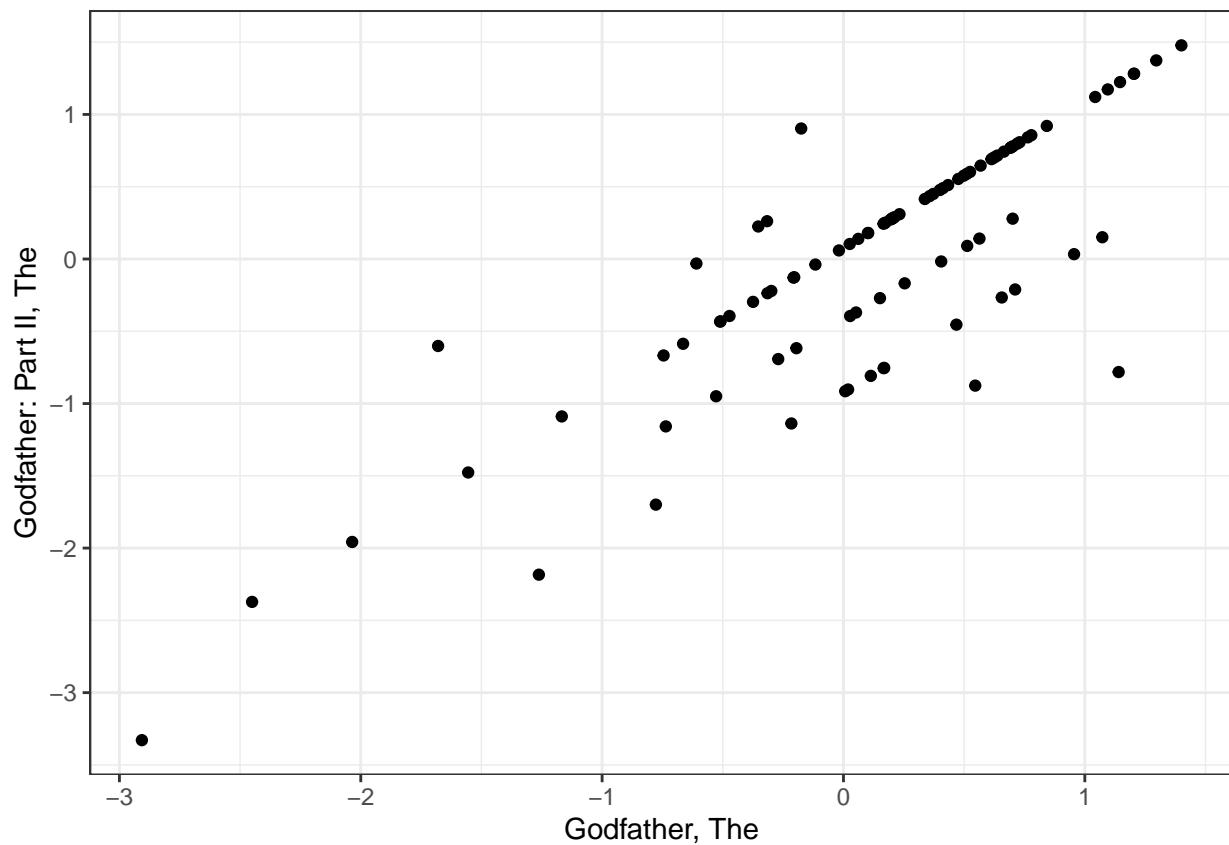
m_1 <- "Godfather, The"

m_2 <- "Godfather: Part II, The"

qplot(y[,m_1], y[,m_2], xlab = m_1, ylab = m_2)

## Warning: Removed 199 rows containing missing values (geom_point).

```



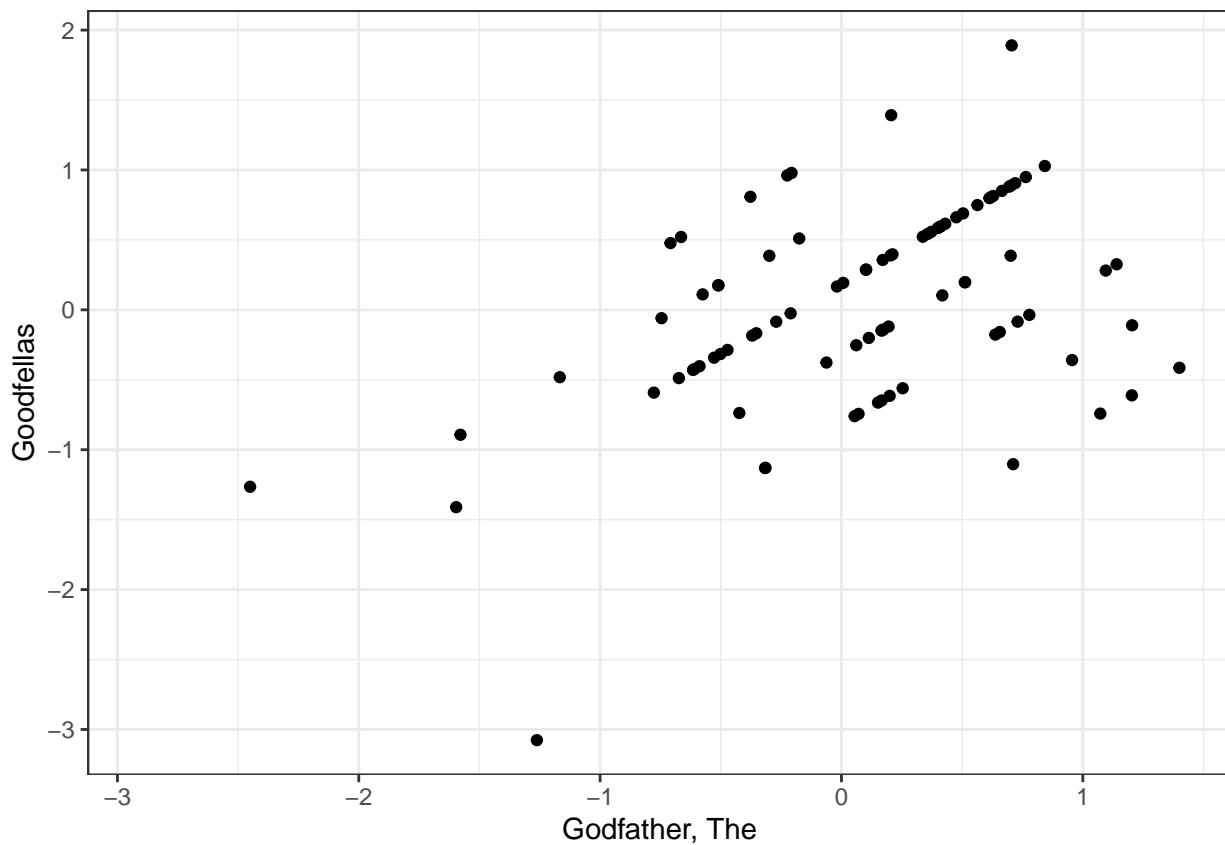
Este gráfico indica que los usuarios a los que les gustó *The Godfather* más de lo que el modelo predeciría también gustaron de más por *The Godfather II*. Lo mismo ocurre para el caso de *The Godfather* y *Goodfellas*:

```
m_1 <- "Godfather, The"
```

```
m_3 <- "Goodfellas"
```

```
qplot(y[,m_1], y[,m_3], xlab = m_1, ylab = m_3)
```

```
## Warning: Removed 204 rows containing missing values (geom_point).
```



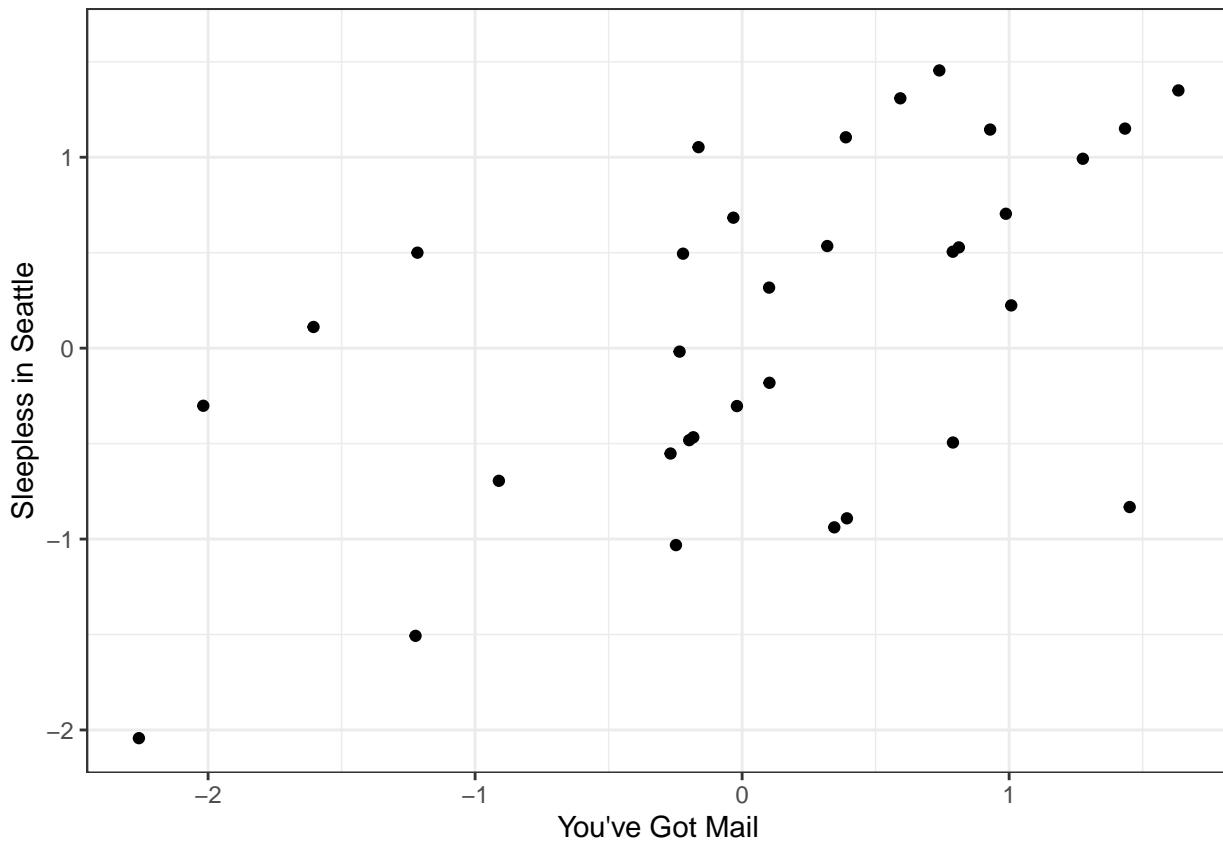
De hecho, encontramos una correlación sustantiva en muchos otros pares de películas:

```
m_4 <- "You've Got Mail"
```

```
m_5 <- "Sleepless in Seattle"
```

```
qplot(y[,m_4], y[,m_5], xlab = m_4, ylab = m_5)
```

```
## Warning: Removed 259 rows containing missing values (geom_point).
```



Si miramos a la correlación entre las siguientes cinco películas, observamos una correlación positiva entre películas de gángsters y las románticas; asimismo, se aprecia una correlación negativa entre las primeras y las segundas:

```
cor(y[, c(m_1, m_2, m_3, m_4, m_5)], use="pairwise.complete") %>%
  knitr::kable()
```

	Godfather, The	Godfather: Part II, The	Goodfellas	You've Got Mail	Sleepless in Seattle
Godfather, The	1.000	0.832	0.454	-0.454	-0.354
Godfather: Part II, The	0.832	1.000	0.540	-0.338	-0.326
Goodfellas	0.454	0.540	1.000	-0.489	-0.367
You've Got Mail	-0.454	-0.338	-0.489	1.000	0.542
Sleepless in Seattle	-0.354	-0.326	-0.367	0.542	1.000

Esto implica que los datos presentan comportamientos que nuestro modelo no contempla. ¿Cómo modelarlos? Para esto, usaremos la **factorización de matrices** y definiremos factores. Suponiendo que existen efectos como el de películas de gángsters y románticas, podemos estrechar las películas a estos dos grupos. Considérese la siguiente ilustración de cómo podríamos usar cierta estructura para predecir los residuales:

```
set.seed(1)

options(digits = 2)

Q <- matrix(c(1, 1, 1, -1, -1), ncol=1)

rownames(Q) <- c(m_1, m_2, m_3, m_4, m_5)

P <- matrix(rep(c(2, 0, -2), c(3, 5, 4)), ncol=1)
```

```
rownames(P) <- 1:nrow(P)

X <- jitter(P %*% t(Q))

X %>%
  knitr::kable(align = "c")
```

Godfather, The	Godfather: Part II, The	Goodfellas	You've Got Mail	Sleepless in Seattle
1.81	2.15	1.81	-1.76	-1.81
1.90	1.91	1.91	-2.31	-1.85
2.06	2.22	1.61	-1.82	-2.02
0.33	0.00	-0.09	-0.07	0.29
-0.24	0.17	0.30	0.26	-0.05
0.32	0.39	-0.13	0.12	-0.20
0.36	-0.10	-0.01	0.23	-0.34
0.13	0.22	0.08	0.04	-0.32
-1.90	-1.65	-2.01	2.02	1.85
-2.35	-2.23	-2.25	2.23	2.01
-2.24	-1.88	-1.74	1.62	2.13
-2.26	-2.30	-1.87	1.98	1.93

Las correlaciones observadas pueden ser explicadas utilizando los siguientes coeficientes (1 para gángsters y -1 a romance).

De hecho, también podemos reducir a los usuarios a tres grupos: aquellos que gustan de las primeras pero no de las segundas, la viceversa, y aquellos indiferentes.

El punto relevante es que podemos reconstruir los datos; así, modelaremos los residuales mediante los parámetros de la siguiente forma.

$$r_{u,i} \approx p_u q_i$$

De ahí viene el nombre de factorización: tenemos una matriz  $r$ , la cual factorizamos en dos elementos, el vector  $p$  y  $q$ . Así, podremos explicar mucho más de la varianza con el siguiente modelo:

$$Y_{u,i} = \mu + b_i + b_u + p_u q_i + \varepsilon_{i,j}$$

Es importante recalcar que nuestra base de datos completa es mucho más compleja que solo dos factores (gángsters y romance)

**8.6.6.2. SVD y PCA** La descomposición matricial que mostramos y que luce como  $r_{u,i} \approx p_{u,1}1_{1,i} + p_{u,2}q_{2,i}$  está relacionada con la descomposición de valores singulares (SVD) y el análisis de componentes principales (PCA), donde el primero es un algoritmo que encuentra vectores  $p$  y  $q$  que permiten escribir una matriz de residuales  $r$  con  $m$  renglones y  $n$  columnas de la siguiente forma:

$$r_{u,i} = p_{u,1}q_{1,i} + p_{u,2}q_{2,i} + \dots + p_{u,m}q_{m,i}$$

Con la ventaja que la variabilidad de estos términos es decreciente y que las  $p$  no están correlacionadas entre sí. El algoritmo también calcula estas variabilidades, de tal forma que sepamos qué tanto de la variabilidad total de la matriz es explicada a medida que agregamos nuevos términos.

En nuestro ejemplo, para calcular la descomposición, haremos a todos los NAs cero

```
y[is.na(y)] <- 0  
y <- sweep(y, 1, rowMeans(y))  
pca <- prcomp(y)
```

Los vectores  $q$  (efecto película) son llamados componentes principales y están alojados en la siguiente matriz:  
`dim(pca$rotation)`

```
## [1] 454 292
```

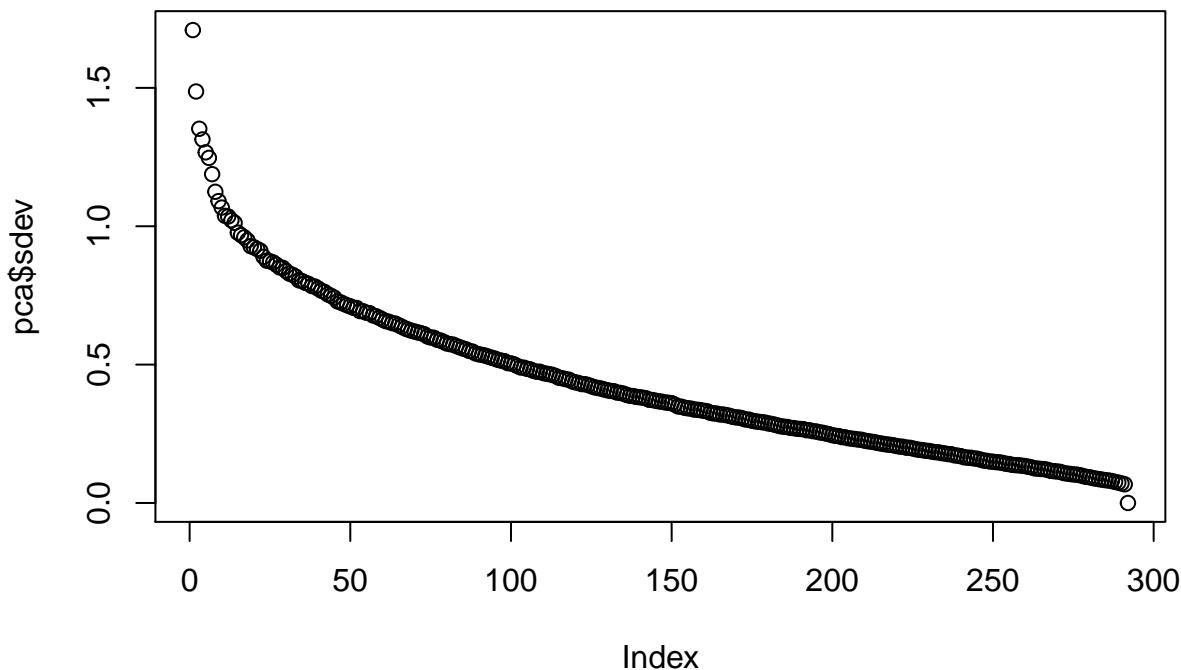
Mientras que los vectores  $p$  (efecto usuario) están en:

```
dim(pca$x)
```

```
## [1] 292 292
```

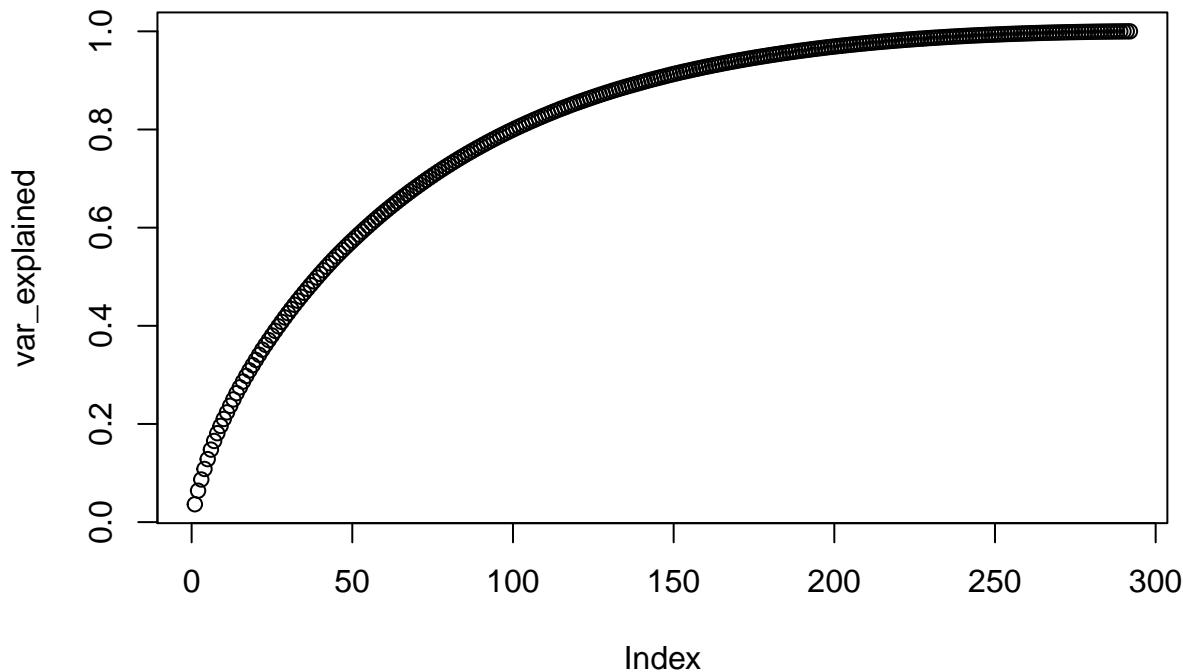
La función “pca()” arroja un elemento con la variabilidad de cada componente principal al cual podemos accesar como sigue y graficarlo:

```
plot(pca$sdev)
```



También podemos ver que con solo algunos de estos componentes principales podemos explicar un gran porcentaje de la variación de los datos:

```
var_explained <- cumsum(pca$sdev^2/sum(pca$sdev^2))  
plot(var_explained)
```

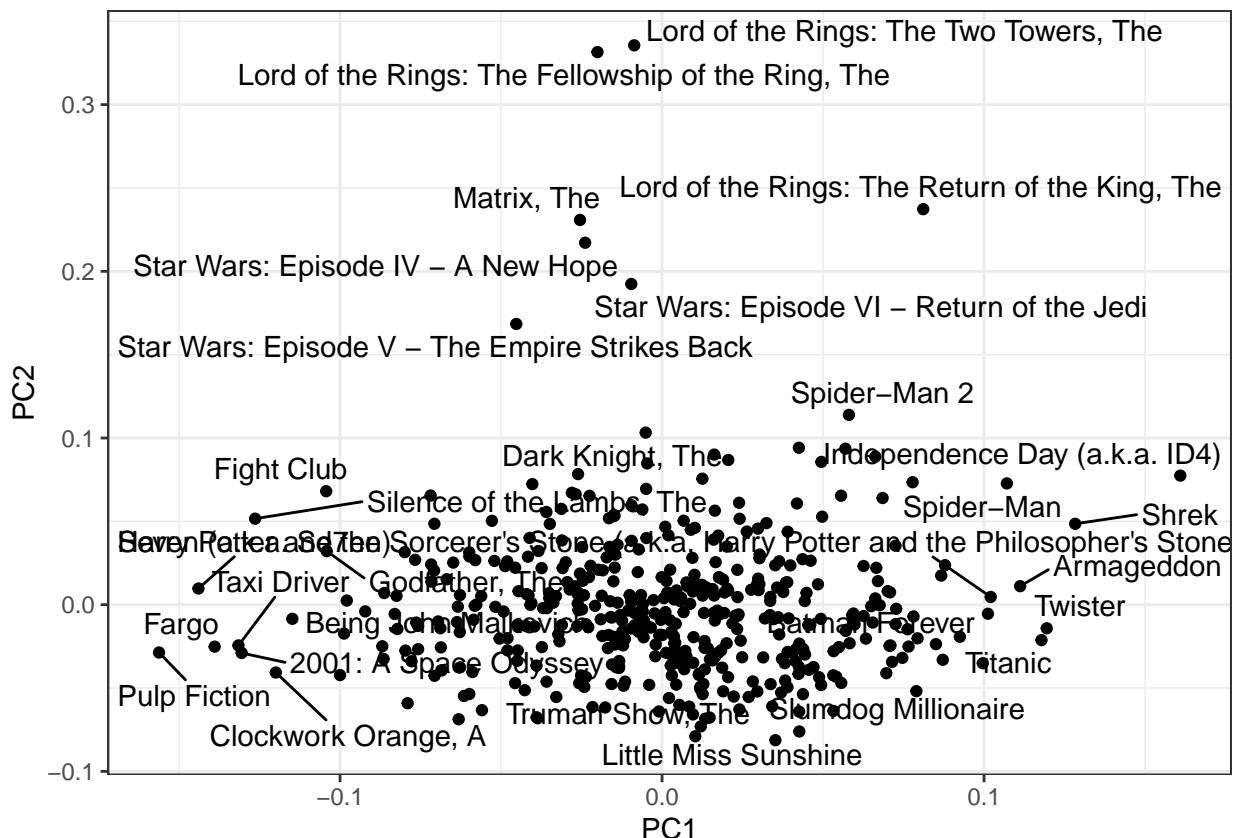


Para observar que los componentes principales efectivamente están capturando algo relevante sobre los datos, podemos graficar, por ejemplo, los primeros dos y etiquetar los puntos con el título de la película:

```
library(ggrepel)

pcs <- data.frame(pca$rotation, name = colnames(y))

pcs %>%
  ggplot(aes(PC1, PC2)) + geom_point() +
  geom_text_repel(aes(PC1, PC2, label=name),
                  data = filter(pcs,
                                PC1 < -0.1 | PC1 > 0.1 | PC2 < -0.075 | PC2 > 0.1))
```



Pero, ¿qué significa este gráfico? El primer CP muestra, en un extremo, películas aclamadas por la crítica, y éxitos taquilleros por el otro:

```

pcs %>%
  select(name, PC1) %>%
  arrange(PC1) %>%
  slice(1:10)

##
## Pulp Fiction
## Seven (a.k.a. Se7en)
## Fargo
## Taxi Driver
## 2001: A Space Odyssey
## Silence of the Lambs, The
## Clockwork Orange, A
## Being John Malkovich
## Fight Club
## Godfather, The

name      PC1
Pulp Fiction -0.16
Seven (a.k.a. Se7en) -0.14
Fargo -0.14
Taxi Driver -0.13
2001: A Space Odyssey -0.13
Silence of the Lambs, The -0.13
Clockwork Orange, A -0.12
Being John Malkovich -0.11
Fight Club -0.10
Godfather, The -0.10

pcs %>%
  select(name, PC1) %>%
  arrange(desc(PC1)) %>%
  slice(1:10)

##
## Independence Day (a.k.a. ID4)
## Shrek

```

```

## Twister
## Titanic
## Armageddon
## Spider-Man
## Harry Potter and the Sorcerer's Stone (a.k.a. Harry Potter and the Philosopher's Stone) Harry Potter
## Batman Forever
## Forrest Gump
## Enemy of the State
## PC1
## Independence Day (a.k.a. ID4) 0.161
## Shrek 0.128
## Twister 0.119
## Titanic 0.118
## Armageddon 0.111
## Spider-Man 0.107
## Harry Potter and the Sorcerer's Stone (a.k.a. Harry Potter and the Philosopher's Stone) 0.102
## Batman Forever 0.101
## Forrest Gump 0.100
## Enemy of the State 0.092

```

También podemos ver que el segundo CP captura información importante sobre la estructura de datos: películas independientes en un extremo y *nerd favorites* en otro:

```

pcs %>%
  select(name, PC2) %>%
  arrange(PC2) %>%
  slice(1:10)

##                                     name
## Little Miss Sunshine           Little Miss Sunshine
## Truman Show, The              Truman Show, The
## Slumdog Millionaire          Slumdog Millionaire
## Mars Attacks!                 Mars Attacks!
## American Beauty                American Beauty
## Amelie (Fabuleux destin d'Amélie Poulain, Le) Amelie (Fabuleux destin d'Amélie Poulain, Le)
## City of God (Cidade de Deus)   City of God (Cidade de Deus)
## Monty Python's Life of Brian Monty Python's Life of Brian
## Shawshank Redemption, The     Shawshank Redemption, The
## Beautiful Mind, A             Beautiful Mind, A
##                                     PC2
## Little Miss Sunshine          -0.081
## Truman Show, The              -0.079
## Slumdog Millionaire          -0.076
## Mars Attacks!                 -0.073
## American Beauty                -0.069
## Amelie (Fabuleux destin d'Amélie Poulain, Le) -0.068
## City of God (Cidade de Deus)   -0.068
## Monty Python's Life of Brian -0.068
## Shawshank Redemption, The     -0.066
## Beautiful Mind, A             -0.064

pcs %>%
  select(name, PC2) %>%
  arrange(desc(PC2)) %>%
  slice(1:10)

```

```

##                                         name
## Lord of the Rings: The Two Towers, The   Lord of the Rings: The Two Towers, The
## Lord of the Rings: The Fellowship of the Ring, The Lord of the Rings: The Fellowship of the Ring, The
## Lord of the Rings: The Return of the King, The   Lord of the Rings: The Return of the King, The
## Matrix, The                                Matrix, The
## Star Wars: Episode IV - A New Hope      Star Wars: Episode IV - A New Hope
## Star Wars: Episode VI - Return of the Jedi  Star Wars: Episode VI - Return of the Jedi
## Star Wars: Episode V - The Empire Strikes Back  Star Wars: Episode V - The Empire Strikes Back
## Spider-Man 2                            Spider-Man 2
## Dark Knight, The                        Dark Knight, The
## X2: X-Men United                      X2: X-Men United

##                                     PC2
## Lord of the Rings: The Two Towers, The    0.336
## Lord of the Rings: The Fellowship of the Ring, The 0.332
## Lord of the Rings: The Return of the King, The 0.237
## Matrix, The                           0.231
## Star Wars: Episode IV - A New Hope     0.217
## Star Wars: Episode VI - Return of the Jedi 0.192
## Star Wars: Episode V - The Empire Strikes Back 0.168
## Spider-Man 2                         0.114
## Dark Knight, The                     0.103
## X2: X-Men United                   0.094

```

Así, mediante el análisis de componentes principales hemos mostrado que la factorización de matrices puede detectar estructuras importantes en nuestros datos.

### 8.6.7. Assessment 33

In this exercise, we will see one of the ways that this decomposition can be useful. To do this, we will construct a dataset that represents grade scores for 100 students in 24 different subjects. The overall average has been removed so this data represents the percentage point each student received above or below the average test score. So a 0 represents an average grade (C), a 25 is a high grade (A+), and a -25 represents a low grade (F). You can simulate the data like this:

```
set.seed(1987, sample.kind="Rounding")

## Warning in set.seed(1987, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
n <- 100

k <- 8

Sigma <- 64 * matrix(c(1, .75, .5, .75, 1, .5, .5, .5, 1), 3, 3)

m <- MASS::mvrnorm(n, rep(0, 3), Sigma)

m <- m[order(rowMeans(m), decreasing = TRUE),]

y <- m %x% matrix(rep(1, k), nrow = 1) + matrix(rnorm(matrix(n*k*3)), n, k*3)

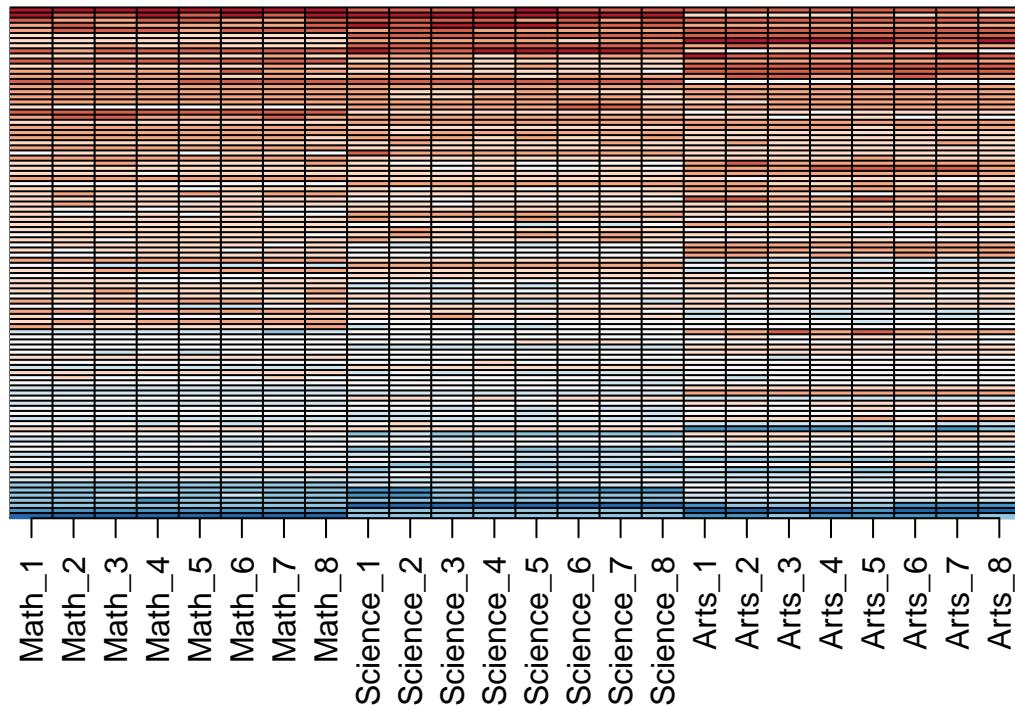
colnames(y) <- c(paste(rep("Math",k), 1:k, sep="_"),
                  paste(rep("Science",k), 1:k, sep="_"),
                  paste(rep("Arts",k), 1:k, sep="_"))
```

Our goal is to describe the student performances as succinctly as possible. For example, we want to know if these test results are all just a random independent numbers. Are all students just about as good? Does being good in one subject imply you will be good in another? How does the SVD help with all this? We will go step by step to show that with just three relatively small pairs of vectors we can explain much of the variability in this 100x24 dataset.

You can visualize the 24 test scores for the 100 students by plotting an image:

```
my_image <- function(x, zlim = range(x), ...){
  colors = rev(RColorBrewer::brewer.pal(9, "RdBu"))
  cols <- 1:ncol(x)
  rows <- 1:nrow(x)
  image(cols, rows, t(x[rev(rows), , drop=FALSE]), xaxt = "n", yaxt = "n",
        xlab="", ylab="", col = colors, zlim = zlim, ...)
  abline(h=rows + 0.5, v = cols + 0.5)
  axis(side = 1, cols, colnames(x), las = 2)
}

my_image(y)
```



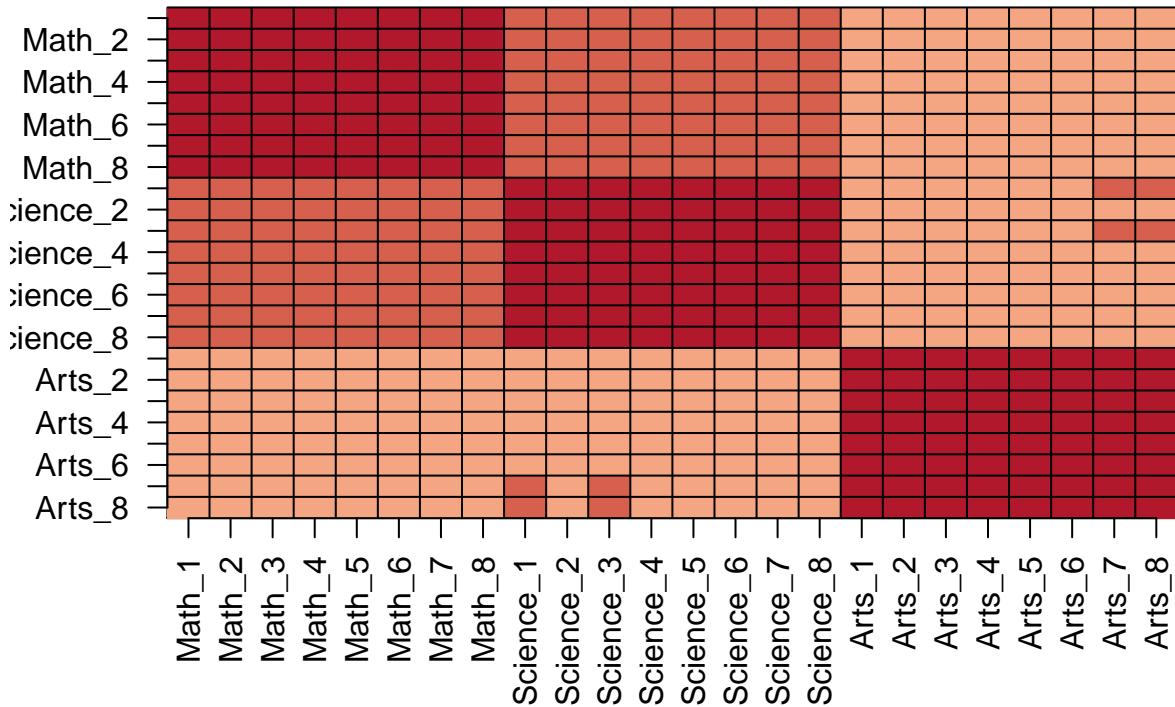
You can examine the correlation between the test scores directly like this:

```
my_image(cor(y), zlim = c(-1,1))

range(cor(y))

## [1] 0.49 1.00

axis(side = 2, 1:ncol(y), rev(colnames(y)), las = 2)
```



Remember that orthogonality means that  $U^T U$  and  $V^T V$  are equal to the identity matrix. This implies that we can also rewrite the decomposition as:

$$YV = UD \quad \text{or} \quad U^T Y = DV^T$$

We can think of  $YV$  and  $U^T Y$  as two transformations of  $Y$  that preserve the total variability of  $Y$  since  $U$  and  $V$  are orthogonal. Use the function `svd()` to compute the SVD of  $y$ . This function will return  $,U$ ,  $V$  and the diagonal entries of  $D$ .

```
s <- svd(y)

names(s)

## [1] "d" "u" "v"
```

You can check that the SVD works by typing:

```
y_svd <- s$u %*% diag(s$d) %*% t(s$v)

max(abs(y - y_svd))

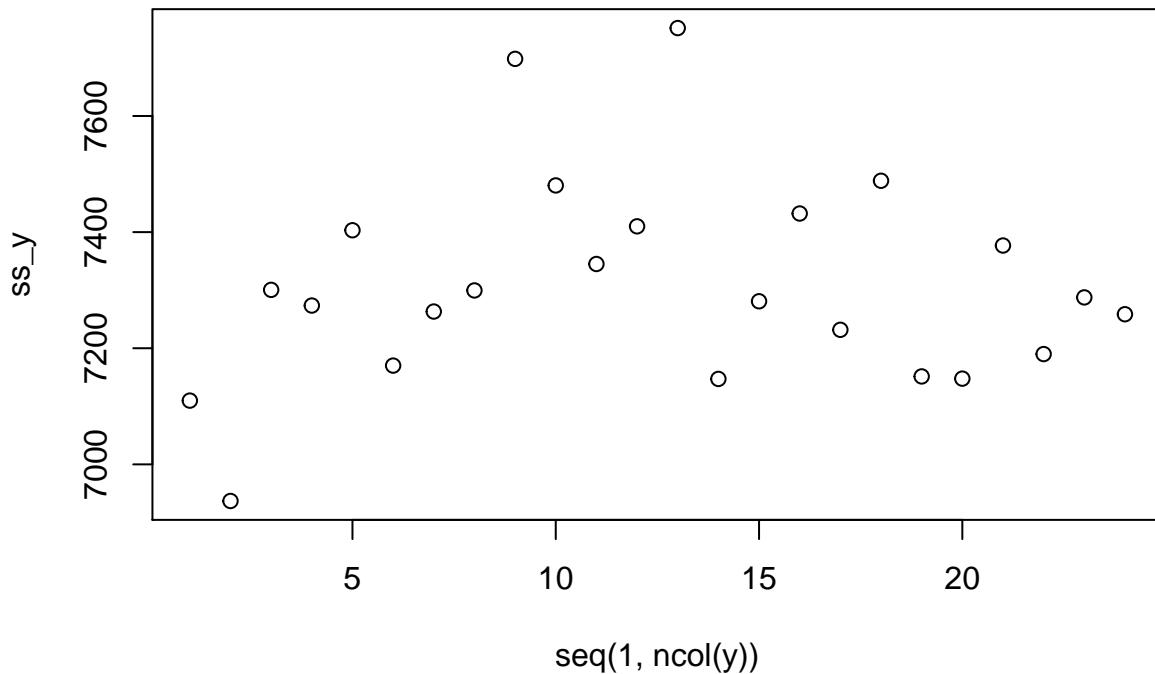
## [1] 5.3e-14
```

Compute the sum of squares of the columns of  $Y$  and store them in `ss_y`. Then compute the sum of squares of columns of the transformed  $YV$  and store them in `ss_yv`. Confirm that `sum(ss_y)` is equal to `sum(ss_yv)`. What is the value of `sum(ss_y)` (and also the value of `sum(ss_yv)`)?

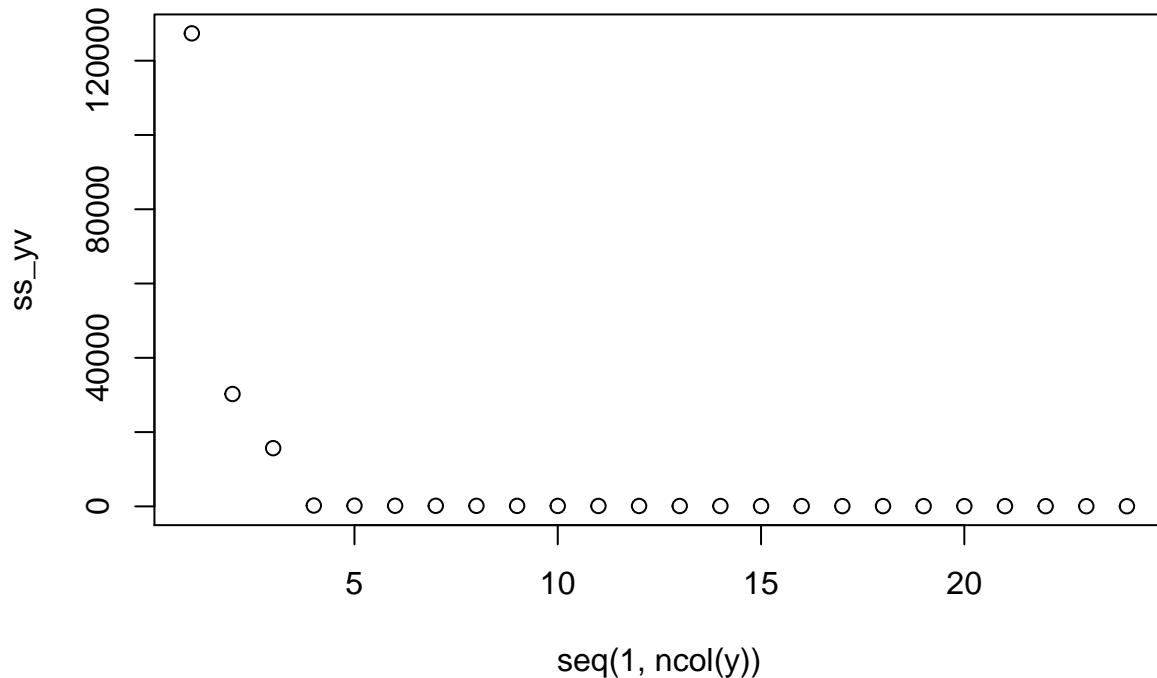
```
ss_y <- apply(y^2, 2, sum)
ss_yv <- apply((y %*% s$v)^2, 2, sum)
sum(ss_y)
## [1] 175435
sum(ss_yv)
## [1] 175435
```

We see that the total sum of squares is preserved. This is because is orthogonal. Now to start understanding how  $YV$  is useful, plot `ss_y` against the column number and then do the same for `ss_yv`.

```
plot(seq(1, ncol(y)), ss_y)
```

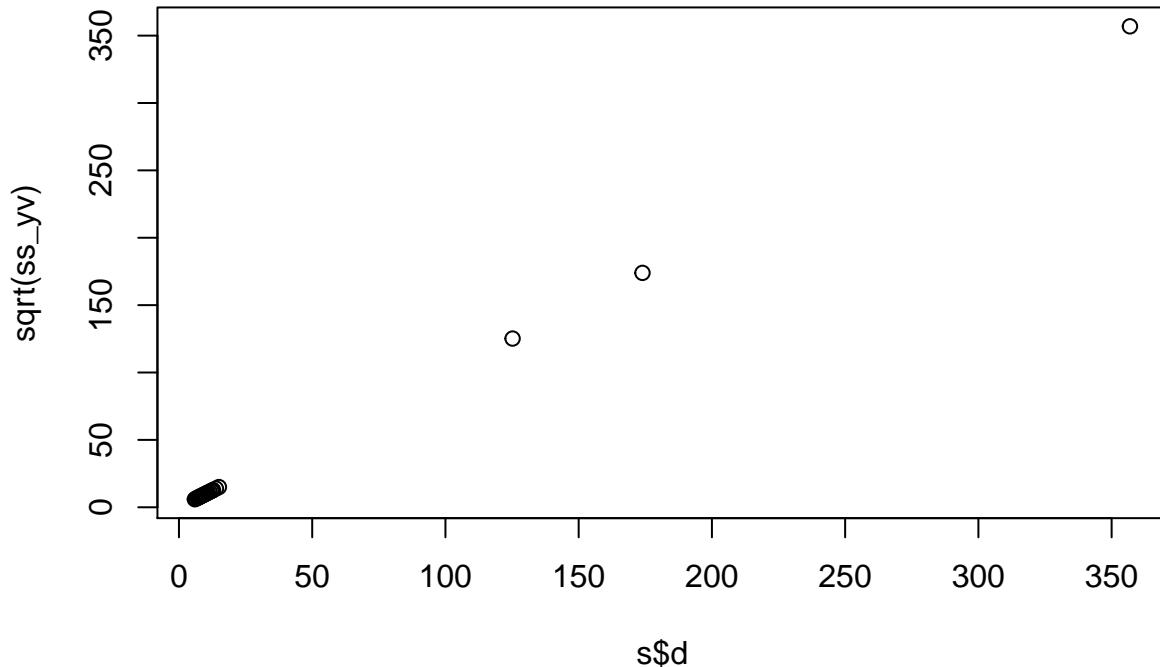


```
plot(seq(1, ncol(y)), ss_yv)
```



Now notice that we didn't have to compute `ss_yv` because we already have the answer. How? Remember that  $YV = UD$  and because  $U$  is orthogonal, we know that the sum of squares of the columns of  $UD$  are the diagonal entries of  $D$  squared. Confirm this by plotting the square root of `ss_yv` versus the diagonal entries of  $D$ .

```
plot(s$d, sqrt(ss_yv))
```



What proportion of the total variability is explained by the first three columns of ?

```
sum(s$d[1:3]^2) / sum(s$d^2)
```

```
## [1] 0.99
```

Before we continue, let's show a useful computational trick to avoid creating the matrix  $\text{diag}(s\$d)$ . To motivate this, we note that if we write  $U$  out in its columns  $[U_1, U_2, \dots, U_p]$  then  $UD$  is equal to:

$$UD = [U_1 d_{1,1}, U_2 d_{2,2}, \dots, U_p d_{p,p}]$$

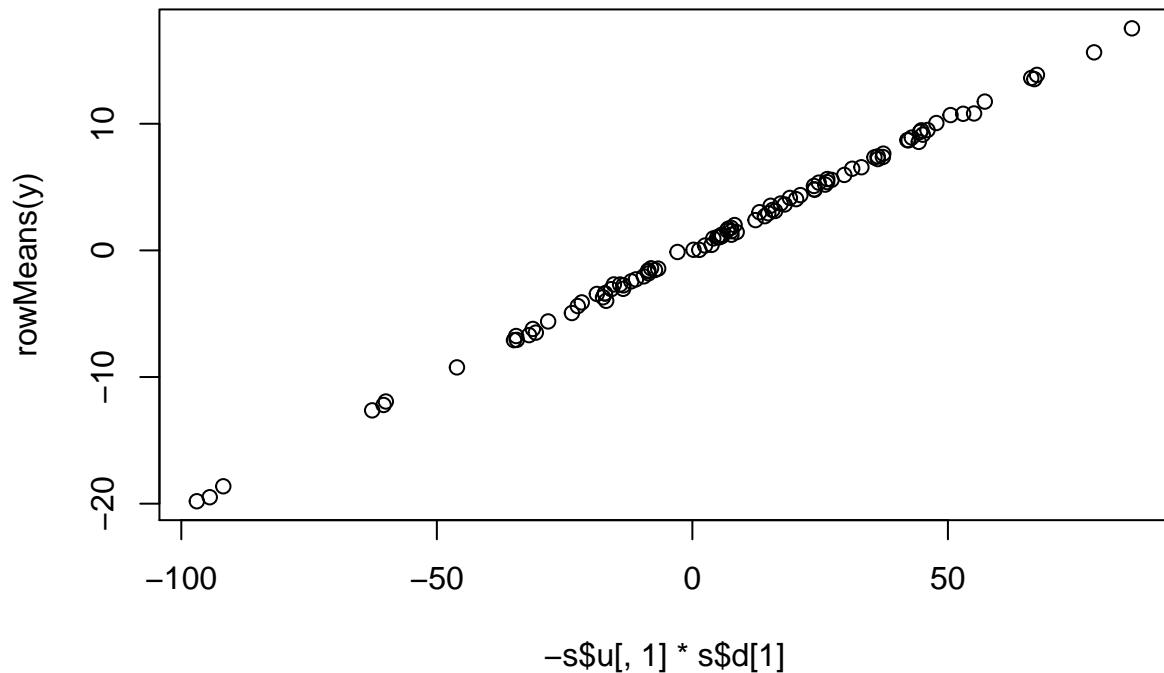
Use the `sweep` function to compute without constructing  $\text{diag}(s\$d)$  or using matrix multiplication.

```
identical(s$u %*% diag(s$d), sweep(s$u, 2, s$d, FUN = "*"))
```

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

We know that  $U_1 d_{1,1}$ , the first column of  $UD$ , has the most variability of all the columns of  $UD$ . Earlier we looked at an image of  $Y$  using `my_image(y)`, in which we saw that the student to student variability is quite large and that students that are good in one subject tend to be good in all. This implies that the average (across all subjects) for each student should explain a lot of the variability. Compute the average score for each student, plot it against  $U_1 d_{1,1}$ , and describe what you find.

```
plot(-s$u[,1]*s$d[1], rowMeans(y))
```



### 8.6.8. Assessment 34

We want to explore the tissue\_gene\_expression predictors by plotting them.

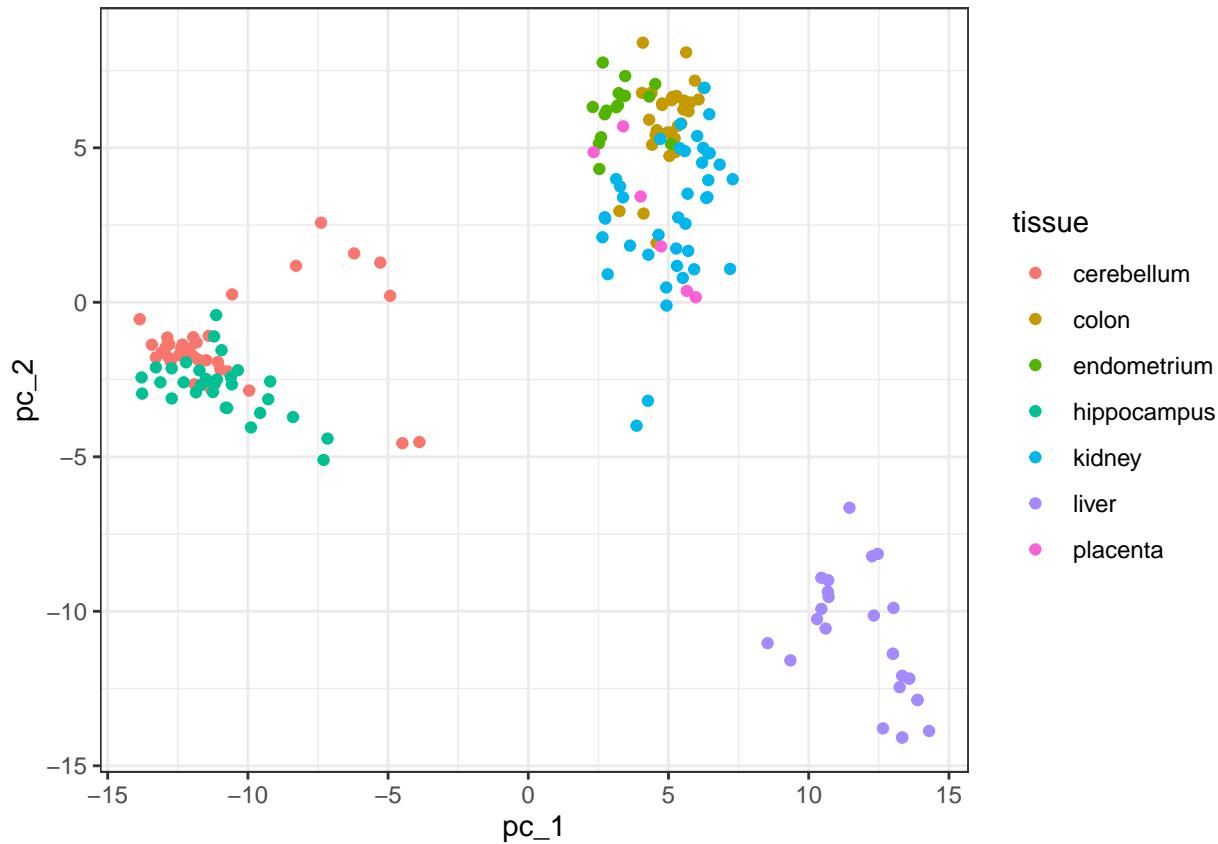
```
data("tissue_gene_expression")
dim(tissue_gene_expression$x)
```

```
## [1] 189 500
```

We want to get an idea of which observations are close to each other, but, as you can see from the dimensions, the predictors are 500-dimensional, making plotting difficult. Plot the first two principal components with color representing tissue type. Which tissue is in a cluster by itself?

```
pc <- prcomp(tissue_gene_expression$x)

data.frame(pc_1 = pc$x[,1], pc_2 = pc$x[,2],
           tissue = tissue_gene_expression$y) %>%
  ggplot(aes(pc_1, pc_2, color = tissue)) +
  geom_point()
```

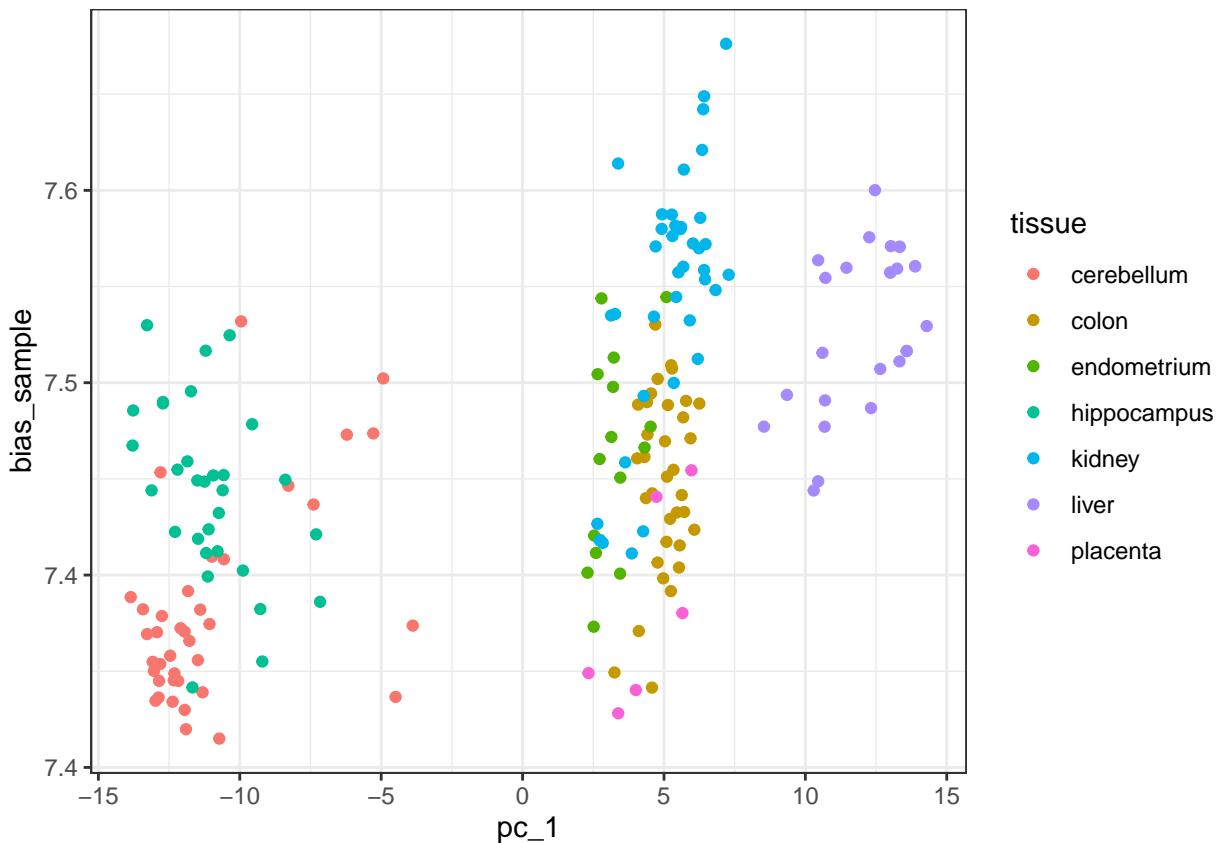


The predictors for each observation are measured using the same device and experimental procedure. This introduces biases that can affect all the predictors from one observation. For each observation, compute the average across all predictors, and then plot this against the first PC with color representing tissue. Report the correlation.

```
bias_sample <- rowMeans(tissue_gene_expression$x)

data.frame(pc_1 = pc$x[,1], bias_sample,
           tissue = tissue_gene_expression$y) %>%
```

```
ggplot(aes(pc_1, bias_sample, color = tissue)) +
  geom_point()
```



```
cor(pc$x[,1], bias_sample)
```

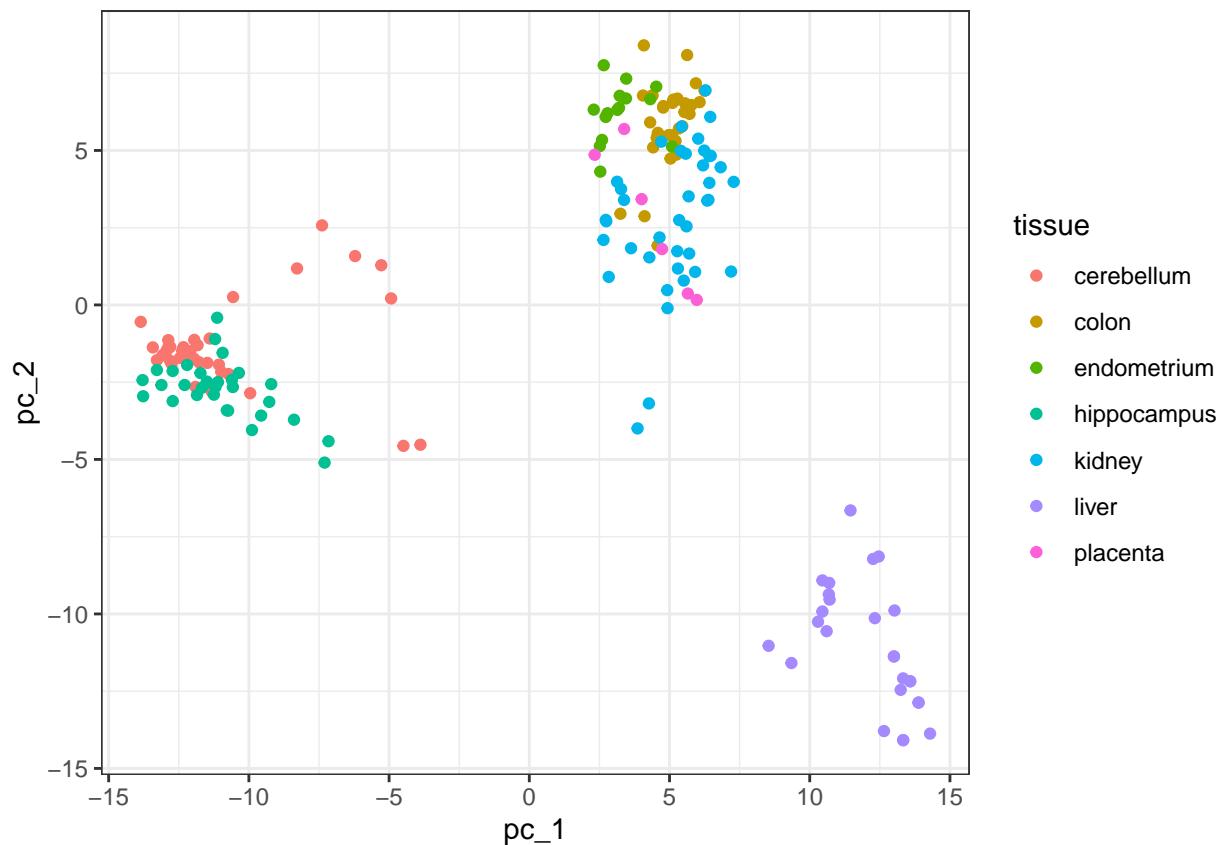
```
## [1] 0.6
```

We see an association with the first PC and the observation averages. Redo the PCA but only after removing the center. Part of the code is provided for you.

```
x <- with(tissue_gene_expression, sweep(x, 1, rowMeans(x)))

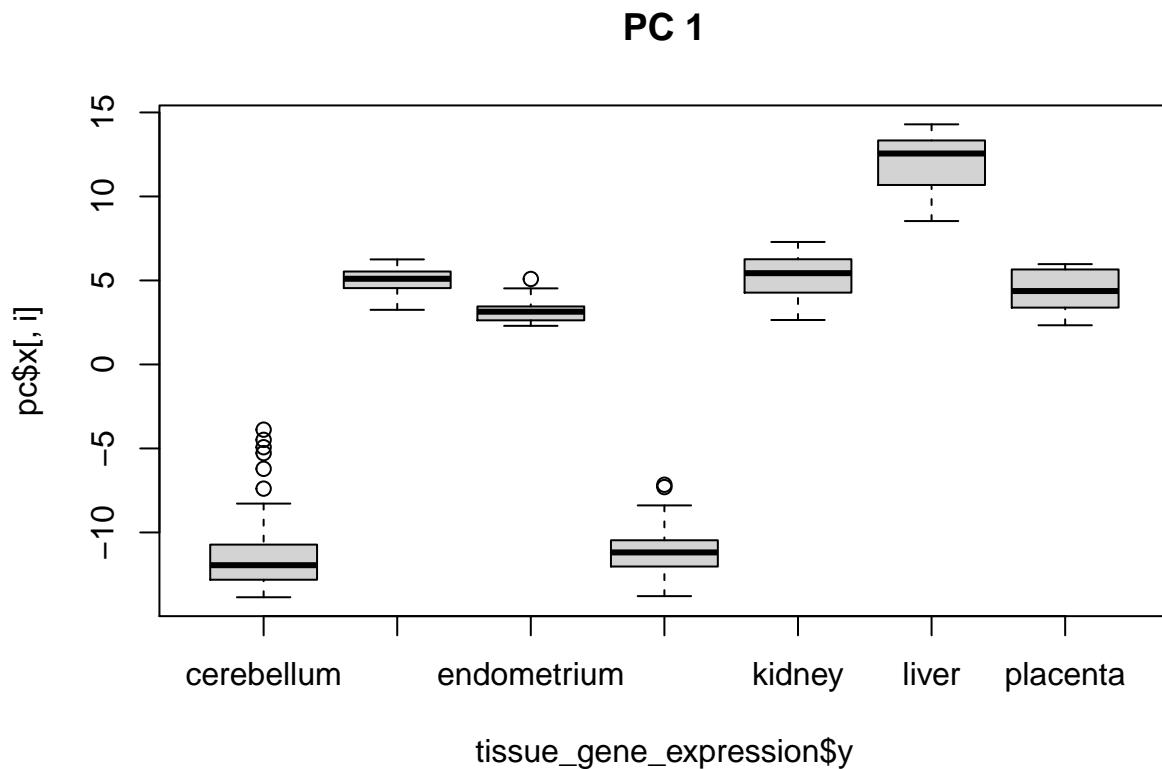
pc2 <- prcomp(x)

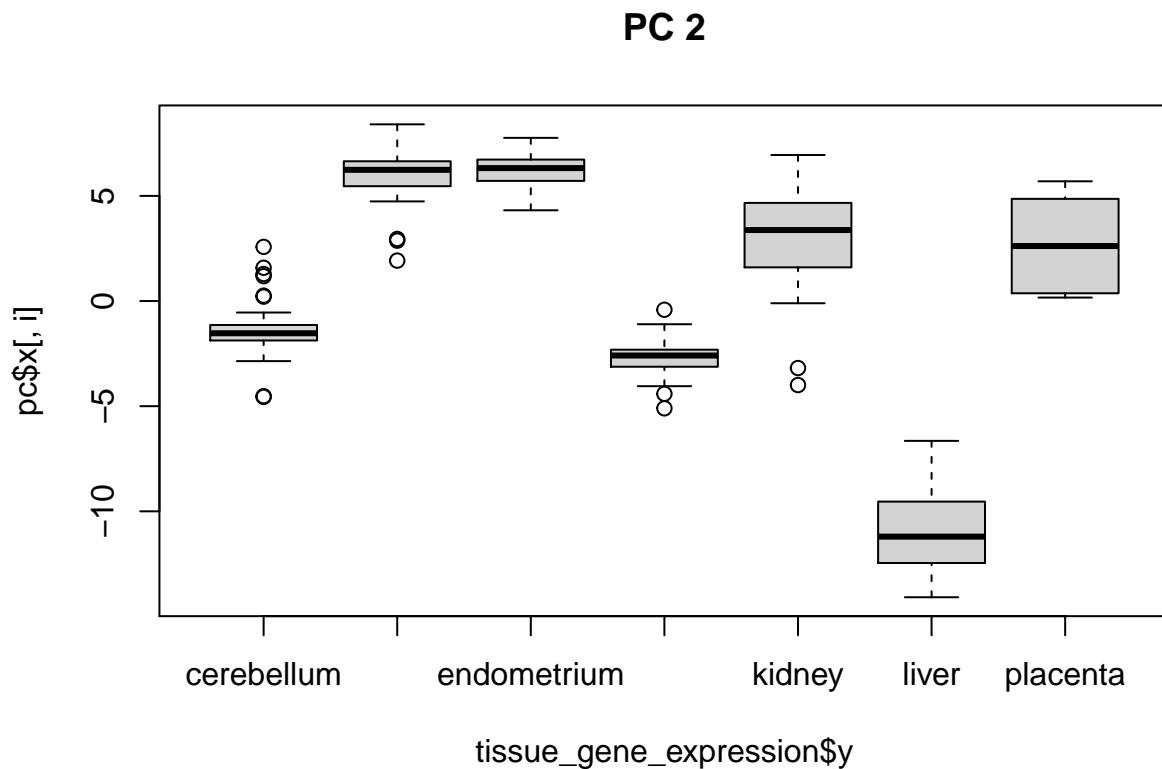
data.frame(pc_1 = pc$x[,1], pc_2 = pc$x[,2],
           tissue = tissue_gene_expression$y) %>%
  ggplot(aes(pc_1, pc_2, color = tissue)) +
  geom_point()
```

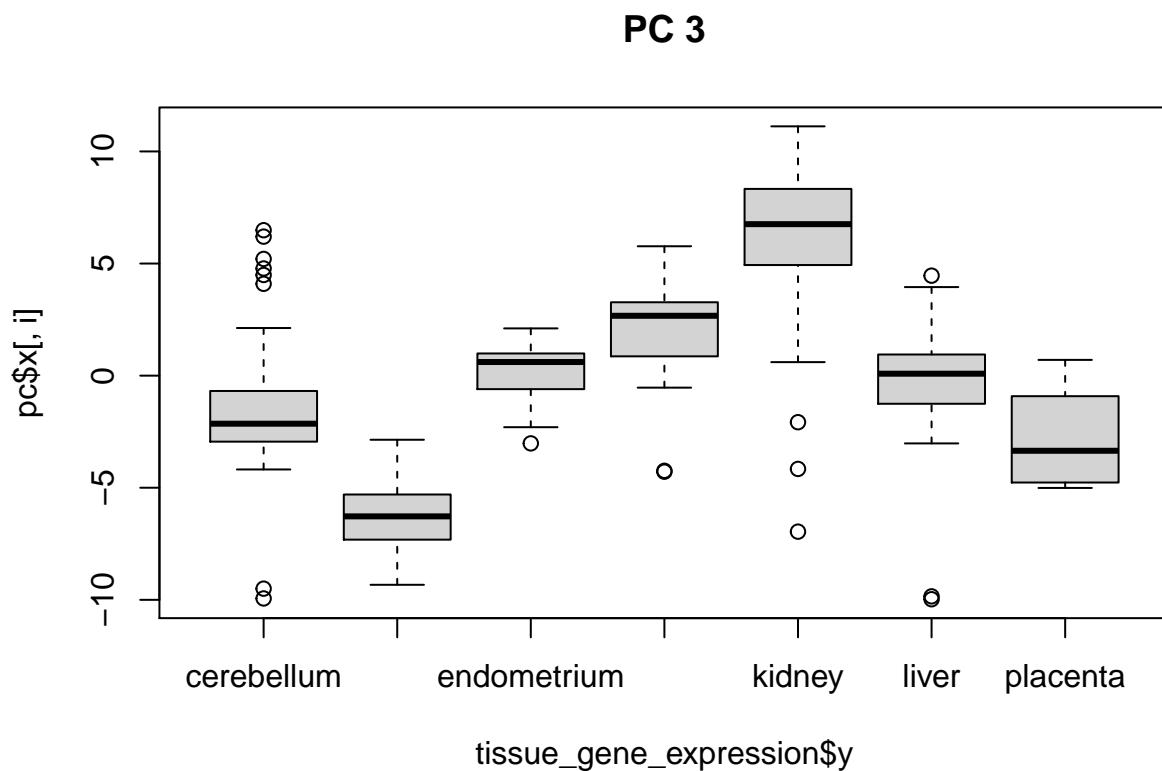


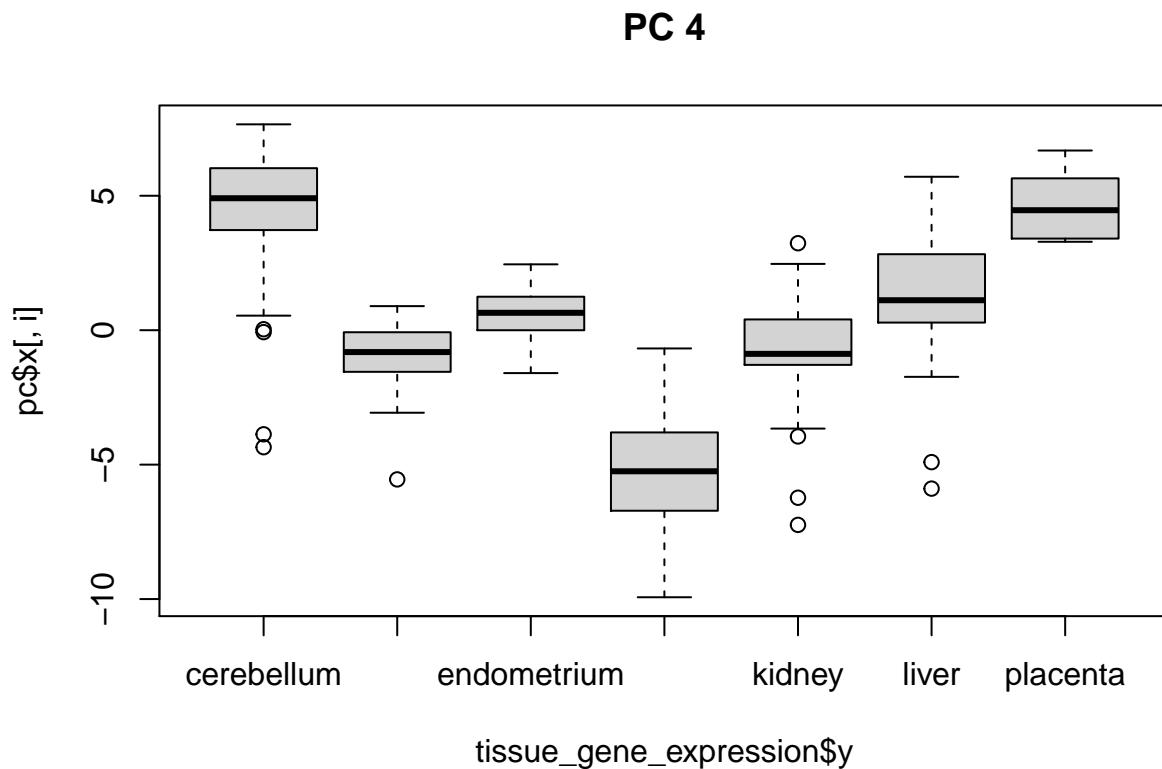
For the first 10 PCs, make a boxplot showing the values for each tissue.

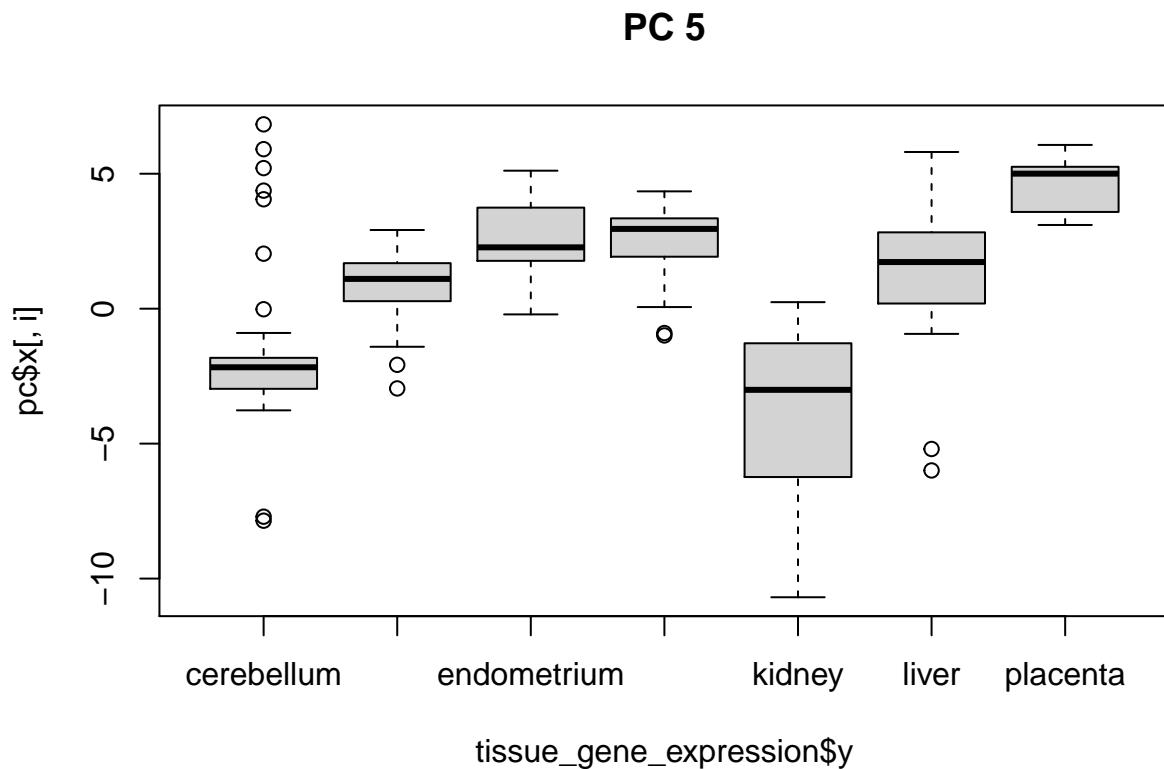
```
for(i in 1:10){  
  boxplot(pc$x[,i] ~ tissue_gene_expression$y, main = paste("PC", i))  
}
```

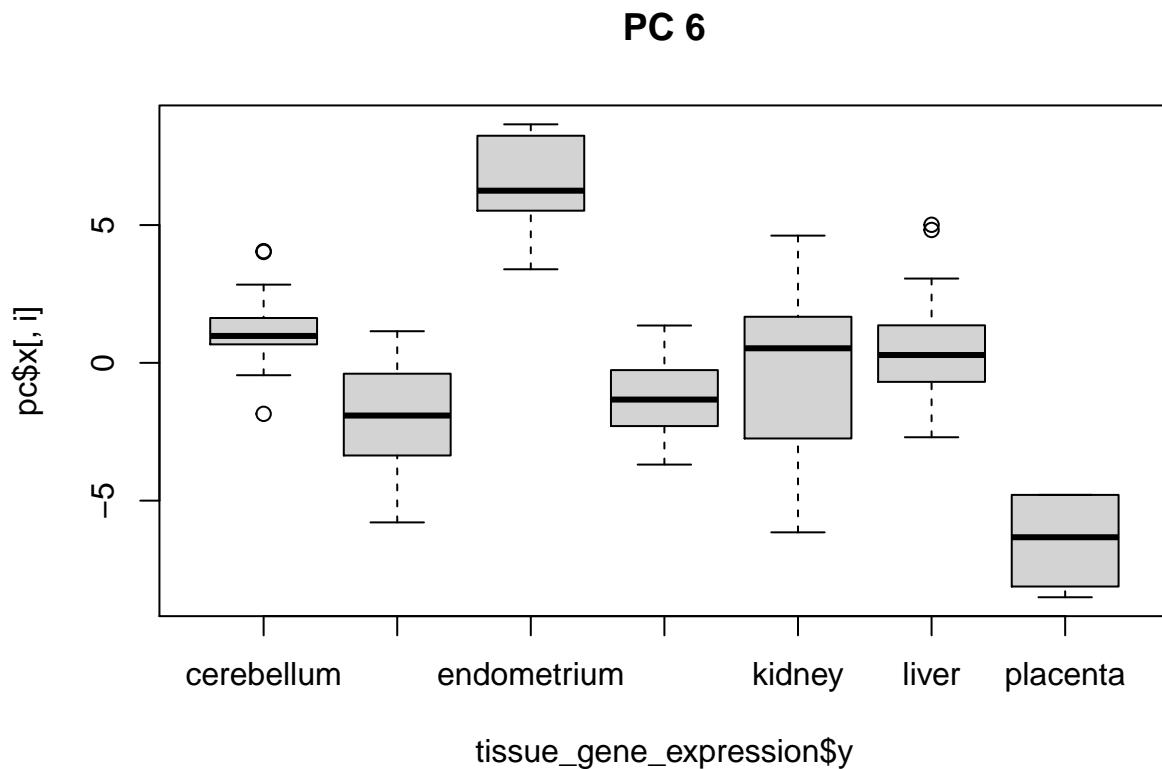


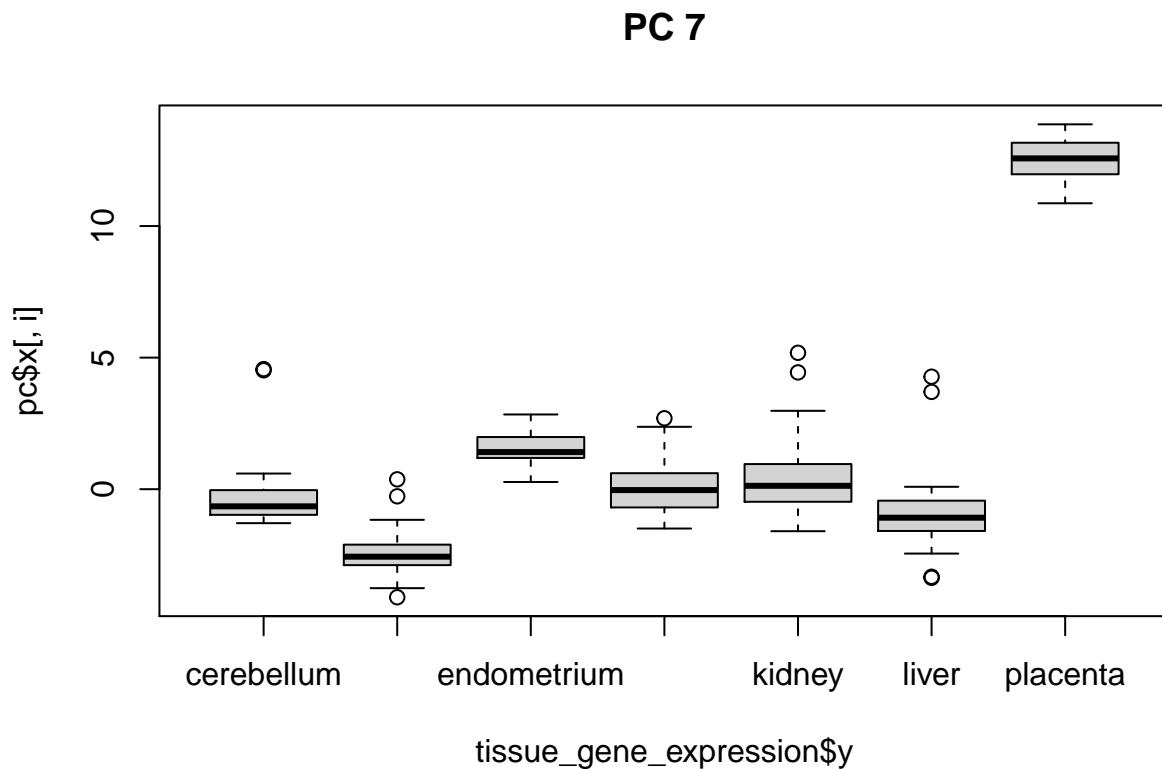


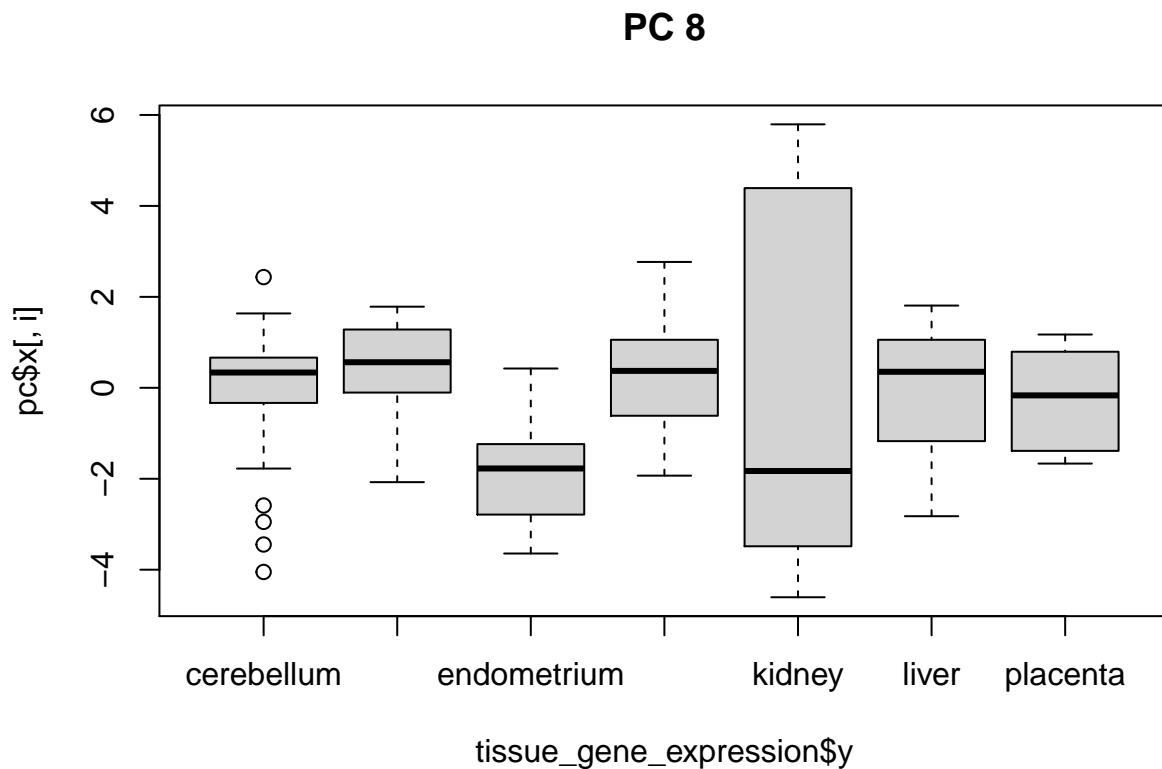


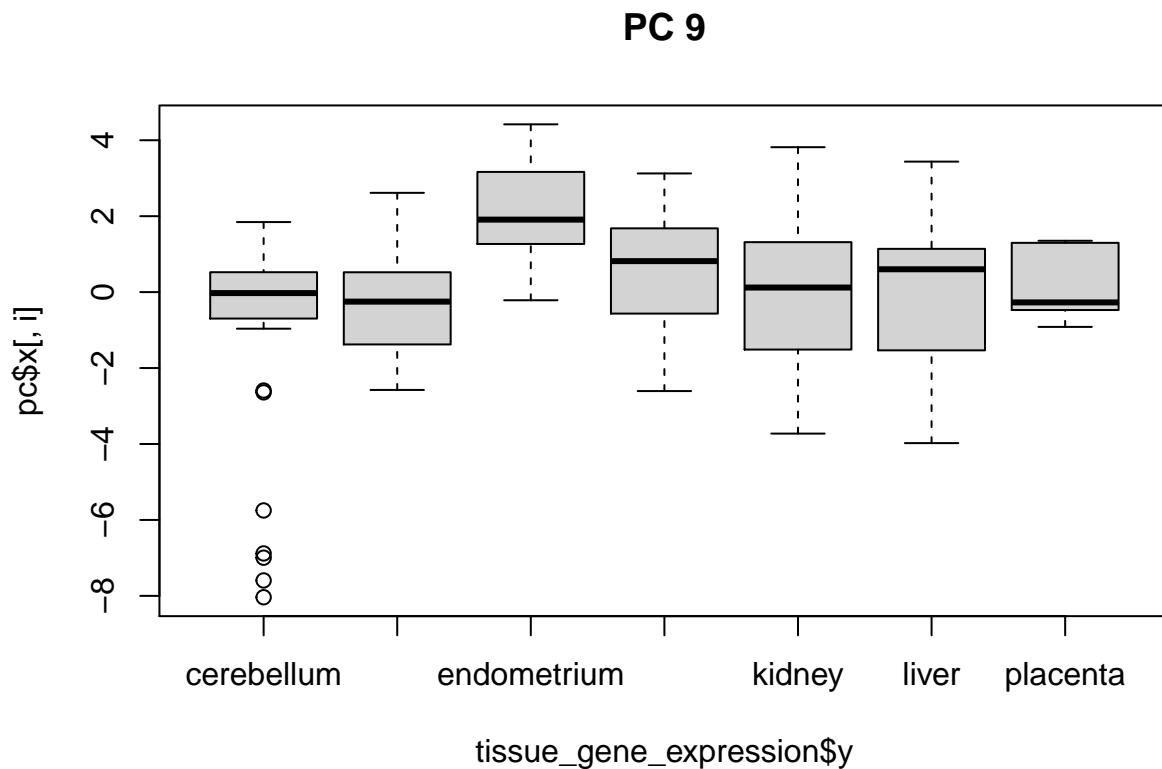


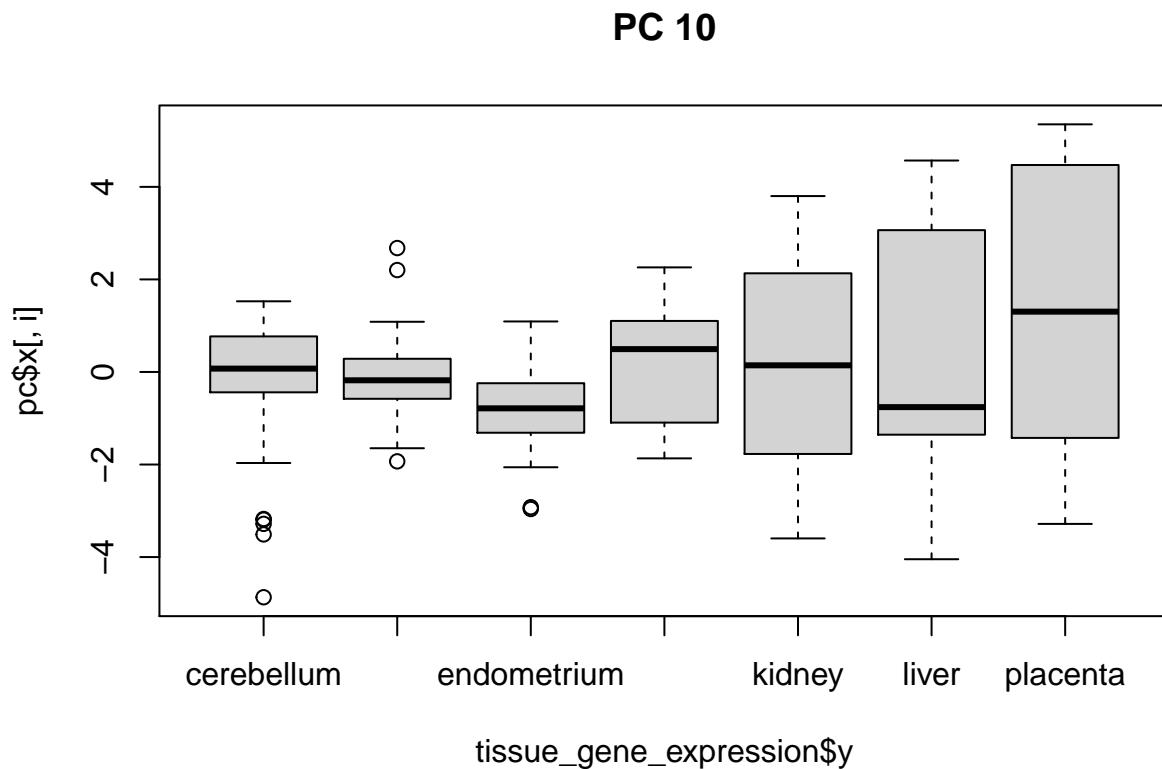






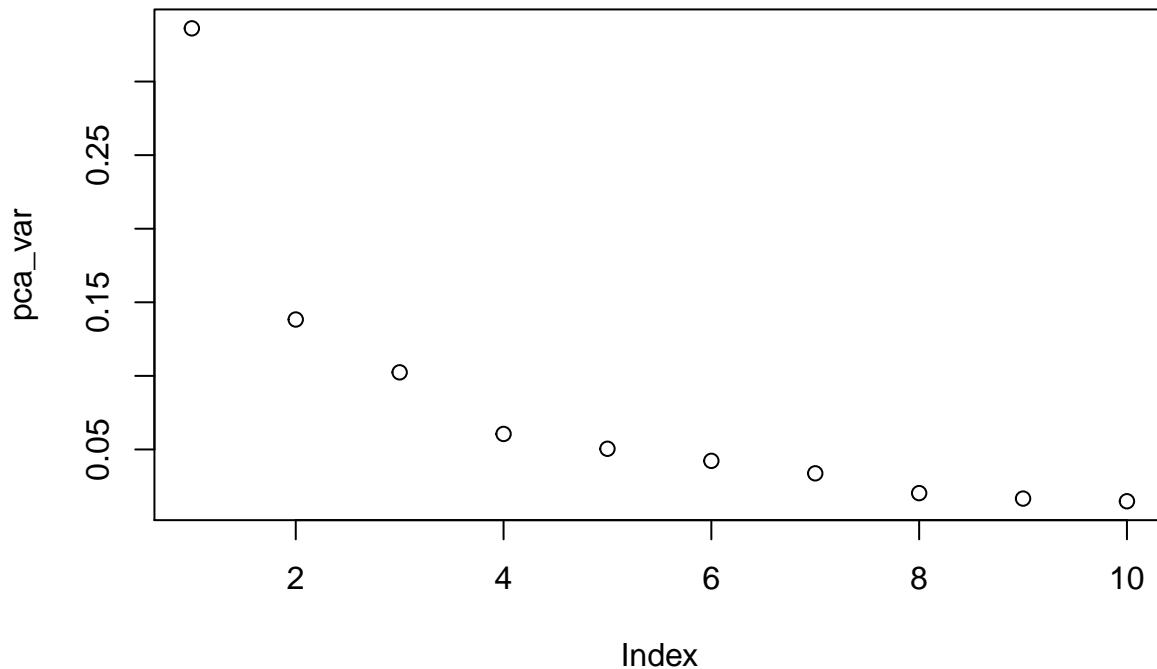






Plot the percent variance explained by PC number. Hint: use the summary function. How many PCs are required to reach a cumulative percent variance explained greater than 50 %?

```
pca_var <- sapply(pc2$sdev[1:10], function(pc_i){  
  pc_i^2 / (t(pc2$sdev) %*% pc2$sdev)  
})  
  
plot(pca_var)
```



```
cumsum(pca_var)
```

```
## [1] 0.34 0.47 0.58 0.64 0.69 0.73 0.76 0.78 0.80 0.82
```

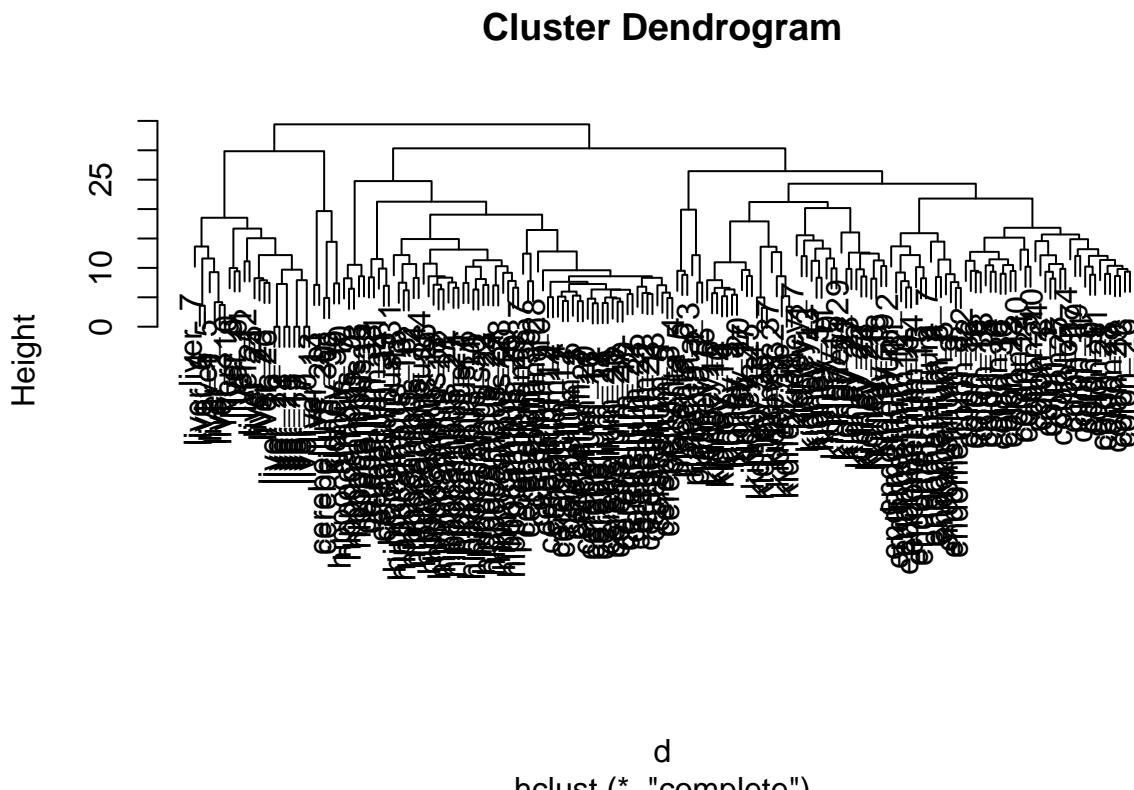
Load the tissue\_gene\_expression dataset. Remove the row means and compute the distance between each observation. Store the result in d. Which of the following lines of code correctly does this computation?

```
d <- dist(tissue_gene_expression$x - rowMeans(tissue_gene_expression$x))
```

Make a hierarchical clustering plot and add the tissue types as labels. You will observe multiple branches. Which tissue type is in the branch farthest to the left?

```
h <- hclust(d)
```

```
plot(h)
```



Select the 50 most variable genes. Make sure the observations show up in the columns, that the predictor are centered, and add a color bar to show the different tissue types. Hint: use the ColSideColors argument to assign colors. Also, use `col = RColorBrewer::brewer.pal(11, "RdBu")` for a better use of colors. Part of the code is provided for you here:

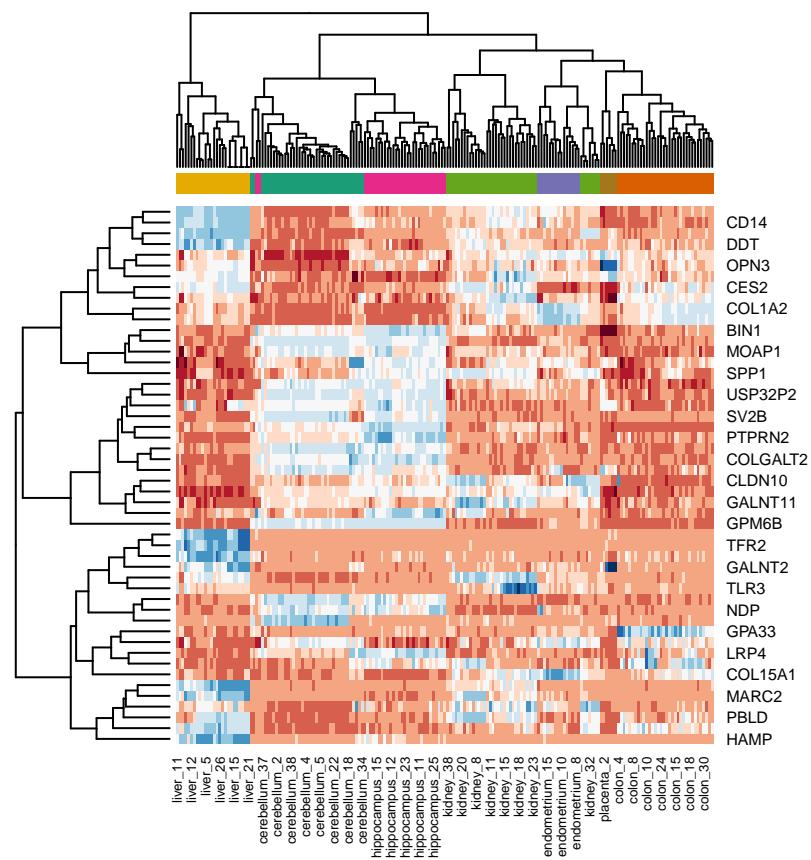
```
library(RColorBrewer)

sds <- matrixStats::colSds(tissue_gene_expression$x)

ind <- order(sds, decreasing = TRUE)[1:50]

colors <- brewer.pal(7, "Dark2")[as.numeric(tissue_gene_expression$y)]

heatmap(t(tissue_gene_expression$x[,ind]), col = brewer.pal(11, "RdBu"), scale = "row", ColSideColors =
```



### 8.6.9. Comprehensive Assessment

The brca dataset from the dslabs package contains information about breast cancer diagnosis biopsy samples for tumors that were determined to be either benign (not cancer) and malignant (cancer). The brca object is a list consisting of:

- brca\$y: a vector of sample classifications (“B” = benign or “M” = malignant)
- brca\$x: a matrix of numeric features describing properties of the shape and size of cell nuclei extracted from biopsy microscope images

For these exercises, load the data by setting your options and loading the libraries and data as shown in the code here:

```
options(digits = 3)
library(matrixStats)
library(tidyverse)
library(caret)
library(dslabs)
data(brca)
```

How many samples are in the dataset? How many predictors are in the matrix?

```
dim(brca[["x"]])
```

```
## [1] 569 30
```

What proportion of the samples are malignant?

```
mean(brca[["y"]]== "M")
```

```
## [1] 0.373
```

Which column number has the highest mean?

```
which.max(colMeans(brca$x))
```

```
## area_worst
##          24
```

Which column number has the lowest standard deviation?

```
which.min(colSds(brca$x))
```

```
## [1] 20
```

Use sweep() two times to scale each column: subtract the column means of brca\$x, then divide by the column standard deviations of brca\$x. After scaling, what is the standard deviation of the first column?

```
x_centered <- sweep(brca$x, 2, colMeans(brca$x))

x_scaled <- sweep(x_centered, 2, colSds(brca$x), FUN = "/")

sd(x_scaled[,1])

## [1] 1
```

After scaling, what is the median value of the first column?

```
median(x_scaled[,1])

## [1] -0.215
```

Calculate the distance between all samples using the scaled matrix. What is the average distance between the first sample, which is benign, and other benign samples?

```
d_samples <- dist(x_scaled)

dist_BtoB <- as.matrix(d_samples)[1, brca$y == "B"]

mean(dist_BtoB[2:length(dist_BtoB)]))

## [1] 4.41
```

What is the average distance between the first sample and malignant samples?

```
dist_BtoM <- as.matrix(d_samples)[1, brca$y == "M"]

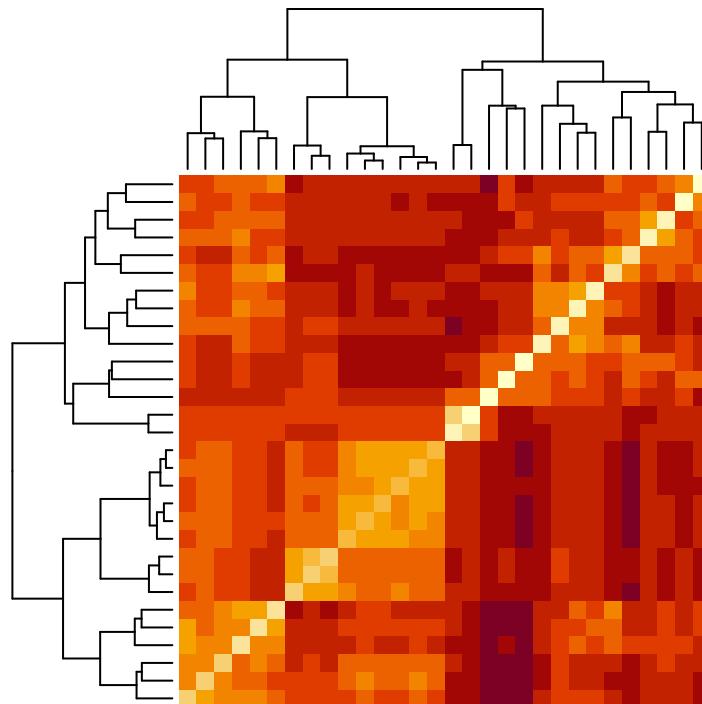
mean(dist_BtoM)

## [1] 7.12
```

Make a heatmap of the relationship between features using the scaled matrix.

```
d_features <- dist(t(x_scaled))

heatmap(as.matrix(d_features), labRow = NA, labCol = NA)
```



Perform hierarchical clustering on the 30 features. Cut the tree into 5 groups.

```
h <- hclust(d_features)

groups <- cutree(h, k = 5)
```

```

split(names(groups), groups)

## $`1`
## [1] "radius_mean"      "perimeter_mean"    "area_mean"
## [4] "concavity_mean"   "concave_pts_mean"  "radius_se"
## [7] "perimeter_se"     "area_se"          "radius_worst"
## [10] "perimeter_worst"  "area_worst"       "concave_pts_worst"
##
## $`2`
## [1] "texture_mean"    "texture_worst"
##
## $`3`
## [1] "smoothness_mean" "compactness_mean" "symmetry_mean"
## [4] "fractal_dim_mean" "smoothness_worst" "compactness_worst"
## [7] "concavity_worst"  "symmetry_worst"  "fractal_dim_worst"
##
## $`4`
## [1] "texture_se"      "smoothness_se"  "symmetry_se"
##
## $`5`
## [1] "compactness_se"  "concavity_se"   "concave_pts_se" "fractal_dim_se"

```

Perform a principal component analysis of the scaled matrix. What proportion of variance is explained by the first principal component?

```
pca <- prcomp(x_scaled)
```

```
summary(pca)
```

```

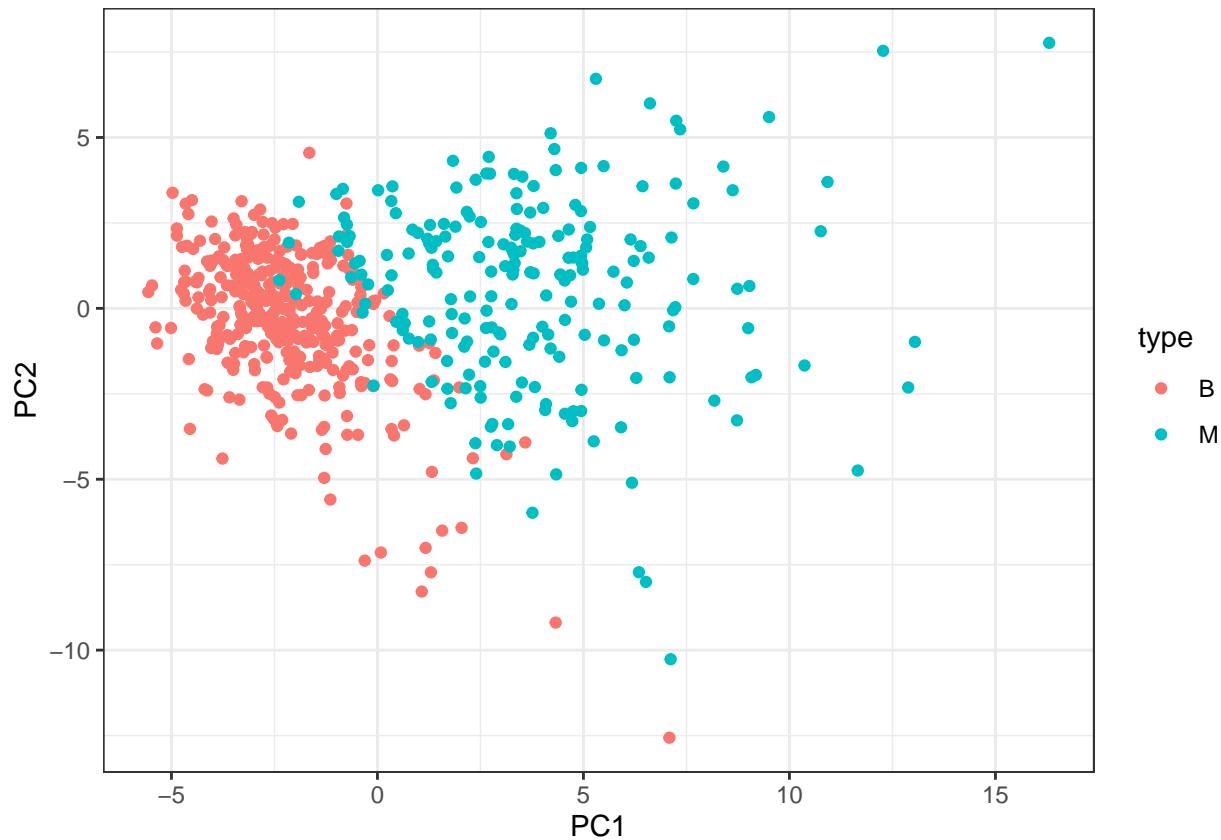
## Importance of components:
##                  PC1    PC2    PC3    PC4    PC5    PC6    PC7    PC8
## Standard deviation 3.644 2.386 1.6787 1.407 1.284 1.0988 0.8217 0.6904
## Proportion of Variance 0.443 0.190 0.0939 0.066 0.055 0.0403 0.0225 0.0159
## Cumulative Proportion 0.443 0.632 0.7264 0.792 0.847 0.8876 0.9101 0.9260
##                  PC9    PC10   PC11   PC12   PC13   PC14   PC15
## Standard deviation 0.6457 0.5922 0.5421 0.51104 0.49128 0.39624 0.30681
## Proportion of Variance 0.0139 0.0117 0.0098 0.00871 0.00805 0.00523 0.00314
## Cumulative Proportion 0.9399 0.9516 0.9614 0.97007 0.97812 0.98335 0.98649
##                  PC16   PC17   PC18   PC19   PC20   PC21   PC22
## Standard deviation 0.28260 0.24372 0.22939 0.22244 0.17652 0.173 0.16565
## Proportion of Variance 0.00266 0.00198 0.00175 0.00165 0.00104 0.001 0.00091
## Cumulative Proportion 0.98915 0.99113 0.99288 0.99453 0.99557 0.997 0.99749
##                  PC23   PC24   PC25   PC26   PC27   PC28   PC29
## Standard deviation 0.15602 0.1344 0.12442 0.09043 0.08307 0.03987 0.02736
## Proportion of Variance 0.00081 0.0006 0.00052 0.00027 0.00023 0.00005 0.00002
## Cumulative Proportion 0.99830 0.9989 0.99942 0.99969 0.99992 0.99997 1.00000
##                  PC30
## Standard deviation 0.0115
## Proportion of Variance 0.0000
## Cumulative Proportion 1.00000

```

Plot the first two principal components with color representing tumor type (benign/malignant).

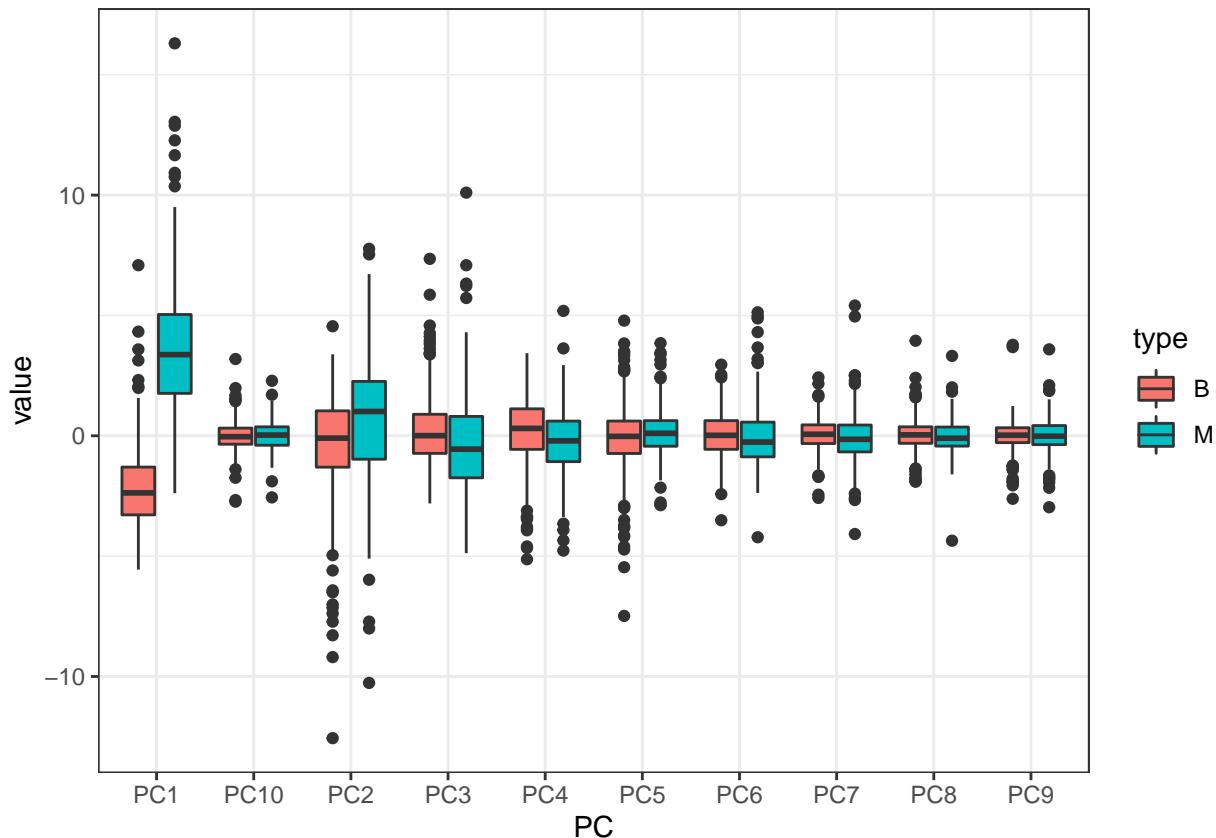
```
data.frame(pca$x[,1:2], type = brca$y) %>%
  ggplot(aes(PC1, PC2, color = type)) +
```

```
geom_point()
```



Make a boxplot of the first 10 PCs grouped by tumor type. Which PCs are significantly different enough by tumor type that there is no overlap in the interquartile ranges (IQRs) for benign and malignant samples?

```
data.frame(type = brca$y, pca$x[,1:10]) %>%
  gather(key = "PC", value = "value", -type) %>%
  ggplot(aes(PC, value, fill = type)) +
  geom_boxplot()
```



Set the seed to 1, then create a data partition splitting `brca$y` and the scaled version of the `brca$x` matrix into a 20 % test set and 80 % train using the following code:

```
set.seed(1, sample.kind = "Rounding")

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used

test_index <- createDataPartition(brca$y, times = 1, p = 0.2, list = FALSE)

test_x <- x_scaled[test_index,]

test_y <- brca$y[test_index]

train_x <- x_scaled[-test_index,]

train_y <- brca$y[-test_index]
```

Check that the training and test sets have similar proportions of benign and malignant tumors. What proportion of the training set is benign?

```
mean(train_y=="B")
```

```
## [1] 0.628
```

What proportion of the test set is benign?

```
mean(test_y == "B")
```

```
## [1] 0.626
```

The predict\_kmeans() function defined here takes two arguments - a matrix of observations x and a k-means object k - and assigns each row of x to a cluster from k.

```
predict_kmeans <- function(x, k) {
  centers <- k$centers
  distances <- sapply(1:nrow(x), function(i){
    apply(centers, 1, function(y) dist(rbind(x[i,], y)))
  })
  max.col(-t(distances))
}
```

Set the seed to 3. Perform k-means clustering on the training set with 2 centers and assign the output to k. Then use the predict\_kmeans() function to make predictions on the test set. What is the overall accuracy?

```
set.seed(3, sample.kind = "Rounding")

## Warning in set.seed(3, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used

k <- kmeans(train_x, centers = 2)

kmeans_preds <- ifelse(predict_kmeans(test_x, k) == 1, "B", "M")

mean(kmeans_preds == test_y)

## [1] 0.922
```

What proportion of benign/malignant tumors are correctly identified?

```
sensitivity(factor(kmeans_preds), test_y, positive = "B")

## [1] 0.986

sensitivity(factor(kmeans_preds), test_y, positive = "M")

## [1] 0.814
```

Fit a logistic regression model on the training set with caret::train() using all predictors. Ignore warnings about the algorithm not converging. Make predictions on the test set. What is the accuracy of the logistic regression model on the test set?

```
train_glm <- train(train_x, train_y,
                     method = "glm")

glm_preds <- predict(train_glm, test_x)

mean(glm_preds == test_y)

## [1] 0.957
```

Train an LDA model and a QDA model on the training set. Make predictions on the test set using each model. What is the accuracy of the LDA model on the test set?

```
train_lda <- train(train_x, train_y,
                     method = "lda")

lda_preds <- predict(train_lda, test_x)

mean(lda_preds == test_y)
```

```
## [1] 0.991
```

**What is the accuracy of the QDA model on the test set?**

```
train_qda <- train(train_x, train_y,
                     method = "qda")

qda_preds <- predict(train_qda, test_x)

mean(qda_preds == test_y)
```

```
## [1] 0.957
```

Set the seed to 5, then fit a loess model on the training set with the caret package. You will need to install the gam package if you have not yet done so. Use the default tuning grid. This may take several minutes; ignore warnings. Generate predictions on the test set. What is the accuracy of the loess model on the test set?

```
library(gam)

train_loess <- train(train_x, train_y,
                      method = "gamLoess")

loess_preds <- predict(train_loess, test_x)

mean(loess_preds == test_y)
```

```
## [1] 0.983
```

Set the seed to 7, then train a k-nearest neighbors model on the training set using the caret package. Try odd values of from 3 to 21. Use the final model to generate predictions on the test set. What is the final value of  $k$  used in the model? What is the accuracy of the kNN model on the test set?

```
set.seed(7,sample.kind = "Rounding")

train_knn <- train(train_x, train_y,
                     method = "knn",
                     tuneGrid = data.frame(k = seq(3, 21, 2)))

knn_preds <- predict(train_knn, test_x)

mean(knn_preds == test_y)
```

```
## [1] 0.957
```

Set the seed to 9, then train a random forest model on the training set using the caret package. Test mtry values of c(3, 5, 7, 9). Use the argument importance = TRUE so that feature importance can be extracted. Generate predictions on the test set. Note: please use c(3, 5, 7, 9) instead of seq(3, 9, 2) in tuneGrid. What value of mtry gives the highest accuracy? What is the accuracy of the random forest model on the test set?

```
set.seed(9,sample.kind = "Rounding")

train_rf <- train(train_x, train_y,
                   method = "rf",
                   tuneGrid = data.frame(mtry = c(3,5,7,9)))
```

```

rf_preds <- predict(train_rf, test_x)

mean(rf_preds == test_y)

## [1] 0.974

train_rf$bestTune

##     mtry
## 1      3

varImp(train_rf)

## rf variable importance
##
## only 20 most important variables shown (out of 30)
##
##                               Overall
## perimeter_worst      100.00
## concave_pts_worst    99.78
## area_worst            90.14
## radius_worst          85.67
## concave_pts_mean     72.50
## area_mean              55.22
## perimeter_mean         54.21
## radius_mean             53.45
## concavity_mean        40.05
## area_se                 39.83
## concavity_worst        35.08
## compactness_worst       24.17
## compactness_mean        15.85
## radius_se                 15.32
## texture_worst            14.27
## perimeter_se              12.54
## texture_mean                10.68
## smoothness_worst           10.23
## symmetry_worst              8.42
## concavity_se                  4.49

```

Create an ensemble using the predictions from the 7 models created in the previous exercises: k-means, logistic regression, LDA, QDA, loess, k-nearest neighbors, and random forest. Use the ensemble to generate a majority prediction of the tumor type (if most models suggest the tumor is malignant, predict malignant). What is the accuracy of the ensemble prediction?

```

ensemble <- cbind(glm = glm_preds == "B",
                    lda = lda_preds == "B",
                    qda = qda_preds == "B",
                    loess = loess_preds == "B",
                    rf = rf_preds == "B",
                    knn = knn_preds == "B",
                    kmeans = kmeans_preds == "B")

ensemble_preds <- ifelse(rowMeans(ensemble) > 0.5, "B", "M")

mean(ensemble_preds == test_y)

```

```
## [1] 0.983
```