

Rafael A. Irizarry

Introducción a la ciencia de datos

Análisis de datos y algoritmos de predicción con R

Contenido

Prefacio	5
Agradecimientos	7
Introducción	9
1 Comenzando con R y RStudio	13
1.1 ¿Por qué R?	13
1.2 La consola R	14
1.3 <i>Scripts</i>	14
1.4 RStudio	15
1.4.1 Paneles	15
1.4.2 Atajos de teclado	17
1.4.3 Cómo ejecutar comandos mientras editan <i>scripts</i>	18
1.4.4 Cómo cambiar las opciones globales	20
1.5 Instalación de paquetes de R	21
I R	23
2 Lo básico de R	25
2.1 Caso de estudio: los asesinatos con armas en EE. UU.	25
2.2 Lo básico	27
2.2.1 Objetos	27
2.2.2 El espacio de trabajo	28
2.2.3 Funciones	28
2.2.4 Otros objetos predefinidos	30
2.2.5 Nombres de variables	31
2.2.6 Cómo guardar su espacio de trabajo	31
2.2.7 <i>Scripts</i> motivantes	31
2.2.8 Cómo comentar su código	32

2.3	Ejercicios	32
2.4	Tipos de datos	33
2.4.1	<i>Data frames</i>	33
2.4.2	Cómo examinar un objeto	34
2.4.3	El operador de acceso: \$	34
2.4.4	Vectores: numéricos, de caracteres y lógicos	35
2.4.5	Factores	36
2.4.6	Listas	37
2.4.7	Matrices	38
2.5	Ejercicios	40
2.6	Vectores	41
2.6.1	Cómo crear vectores	41
2.6.2	Nombres	42
2.6.3	Secuencias	42
2.6.4	Cómo crear un subconjunto	43
2.7	La conversión forzada	44
2.7.1	<i>Not available</i> (NA)	44
2.8	Ejercicios	45
2.9	<i>Sorting</i>	46
2.9.1	<code>sort</code>	46
2.9.2	<code>order</code>	46
2.9.3	<code>max</code> y <code>which.max</code>	47
2.9.4	<code>rank</code>	48
2.9.5	Cuidado con el reciclaje	48
2.10	Ejercicios	48
2.11	Aritmética de vectores	50
2.11.1	<i>Rescaling</i> un vector	50
2.11.2	Dos vectores	50
2.12	Ejercicios	51
2.13	Indexación	51
2.13.1	Cómo crear subconjuntos con lógicos	52
2.13.2	Operadores lógicos	52
2.13.3	<code>which</code>	53
2.13.4	<code>match</code>	53

<i>0.0 Contents</i>	5
2.13.5 <code>%in%</code>	54
2.14 Ejercicios	54
2.15 Gráficos básicos	55
2.15.1 <code>plot</code>	55
2.15.2 <code>hist</code>	55
2.15.3 <code>boxplot</code>	56
2.15.4 <code>image</code>	57
2.16 Ejercicios	57
3 Conceptos básicos de programación	59
3.1 Expresiones condicionales	59
3.2 Cómo definir funciones	61
3.3 <i>Namespaces</i>	62
3.4 Bucles-for	63
3.5 Vectorización y funcionales	64
3.6 Ejercicios	65
4 <i>tidyverse</i>	67
4.1 Datos <code>tidy</code>	67
4.2 Ejercicios	68
4.3 Cómo manipular los <i>data frames</i>	69
4.3.1 Cómo añadir una columna con <code>mutate</code>	69
4.3.2 Cómo crear subconjuntos con <code>filter</code>	70
4.3.3 Cómo seleccionar columnas con <code>select</code>	70
4.4 Ejercicios	71
4.5 El <i>pipe</i> : <code>%>%</code>	72
4.6 Ejercicios	73
4.7 Cómo resumir datos	74
4.7.1 <code>summarize</code>	75
4.7.2 Resúmenes múltiples	76
4.7.3 Cómo agrupar y luego resumir con <code>group_by</code>	77
4.8 <code>pull</code>	78
4.9 Cómo ordenar los <i>data frames</i>	79
4.9.1 Cómo ordenar anidadadamente	79
4.9.2 Los primeros <i>n</i>	80

4.10 Ejercicios	80
4.11 <i>Tibbles</i>	82
4.11.1 Los <i>tibbles</i> se ven mejor	82
4.11.2 Los subconjuntos de <i>tibbles</i> son <i>tibbles</i>	83
4.11.3 Los <i>tibbles</i> pueden tener entradas complejas	83
4.11.4 Los <i>tibbles</i> se pueden agrupar	84
4.11.5 Cómo crear un <i>tibble</i> usando <code>tibble</code> en lugar de <code>data.frame</code>	84
4.12 El operador punto	84
4.13 El paquete purrr	85
4.14 Los condicionales de <i>tidyverse</i>	86
4.14.1 <code>case_when</code>	86
4.14.2 <code>between</code>	87
4.15 Ejercicios	87
5 Importando datos	89
5.1 Las rutas y el directorio de trabajo	90
5.1.1 El sistema de archivos	91
5.1.2 Las rutas relativas y completas	91
5.1.3 El directorio de trabajo	92
5.1.4 Cómo generar los nombres de ruta	92
5.1.5 Cómo copiar los archivos usando rutas	92
5.2 Los paquetes <code>readr</code> y <code>readxl</code>	93
5.2.1 <code>readr</code>	93
5.2.2 <code>readxl</code>	94
5.3 Ejercicios	95
5.4 Cómo descargar archivos	95
5.5 Las funciones de importación de base R	96
5.5.1 <code>scan</code>	96
5.6 Archivos de texto versus archivos binarios	97
5.7 Unicode versus ASCII	97
5.8 Cómo organizar datos con hojas de cálculo	98
5.9 Ejercicios	99
II Visualización de datos	101
6 Introducción a la visualización de datos	103

<i>0.0 Contents</i>	7
7 ggplot2	107
7.1 Los componentes de un gráfico	108
7.2 Objetos <code>ggplot</code>	109
7.3 Geometrías	110
7.4 Mapeos estéticos	111
7.5 Capas	112
7.5.1 Cómo probar varios argumentos	113
7.6 Mapeos estéticos globales versus locales	115
7.7 Escalas	116
7.8 Etiquetas y títulos	116
7.9 Categorías como colores	117
7.10 Anotación, formas y ajustes	119
7.11 Paquetes complementarios	120
7.12 Cómo combinarlo todo	120
7.13 Gráficos rápidos con <code>qplot</code>	121
7.14 Cuadrículas de gráficos	122
7.15 Ejercicios	123
8 Cómo visualizar distribuciones de datos	127
8.1 Tipos de variables	127
8.2 Estudio de caso: describiendo alturas de estudiantes	128
8.3 La función de distribución	128
8.4 Funciones de distribución acumulada	129
8.5 Histogramas	130
8.6 Densidad suave	131
8.6.1 Cómo interpretar el eje-y	135
8.6.2 Densidades permiten estratificación	136
8.7 Ejercicios	137
8.8 La distribución normal	141
8.9 Unidades estándar	143
8.10 Gráficos Q-Q	143
8.11 Percentiles	146
8.12 Diagramas de caja	146
8.13 Estratificación	147
8.14 Estudio de caso: describiendo alturas de estudiantes (continuación)	148

8.15 Ejercicios	149
8.16 Geometrías ggplot2	151
8.16.1 Diagramas de barras	152
8.16.2 Histogramas	153
8.16.3 Gráficos de densidad	154
8.16.4 Diagramas de caja	155
8.16.5 Gráficos Q-Q	155
8.16.6 Imágenes	156
8.16.7 Gráficos rápidos	157
8.17 Ejercicios	159
9 Visualización de datos en la práctica	161
9.1 Estudio de caso: nuevas ideas sobre la pobreza	161
9.1.1 La prueba de Hans Rosling	162
9.2 Diagramas de dispersión	163
9.3 Separar en facetas	165
9.3.1 <code>facet_wrap</code>	166
9.3.2 Escalas fijas para mejores comparaciones	167
9.4 Gráficos de series de tiempo	168
9.4.1 Etiquetas en lugar de leyendas	171
9.5 Transformaciones de datos	172
9.5.1 Transformación logarítmica	173
9.5.2 ¿Qué base?	174
9.5.3 ¿Transformar los valores o la escala?	175
9.6 Cómo visualizar distribuciones multimodales	176
9.7 Cómo comparar múltiples distribuciones con diagramas de caja y gráficos <i>ridge</i>	177
9.7.1 Diagramas de caja	178
9.7.2 Gráficos <i>ridge</i>	179
9.7.3 Ejemplo: distribuciones de ingresos de 1970 versus 2010	181
9.7.4 Cómo obtener acceso a variables calculadas	187
9.7.5 Densidades ponderadas	190
9.8 La falacia ecológica y la importancia de mostrar los datos	190
9.8.1 Transformación logística	191
9.8.2 Mostrar los datos	191

<i>0.0 Contents</i>	9
10 Principios de visualización de datos	193
10.1 Cómo codificar datos utilizando señales visuales	193
10.2 Cuándo incluir 0	196
10.3 No distorsionar cantidades	200
10.4 Ordenar categorías por un valor significativo	201
10.5 Mostrar los datos	203
10.6 Cómo facilitar comparaciones	206
10.6.1 Usen ejes comunes	206
10.6.2 Alineen gráficos verticalmente para ver cambios horizontales y horizontalmente para ver cambios verticales	207
10.6.3 Consideren transformaciones	208
10.6.4 Señales visuales comparadas deben estar adyacentes	210
10.6.5 Usen color	211
10.7 Consideren los daltónicos	211
10.8 Gráficos para dos variables	212
10.8.1 <i>Slope charts</i>	212
10.8.2 Gráfico Bland-Altman	214
10.9 Cómo codificar una tercera variable	214
10.10 Eviten los gráficos pseudo-tridimensionales	216
10.11 Eviten demasiados dígitos significativos	218
10.12 Consideren a su audiencia	219
10.13 Ejercicios	219
10.14 Estudio de caso: las vacunas y las enfermedades infecciosas	224
10.15 Ejercicios	227
11 Resúmenes robustos	229
11.1 Valores atípicos	229
11.2 Mediana	230
11.3 El rango intercuartil (IQR)	230
11.4 La definición de Tukey de un valor atípico	231
11.5 Desviación absoluta mediana	232
11.6 Ejercicios	232
11.7 Estudio de caso: alturas autoreportadas de estudiantes	233
III Estadísticas con R	237

12 Introducción a las estadísticas con R	239
13 Probabilidad	241
13.1 Probabilidad discreta	241
13.1.1 Frecuencia relativa	241
13.1.2 Notación	242
13.1.3 Distribuciones de probabilidad	242
13.2 Simulaciones Monte Carlo para datos categóricos	243
13.2.1 Fijar la semilla aleatoria	244
13.2.2 Con y sin reemplazo	244
13.3 Independencia	245
13.4 Probabilidades condicionales	246
13.5 Reglas de la adición y de la multiplicación	246
13.5.1 Regla de la multiplicación	246
13.5.2 Regla de la multiplicación bajo independencia	247
13.5.3 Regla de la adición	247
13.6 Combinaciones y permutaciones	248
13.6.1 Ejemplo Monte Carlo	251
13.7 Ejemplos	252
13.7.1 Problema Monty Hall	252
13.7.2 Problema de cumpleaños	253
13.8 Infinito en la práctica	256
13.9 Ejercicios	257
13.10 Probabilidad continua	259
13.11 Distribuciones teóricas continuas	260
13.11.1 Distribuciones teóricas como aproximaciones	260
13.11.2 La densidad de probabilidad	262
13.12 Simulaciones Monte Carlo para variables continuas	263
13.13 Distribuciones continuas	265
13.14 Ejercicios	265
14 Variables aleatorias	267
14.1 Variables aleatorias	267
14.2 Modelos de muestreo	268
14.3 La distribución de probabilidad de una variable aleatoria	269

<i>0.0 Contents</i>	11
14.4 Distribuciones versus distribuciones de probabilidad	271
14.5 Notación para variables aleatorias	272
14.6 El valor esperado y el error estándar	272
14.6.1 Población SD versus la muestra SD	274
14.7 Teorema del límite central	275
14.7.1 ¿Cuán grande es grande en el teorema del límite central?	276
14.8 Propiedades estadísticas de promedios	277
14.9 Ley de los grandes números	278
14.9.1 Malinterpretando la ley de promedios	278
14.10 Ejercicios	279
14.11 Estudio de caso: <i>The Big Short</i>	280
14.11.1 Tasas de interés explicadas con modelo de oportunidad	280
14.11.2 <i>The Big Short</i>	283
14.12 Ejercicios	286
15 Inferencia estadística	289
15.1 Encuestas	289
15.1.1 El modelo de muestreo para encuestas	290
15.2 Poblaciones, muestras, parámetros y estimadores	292
15.2.1 El promedio de la muestra	292
15.2.2 Parámetros	293
15.2.3 Encuesta versus pronóstico	294
15.2.4 Propiedades de nuestro estimador: valor esperado y error estándar .	294
15.3 Ejercicios	295
15.4 Teorema del límite central en la práctica	296
15.4.1 Una simulación Monte Carlo	298
15.4.2 La diferencia	299
15.4.3 Sesgo: ¿por qué no realizar una encuesta bien grande?	300
15.5 Ejercicios	301
15.6 Intervalos de confianza	302
15.6.1 Una simulación Monte Carlo	305
15.6.2 El idioma correcto	306
15.7 Ejercicios	306
15.8 Poder	308
15.9 valores-p	308

15.10 Pruebas de asociación	309
15.10.1 Lady Tasting Tea	310
15.10.2 Tablas 2x2	311
15.10.3 Prueba de chi-cuadrado	311
15.10.4 Riesgo relativo	313
15.10.5 Intervalos de confianza para el riesgo relativo	314
15.10.6 Corrección de recuento pequeño	315
15.10.7 Muestras grandes, valores-p pequeños	315
15.11 Ejercicios	315
16 Modelos estadísticos	317
16.1 Agregadores de encuestas	318
16.1.1 Datos de encuesta	321
16.1.2 Sesgo de los encuestadores	322
16.2 Modelos basados en datos	323
16.3 Ejercicios	325
16.4 Estadísticas bayesianas	328
16.4.1 Teorema de Bayes	329
16.5 Simulación del teorema de Bayes	330
16.5.1 Bayes en la práctica	330
16.6 Modelos jerárquicos	332
16.7 Ejercicios	334
16.8 Estudio de caso: pronóstico de elecciones	336
16.8.1 Enfoque bayesiano	336
16.8.2 El sesgo general	337
16.8.3 Representaciones matemáticas de modelos	338
16.8.4 Prediciendo el colegio electoral	341
16.8.5 Pronósticos	345
16.9 Ejercicios	348
16.10 La distribución t	349
17 Regresión	353
17.1 Estudio de caso: ¿la altura es hereditaria?	353
17.2 El coeficiente de correlación	355
17.2.1 La correlación de muestra es una variable aleatoria	356

0.0 Contents	13
17.2.2 La correlación no siempre es un resumen útil	358
17.3 Valor esperado condicional	358
17.4 La línea de regresión	361
17.4.1 La regresión mejora la precisión	362
17.4.2 Distribución normal de dos variables (avanzada)	363
17.4.3 Varianza explicada	365
17.4.4 Advertencia: hay dos líneas de regresión	365
17.5 Ejercicios	366
18 Modelos lineales	367
18.1 Estudio de caso: <i>Moneyball</i>	367
18.1.1 Sabermetrics	368
18.1.2 Conceptos básicos de béisbol	369
18.1.3 No hay premios para BB	370
18.1.4 ¿Base por bolas o bases robadas?	371
18.1.5 Regresión aplicada a las estadísticas de béisbol	374
18.2 Confusión	377
18.2.1 Cómo entender la confusión a través de la estratificación	378
18.2.2 Regresión lineal múltiple	381
18.3 Estimaciones de mínimos cuadrados	381
18.3.1 Interpretando modelos lineales	382
18.3.2 Estimadores de mínimos cuadrados (LSE)	382
18.3.3 La función <code>lm</code>	384
18.3.4 El LSE consiste de variables aleatorias	385
18.3.5 Valores pronosticados son variables aleatorias	386
18.4 Ejercicios	387
18.5 Regresión lineal en el tidyverse	388
18.5.1 El paquete broom	390
18.6 Ejercicios	392
18.7 Estudio de caso: <i>Moneyball</i> (continuación)	393
18.7.1 Añadiendo información sobre salario y posición	396
18.7.2 Escoger nueve jugadores	398
18.8 La falacia de la regresión	399
18.9 Modelos de error de medición	401
18.10 Ejercicios	403

19 La correlación no implica causalidad	405
19.1 Correlación espuria	405
19.2 Valores atípicos	408
19.3 Inversión de causa y efecto	410
19.4 Factores de confusión	411
19.4.1 Ejemplo: admisiones a la Universidad de California, Berkeley	411
19.4.2 Confusión explicada gráficamente	413
19.4.3 Calcular promedio luego de estratificar	413
19.5 La paradoja de Simpson	414
19.6 Ejercicios	415
IV Wrangling de datos	417
20 Introducción al wrangling de datos	419
21 Cómo cambiar el formato de datos	421
21.1 pivot_longer	421
21.2 pivot_wider	423
21.3 separate	423
21.4 unite	426
21.5 Ejercicios	427
22 Unir tablas	429
22.1 Funciones para unir	430
22.1.1 Left join	431
22.1.2 Right join	432
22.1.3 Inner join	432
22.1.4 Full join	432
22.1.5 Semi join	433
22.1.6 Anti join	433
22.2 Binding	434
22.2.1 Pegando columnas	434
22.2.2 Pegando filas	434
22.3 Operadores de sets	435
22.3.1 Intersecar	435
22.3.2 Unión	436

<i>0.0 Contents</i>	15
22.3.3 <code>setdiff</code>	436
22.3.4 <code>setequal</code>	436
22.4 Ejercicios	437
23 Extracción de la web	439
23.1 HTML	440
23.2 El paquete <code>rvest</code>	441
23.3 Selectores CSS	443
23.4 JSON	444
23.5 Ejercicios	445
24 Procesamiento de cadenas	447
24.1 El paquete <code>stringr</code>	447
24.2 Estudio de caso 1: datos de asesinatos en EE. UU.	449
24.3 Estudio de caso 2: alturas autoreportadas	451
24.4 Cómo <i>escapar</i> al definir cadenas	453
24.5 Expresiones regulares	455
24.5.1 Las cadenas son expresiones regulares	456
24.5.2 Caracteres especiales	456
24.5.3 Clases de caracteres	457
24.5.4 Anclas	458
24.5.5 Cuantificadores	459
24.5.6 Espacio en blanco <code>\s</code>	460
24.5.7 Cuantificadores: <code>*</code> , <code>?</code> , <code>+</code>	460
24.5.8 Todo excepto	461
24.5.9 Grupos	462
24.6 Buscar y reemplazar con expresiones regulares	463
24.6.1 Buscar y reemplazar usando grupos	464
24.7 Probar y mejorar	465
24.8 Podar	468
24.9 Cómo cambiar de mayúsculas o minúsculas	469
24.10 Estudio de caso 2: alturas autoreportadas (continuación)	469
24.10.1 La función <code>extract</code>	470
24.10.2 Juntando todas las piezas	471
24.11 División de cadenas	472

24.12 Estudio de caso 3: extracción de tablas de un PDF	475
24.13 Recodificación	478
24.14 Ejercicios	480
25 Cómo leer y procesar fechas y horas	483
25.1 Datos tipo fecha	483
25.2 El paquete lubridate	484
25.3 Ejercicios	487
26 Minería de textos	491
26.1 Estudio de caso: tuits de Trump	491
26.2 Texto como datos	494
26.3 Análisis de sentimiento	499
26.4 Ejercicios	503
V Machine Learning	505
27 Introducción a <i>machine learning</i>	507
27.1 Notación	507
27.2 Un ejemplo	508
27.3 Ejercicios	510
27.4 Métricas de evaluación	511
27.4.1 Sets de entrenamiento y de evaluación	512
27.4.2 Exactitud general	512
27.4.3 Matriz de confusión	515
27.4.4 Sensibilidad y especificidad	516
27.4.5 Exactitud equilibrada y medida F_1	517
27.4.6 La prevalencia importa en la práctica	519
27.4.7 Curvas ROC y precision-recall	520
27.4.8 Función de pérdida	522
27.5 Ejercicios	523
27.6 Probabilidades y expectativas condicionales	524
27.6.1 Probabilidades condicionales	524
27.6.2 Expectativas condicionales	525
27.6.3 La expectativa condicional minimiza la función de pérdida cuadrática	526
27.7 Ejercicios	526
27.8 Estudio de caso: ¿es un 2 o un 7?	527

<i>0.0 Contents</i>	17
28 Suavización	531
28.1 Suavización de compartimientos	533
28.2 Kernels	535
28.3 Regresión ponderada local (loess)	537
28.3.1 Ajustando con paráolas	540
28.3.2 Cuidado con los parámetros de suavización predeterminados	541
28.4 Conectando la suavización al <i>machine learning</i>	542
28.5 Ejercicios	543
29 Validación cruzada	545
29.1 Motivación con k vecinos más cercanos	545
29.1.1 Sobreentrenamiento	547
29.1.2 Sobre-suavización	549
29.1.3 Escogiendo la <i>k</i> en kNN	549
29.2 Descripción matemática de validación cruzada	551
29.3 Validación cruzada K-fold	552
29.4 Ejercicios	556
29.5 Bootstrap	557
29.6 Ejercicios	559
30 El paquete caret	561
30.1 La función <code>train</code> de caret	561
30.2 Validación cruzada	562
30.3 Ejemplo: ajuste con loess	565
31 Ejemplos de algoritmos	569
31.1 Regresión lineal	569
31.1.1 La función <code>predict</code>	570
31.2 Ejercicios	571
31.3 Regresión logística	573
31.3.1 Modelos lineales generalizados	575
31.3.2 Regresión logística con más de un predictor	578
31.4 Ejercicios	579
31.5 k vecinos más cercanos (kNN)	580
31.6 Ejercicios	581
31.7 Modelos generativos	581

31.7.1	Naive Bayes	582
31.7.2	Controlando la prevalencia	583
31.7.3	Análisis discriminante cuadrático	585
31.7.4	Análisis discriminante lineal	588
31.7.5	Conexión a distancia	590
31.8	Estudio de caso: más de tres clases	590
31.9	Ejercicios	594
31.10	Árboles de clasificación y regresión (CART)	595
31.10.1	La maldición de la dimensionalidad	595
31.10.2	Motivación CART	597
31.10.3	Árboles de regresión	599
31.10.4	Árboles de clasificación (decisión)	605
31.11	Bosques aleatorios	607
31.12	Ejercicios	612
32	Machine learning en la práctica	615
32.1	Preprocesamiento	616
32.2	k-vecino más cercano y bosque aleatorio	617
32.3	Importancia variable	619
32.4	Evaluaciones visuales	620
32.5	Conjuntos	621
32.6	Ejercicios	622
33	Sets grandes de datos	623
33.1	Álgebra matricial	623
33.1.1	Notación	624
33.1.2	Convertir un vector en una matriz	626
33.1.3	Resúmenes de filas y columnas	627
33.1.4	<code>apply</code>	628
33.1.5	Filtrar columnas basado en resúmenes	629
33.1.6	Indexación con matrices	631
33.1.7	Binarizar los datos	632
33.1.8	Vectorización para matrices	632
33.1.9	Operaciones de álgebra matricial	633
33.2	Ejercicios	633

0.0	Contents	19
33.3	Distancia	634
33.3.1	Distancia euclíadiana	634
33.3.2	Distancia en dimensiones superiores	635
33.3.3	Ejemplo de distancia euclíadiana	635
33.3.4	Espacio predictor	637
33.3.5	Distancia entre predictores	638
33.4	Ejercicios	638
33.5	Reducción de dimensiones	639
33.5.1	Preservando la distancia	639
33.5.2	Transformaciones lineales (avanzado)	643
33.5.3	Transformaciones ortogonales (avanzado)	643
33.5.4	Análisis de componentes principales	645
33.5.5	Ejemplo de lirios	647
33.5.6	Ejemplo de MNIST	650
33.6	Ejercicios	653
33.7	Sistemas de recomendación	654
33.7.1	Datos de MovieLens	654
33.7.2	Sistemas de recomendación como un desafío de <i>machine learning</i>	656
33.7.3	Función de pérdida	656
33.7.4	Un primer modelo	657
33.7.5	Modelando los efectos de películas	658
33.7.6	Efectos de usuario	659
33.8	Ejercicios	661
33.9	Regularización	662
33.9.1	Motivación	662
33.9.2	Mínimos cuadrados penalizados	664
33.9.3	Cómo elegir los términos de penalización	666
33.10	Ejercicios	669
33.11	Factorización de matrices	670
33.11.1	Análisis de factores	673
33.11.2	Conexión a SVD y PCA	675
33.12	Ejercicios	678

34 Agrupación	685
34.1 Agrupación jerárquica	686
34.2 k-means	688
34.3 Mapas de calor	688
34.4 Filtrando atributos	689
34.5 Ejercicios	690
VI Herramientas de productividad	691
35 Introducción a las herramientas de productividad	693
36 Instalación de R y RStudio	695
36.1 Instalando R	695
36.2 Instalación de RStudio	702
37 Accediendo al terminal e instalando Git	709
37.1 Accediendo al terminal en una Mac	709
37.2 Instalando Git en la Mac	710
37.3 Instalación de Git y Git Bash en Windows	713
37.4 Accediendo el terminal en Windows	717
38 Organizando con Unix	721
38.1 Convención de nomenclatura	721
38.2 El terminal	722
38.3 El sistema de archivos	722
38.3.1 Directorios y subdirectorios	723
38.3.2 El directorio <i>home</i>	723
38.3.3 Directorio de trabajo	724
38.3.4 Rutas	725
38.4 Comandos de Unix	726
38.4.1 ls : Listado de contenido del directorio	726
38.4.2 mkdir y rmdir : crear y eliminar un directorio	726
38.4.3 cd : navegando por el sistema de archivos cambiando directorios	727
38.5 Algunos ejemplos	729
38.6 Más comandos de Unix	730
38.6.1 mv : mover archivos	730
38.6.2 cp : copiando archivos	731

<i>0.0 Contents</i>	21
38.6.3 <code>rm</code> : eliminar archivos	732
38.6.4 <code>less</code> : mirando un archivo	732
38.7 Preparación para un proyecto de ciencia de datos	732
38.8 Unix avanzado	733
38.8.1 Argumentos	733
38.8.2 Obtener ayuda	735
38.8.3 El <i>pipe</i>	735
38.8.4 Comodines	735
38.8.5 Variables de entorno	736
38.8.6 <i>Shells</i>	737
38.8.7 Ejecutables	737
38.8.8 Permisos y tipos de archivo	738
38.8.9 Comandos que deben aprender	738
38.8.10 Manipulación de archivos en R	739
39 Git y GitHub	741
39.1 ¿Por qué usar Git y GitHub?	741
39.2 Cuentas GitHub	742
39.3 Repositorios de GitHub	744
39.4 Descripción general de Git	745
39.4.1 Clonar	746
39.5 Inicializando un directorio Git	750
39.6 Usando Git y GitHub en RStudio	752
40 Proyectos reproducibles con RStudio y R Markdown	757
40.1 Proyectos de RStudio	757
40.2 R Markdown	760
40.2.1 El encabezado	762
40.2.2 Fragmentos de código R	762
40.2.3 Opciones globales	763
40.2.4 <code>knitr</code>	763
40.2.5 Más sobre R Markdown	764
40.3 Organizando un proyecto de ciencia de datos	764
40.3.1 Crear directorios en Unix	765
40.3.2 Crear un proyecto en RStudio	765

40.3.3 Editar algunos scripts de R	766
40.3.4 Crear más directorios usando Unix	767
40.3.5 Agregar un archivo README	768
40.3.6 Inicializando un directorio Git	768

Prefacio

Este libro comenzó como las notas utilizadas para enseñar las clases de HarvardX Data Science Series¹.

El código RMarkdown que se usó para generar el libro está disponible en [GitHub²](#). El tema gráfico utilizado para los gráficos a lo largo del libro se pueden recrear utilizando la función `ds_theme_set()` del paquete **dslabs**.

Un PDF de la versión en inglés de este libro está disponible en [Leanpub³](#).

Una copia impresa de la versión en inglés de este libro está disponible en [CRC Press⁴](#).

Este trabajo se publica bajo la licencia Creative Commons Attribution-NonCommercial-ShareAlike 4.0 Internacional [CC BY-NC-SA 4.0](#).

Hacemos anuncios relacionados al libro en Twitter. Para la información más reciente, siga [@rafalab](#).

¹<https://www.edx.org/professional-certificate/harvardx-data-science>

²<https://github.com/rafalab/dsbook>

³<https://leanpub.com/datasciencobook>

⁴<https://www.crcpress.com/Introduction-to-Data-Science-Data-Analysis-and-Prediction-Algorithms-with/Irizarry/p/book/9780367357986>

Agradecimientos

Este libro está dedicado a todas las personas involucradas en la construcción y el mantenimiento de R y los paquetes R que utilizamos en el texto. Un agradecimiento especial a los desarrolladores y los mantenedores de base R, el *tidyverse* y el paquete **caret**.

Un agradecimiento especial a mi *tidyverse* gurú David Robinson y a Amy Gill por docenas de comentarios, cambios y sugerencias. Además, muchas gracias a Stephanie Hicks, que dos veces sirvió como co-instructora en mis clases de ciencias de datos, y a Yihui Xie, que pacientemente toleró mis múltiples preguntas sobre bookdown. Gracias también a Karl Broman, de quien tomé prestadas ideas para las secciones sobre la visualización de datos y las herramientas de productividad, y a Hector Corrada-Bravo, por sus consejos sobre cómo mejor enseñar *machine learning*. Gracias a Peter Aldhous, de quien tomé prestadas ideas para la sección sobre los principios de la visualización de datos y a Jenny Bryan por escribir *Happy Git* y *GitHub for the useR*, que influyeron en nuestros capítulos de Git. Gracias a Alyssa Frazee por ayudar a crear el problema de tarea que se convirtió en el capítulo sobre los sistemas de recomendación y a Amanda Cox por proporcionar los datos de los exámenes de los Regentes de Nueva York. Además, muchas gracias a Jeff Leek, Roger Peng y Brian Caffo, cuya clase inspiró la forma en que se divide este libro y a Garrett Grolemund y Hadley Wickham por abrir el código para su libro R for Data Science. Finalmente, gracias a Alex Nones por corregir el manuscrito durante sus diversas etapas.

Este libro fue concebido durante la enseñanza de varios cursos de estadística aplicada, comenzando hace más de quince años. Los profesores asistentes que trabajaron conmigo a lo largo de los años hicieron importantes contribuciones indirectas a este libro. La última versión de este curso es una serie de HarvardX coordinada por Heather Sternshein y Zofia Gajdos. Les agradecemos sus contribuciones. También estamos agradecidos a todos los estudiantes cuyas preguntas y comentarios nos ayudaron a mejorar el libro. Los cursos fueron parcialmente financiados por el subsidio del NIH R25GM114818. Agradecemos los Institutos Nacionales de Salud por su apoyo.

Un agradecimiento especial a todos aquellos que editaron el libro a través de *pull requests* de GitHub o hicieron sugerencias creando un *issue* o enviando un correo electrónico: **nickyfoto** (Huang Qiang) **desautm** (Marc-André Désautels), **michaschwab** (Michail Schwab) **alvarolarreategui** (Alvaro Larreategui), **jakevc** (Jake VanCappelen), **omerta** (Guillermo Lengemann), **espinielli** (Enrico Spinielli), **asimumba** (Aaron Simumba) **braunschweig** (Maledwar), **gwierzchowski** (Grzegorz Wierzchowski), **technocrat** (Richard Careaga) **atzakas**, **defeit** (David Emerson Feit), **shiraamitchell** (Shira Mitchell) **Nathalie-S**, **andreashandel** (Andreas Handel) **berkowitz** (Elias Berkowitz) **Dean-Webb** (Dean Webber), **mohayusuf**, **jimrothstein**, **mPloenzke** (Matthew Ploenzke), **NicholasDowand** (Nicholas Dow) **kant** (Darió Hereñú), **debbieyuster** (Debbie Yuster), **tuanchauict** (Tuan Chau), **phzeller**, David D. Kane, El Mustapha El Abbassi y Vadim Zipunnikov.

La traducción del libro al español estuvo a cargo de Alex Nones. Agradecemos a todos los que contribuyeron a esta traducción. Ilia Ushkin y Dustin Tingley generaron un primer

borrador usando un programa de traducción automática. A través de Twitter @R4DS_es y @lacion (Laura Ación) proveyeron importante información sobre recursos existentes. Varios otros contribuyeron a través de Twitter, GitHub, o email: @hortizzuazaga (Humberto Ortiz), @ribnikov (Jose Matestadístico), @jarangoo (Julián A.), @DiegoV_O_ (Diego), @Bruno-ContrerasM (BContreras Moreira), @a2kimura (Alejandro Kimura), @Emilio_NTN (Emilio García Morán), @beto_bfr (betofogo), @jdieramon (Jose V. Die), @yabellini (Yanina Bellini Saibene), @symusicgroup (Ismael Rudas), @cristinaz (Cristina Zenteno), @controlnegativo (Cristina de Dios), @d_olivaw (Elio Campitelli), @aguerri_jc (Jesús C. Aguerri), @pincheipie (Francisco, en casa) @compBiology (Pedro Madrigal), @RLadiesCuerna (RLadies Cuernavaca), @thecarpentries, @midnucas, eead-csic-compbio (CSIC & Fundación ARAID), pablormier (Pablo R. Mier), josschavezf (Joselyn Chavez), jmcastagnetto (Jesus M. Castagnetto), ismaelrudas, AnaBVA (Ana B. Villaseñor Altamirano), @pabloguti3rr3z (Pablo Gutiérrez), Héctor Corrada-Bravo, Rafael A. Arce Nazario, Luis R. Pericchi Guerra, María E. Perez Hernández, Juan Carlos Perdomo, Anamari Irizarry y Amed Irizarry.

Introducción

La demanda de profesionales cualificados en ciencias de datos en la industria, la academia y el gobierno está creciendo rápidamente. Este libro presenta conceptos y destrezas que pueden ayudarles a enfrentar los desafíos del análisis de datos en situaciones reales. El texto abarca los conceptos de probabilidad, inferencia estadística, regresión lineal y *machine learning*. También les ayudará a desarrollar destrezas como la programación en R, el *wrangling* de datos, **dplyr**, la visualización de datos con **ggplot2**, la creación de algoritmos con **caret**, la organización de archivos con UNIX/Linux *shell*, el control de versiones con Git y GitHub y la preparación de documentos reproducibles con **knitr** y R markdown. El libro se divide en seis partes: **R**, **Visualización de datos**, **Wrangling de datos**, **Estadísticas con R**, **Machine Learning** y **Herramientas de productividad**. Cada parte tiene varios capítulos que se deben presentar como una sola clase e incluye docenas de ejercicios distribuidos a través de los capítulos.

Los casos de estudio

A lo largo del libro, utilizamos casos de estudio motivantes. En cada caso de estudio, intentamos imitar de manera realista la experiencia de los científicos de datos. Para cada uno de los conceptos que discutimos, comenzamos haciendo preguntas específicas a las que entonces respondemos mediante un análisis de datos. Aprendemos los conceptos como un medio para responder a las preguntas. Ejemplos de los casos de estudio que incluimos en este libro son:

Caso de estudio	Concepto
Tasas de asesinatos en Estados Unidos por estado	Conceptos básicos de R
Alturas de estudiantes	Resúmenes estadísticos
Tendencias en la salud y la economía mundial	Visualización de datos
El impacto de las vacunas en las tasas de enfermedades infecciosas	Visualización de datos
La crisis financiera de 2007-2008	Probabilidad
Previsión de elecciones	Inferencia estadística
Alturas autoreportadas de estudiantes	<i>Wrangling</i> de datos
<i>Moneyball</i> : Construyendo un equipo de béisbol	Regresión lineal
MNIST: Procesamiento de imagen de dígitos escritos a mano	<i>Machine Learning</i>

Caso de estudio	Concepto
Sistemas de recomendación de películas	<i>Machine Learning</i>

¿Quién encontrará útil este libro?

El propósito de este libro es servir como un texto para un primer curso de ciencia de datos. No es necesario tener conocimientos previos de R, aunque algo de experiencia en la programación puede ser útil. Los conceptos estadísticos utilizados para responder a las preguntas de los casos de estudio se presentan solo brevemente y, por lo tanto, recomendamos un libro de texto de probabilidad y estadística para los que quieran entender a fondo estos conceptos. Al leer y comprender todos los capítulos y completar todos los ejercicios, los estudiantes estarán bien posicionados para realizar tareas básicas de análisis de datos y aprender los conceptos y las destrezas más avanzadas que son necesarios para convertirse en expertos.

¿Qué cubre este libro?

Comenzamos repasando los **conceptos básicos de R** y el **tidyverse**. Aprenderán R a lo largo del libro, pero en la primera parte nos dedicamos a revisar los componentes básicos necesarios para seguir aprendiendo.

La creciente disponibilidad de sets de datos informativos y de herramientas de software ha conducido a que más y más campos dependan de la **visualización de datos**. En la segunda parte, demostramos cómo usar **ggplot2** para generar gráficos y describir principios importantes de la visualización de datos.

En la tercera parte, demostramos la importancia de las estadísticas en el análisis de datos respondiendo a preguntas de estudios de caso usando la **probabilidad**, la **inferencia** y la **regresión** con R.

La cuarta parte utiliza varios ejemplos para familiarizar a los lectores con el **wrangling de datos**. Entre las destrezas específicas que estudiamos están la extracción de la web (*web scraping* en inglés), el uso de expresiones regulares y la unión y el cambio de formato de tablas de datos. Hacemos esto usando las herramientas de **tidyverse**.

En la quinta parte presentamos varios desafíos que nos llevan a introducir **machine learning**. Aprendemos a usar el paquete **caret** para construir algoritmos de predicción que incluyen k vecinos más cercanos y bosques aleatorios.

En la parte final, ofrecemos una breve introducción a las **herramientas de productividad** que usamos diariamente en los proyectos de ciencia de datos. Estas son RStudio, UNIX/Linux shell, Git y GitHub, y **knitr** y R Markdown.

¿Qué no cubre este libro?

Este libro se enfoca en los aspectos del análisis de datos de la ciencia de datos. Por consiguiente, no discutimos aspectos relacionados con el manejo de datos (*data management* en inglés) o la ingeniería. Aunque la programación en R es una parte esencial del libro, no enseñamos temas informáticos más avanzados como las estructuras de datos, la optimización y la teoría de algoritmos. Del mismo modo, no discutimos temas como los servicios web, los gráficos interactivos, la computación paralela y el procesamiento de flujos de datos (*data streaming processing* en inglés). Los conceptos estadísticos se presentan principalmente como herramientas para resolver problemas y no se incluyen descripciones teóricas detalladas en este libro.

1

Comenzando con R y RStudio

1.1 ¿Por qué R?

R no es un lenguaje de programación como C o Java. No fue creado por ingenieros de software para el desarrollo de software, sino por estadísticos como un ambiente interactivo para el análisis de datos. Pueden leer la historia completa en el artículo *A Brief History of S*¹. La interactividad es una característica indispensable en la ciencia de datos porque, como pronto aprenderán, la capacidad de explorar rápidamente los datos es necesario para el éxito en este campo. Sin embargo, igual que en otros lenguajes de programación, en R pueden guardar su trabajo como una secuencia de comandos, conocida como un *script*, que se pueden ejecutar fácilmente en cualquier momento. Estos *scripts* sirven como un registro del análisis que realizaron, una característica clave que facilita el trabajo reproducible. Los programadores expertos no deben esperar que R siga las convenciones a que están acostumbrados, ya que se sentirán decepcionados. Si son pacientes, apreciarán la gran ventaja de R cuando se trata del análisis de datos y, específicamente, de la visualización de datos.

Otras características atractivas de R son:

1. R es gratuito y de código abierto².
2. Se ejecuta en todas las plataformas principales: Windows, Mac Os, UNIX/Linux.
3. Los *scripts* y los objetos de datos se pueden compartir sin problemas entre plataformas.
4. Existe una comunidad grande, creciente y activa de usuarios de R y, como resultado, hay numerosos recursos para aprender y hacer preguntas^{3 4}.
5. Es fácil para otras personas contribuir complementos (*add-ons* en inglés) que les permiten a los desarrolladores compartir implementaciones de software de nuevas metodologías de ciencia de datos. Esto les da a los usuarios de R acceso temprano a los métodos y herramientas más recientes que se desarrollan para una amplia variedad de disciplinas, incluyendo la ecología, la biología molecular, las ciencias sociales y la geografía, entre otros campos.

¹<https://pdfs.semanticscholar.org/9b48/46f192aa37ca122cfabb1ed1b59866d8bfda.pdf>

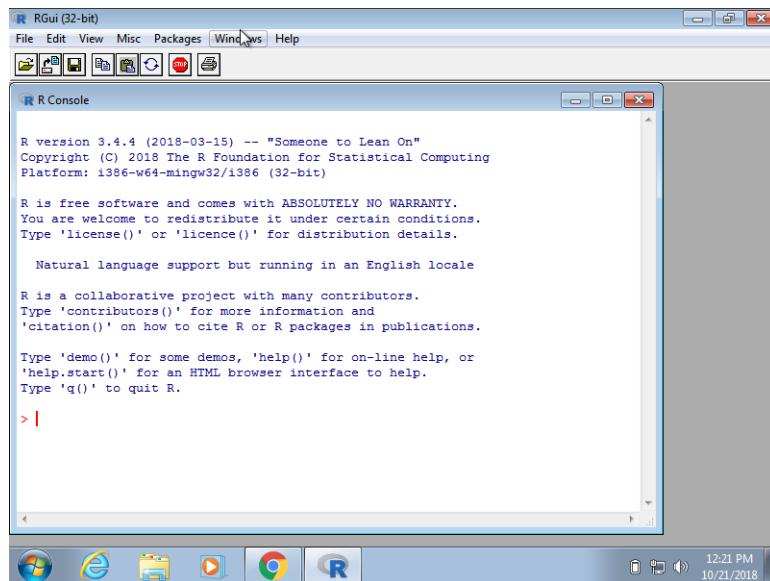
²<https://opensource.org/history>

³<https://stats.stackexchange.com/questions/138/free-resources-for-learning-r>

⁴<https://www.r-project.org/help.html>

1.2 La consola R

El análisis de datos interactivo generalmente ocurre en la consola R que ejecuta comandos a medida que los escriban. Hay varias formas de obtener acceso a una consola R. Una es simplemente iniciando R en su computadora. La consola se ve así:



Como ejemplo rápido, intenten usar la consola para calcular una propina de 15% en una comida que cuesta \$19.71:

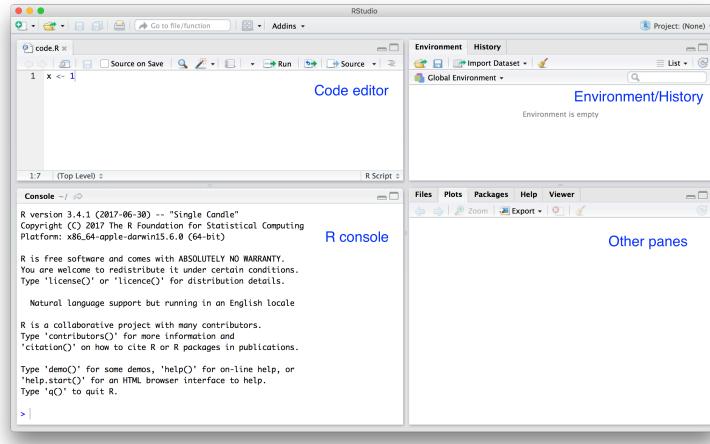
```
0.15 * 19.71
#> [1] 2.96
```

Ojo: En este libro, los cuadros grises se utilizan para mostrar el código R escrito en la consola R. El símbolo `#>` se usa para denotar el *output* de la consola R.

1.3 Scripts

Una de las grandes ventajas de R sobre el software de análisis de apuntar y hacer clic es que pueden guardar su trabajo como *scripts*, que entonces pueden editar y guardar con un editor de texto. El material de este libro se desarrolló utilizando el *Integrated Development Environment* (IDE) de RStudio⁵. RStudio incluye un editor con muchas características específicas de R, una consola para ejecutar su código y otros paneles útiles, incluso uno para mostrar figuras.

⁵<https://www.rstudio.com/>



La mayoría de consolas de R disponibles en la web también incluyen un panel para editar *scripts*, pero no todas les permiten guardar los *scripts* para su uso posterior.

Todos los *scripts* de R utilizados para generar este libro se pueden encontrar en GitHub⁶.

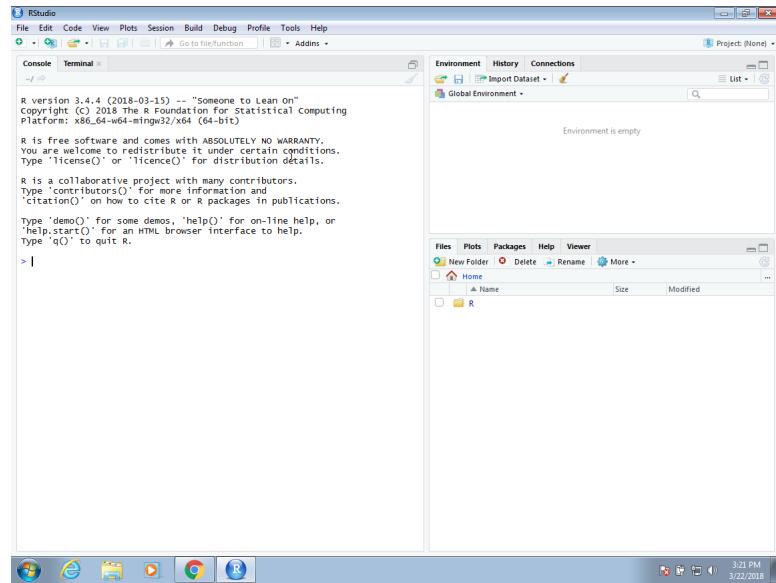
1.4 RStudio

RStudio será nuestra plataforma de lanzamiento para los proyectos de ciencia de datos. No sólo nos provee un editor para crear y editar nuestros *scripts*, sino que también ofrece muchas otras herramientas útiles. En esta sección repasaremos algunos de los conceptos básicos.

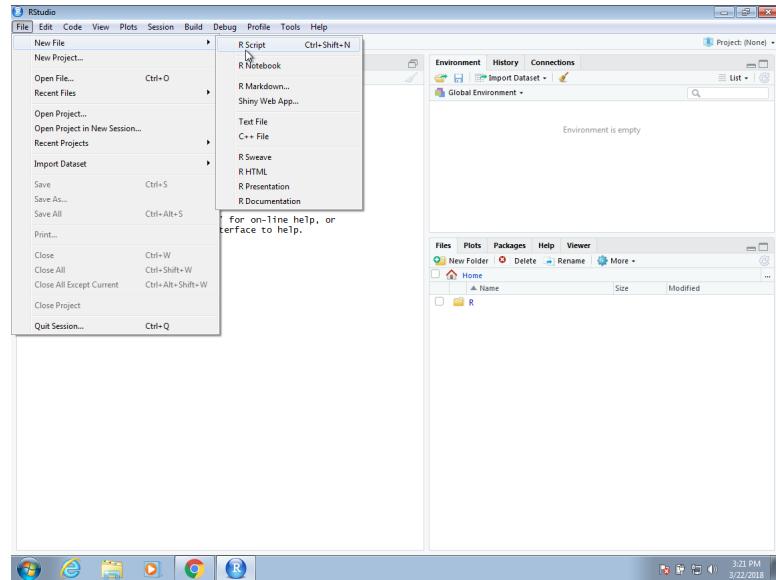
1.4.1 Paneles

Cuando inicien RStudio por primera vez, verán tres paneles. El panel izquierdo muestra la consola R. A la derecha, el panel superior incluye pestañas como *Environment* y *History*, mientras que el panel inferior muestra cinco pestañas: *File*, *Plots*, *Packages*, *Help* y *Viewer* (estas pestañas pueden ser diferentes en las nuevas versiones de RStudio). Pueden hacer clic en cada pestaña para moverse por las diferentes opciones.

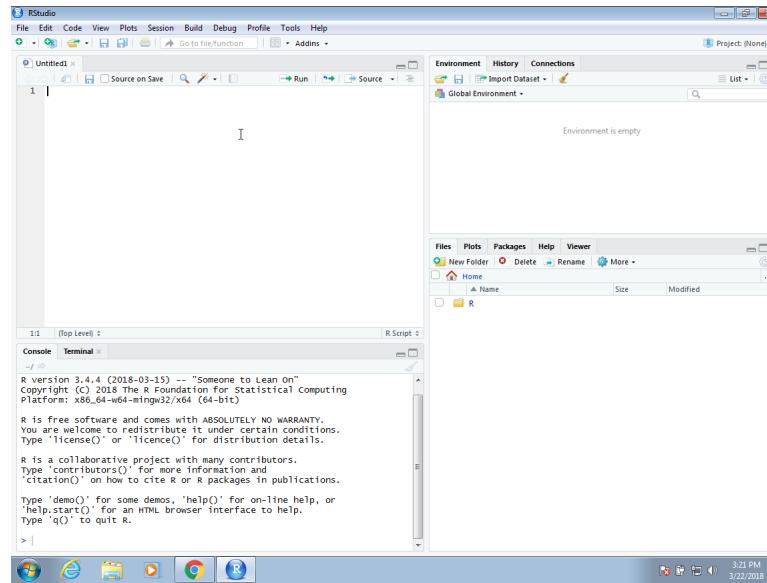
⁶<https://github.com/rafalab/dsbook>



Para iniciar un nuevo *script*, hagan clic en *File*, entonces *New File* y luego *R Script*.

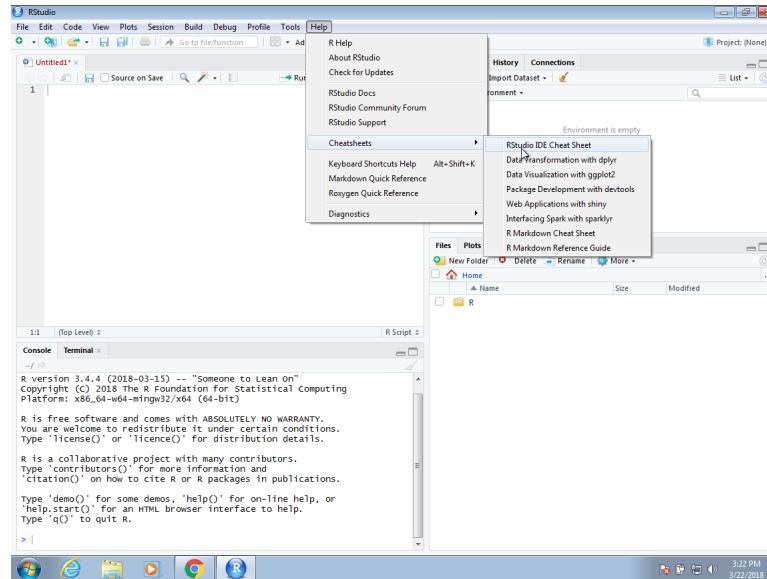


Esto inicia un nuevo panel a la izquierda y es aquí donde pueden comenzar a escribir su *script*.



1.4.2 Atajos de teclado

Aunque en este tutorial a menudo mostramos cómo usar el mouse, **les recomendamos que memoricen los atajos de teclado (*key bindings* en inglés) para las operaciones que usan con mayor frecuencia.** RStudio incluye una hoja de referencia (*cheat sheet* en inglés) útil con los comandos más utilizados. Pueden obtenerla directamente de RStudio así:



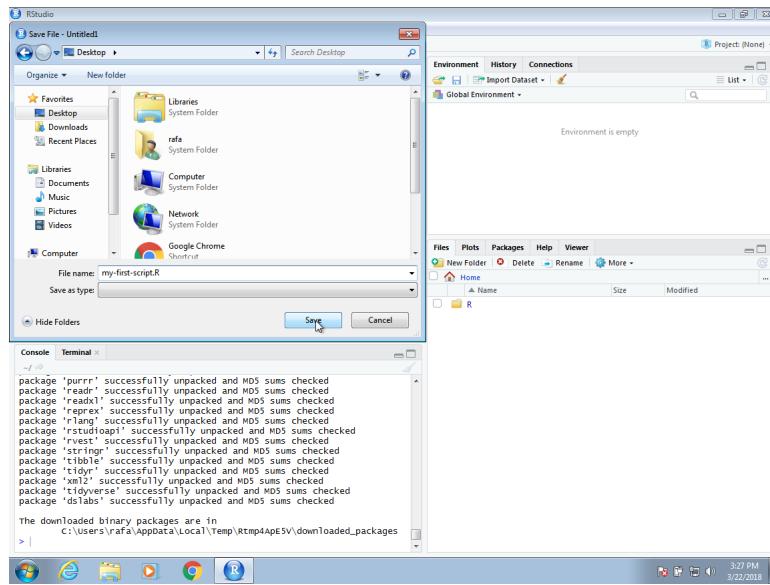
Recomendamos tener esto a mano para poder buscar las combinaciones de teclas cuando se encuentren apuntando y haciendo clic repetidas veces. Pueden encontrar versiones en español aquí: <https://www.rstudio.com/resources/cheatsheets/>.

1.4.3 Cómo ejecutar comandos mientras editan *scripts*

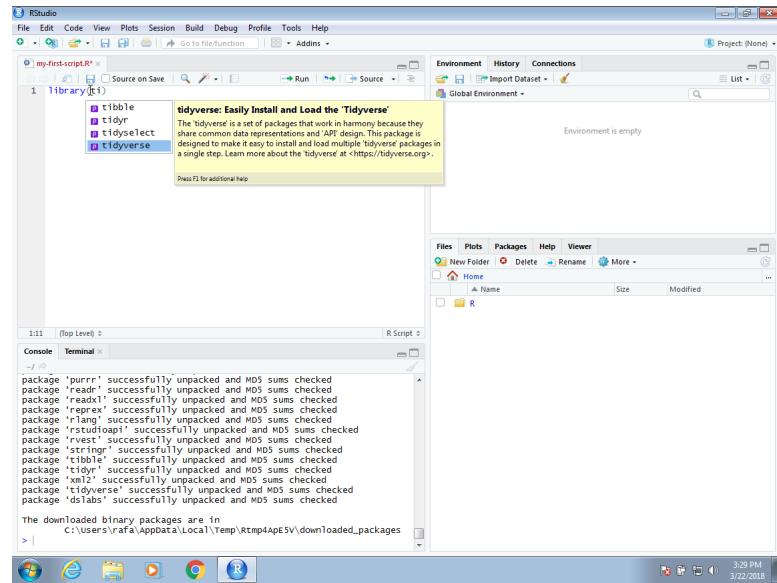
Hay muchos editores diseñados específicamente para la codificación. Estos son útiles porque el color y la indentación se agregan automáticamente para que el código sea más legible. RStudio es uno de estos editores y se desarrolló específicamente para R. Una de las principales ventajas que RStudio tiene sobre otros editores es que podemos probar nuestro código fácilmente mientras editamos nuestros *scripts*. A continuación ofrecemos un ejemplo.

Comencemos abriendo un nuevo *script* como lo hicimos antes. Entonces, nombremos el *script*. Podemos hacer esto a través del editor guardando el nuevo *script* actual sin nombre. Para empezar, hagan clic en el ícono de guardar o usando la combinación de teclas Ctrl + S en Windows y Command + S en Mac.

Al intentar guardar el documento por primera vez, RStudio le pedirá un nombre. Una buena convención es usar un nombre descriptivo, con letras minúsculas, sin espacios, sólo guiones para separar las palabras y luego seguido del sufijo *.R*. Llamaremos a este *script*: *my-first-script.R*.



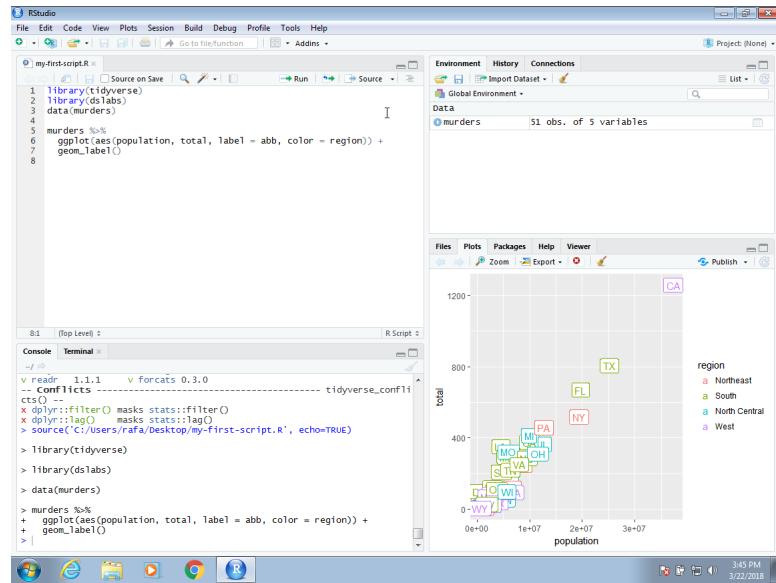
Ahora estamos listos para comenzar a editar nuestro primer *script*. Las primeras líneas de código en un *script* de R se dedican a cargar los paquetes que usaremos. Otra característica útil de RStudio es que una vez escribimos `library()`, RStudio comienza a completar automáticamente lo que estamos escribiendo con los paquetes que hemos instalado. Observen lo que sucede cuando escribimos `library(ti)`:



Otra característica que pueden haber notado es que cuando escriben `library()` el segundo paréntesis se agrega automáticamente. Esto les ayudará a evitar uno de los errores más comunes en la codificación: olvidar de cerrar un paréntesis.

Ahora podemos continuar escribiendo código. Como ejemplo, crearemos un gráfico que muestre los totales de asesinatos versus los totales de población por estado de EE.UU. Una vez que hayan terminado de escribir el código necesario para hacer este gráfico, pueden probarlo *ejecutando* el código. Para hacer esto, hagan clic en el botón *Run* en la parte derecha superior del panel de edición. También pueden usar la combinación de teclas: Ctrl + Shift + Enter en Windows o Command + Shift + Return en Mac.

Tan pronto corran el código, verán que este aparece en la consola R y, en este caso, el gráfico que resulta aparece en la consola de gráficos. Noten que la consola de gráficos tiene una interfaz útil que les permite hacer clic hacia delante o hacia atrás en diferentes gráficos, hacer zoom en el gráfico o guardar los gráficos como archivos.



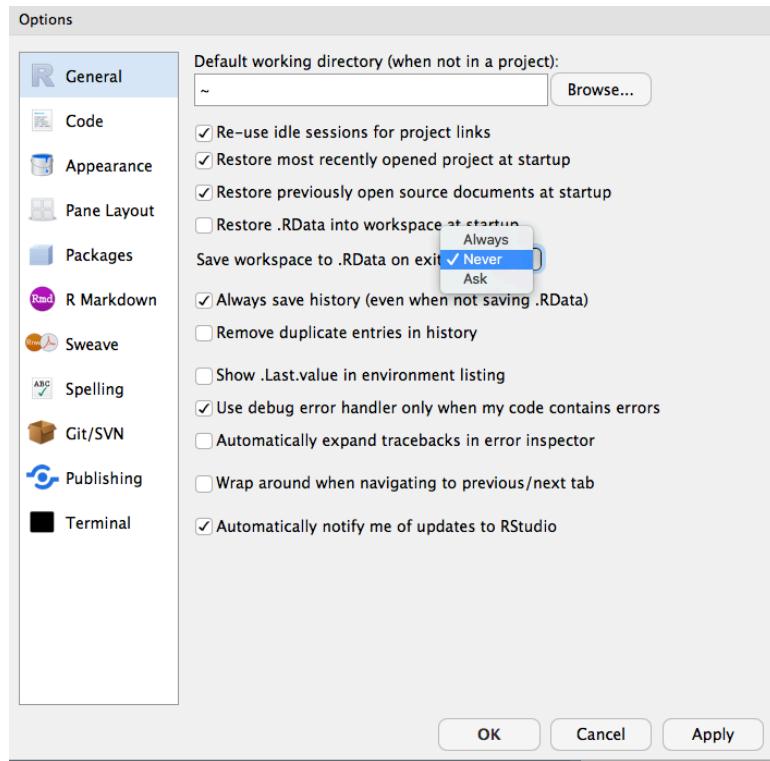
Para ejecutar una línea a la vez en lugar del *script* completo, pueden usar Control-Enter en Windows y Command-Return en Mac.

1.4.4 Cómo cambiar las opciones globales

Pueden cambiar bastante el aspecto y la funcionalidad de RStudio.

Para cambiar las opciones globales, hagan clic en *Tools* y luego en *Global Options ...*.

Como ejemplo, mostramos cómo hacer un cambio que **sumamente recomendamos**: cambiar el *Save workspace to .RData on exit* a *Never* y desmarcar *Restore .RData into workspace at start*. Por defecto, cuando salen de R, el programa guarda todos los objetos que han creado en un archivo llamado *.RData*. Esto ocurre para que cuando reinician la sesión en el mismo archivo, el programa cargue estos objetos. Sin embargo, encontramos que esto causa confusión, especialmente cuando compartimos código con colegas y suponemos que tienen este archivo *.RData*. Para cambiar estas opciones, hagan que su configuración *General* se vea así:



1.5 Instalación de paquetes de R

La funcionalidad que una nueva instalación de R ofrece es sólo una pequeña fracción de lo que es posible. De hecho, nos referimos a lo que obtienen después de su primera instalación como *base R*. La funcionalidad adicional proviene de complementos disponibles de los desarrolladores. Actualmente hay cientos de estos disponibles de CRAN y muchos otros compartidos a través de otros repositorios como GitHub. Sin embargo, debido a que no todo el mundo necesita todas las funciones disponibles, R pone a disposición diferentes componentes a través de paquetes (*packages* en inglés). R facilita la instalación de paquetes desde R. Por ejemplo, para instalar el paquete **dslabs**, que usamos para compartir los sets de datos y códigos relacionados con este libro, deben escribir:

```
install.packages("dslabs")
```

En RStudio pueden navegar a la pestaña *Tools* y seleccionar *Install packages*. Luego, podemos cargar el paquete en nuestras sesiones de R usando la función **library**:

```
library(dslabs)
```

A medida que vayan leyendo este libro, verán que cargamos paquetes sin instalarlos. Esto se debe a que una vez que instalen un paquete, permanece instalado y sólo necesita cargarse con

`library`. El paquete permanece cargado hasta que terminemos con la sesión R. Si intentan cargar un paquete y obtienen un error, probablemente significa que no lo han instalado.

Podemos instalar más de un paquete a la vez al proveerle un vector de caracteres a esta función:

```
install.packages(c("tidyverse", "dslabs"))
```

Tengan en cuenta que la instalación de **tidyverse** instala varios paquetes. Esto ocurre comúnmente cuando un paquete tiene *dependencias*, es decir usa funciones de otros paquetes. Cuando cargan un paquete usando `library`, también cargan sus dependencias.

Una vez que los paquetes estén instalados, pueden cargarlos en R y no necesitan instalarlos nuevamente, a menos que instalen una versión nueva de R. Recuerden que los paquetes están instalados en R y no en RStudio.

Es útil mantener una lista de todos los paquetes que necesitan para su trabajo en un *script* porque si tienen que realizar una instalación nueva de R, pueden reinstalar todos sus paquetes simplemente ejecutando un *script*.

Pueden ver todos los paquetes que han instalado utilizando la siguiente función:

```
installed.packages()
```

Part I

R

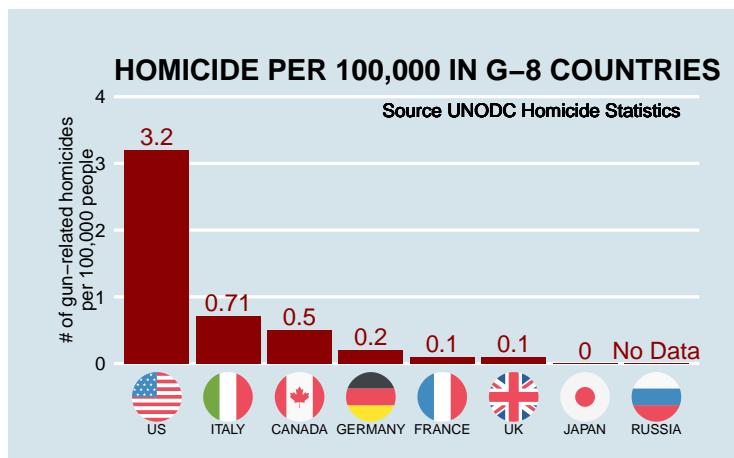
2

Lo básico de R

En este libro, utilizaremos el ambiente de software R para todo nuestro análisis. Aprenderán R y las técnicas de análisis de datos simultáneamente. Por lo tanto, para continuar necesitarán acceso a R. También recomendamos el uso de un *Entorno de Desarrollo Integrado* (IDE), como RStudio, para guardar su trabajo. Recuerden que es común que un curso o taller ofrezca acceso a un ambiente de R y a un IDE a través de su navegador de web, como lo hace RStudio cloud¹. Si tienen acceso a dicho recurso, no necesitan instalar R ni RStudio. Sin embargo, si eventualmente quieren convertirse en analistas expertos de datos, recomendamos instalar estas herramientas en su computadora². Tanto R como RStudio son gratuitos y están disponibles en línea.

2.1 Caso de estudio: los asesinatos con armas en EE. UU.

Imagínense que viven en Europa y se les ofrece un trabajo en una empresa estadounidense con muchas ubicaciones por todo EE. UU. Es un gran trabajo, pero noticias con titulares como **Tasa de homicidios con armas de fuego de EE. UU. más alta que en otros países desarrollados**³. ¿Se preocupan? Gráficos como el siguiente pueden preocuparle aún más:



¹<https://rstudio.cloud>

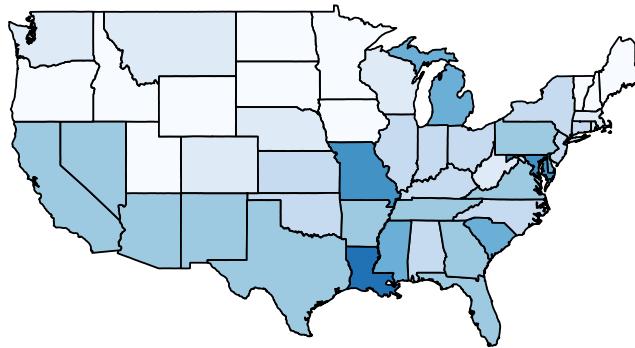
²<https://rafalab.github.io/dsbook/installing-r-rstudio.html>

³<http://abcnews.go.com/blogs/headlines/2012/12/us-gun-ownership-homicide-rate-higher-than-other-developed-countries/>

O peor aún, esta versión de everytown.org:



Pero entonces se recuerdan que Estados Unidos es un país grande y diverso, con 50 estados muy diferentes, además del Distrito de Columbia (DC).



California, por ejemplo, tiene una población más grande que Canadá, y 20 estados de EE. UU. tienen poblaciones más grandes que la de Noruega. En algunos aspectos, la variabilidad entre los estados de EE. UU. es parecida a la variabilidad entre los países de Europa. Además, aunque no se incluyen en los cuadros anteriores, las tasas de asesinatos en Lituania, Ucrania y Rusia son superiores a cuatro por cada 100,000. Entonces, es posible que las noticias que les preocupan sean demasiado superficiales. Tienen opciones de dónde pueden vivir y desean determinar la seguridad de cada estado en particular. Obtendremos algunas ideas al examinar los datos relacionados con asesinatos con armas de fuego de EE. UU. en 2010 usando R.

Antes de comenzar con nuestro ejemplo, necesitamos discutir la logística, así como algunos de los componentes básicos necesarios para obtener destrezas más avanzadas de R. Recuerden que la utilidad de algunos de estos componentes básicos no siempre es inmediatamente obvia, pero más tarde en el libro apreciarán haber dominado estas destrezas.

2.2 Lo básico

Antes de empezar con el set de datos motivante, necesitamos repasar los conceptos básicos de R.

2.2.1 Objetos

Supongan que unos estudiantes de secundaria nos piden ayuda para resolver varias ecuaciones cuadráticas de la forma $ax^2 + bx + c = 0$. La fórmula cuadrática nos ofrece las soluciones:

$$\frac{-b - \sqrt{b^2 - 4ac}}{2a} \text{ and } \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

que por supuesto cambian dependiendo de los valores de a , b y c . Una ventaja de los lenguajes de programación es poder definir variables y escribir expresiones con estas, como se hace en las matemáticas, para obtener una solución numérica. Escribiremos un código general para la ecuación cuadrática a continuación, pero si nos piden resolver $x^2 + x - 1 = 0$, entonces definimos:

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

que almacena los valores para su uso posterior. Usamos `<-` para asignar valores a las variables.

También podemos asignar valores usando `=` en lugar de `<-`, pero recomendamos no usar `=` para evitar confusión.

Copien y peguen el código anterior en su consola para definir las tres variables. Tengan en cuenta que R no imprime nada cuando hacemos esta asignación. Esto significa que los objetos se definieron con éxito. Si hubieran cometido un error, recibirían un mensaje de error.

Para ver el valor almacenado en una variable, simplemente le pedimos a R que evalúe `a` y R nos muestra el valor almacenado:

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

Una forma más explícita de pedirle a R que nos muestre el valor almacenado en `a` es usar `print` así:

```
print(a)
#> [1] 1
```

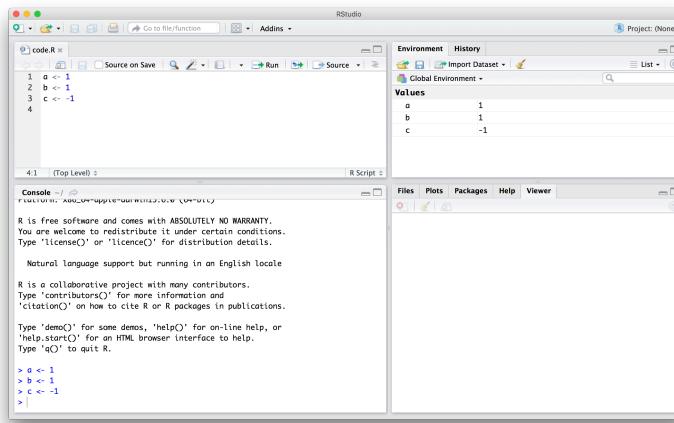
Usamos el término *objeto* para describir cosas que están almacenadas en R. Las variables son un ejemplo, pero los objetos también pueden ser entidades más complicadas como las funciones, que se describen más adelante.

2.2.2 El espacio de trabajo

A medida que definimos objetos en la consola, estamos cambiando el *espacio de trabajo* (*workspace* en inglés). Pueden ver todas las variables guardadas en su espacio de trabajo al escribir:

```
ls()
#> [1] "a"          "b"          "c"          "dat"        "img_path"   "murders"
```

En RStudio, la pestaña *Environment* muestra los valores:



Deberíamos ver **a**, **b** y **c**. Si intentan obtener el valor de una variable que no está en su espacio de trabajo, recibirán un mensaje de error. Por ejemplo, si escriben **x**, verán lo siguiente: **Error: object 'x' not found**.

Ahora, dado que estos valores se guardan en variables, para resolver nuestra ecuación, utilizamos la fórmula cuadrática:

```
(-b + sqrt(b^2 - 4*a*c) )/ ( 2*a )
#> [1] 0.618
(-b - sqrt(b^2 - 4*a*c) )/ ( 2*a )
#> [1] -1.62
```

2.2.3 Funciones

Una vez que definan las variables, el proceso de análisis de datos generalmente se puede describir como una serie de funciones aplicadas a los datos. R incluye varias funciones predefinidas y la mayoría de las líneas de análisis que construimos hacen uso extensivo de ellas.

Ya usamos las funciones `install.packages`, `library` y `ls`. También usamos la función `sqrt` para solucionar la ecuación cuadrática anterior. Hay muchas más funciones predefinidas y se pueden añadir hasta más a través de paquetes. Estas no aparecen en sus espacios de trabajo porque no las definieron, pero están disponibles para su uso inmediato.

En general, necesitamos usar paréntesis para evaluar una función. Si escriben `ls`, la función no se evalúa y en cambio R les muestra el código que la define. Si escriben `ls()`, la función se evalúa y, como ya se mostró, vemos objetos en el espacio de trabajo.

A diferencia de `ls`, la mayoría de las funciones requieren uno o más argumentos. A continuación se muestra un ejemplo de cómo asignamos un objeto al argumento de la función `log`. Recuerden que anteriormente definimos `a` como 1:

```
log(8)
#> [1] 2.08
log(a)
#> [1] 0
```

Pueden averiguar lo que la función espera y lo que hace revisando unos manuales muy útiles incluidos en R. Pueden obtener ayuda utilizando la función `help` así:

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

Para la mayoría de las funciones, también podemos usar esta abreviatura:

```
?log
```

La página de ayuda les mostrará qué argumentos espera la función. Por ejemplo, `log` necesita `x` y `base` para correr. Sin embargo, algunos argumentos son obligatorios y otros son opcionales. Pueden determinar cuáles son opcionales notando en el documento de ayuda cuáles valores predeterminados se asignan con `=`. Definir estos es opcional. Por ejemplo, la base de la función `log` por defecto es `base = exp(1)` que hace `log` el logaritmo natural por defecto.

Para echar un vistazo rápido a los argumentos sin abrir el sistema de ayuda, pueden escribir:

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

Pueden cambiar los valores predeterminados simplemente asignando otro objeto:

```
log(8, base = 2)
#> [1] 3
```

Recuerden que no hemos estado especificando el argumento `x` como tal:

```
log(x = 8, base = 2)
#> [1] 3
```

El código anterior funciona, pero también podemos ahorrarnos un poco de escritura: si no usan un nombre de argumento, R supone que están ingresando argumentos en el orden en que se muestran en la página de ayuda o por `args`. Entonces, al no usar los nombres, R supone que los argumentos son `x` seguido por `base`:

```
log(8,2)
#> [1] 3
```

Si usan los nombres de los argumentos, podemos incluirlos en el orden en que queramos:

```
log(base = 2, x = 8)
#> [1] 3
```

Para especificar argumentos, debemos usar `=` y no `<-`.

Hay algunas excepciones a la regla de que las funciones necesitan los paréntesis para ser evaluadas. Entre estas, los más utilizados son los operadores aritméticos y relacionales. Por ejemplo:

```
2^3
#> [1] 8
```

Pueden ver los operadores aritméticos al escribir:

```
help("+")
```

O

```
? "+"
```

y los operadores relacionales al escribir:

```
help(">")
```

O

```
? ">"
```

2.2.4 Otros objetos predefinidos

Hay varios sets de datos que se incluyen para que los usuarios practiquen y prueben las funciones. Pueden ver todos los sets de datos disponibles escribiendo:

```
data()
```

Esto les muestra el nombre del objeto para estos sets de datos. Estos sets de datos son objetos que se pueden usar simplemente escribiendo el nombre. Por ejemplo, si escriben:

```
co2
```

R les mostrará los datos de concentración de CO₂ atmosférico de Mauna Loa.

Otros objetos predefinidos son cantidades matemáticas, como la constante π e ∞ :

```
pi
#> [1] 3.14
Inf+1
#> [1] Inf
```

2.2.5 Nombres de variables

Hemos usado las letras `a`, `b` y `c` como nombres de variables, pero estos pueden ser casi cualquier cosa. Algunas reglas básicas en R son que los nombres de variables tienen que comenzar con una letra, no pueden contener espacios y no deben ser variables predefinidas en R. Por ejemplo, no nombren una de sus variables `install.packages` escribiendo algo como: `install.packages <- 2`.

Una buena convención a seguir es usar palabras significativas que describan lo que están almacenado, usar solo minúsculas y usar guiones bajos como sustituto de espacios. Para las ecuaciones cuadráticas, podríamos usar algo como:

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

Para obtener más consejos, recomendamos estudiar la guía de estilo de Hadley Wickham⁴.

2.2.6 Cómo guardar su espacio de trabajo

Los valores permanecen en el espacio de trabajo hasta que finalicen sus sesiones o las borren con la función `rm`. Pero los espacios de trabajo también se pueden guardar para su uso posterior. De hecho, al salir de R, el programa les pregunta si desean guardar su espacio de trabajo. Si lo guardan, la próxima vez que inicien R, el programa restaurará el espacio de trabajo.

Sin embargo, no recomendamos guardar el espacio de trabajo así porque, a medida que comiencen a trabajar en diferentes proyectos, será más difícil darle seguimiento de lo que guardan. En cambio, les recomendamos que le asignen al espacio de trabajo un nombre específico. Pueden hacer esto usando las funciones `save` o `save.image`. Para cargar, usen la función `load`. Al guardar un espacio de trabajo, recomendamos el sufijo `rda` o `RData`. En RStudio, también pueden hacerlo navegando a la pestaña *Session* y eligiendo *Save Workspace as*. Luego pueden cargarlo usando las opciones *Load Workspace* en la misma pestaña. Para aprender más, lean las páginas de ayuda (*help pages* en inglés) en `save`, `save.image` y `load`.

2.2.7 Scripts motivantes

Para resolver otra ecuación como $3x^2 + 2x - 1$, podemos copiar y pegar el código anterior, redefinir las variables y volver a calcular la solución:

⁴<http://adv-r.had.co.nz/Style.html>

```
a <- 3
b <- 2
c <- -1
(-b + sqrt(b^2 - 4*a*c))/ (2*a)
(-b - sqrt(b^2 - 4*a*c))/ (2*a)
```

Al crear y guardar un *script* con el código anterior, ya no tendrán que volver a escribirlo todo cada vez, sino simplemente cambiar los nombres de las variables. Intenten escribir la secuencia de comandos anteriores en un editor y observen lo fácil que es cambiar las variables y recibir una respuesta.

2.2.8 Cómo comentar su código

Si una línea de código R comienza con el símbolo `#`, no se evalúa. Podemos usar esto para escribir recordatorios de por qué escribimos un código particular. Por ejemplo, en el *script* anterior, podríamos añadir:

```
## Código para calcular la solución a la
## ecuación cuadrática de la forma ax^2 + bx + c
## definir las variables
a <- 3
b <- 2
c <- -1

## ahora calcule la solución
(-b + sqrt(b^2 - 4*a*c)) / (2*a)
(-b - sqrt(b^2 - 4*a*c)) / (2*a)
```

2.3 Ejercicios

1. ¿Cuál es la suma de los primeros 100 números enteros positivos? La fórmula para la suma de enteros 1 a n es $n(n + 1)/2$. Defina $n = 100$ y luego use R para calcular la suma de 1 a 100 usando la fórmula. ¿Cuál es la suma?

2. Ahora use la misma fórmula para calcular la suma de los enteros del 1 a 1000.

3. Mire el resultado de escribir el siguiente código en R:

```
n <- 1000
x <- seq(1, n)
sum(x)
```

Basado en el resultado, ¿qué cree que hacen las funciones `seq` y `sum`? Puede usar `help`.

- `sum` crea una lista de números y `seq` los suma.
- `seq` crea una lista de números y `sum` los suma.

- c. `seq` crea una lista aleatoria y `sum` calcula la suma de 1 a 1000.
d. `sum` siempre devuelve el mismo número.
4. En las matemáticas y la programación decimos que evaluamos una función cuando reemplazamos el argumento con un número dado. Entonces si escribimos `sqrt(4)`, evaluamos la función `sqrt`. En R se puede evaluar una función dentro de otra función. Las evaluaciones suceden de adentro hacia afuera. Use una línea de código para calcular el logaritmo, en base 10, de la raíz cuadrada de 100.
5. ¿Cuál de los siguientes siempre devolverá el valor numérico almacenado en `x`? Puede intentar los ejemplos y usar el sistema de ayuda si lo desea.
- `log(10^x)`
 - `log10(x^10)`
 - `log(exp(x))`
 - `exp(log(x, base = 2))`

2.4 Tipos de datos

Las variables en R pueden ser de diferentes tipos. Por ejemplo, necesitamos distinguir números de cadenas de caracteres y tablas de listas sencillas de números. La función `class` nos ayuda a determinar qué tipo de objeto tenemos:

```
a <- 2
class(a)
#> [1] "numeric"
```

Para trabajar eficientemente en R, es importante aprender los diferentes tipos de variables y qué podemos hacer con ellos.

2.4.1 Data frames

Hasta ahora, las variables que hemos definido son solo un número. Esto no es muy útil para almacenar datos. La forma más común de almacenar un set de datos en R es usando un *data frame*. Conceptualmente, podemos pensar en un *data frame* como una tabla con filas que representan observaciones y con columnas que representan las diferentes variables recopiladas para cada observación. Los *data frames* son particularmente útiles para sets de datos porque podemos combinar diferentes tipos de datos en un solo objeto.

Una gran proporción de los retos del análisis de datos comienza con datos almacenados en un *data frame*. Por ejemplo, almacenamos los datos para nuestro ejemplo motivante en un *data frame*. Pueden tener acceso a este set de datos cargando el paquete `dslabs` y entonces utilizando la función `data` para cargar el set de datos `murders`:

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

Para verificar que esto es un *data frame*, escribimos:

```
class(murders)
#> [1] "data.frame"
```

2.4.2 Cómo examinar un objeto

La función `str` es útil para obtener más información sobre la estructura de un objeto:

```
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 ...
```

Esto nos dice mucho más sobre el objeto. Vemos que la tabla tiene 51 filas (50 estados más DC) y cinco variables. Podemos mostrar las primeras seis líneas usando la función `head`:

```
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
```

En este set de datos, cada estado se considera una observación y se incluyen cinco variables para cada estado.

Antes de continuar respondiendo a nuestra pregunta original sobre los diferentes estados, repasemos más sobre los componentes de este objeto.

2.4.3 El operador de acceso: \$

Para nuestro análisis, necesitaremos acceso a las diferentes variables representadas por columnas incluidas en este *data frame*. Para hacer esto, utilizamos el operador de acceso `$` de la siguiente manera:

```
murders$population
#> [1] 4779736 710231 6392017 2915918 37253956 5029196 3574097
#> [8] 897934 601723 19687653 9920000 1360301 1567582 12830632
#> [15] 6483802 3046355 2853118 4339367 4533372 1328361 5773552
#> [22] 6547629 9883640 5303925 2967297 5988927 989415 1826341
#> [29] 2700551 1316470 8791894 2059179 19378102 9535483 672591
```

```
#> [36] 11536504 3751351 3831074 12702379 1052567 4625364 814180
#> [43] 6346105 25145561 2763885 625741 8001024 6724540 1852994
#> [50] 5686986 563626
```

¿Pero cómo supimos usar `population`? Anteriormente, aplicando la función `str` al objeto `murders`, revelamos los nombres de cada una de las cinco variables almacenadas en esta tabla. Podemos tener acceso rápido a los nombres de las variables usando:

```
names(murders)
#> [1] "state"      "abb"        "region"     "population" "total"
```

Es importante saber que el orden de las entradas en `murders$population` conserva el orden de las filas en nuestra tabla de datos. Esto luego nos permitirá manipular una variable basada en los resultados de otra. Por ejemplo, podremos ordenar los nombres de los estados según el número de asesinatos.

Consejo: R viene con una muy buena funcionalidad de autocompletar que nos ahorra la molestia de escribir todos los nombres. Escriban `murders$p` y luego presionen la tecla `tab` en su teclado. Esta funcionalidad y muchas otras características útiles de autocompletar están disponibles en RStudio.

2.4.4 Vectores: numéricos, de caracteres y lógicos

El objeto `murders$population` no es un número sino varios. Llamamos *vectores* a este tipo de objeto. Un solo número es técnicamente un vector de longitud 1, pero en general usamos el término vectores para referirnos a objetos con varias entradas. La función `length` les dice cuántas entradas hay en el vector:

```
pop <- murders$population
length(pop)
#> [1] 51
```

Este vector particular es *numérico* ya que los tamaños de la población son números:

```
class(pop)
#> [1] "numeric"
```

En un vector numérico cada entrada debe ser un número.

Para almacenar una cadena de caracteres, los vectores también pueden ser de la clase *caracter*. Por ejemplo, los nombres de los estados son caracteres:

```
class(murders$state)
#> [1] "character"
```

Al igual que con los vectores numéricos, todas las entradas en un vector de caracteres deben ser un carácter.

Otro tipo importante de vectores son los *vectores lógicos*. Estos deben ser TRUE o FALSE.

```
z <- 3 == 2
z
#> [1] FALSE
class(z)
#> [1] "logical"
```

Aquí el `==` es un operador relacional que pregunta si 3 es igual a 2. En R, usar solo un `=` asigna una variable, pero si usan dos `==`, entonces evalúa si los objetos son iguales.

Pueden ver esto al escribir:

```
?Comparison
```

En futuras secciones, observarán lo útil que pueden ser los operadores relacionales.

Discutimos las características más importantes de los vectores después de los siguientes ejercicios.

Avanzado: Matemáticamente, los valores en `pop` son números enteros y hay una clase de enteros en R. Sin embargo, por defecto, a los números se les asigna una clase numérica incluso cuando son enteros redondos. Por ejemplo, `class(1)` devuelve numérico. Pueden convertirlo en un entero de clase con la función `as.integer()` o agregando un L así: `1L`. Tengan en cuenta la clase escribiendo: `class(1L)`.

2.4.5 Factores

En el set de datos `murders`, se podría esperar que la región también sea un vector de caracteres. Sin embargo, no lo es:

```
class(murders$region)
#> [1] "factor"
```

Es un *factor*. Los factores son útiles para almacenar datos categóricos. Podemos ver que solo hay cuatro regiones al utilizar la función `levels`:

```
levels(murders$region)
#> [1] "Northeast"      "South"           "North Central"   "West"
```

En el fondo, R almacena estos *levels* como números enteros y mantiene un mapa para llevar un registro de las etiquetas. Esto es más eficiente en términos de memoria que almacenar todos los caracteres.

Tengan en cuenta que los niveles tienen un orden diferente al orden de aparición en el factor. En R, por defecto, los niveles se ordenan alfabéticamente. Sin embargo, a menudo queremos que los niveles sigan un orden diferente. Pueden especificar un orden a través del argumento `levels` cuando crean el factor con la función `factor`. Por ejemplo, en el set de datos de asesinatos, las regiones se ordenan de este a oeste. La función `reorder` nos permite cambiar el orden de los niveles de un factor según un resumen calculado en un vector numérico. Demostraremos esto con un ejemplo sencillo y veremos otros más avanzados en la parte Visualización de Datos del libro.

Supongan que queremos que los *levels* de la región se ordenen según el número total de asesinatos en vez de por orden alfabético. Si hay valores asociados con cada *level*, podemos usar `reorder` y especificar un resumen de datos para determinar el orden. El siguiente código toma la suma del total de asesinatos en cada región y reordena el factor según estas sumas.

```
region <- murders$region
value <- murders$total
region <- reorder(region, value, FUN = sum)
levels(region)
#> [1] "Northeast"      "North Central" "West"           "South"
```

El nuevo orden concuerda con el hecho de que hay menos asesinatos en el Noreste y más en el Sur.

Advertencia: Los factores pueden causar confusión ya que a veces se comportan como caracteres y otras veces no. Como resultado, estos son una fuente común de errores.

2.4.6 Listas

Los *data frames* son un caso especial de *listas*. Las listas son útiles porque pueden almacenar cualquier combinación de diferentes tipos de datos. Pueden crear una lista utilizando la función `lista` así:

```
record <- list(name = "John Doe",
                 student_id = 1234,
                 grades = c(95, 82, 91, 97, 93),
                 final_grade = "A")
```

La función `c` se describe en la sección 2.6.

Esta lista incluye un carácter, un número, un vector con cinco números y otro carácter.

```
record
#> $name
#> [1] "John Doe"
#>
#> $student_id
#> [1] 1234
#>
#> $grades
#> [1] 95 82 91 97 93
#>
#> $final_grade
#> [1] "A"
class(record)
#> [1] "list"
```

Al igual que con los *data frames*, pueden extraer los componentes de una lista con el operador de acceso: `$`.

```
record$student_id
#> [1] 1234
```

También podemos usar corchetes dobles ([]) así:

```
record[["student_id"]]
#> [1] 1234
```

Deben acostumbrarse al hecho de que, en R, frecuentemente hay varias formas de hacer lo mismo, como tener acceso a las entradas.

También pueden encontrar listas sin nombres de variables:

```
record2 <- list("John Doe", 1234)
record2
#> [[1]]
#> [1] "John Doe"
#>
#> [[2]]
#> [1] 1234
```

Si una lista no tiene nombres, no pueden extraer los elementos con \$, pero todavía pueden usar el método de corchetes. En vez de proveer el nombre de la variable, pueden proveer el índice de la lista de la siguiente manera:

```
record2[[1]]
#> [1] "John Doe"
```

No usaremos listas hasta más tarde, pero es posible que encuentren una en sus propias exploraciones de R. Por eso, les mostramos algunos conceptos básicos aquí.

2.4.7 Matrices

Las matrices son otro tipo de objeto común en R. Las matrices son similares a los *data frames* en que son bidimensionales: tienen filas y columnas. Sin embargo, al igual que los vectores numéricos, de caracteres y lógicos, las entradas en las matrices deben ser del mismo tipo. Por esta razón, los *data frames* son mucho más útiles para almacenar datos, ya que podemos tener caracteres, factores y números en ellos.

No obstante, las matrices tienen una gran ventaja sobre los *data frames*: podemos realizar operaciones de álgebra de matrices, una técnica matemática poderosa. No describimos estas operaciones en este libro, pero gran parte de lo que sucede en segundo plano cuando se realiza un análisis de datos involucra matrices. Cubrimos las matrices con más detalle en el Capítulo 33.1 pero también las discutimos brevemente aquí, ya que algunas de las funciones que aprenderemos devuelven matrices.

Podemos definir una matriz usando la función `matrix`. Necesitamos especificar el número de filas y columnas:

```
mat <- matrix(1:12, 4, 3)
mat
#>      [,1] [,2] [,3]
#> [1,]    1    5    9
#> [2,]    2    6   10
#> [3,]    3    7   11
#> [4,]    4    8   12
```

Pueden acceder a entradas específicas en una matriz usando corchetes ([]). Si desean la segunda fila, tercera columna, escriban:

```
mat[2, 3]
#> [1] 10
```

Si desean toda la segunda fila, dejen vacío el lugar de la columna:

```
mat[2, ]
#> [1] 2 6 10
```

Noten que esto devuelve un vector, no una matriz.

Del mismo modo, si desean la tercera columna completa, dejen el lugar de la fila vacío:

```
mat[, 3]
#> [1] 9 10 11 12
```

Esto también es un vector, no una matriz.

Pueden acceder a más de una columna o más de una fila si lo desean. Esto les dará una nueva matriz:

```
mat[, 2:3]
#>      [,1] [,2]
#> [1,]    5    9
#> [2,]    6   10
#> [3,]    7   11
#> [4,]    8   12
```

Pueden crear subconjuntos basados tanto en las filas como en las columnas:

```
mat[1:2, 2:3]
#>      [,1] [,2]
#> [1,]    5    9
#> [2,]    6   10
```

Podemos convertir las matrices en *data frames* usando la función `as.data.frame`:

```
as.data.frame(mat)
#>   V1 V2 V3
#> 1  1  5  9
#> 2  2  6 10
#> 3  3  7 11
#> 4  4  8 12
```

También podemos usar corchetes individuales (`[]`) para acceder a las filas y las columnas de un *data frame*:

```
data("murders")
murders[25, 1]
#> [1] "Mississippi"
murders[2:3, ]
#>   state abb region population total
#> 2 Alaska AK    West      710231    19
#> 3 Arizona AZ    West     6392017   232
```

2.5 Ejercicios

1. Cargue el set de datos de asesinatos de EE.UU.

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

Use la función `str` para examinar la estructura del objeto `murders`. ¿Cuál de las siguientes opciones describe mejor las variables representadas en este *data frame*?

- a. Los 51 estados.
 - b. Las tasas de asesinatos para los 50 estados y DC.
 - c. El nombre del estado, la abreviatura del nombre del estado, la región del estado y la población y el número total de asesinatos para 2010 del estado.
 - d. `str` no muestra información relevante.
2. ¿Cuáles son los nombres de las columnas utilizados por el *data frame* para estas cinco variables?
 3. Use el operador de acceso `$` para extraer las abreviaturas de los estados y asignarlas al objeto `a`. ¿Cuál es la clase de este objeto?
 4. Ahora use los corchetes para extraer las abreviaturas de los estados y asignarlas al objeto `b`. Utilice la función `identical` para determinar si `a` y `b` son iguales.
 5. Vimos que la columna `region` almacena un factor. Puede corroborar esto escribiendo:

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

Con una línea de código, use las funciones `levels` y `length` para determinar el número de regiones definidas por este set de datos.

6. La función `table` toma un vector y devuelve la frecuencia de cada elemento. Puede ver rápidamente cuántos estados hay en cada región aplicando esta función. Use esta función en una línea de código para crear una tabla de estados por región.

2.6 Vectores

En R, los objetos más básicos disponibles para almacenar datos son *vectores*. Como hemos visto, los sets de datos complejos generalmente se pueden dividir en componentes que son vectores. Por ejemplo, en un *data frame*, cada columna es un vector. Aquí aprendemos más sobre esta clase importante.

2.6.1 Cómo crear vectores

Podemos crear vectores usando la función `c`, que significa *concatenar*. Usamos `c` para concatenar entradas de la siguiente manera:

```
codes <- c(380, 124, 818)
codes
#> [1] 380 124 818
```

También podemos crear vectores de caracteres. Usamos las comillas para denotar que las entradas son caracteres en lugar de nombres de variables.

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

En R, también pueden usar comillas sencillas:

```
country <- c('italy', 'canada', 'egypt')
```

Pero tengan cuidado de no confundir la comilla sencilla ' con el *back quote* `.

A estas alturas ya deberían saber que si escriben:

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

recibirán un error porque las variables `italy`, `canada` y `egypt` no están definidas. Si no usamos las comillas, R busca variables con esos nombres y devuelve un error.

2.6.2 Nombres

A veces es útil nombrar las entradas de un vector. Por ejemplo, al definir un vector de códigos de países, podemos usar los nombres para conectar los dos:

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

El objeto `codes` sigue siendo un vector numérico:

```
class(codes)
#> [1] "numeric"
```

pero con nombres:

```
names(codes)
#> [1] "italy"  "canada" "egypt"
```

Si el uso de cadenas sin comillas parece confuso, sepán que también pueden usar las comillas:

```
codes <- c("italy" = 380, "canada" = 124, "egypt" = 818)
codes
#> italy canada egypt
#> 380    124    818
```

No hay diferencia entre esta llamada a una función (*function call* en inglés) y la anterior. Esta es una de las muchas formas en que R es peculiar en comparación con otros lenguajes.

También podemos asignar nombres usando la función `names`:

```
codes <- c(380, 124, 818)
country <- c("italy", "canada", "egypt")
names(codes) <- country
codes
#> italy canada egypt
#> 380    124    818
```

2.6.3 Secuencias

Otra función útil para crear vectores genera secuencias:

```
seq(1, 10)
#> [1] 1 2 3 4 5 6 7 8 9 10
```

El primer argumento define el inicio y el segundo define el final que se incluye en el vector. El valor por defecto es subir en incrementos de 1, pero un tercer argumento nos permite determinar cuánto saltar:

```
seq(1, 10, 2)
#> [1] 1 3 5 7 9
```

Si queremos enteros consecutivos, podemos usar la siguiente abreviación:

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

Cuando usamos estas funciones, R produce números enteros, no numéricos, porque generalmente se usan para indexar algo:

```
class(1:10)
#> [1] "integer"
```

Sin embargo, si creamos una secuencia que incluye no enteros, la clase cambia:

```
class(seq(1, 10, 0.5))
#> [1] "numeric"
```

2.6.4 Cómo crear un subconjunto

Usamos los corchetes para tener acceso a elementos específicos de un vector. Para el vector `codes` que definimos anteriormente, podemos tener acceso al segundo elemento usando:

```
codes[2]
#> canada
#> 124
```

Pueden obtener más de una entrada utilizando un vector de entradas múltiples como índice:

```
codes[c(1,3)]
#> italy egypt
#> 380 818
```

Las secuencias definidas anteriormente son particularmente útiles si necesitamos acceso, digamos, a los dos primeros elementos:

```
codes[1:2]
#> italy canada
#> 380 124
```

Si los elementos tienen nombres, también podemos acceder a las entradas utilizando estos nombres. A continuación ofrecemos dos ejemplos:

```
codes["canada"]
#> canada
#> 124
codes[c("egypt","italy")]
#> egypt italy
#> 818 380
```

2.7 La conversión forzada

En general, la *conversión forzada* (*coercion* en inglés) es un intento de R de ser flexible con los tipos de datos. Cuando una entrada no coincide con lo esperado, algunas de las funciones predefinidas de R tratan de adivinar lo que uno intentaba antes de devolver un mensaje de error. Esto también puede causar confusión. Al no entender la conversión forzada, los programadores pueden volverse locos cuando codifican en R, ya que R se comporta de manera bastante diferente a la mayoría de los otros idiomas en cuanto a esto. Aprendamos más con unos ejemplos.

Dijimos que los vectores deben ser todos del mismo tipo. Entonces, si tratamos de combinar, por ejemplo, números y caracteres, pueden esperar un error:

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

¡Pero no nos da un error, ni siquiera una advertencia! ¿Qué pasó? Miren x y su clase:

```
x
#> [1] "1"      "canada" "3"
class(x)
#> [1] "character"
```

R forzó una conversión de los datos a caracteres. Como pusimos una cadena de caracteres en el vector, R adivinó que nuestra intención era que el 1 y el 3 fueran las cadenas de caracteres "1" y "3". El hecho de que ni siquiera emitiera una advertencia es un ejemplo de cómo la conversión forzada puede causar muchos errores inadvertidos en R.

R también ofrece funciones para cambiar de un tipo a otro. Por ejemplo, pueden convertir números en caracteres con:

```
x <- 1:5
y <- as.character(x)
y
#> [1] "1" "2" "3" "4" "5"
```

Pueden revertir a lo anterior con `as.numeric`:

```
as.numeric(y)
#> [1] 1 2 3 4 5
```

Esta función es muy útil ya que los sets de datos que incluyen números como cadenas de caracteres son comunes.

2.7.1 *Not available* (NA)

Cuando una función intenta forzar una conversión de un tipo a otro y encuentra un caso imposible, generalmente nos da una advertencia y convierte la entrada en un valor especial llamado NA que significa *no disponible* (*Not Available* en inglés). Por ejemplo:

```
x <- c("1", "b", "3")
as.numeric(x)
#> Warning: NAs introducidos por coerción
#> [1] 1 NA 3
```

R no sabe el número que querían cuando escribieron `b` y no lo intenta adivinar.

Como científicos de datos, se encontrarán con el `NA` frecuentemente ya que se usa generalmente para datos faltantes (*missing data* en inglés), un problema común en los sets de datos del mundo real.

2.8 Ejercicios

1. Use la función `c` para crear un vector con las temperaturas altas promedio en enero para Beijing, Lagos, París, Río de Janeiro, San Juan y Toronto, que son 35, 88, 42, 84, 81 y 30 grados Fahrenheit. Llame al objeto `temp`.
2. Ahora cree un vector con los nombres de las ciudades y llame al objeto `city`.
3. Utilice la función `names` y los objetos definidos en los ejercicios anteriores para asociar los datos de temperatura con su ciudad correspondiente.
4. Utilice los operadores `[` y `:` para acceder a la temperatura de las tres primeras ciudades de la lista.
5. Utilice el operador `[` para acceder a la temperatura de París y San Juan.
6. Utilice el operador `:` para crear una secuencia de números $12, 13, 14, \dots, 73$.
7. Cree un vector que contenga todos los números impares positivos menores que 100.
8. Cree un vector de números que comience en 6, no pase 55 y agregue números en incrementos de $4/7$: $6, 6 + 4/7, 6 + 8/7$, y así sucesivamente. ¿Cuántos números tiene la lista? Sugerencia: use `seq` y `length`.
9. ¿Cuál es la clase del siguiente objeto `a <- seq(1, 10, 0.5)`?
10. ¿Cuál es la clase del siguiente objeto `a <- seq(1, 10)`?
11. La clase de `class(a<-1)` es numérica, no entero. R por defecto es numérico y para forzar un número entero, debe añadir la letra `L`. Confirme que la clase de `1L` es entero.
12. Defina el siguiente vector:

```
x <- c("1", "3", "5")
```

y oblíguelo a obtener enteros.

2.9 Sorting

Ahora que hemos dominado algunos conocimientos básicos de R, intentemos obtener algunos conocimientos sobre la seguridad de los distintos estados en el contexto de los asesinatos con armas de fuego.

2.9.1 sort

Digamos que queremos clasificar los estados desde el menor hasta el mayor según los asesinatos con armas de fuego. La función `sort` ordena un vector en orden creciente. Por lo tanto, podemos ver la mayor cantidad de asesinatos escribiendo:

```
library(dslabs)
data(murders)
sort(murders$total)
#> [1] 2 4 5 5 7 8 11 12 12 16 19 21 22
#> [14] 27 32 36 38 53 63 65 67 84 93 93 97 97
#> [27] 99 111 116 118 120 135 142 207 219 232 246 250 286
#> [40] 293 310 321 351 364 376 413 457 517 669 805 1257
```

Sin embargo, esto no nos da información sobre qué estados tienen qué total de asesinatos. Por ejemplo, no sabemos qué estado tuvo 1257.

2.9.2 order

La función `order` es mas apropiada para lo que queremos hacer. `order` toma un vector como entrada y devuelve el vector de índices que clasifica el vector de entrada. Esto puede ser un poco confuso, así que estudiemos un ejemplo sencillo. Podemos crear un vector y ordenarlo (`sort` en inglés):

```
x <- c(31, 4, 15, 92, 65)
sort(x)
#> [1] 4 15 31 65 92
```

En lugar de ordenar el vector de entrada, la función `order` devuelve el índice que ordena el vector de entrada:

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

Este es el mismo resultado que le devuelve `sort(x)`. Si miramos este índice, vemos por qué funciona:

```
x
#> [1] 31 4 15 92 65
order(x)
#> [1] 2 3 1 5 4
```

La segunda entrada de `x` es la más pequeña, entonces `order(x)` comienza con 2. La siguiente más pequeña es la tercera entrada, por lo que la segunda entrada es 3 y así sigue.

¿Cómo nos ayuda esto a ordenar los estados por asesinatos? Primero, recuerden que las entradas de vectores a las que acceden con \$ siguen el mismo orden que las filas de la tabla. Por ejemplo, estos dos vectores que contienen el nombre de los estados y sus abreviaturas, respectivamente, siguen el mismo orden:

```
murders$state[1:6]
#> [1] "Alabama"      "Alaska"       "Arizona"      "Arkansas"     "California"
#> [6] "Colorado"
murders$abb[1:6]
#> [1] "AL" "AK" "AZ" "AR" "CA" "CO"
```

Esto significa que podemos ordenar los nombres de estado según el total de asesinatos. Primero obtenemos el índice que ordena los vectores por el total de asesinatos y luego ponemos el vector de nombres de estado en un índice:

```
ind <- order(murders$total)
murders$abb[ind]
#> [1] "VT" "ND" "NH" "WY" "HI" "SD" "ME" "ID" "MT" "RI" "AK" "IA" "UT"
#> [14] "WV" "NE" "OR" "DE" "MN" "KS" "CO" "NM" "NV" "AR" "WA" "CT" "WI"
#> [27] "DC" "OK" "KY" "MA" "MS" "AL" "IN" "SC" "TN" "AZ" "NJ" "VA" "NC"
#> [40] "MD" "OH" "MO" "LA" "IL" "GA" "MI" "PA" "NY" "FL" "TX" "CA"
```

De acuerdo con lo anterior, California tuvo la mayor cantidad de asesinatos.

2.9.3 max y which.max

Si solo estamos interesados en la entrada con el mayor valor, podemos usar `max`:

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

y `which.max` para el índice del valor mayor:

```
i_max <- which.max(murders$total)
murders$state[i_max]
#> [1] "California"
```

Para el mínimo, podemos usar `min` y `which.min` del mismo modo.

¿Esto significa que California es el estado más peligroso? En una próxima sección, planteamos que deberíamos considerar las tasas en lugar de los totales. Antes de hacer eso, presentamos una última función relacionada con el orden: `rank`.

2.9.4 rank

Aunque no se usa con tanta frecuencia como `order` y `sort`, la función `rank` también está relacionada con el orden y puede ser útil. Para cualquier vector dado, `rank` devuelve un vector con el rango de la primera entrada, segunda entrada, etc., del vector de entrada. Aquí tenemos un ejemplo sencillo:

```
x <- c(31, 4, 15, 92, 65)
rank(x)
#> [1] 3 1 2 5 4
```

Para resumir, veamos los resultados de las tres funciones que hemos discutido:

original	sort	order	rank
31	4	2	3
4	15	3	1
15	31	1	2
92	65	5	5
65	92	4	4

2.9.5 Cuidado con el reciclaje

Otra fuente común de errores inadvertidos en R es el uso de *reciclaje* (*recycling* en inglés). Hemos visto como los vectores se agregan por elementos. Entonces, si los vectores no coinciden en longitud, es natural suponer que vamos a ver un error. Pero ese no es el caso. Vean lo que pasa:

```
x <- c(1, 2, 3)
y <- c(10, 20, 30, 40, 50, 60, 70)
x+y
#> Warning in x + y: longitud de objeto mayor no es múltiplo de la
#> longitud de uno menor
#> [1] 11 22 33 41 52 63 71
```

Recibimos una advertencia, pero no hay error. Para el *output*, R ha reciclado los números en `x`. Observen el último dígito de números en el *output*.

2.10 Ejercicios

Para estos ejercicios usaremos el set de datos de asesinatos de EE. UU. Asegúrense de cargarlo antes de comenzar.

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

1. Utilice el operador `$` para acceder a los datos del tamaño de la población y almacenarlos como el objeto `pop`. Luego, use la función `sort` para redefinir `pop` para que esté en orden alfabético. Finalmente, use el operador `[` para indicar el tamaño de población más pequeño.
2. Ahora, en lugar del tamaño de población más pequeño, encuentre el índice de la entrada con el tamaño de población más pequeño. Sugerencia: use `order` en lugar de `sort`.
3. Podemos realizar la misma operación que en el ejercicio anterior usando la función `which.min`. Escriba una línea de código que haga esto.
4. Ahora sabemos cuán pequeño es el estado más pequeño y qué fila lo representa. ¿Qué estado es? Defina una variable `states` para ser los nombres de los estados del *data frame* `murders`. Indique el nombre del estado con la población más pequeña.
5. Puede crear un *data frame* utilizando la función `data.frame`. Aquí un ejemplo:

```
temp <- c(35, 88, 42, 84, 81, 30)
city <- c("Beijing", "Lagos", "Paris", "Rio de Janeiro",
         "San Juan", "Toronto")
city_temps <- data.frame(name = city, temperature = temp)
```

Utilice la función `rank` para determinar el rango de población de cada estado desde el menos poblado hasta el más poblado. Guarde estos rangos en un objeto llamado `ranks`. Luego, cree un *data frame* con el nombre del estado y su rango. Nombre el *data frame* `my_df`.

6. Repita el ejercicio anterior, pero esta vez ordene `my_df` para que los estados estén en orden de menos poblado a más poblado. Sugerencia: cree un objeto `ind` que almacene los índices necesarios para poner en orden los valores de la población. Luego, use el operador de corchete `[` para reordenar cada columna en el *data frame*.
7. El vector `na_example` representa una serie de conteos. Puede examinar rápidamente el objeto usando:

```
data("na_example")
str(na_example)
#>  int [1:1000] 2 1 3 2 1 3 1 4 3 2 ...
```

Sin embargo, cuando calculamos el promedio con la función `mean`, obtenemos un `NA`:

```
mean(na_example)
#> [1] NA
```

La función `is.na` devuelve un vector lógico que nos dice qué entradas son `NA`. Asigne este vector lógico a un objeto llamado `ind` y determine cuántos `NAs` tiene `na_example`.

8. Ahora calcule nuevamente el promedio, pero solo para las entradas que no son `NA`. Sugerencia: recuerde el operador `!`.

2.11 Aritmética de vectores

California tuvo la mayor cantidad de asesinatos, pero ¿esto significa que es el estado más peligroso? ¿Qué pasa si solo tiene muchas más personas que cualquier otro estado? Podemos confirmar rápidamente que California tiene la mayor población:

```
library(dslabs)
data("murders")
murders$state[which.max(murders$population)]
#> [1] "California"
```

con más de 37 millones de habitantes. Por lo tanto, es injusto comparar los totales si estamos interesados en saber cuán seguro es el estado. Lo que realmente deberíamos calcular son los asesinatos per cápita. Los informes que describimos en la sección motivante utilizan asesinatos por cada 100,000 como la unidad. Para calcular esta cantidad, usamos las poderosas capacidades aritméticas de vectores de R.

2.11.1 *Rescaling* un vector

En R, las operaciones aritméticas en vectores ocurren elemento por elemento. Como ejemplo, supongan que tenemos la altura en pulgadas (*inches* en inglés):

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

y queremos convertirla a centímetros. Observen lo que sucede cuando multiplicamos *inches* por 2.54:

```
inches * 2.54
#> [1] 175 157 168 178 178 185 170 185 170 178
```

Arriba, multiplicamos cada elemento por 2.54. Del mismo modo, si para cada entrada queremos calcular cuántas pulgadas más alto, o cuántas más corto, que 69 pulgadas (la altura promedio para hombres), podemos restarlo de cada entrada de esta manera:

```
inches - 69
#> [1] 0 -7 -3 1 1 4 -2 4 -2 1
```

2.11.2 Dos vectores

Si tenemos dos vectores de la misma longitud y los sumamos en R, se agregarán entrada por entrada de la siguiente manera:

$$\begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} + \begin{pmatrix} e \\ f \\ g \\ h \end{pmatrix} = \begin{pmatrix} a+e \\ b+f \\ c+g \\ d+h \end{pmatrix}$$

Lo mismo aplica para otras operaciones matemáticas, como `-`, `*` y `/`.

Esto implica que para calcular las tasas de asesinatos (*murder rates* en inglés) simplemente podemos escribir:

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

Al hacer esto, notamos que California ya no está cerca de la parte superior de la lista. De hecho, podemos usar lo que hemos aprendido para poner a los estados en orden por tasa de asesinatos:

```
murders$abb[order(murder_rate)]
#> [1] "VT" "NH" "HI" "ND" "IA" "ID" "UT" "ME" "WY" "OR" "SD" "MN" "MT"
#> [14] "CO" "WA" "WV" "RI" "WI" "NE" "MA" "IN" "KS" "NY" "KY" "AK" "OH"
#> [27] "CT" "NJ" "AL" "IL" "OK" "NC" "NV" "VA" "AR" "TX" "NM" "CA" "FL"
#> [40] "TN" "PA" "AZ" "GA" "MS" "MI" "DE" "SC" "MD" "MO" "LA" "DC"
```

2.12 Ejercicios

1. Anteriormente, creamos este *data frame*:

```
temp <- c(35, 88, 42, 84, 81, 30)
city <- c("Beijing", "Lagos", "Paris", "Rio de Janeiro",
        "San Juan", "Toronto")
city_temps <- data.frame(name = city, temperature = temp)
```

Vuelva a crear el *data frame* utilizando el código anterior, pero agregue una línea que convierta la temperatura de Fahrenheit a Celsius. La conversión es $C = \frac{5}{9} \times (F - 32)$.

2. ¿Cuál es la siguiente suma $1 + 1/2^2 + 1/3^2 + \dots + 1/100^2$? Sugerencia: gracias a Euler, sabemos que debería estar cerca de $\pi^2/6$.

3. Calcule la tasa de asesinatos por cada 100,000 para cada estado y almacénela en el objeto `murder_rate`. Luego, calcule la tasa promedio de asesinatos para EE. UU. con la función `mean`. ¿Cuál es el promedio?

2.13 Indexación

R provee una forma poderosa y conveniente de indexar vectores. Podemos, por ejemplo, crear un subconjunto de un vector según las propiedades de otro vector. En esta sección, continuamos trabajando con nuestro ejemplo de asesinatos en EE. UU., que podemos cargar así:

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

2.13.1 Cómo crear subconjuntos con lógicos

Ahora hemos calculado la tasa de asesinatos usando:

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

Imaginen que se mudan de Italia donde, según un informe de noticias, la tasa de asesinatos es solo 0.71 por 100,000. Preferirían mudarse a un estado con una tasa de asesinatos similar. Otra característica poderosa de R es que podemos usar lógicas para indexar vectores. Si comparamos un vector con un solo número, R realiza la prueba para cada entrada. Aquí tenemos un ejemplo relacionado con la pregunta anterior:

```
ind <- murder_rate < 0.71
```

Si en cambio queremos saber si un valor es menor o igual, podemos usar:

```
ind <- murder_rate <= 0.71
```

Recuerden que devuelve un vector lógico con TRUE para cada entrada menor o igual a 0.71. Para ver qué estados son estos, podemos aprovechar el hecho de que los vectores se pueden indexar con lógicos.

```
murders$state[ind]
#> [1] "Hawaii"           "Iowa"              "New Hampshire" "North Dakota"
#> [5] "Vermont"
```

Para contar cuántos son *TRUE*, la función `sum` devuelve la suma de las entradas de un vector y fuerza una conversión de los vectores lógicos a numéricos con *TRUE* codificado como 1 y *FALSE* como 0. Así podemos contar los estados usando:

```
sum(ind)
#> [1] 5
```

2.13.2 Operadores lógicos

Supongan que nos gustan las montañas y queremos mudarnos a un estado seguro en la región occidental del país. Queremos que la tasa de asesinatos sea como máximo 1. En este caso, queremos que dos cosas diferentes sean ciertas. Aquí podemos usar el operador lógico *and*, que en R se representa con `&`. Esta operación da como resultado *TRUE* solo cuando ambos lógicos son *TRUE*, es decir ciertos. Para ver esto, consideren este ejemplo:

```
TRUE & TRUE
#> [1] TRUE
TRUE & FALSE
#> [1] FALSE
FALSE & FALSE
#> [1] FALSE
```

Para nuestro ejemplo, podemos formar dos lógicos:

```
west <- murders$region == "West"
safe <- murder_rate <= 1
```

y podemos usar `&` para obtener un vector lógico que nos dice qué estados satisfacen ambas condiciones:

```
ind <- safe & west
murders$state[ind]
#> [1] "Hawaii"   "Idaho"    "Oregon"   "Utah"     "Wyoming"
```

2.13.3 which

Supongan que queremos ver la tasa de asesinatos de California. Para este tipo de operación, es conveniente convertir vectores lógicos en índices en lugar de mantener vectores lógicos largos. La función `which` nos dice qué entradas de un vector lógico son TRUE. Entonces podemos escribir:

```
ind <- which(murders$state == "California")
murder_rate[ind]
#> [1] 3.37
```

2.13.4 match

Si en lugar de un solo estado queremos averiguar las tasas de asesinatos de varios estados, digamos Nueva York, Florida y Texas, podemos usar la función `match`. Esta función nos dice qué índices de un segundo vector coinciden con cada una de las entradas de un primer vector:

```
ind <- match(c("New York", "Florida", "Texas"), murders$state)
ind
#> [1] 33 10 44
```

Ahora podemos ver las tasas de asesinatos:

```
murder_rate[ind]
#> [1] 2.67 3.40 3.20
```

2.13.5 %in%

Si en lugar de un índice queremos un lógico que nos diga si cada elemento de un primer vector está en un segundo vector, podemos usar la función `%in%`. Imaginen que no están seguros si Boston, Dakota y Washington son estados. Pueden averiguar así:

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

Tengan en cuenta que estaremos usando `%in%` frecuentemente a lo largo del libro.

Avanzado: Hay una conexión entre `match` y `%in%` mediante `which`. Para ver esto, observen que las siguientes dos líneas producen el mismo índice (aunque en orden diferente):

```
match(c("New York", "Florida", "Texas"), murders$state)
#> [1] 33 10 44
which(murders$state %in% c("New York", "Florida", "Texas"))
#> [1] 10 33 44
```

2.14 Ejercicios

Empiece cargando el paquete y los datos.

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

1. Calcule la tasa de asesinatos por cada 100,000 para cada estado y almacénela en un objeto llamado `murder_rate`. Luego, use operadores lógicos para crear un vector lógico llamado `low` que nos dice qué entradas de `murder_rate` son inferiores a 1.
2. Ahora use los resultados del ejercicio anterior y la función `which` para determinar los índices de `murder_rate` asociados con valores inferiores a 1.
3. Use los resultados del ejercicio anterior para indicar los nombres de los estados con tasas de asesinatos inferiores a 1.
4. Ahora extienda el código de los ejercicios 2 y 3 para indicar los estados del noreste con tasas de asesinatos inferiores a 1. Sugerencia: use el vector lógico predefinido `low` y el operador lógico `&`.
5. En un ejercicio anterior, calculamos la tasa de asesinatos para cada estado y el promedio de estos números. ¿Cuántos estados están por debajo del promedio?
6. Use la función `match` para identificar los estados con abreviaturas AK, MI e IA. Sugerencia: comience definiendo un índice de las entradas de `murders$abb` que coinciden con las tres abreviaturas. Entonces use el operador `[` para extraer los estados.
7. Utilice el operador `%in%` para crear un vector lógico que responda a la pregunta: ¿cuáles de las siguientes son abreviaturas reales: MA, ME, MI, MO, MU?

8. Extienda el código que usó en el ejercicio 7 para averiguar la única entrada que **no** es una abreviatura real. Sugerencia: use el operador **!**, que convierte **FALSE** a **TRUE** y viceversa, y entonces **which** para obtener un índice.

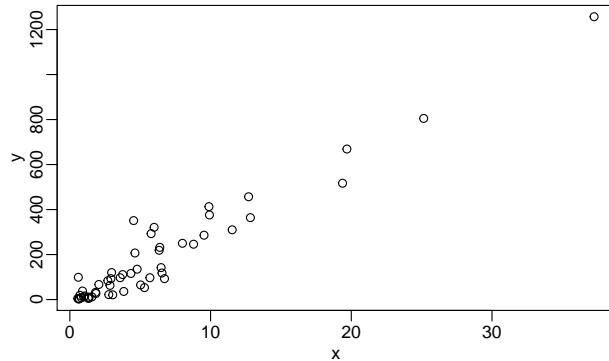
2.15 Gráficos básicos

En el capítulo 7, describimos un paquete complementario que ofrece un enfoque muy útil para producir gráficos (*plots* en inglés) en R. Luego tenemos una parte entera, “Visualización de datos”, en la que ofrecemos muchos ejemplos. Aquí describimos brevemente algunas de las funciones disponibles en una instalación básica de R.

2.15.1 plot

La función **plot** se puede utilizar para hacer diagramas de dispersión (*scatterplots* en inglés). Abajo tenemos un gráfico de total de asesinatos versus población.

```
x <- murders$population/ 10^6  
y <- murders$total  
plot(x, y)
```



Para crear un gráfico rápido que no accede a las variables dos veces, podemos usar la función **with**:

```
with(murders, plot(population, total))
```

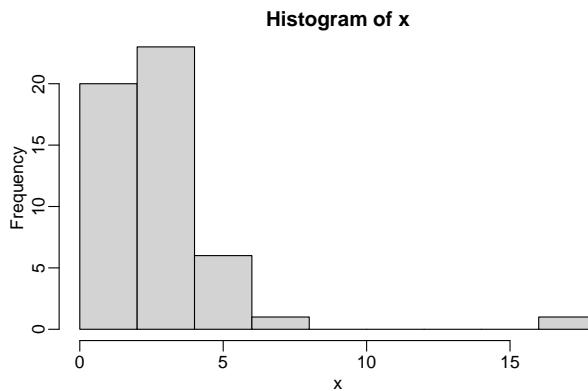
La función **with** nos permite usar los nombres de la columna **murders** en la función **plot**. También funciona con cualquier *data frame* y cualquier función.

2.15.2 hist

Describiremos los histogramas en relación con las distribuciones en la parte del libro “Visualización de datos”. Aquí simplemente notaremos que los histogramas son un resumen

gráfico eficaz de una lista de números que nos ofrece una descripción general de los tipos de valores que tenemos. Podemos hacer un histograma de nuestras tasas de asesinatos al simplemente escribir:

```
x <- with(murders, total/ population * 100000)
hist(x)
```



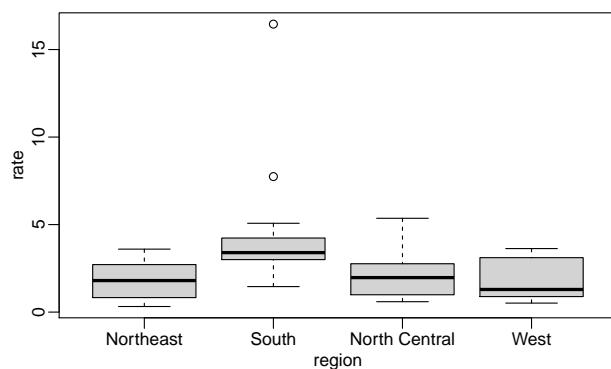
Podemos ver que hay una amplia gama de valores con la mayoría de ellos entre 2 y 3 y un caso muy extremo con una tasa de asesinatos de más de 15:

```
murders$state[which.max(x)]
#> [1] "District of Columbia"
```

2.15.3 boxplot

Los diagramas de caja (*boxplots* en inglés) también se describirán en la parte del libro “Visualización de datos”. Estos proveen un resumen más conciso que los histogramas, pero son más fáciles de apilar con otros diagramas de caja. Por ejemplo, aquí podemos usarlos para comparar las diferentes regiones:

```
murders$rate <- with(murders, total/ population * 100000)
boxplot(rate~region, data = murders)
```

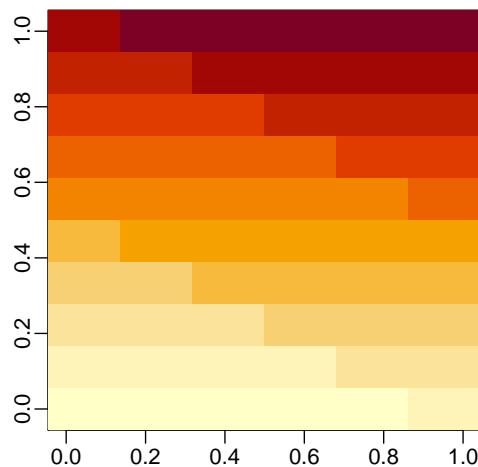


Podemos ver que el Sur (*South* en inglés) tiene tasas de asesinatos más altas que las otras tres regiones.

2.15.4 `image`

La función `image` muestra los valores en una matriz usando color. Aquí mostramos un ejemplo rápido:

```
x <- matrix(1:120, 12, 10)
image(x)
```



2.16 Ejercicios

1. Hicimos un gráfico de asesinatos totales versus población y notamos una fuerte relación. No es sorprendente que los estados con poblaciones más grandes hayan tenido más asesinatos.

```
library(dslabs)
data(murders)
population_in_millions <- murders$population/10^6
total_gun_murders <- murders$total
plot(population_in_millions, total_gun_murders)
```

Recuerden que muchos estados tienen poblaciones inferiores a 5 millones y están agrupados. Podemos obtener más información al hacer este gráfico en la escala logarítmica. Transforme las variables usando la transformación `log10` y luego cree un gráfico de los resultados.

2. Cree un histograma de las poblaciones estatales.
3. Genere diagramas de caja de las poblaciones estatales por región.

3

Conceptos básicos de programación

3.1 Expresiones condicionales

Las expresiones condicionales son una de las características básicas de la programación. Se utilizan para lo que se denomina *flow control*. La expresión condicional más común es la declaración *if-else*. En R, podemos realizar mucho análisis de datos sin condicionales. Sin embargo, aparecen ocasionalmente y los necesitarán una vez comiencen a escribir sus propias funciones y paquetes.

Aquí presentamos un ejemplo muy sencillo que muestra la estructura general de una instrucción *if-else*. La idea básica es imprimir el recíproco de *a* a menos que *a* sea 0:

```
a <- 0

if(a!=0){
  print(1/a)
} else{
  print("No reciprocal for 0.")
}
#> [1] "No reciprocal for 0."
```

Veamos otro ejemplo usando el set de datos de asesinatos de EE. UU.:

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

Aquí ofrecemos un ejemplo muy sencillo que nos dice qué estados, si los hay, tienen una tasa de homicidios inferior a 0.5 por 100,000. Las declaraciones *if* nos protegen del caso en el que ningún estado satisface la condición.

```
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"
```

Si lo intentamos nuevamente con una tasa de 0.25, obtenemos una respuesta diferente:

```
if(murder_rate[ind] < 0.25){
  print(murders$state[ind])
} else{
  print("No state has a murder rate that low.")
}
#> [1] "No state has a murder rate that low."
```

Una función relacionada que es muy útil es `ifelse`. Esta función toma tres argumentos: un lógico y dos posibles respuestas. Si el lógico es TRUE, devuelve el valor en el segundo argumento y, si es FALSE, devuelve el valor en el tercer argumento. Aquí tenemos un ejemplo:

```
a <- 0
ifelse(a > 0, 1/a, NA)
#> [1] NA
```

Esta función es particularmente útil porque sirve para vectores. Examina cada entrada del vector lógico y devuelve elementos del vector proporcionado en el segundo argumento, si la entrada es TRUE, o elementos del vector proporcionado en el tercer argumento, si la entrada es FALSE.

```
a <- c(0, 1, 2, -4, 5)
result <- ifelse(a > 0, 1/a, NA)
```

Esta tabla nos ayuda a ver qué sucedió:

a	is_a_positive	answer1	answer2	result
0	FALSE	Inf	NA	NA
1	TRUE	1.00	NA	1.0
2	TRUE	0.50	NA	0.5
-4	FALSE	-0.25	NA	NA
5	TRUE	0.20	NA	0.2

Aquí hay un ejemplo de cómo esta función se puede usar fácilmente para reemplazar todos los valores faltantes en un vector con ceros:

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

Otras dos funciones útiles son `any` y `all`. La función `any` toma un vector de lógicos y devuelve TRUE si alguna de las entradas es TRUE. La función `all` toma un vector de lógicos y devuelve TRUE si todas las entradas son TRUE. Aquí ofrecemos un ejemplo:

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

3.2 Cómo definir funciones

A medida que adquieran más experiencia, necesitarán realizar las mismas operaciones una y otra vez. Un ejemplo sencillo es el cálculo de promedios. Podemos calcular el promedio de un vector `x` utilizando las funciones `sum` y `length`: `sum(x)/length(x)`. Debido a que hacemos esto repetidas veces, es mucho más eficiente escribir una función que realice esta operación. Esta operación particular es tan común que alguien ya escribió la función `mean` y se incluye en la base R. Sin embargo, se encontrarán con situaciones en las que la función aún no existe, por lo que R les permite escribir una. Se puede definir una versión sencilla de una función que calcula el promedio así:

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

Ahora `avg` es una función que calcula el promedio:

```
x <- 1:100  
identical(mean(x), avg(x))  
#> [1] TRUE
```

Observen que las variables definidas dentro de una función no se guardan en el espacio de trabajo. Por lo tanto, mientras usamos `s` y `n` cuando llamamos (*call* en inglés) `avg`, los valores se crean y cambian solo durante la llamada. Aquí podemos ver un ejemplo ilustrativo:

```
s <- 3  
avg(1:10)  
#> [1] 5.5  
s  
#> [1] 3
```

Noten como `s` todavía es 3 después de que llamamos `avg`.

En general, las funciones son objetos, por lo que les asignamos nombres de variables con `<-`. La función `function` le dice a R que están a punto de definir una función. La forma general de la definición de una función es así:

```
my_function <- function(VARIABLE_NAME){  
  perform operations on VARIABLE_NAME and calculate VALUE  
  VALUE  
}
```

Las funciones que definen pueden tener múltiples argumentos, así como valores predeterminados. Por ejemplo, podemos definir una función que calcule el promedio aritmético o geométrico dependiendo de una variable definida por usuarios como esta:

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

Aprenderemos más sobre cómo crear funciones a través de la experiencia a medida que nos enfrentemos a tareas más complejas.

3.3 Namespaces

Una vez que comiencen a convertirse en usuarios expertos de R, es probable que necesiten cargar varios complementos de paquetes (*add-ons* en inglés) para algunos de sus análisis. Tan pronto hagan eso, es probable que descubran que dos paquetes a veces usan el mismo nombre para dos funciones diferentes. Y a menudo estas funciones hacen cosas completamente diferentes. De hecho, ya hemos visto esto porque ambos paquetes de base R **dplyr** y **stats** definen una función **filter**. Hay otros cinco ejemplos en **dplyr**. Sabemos esto porque cuando cargamos **dplyr** por primera vez, vemos el siguiente mensaje:

The following objects are masked from ‘package:stats’:

filter, **lag**

The following objects are masked from ‘package:base’:

intersect, **setdiff**, **setequal**, **union**

Entonces, ¿qué hace R cuando escribimos **filter**? ¿Utiliza la función **dplyr** o la función **stats**? De nuestro trabajo anterior sabemos que usa **dplyr**. Pero, ¿qué pasa si queremos usar **stats**?

Estas funciones viven en diferentes *namespaces*. R seguirá un cierto orden cuando busque una función en estos *namespaces*. Pueden ver el orden escribiendo:

```
search()
```

La primera entrada en esta lista es el ambiente global que incluye todos los objetos que definen.

Entonces, ¿qué pasa si queremos usar el **filter stats** en lugar del **filter dplyr** pero **dplyr** aparece primero en la lista de búsqueda? Pueden forzar el uso de un *namespace* específico utilizando dos puntos dobles (::) así:

```
stats::filter
```

Si queremos estar absolutamente seguros de que usamos el **filter** de **dplyr**, podemos usar:

```
dplyr::filter
```

Recuerden que si queremos usar una función en un paquete sin cargar el paquete completo, también podemos usar los dos puntos dobles.

Para más información sobre este tema más avanzado, recomendamos el libro de paquetes R¹.

3.4 Bucles-for

La fórmula para la suma de la serie $1 + 2 + \dots + n$ es $n(n + 1)/2$. ¿Qué pasaría si no estuviéramos seguros de que esa era la función correcta? ¿Cómo podríamos verificar? Usando lo que aprendimos sobre las funciones, podemos crear una que calcule S_n :

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

¿Cómo podemos calcular S_n para varios valores de n , digamos $n = 1, \dots, 25$? ¿Escribimos 25 líneas de código llamando `compute_s_n`? No. Para eso están los bucles-for (*for-loops* en inglés) en la programación. En este caso, estamos realizando exactamente la misma tarea una y otra vez, y lo único que está cambiando es el valor de n . Los bucles-for nos permiten definir el rango que toma nuestra variable (en nuestro ejemplo $n = 1, \dots, 10$), luego cambiar el valor y evaluar la expresión a medida que realice un *bucle*.

Quizás el ejemplo más sencillo de un bucle-for es este código inútil:

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

Aquí está el bucle-for que escribiríamos para nuestro ejemplo S_n :

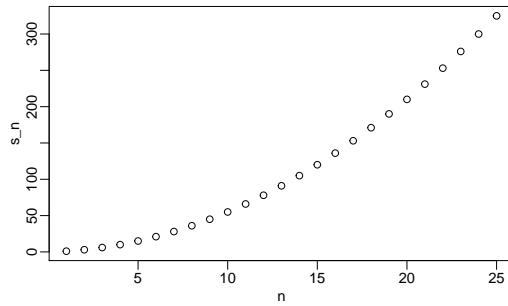
```
m <- 25
s_n <- vector(length = m) # create an empty vector
for(n in 1:m){
  s_n[n] <- compute_s_n(n)
}
```

¹<http://r-pkgs.had.co.nz/namespace.html>

En cada iteración $n = 1, n = 2$, etc ..., calculamos S_n y lo guardamos en la entrada n de `s_n`.

Ahora podemos crear un gráfico para buscar un patrón:

```
n <- 1:m
plot(n, s_n)
```



Si notaron que parece ser cuadrático, van por buen camino porque la fórmula es $n(n + 1)/2$.

3.5 Vectorización y funcionales

Aunque los bucles-for son un concepto importante para entender, no se usan mucho en R. A medida que aprendan más R, se darán cuenta de que la *vectorización* es preferible a los bucles-for puesto que resulta en un código más corto y claro. Ya vimos ejemplos en la sección de aritmética de vectores. Una función *vectorizada* es una función que aplicará la misma operación en cada uno de los vectores.

```
x <- 1:10
sqrt(x)
#> [1] 1.00 1.41 1.73 2.00 2.24 2.45 2.65 2.83 3.00 3.16
y <- 1:10
x*y
#> [1] 1 4 9 16 25 36 49 64 81 100
```

Para hacer este cálculo, no necesitamos los bucles-for. Sin embargo, no todas las funciones funcionan de esta manera. Por ejemplo, la función que acabamos de escribir, `compute_s_n`, no funciona elemento por elemento ya que espera un escalar. Este fragmento de código no ejecuta la función en cada entrada de `n`:

```
n <- 1:25
compute_s_n(n)
```

Los *funcionales* son funciones que nos ayudan a aplicar la misma función a cada entrada

en un vector, matriz, *data frame* o lista. Aquí cubrimos el funcional que opera en vectores numéricos, lógicos y de caracteres: **sapply**.

La función **sapply** nos permite realizar operaciones basadas en elementos (*element-wise* en inglés) en cualquier función. Aquí podemos ver como funciona:

```
x <- 1:10
sapply(x, sqrt)
#> [1] 1.00 1.41 1.73 2.00 2.24 2.45 2.65 2.83 3.00 3.16
```

Cada elemento de **x** se pasa a la función **sqrt** y devuelve el resultado. Estos resultados se concatenan. En este caso, el resultado es un vector de la misma longitud que el original, **x**. Esto implica que el bucle-for anterior puede escribirse de la siguiente manera:

```
n <- 1:25
s_n <- sapply(n, compute_s_n)
```

Otros funcionales son **apply**, **lapply**, **tapply**, **mapply**, **vapply** y **replicate**. Usamos principalmente **sapply**, **apply** y **replicate** en este libro, pero recomendamos familiarizarse con los demás ya que pueden ser muy útiles.

3.6 Ejercicios

1. ¿Qué devolverá esta expresión condicional?

```
x <- c(1,2,-3,4)

if(all(x>0)){
  print("All Postives")
} else{
  print("Not all positives")
}
```

2. ¿Cuál de las siguientes expresiones es siempre FALSE cuando al menos una entrada de un vector lógico **x** es TRUE?

- a. **all(x)**
- b. **any(x)**
- c. **any(!x)**
- d. **all(!x)**

3. La función **nchar** le dice cuántos caracteres tiene un vector de caracteres. Escriba una línea de código que le asigne al objeto **new_names** la abreviatura del estado cuando el nombre del estado tiene más de 8 caracteres.

4. Cree una función **sum_n** que para cualquier valor dado, digamos **n**, calcula la suma de los

enteros de 1 a n (inclusivo). Use la función para determinar la suma de los enteros de 1 a 5,000.

5. Cree una función `altman_plot` que toma dos argumentos, `x` y `y`, y grafica la diferencia contra la suma.

6. Después de ejecutar el siguiente código, ¿cuál es el valor de `x`?

```
x <- 3
my_func <- function(y){
  x <- 5
  y+5
}
```

7. Escriba una función `compute_s_n` que para cualquier n calcula la suma $S_n = 1^2 + 2^2 + 3^2 + \dots + n^2$. Indique el valor de la suma cuando $n = 10$.

8. Defina un vector numérico vacío `s_n` de tamaño 25 usando `s_n <- vector("numeric", 25)` y almacene los resultados de S_1, S_2, \dots, S_{25} usando un bucle-for.

9. Repita el ejercicio 8, pero esta vez use `sapply`.

10. Repita el ejercicio 8, pero esta vez use `map dbl`.

11. Grafique S_n versus n . Use puntos definidos por $n = 1, \dots, 25$.

12. Confirme que la fórmula para esta suma es $S_n = n(n + 1)(2n + 1)/6$.

4

tidyverse

Hasta ahora hemos estado manipulando vectores reordenándolos y creando subconjuntos mediante la indexación. Sin embargo, una vez comencemos los análisis más avanzados, la unidad preferida para el almacenamiento de datos no es el vector sino el *data frame*. En este capítulo aprenderemos a trabajar directamente con *data frames*, que facilitan enormemente la organización de información. Utilizaremos *data frames* para la mayoría de este libro. Nos enfocaremos en un formato de datos específico denominado *tidy* y en una colección específica de paquetes que son particularmente útiles para trabajar con datos *tidy* y que se denomina el *tidyverse*.

Podemos cargar todos los paquetes del *tidyverse* a la vez al instalar y cargar el paquete **tidyverse**:

```
library(tidyverse)
```

Aprenderemos cómo implementar el enfoque *tidyverse* a lo largo del libro, pero antes de profundizar en los detalles, en este capítulo presentamos algunos de los aspectos más utilizadas del *tidyverse*, comenzando con el paquete **dplyr** para manipular los *data frames* y el paquete **purrr** para trabajar con las funciones. Tengan en cuenta que el *tidyverse* también incluye un paquete para graficar, **ggplot2**, que presentaremos más adelante en el Capítulo 7 en la parte de visualización de datos del libro, el paquete **readr** discutido en el Capítulo 5 y muchos otros. En este capítulo, primero presentamos el concepto de datos *tidy* y luego demostramos cómo usamos el *tidyverse* para trabajar con *data frames* en este formato.

4.1 Datos *tidy*

Decimos que una tabla de datos está en formato *tidy* si cada fila representa una observación y las columnas representan las diferentes variables disponibles para cada una de estas observaciones. El set de datos **murders** es un ejemplo de un *data frame* *tidy*.

```
#>      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
```

Cada fila representa un estado con cada una de las cinco columnas proveyendo una variable

diferente relacionada con estos estados: nombre, abreviatura, región, población y total de asesinatos.

Para ver cómo se puede proveer la misma información en diferentes formatos, consideren el siguiente ejemplo:

```
#>      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
```

Este set de datos *tidy* ofrece tasas de fertilidad para dos países a lo largo de los años. Se considera un set de datos *tidy* porque cada fila presenta una observación con las tres variables: país, año y tasa de fecundidad. Sin embargo, este set de datos originalmente vino en otro formato y le cambiamos la forma para distribuir a través del paquete **dslabs**. Originalmente, los datos estaban en el siguiente formato:

```
#>      country 1960 1961 1962
#> 1    Germany 2.41 2.44 2.47
#> 2 South Korea 6.16 5.99 5.79
```

Se provee la misma información, pero hay dos diferencias importantes en el formato: 1) cada fila incluye varias observaciones y 2) una de las variables, año, se almacena en el encabezado. Para que los paquetes del *tidyverse* se utilicen de manera óptima, le tenemos que cambiar la forma a los datos para que estén en formato *tidy*, que aprenderán a hacer en la sección “*Wrangling* de datos” del libro. Hasta entonces, utilizaremos ejemplos de sets de datos que ya están en formato *tidy*.

Aunque no es inmediatamente obvio, a medida que avancen en el libro comenzarán a apreciar las ventajas de trabajar usando un acercamiento en el que las funciones usan formatos *tidy* tanto para *inputs* como para *outputs*. Verán cómo esto permite que los analistas de datos se enfoquen en los aspectos más importantes del análisis en lugar del formato de los datos.

4.2 Ejercicios

1. Examine el set de datos `co2` incluidos en base R. ¿Cuál de los siguientes es cierto?

- a. `co2` son datos *tidy*: tiene un año para cada fila.
- b. `co2` no es *tidy*: necesitamos al menos una columna con un vector de caracteres.
- c. `co2` no es *tidy*: es una matriz en lugar de un *data frame*.
- d. `co2` no es *tidy*: para ser *tidy* tendríamos que cambiarle la forma (*wrangle it* en inglés) para tener tres columnas (año, mes y valor), y entonces cada observación de CO2 tendría una fila.

2. Examine el set de datos `ChickWeight` incluidos en base R. ¿Cuál de los siguientes es cierto?

- a. `ChickWeight` no es *tidy*: cada pollito tiene más de una fila.
- b. `ChickWeight` es *tidy*: cada observación (un peso) está representada por una fila.
El pollito de donde provino esta medida es una de las variables.
- c. `ChickWeight` no es *tidy*: nos falta la columna del año.
- d. `ChickWeight` es *tidy*: se almacena en un *data frame*.

3. Examine el set de datos predefinido `BOD`. ¿Cuál de los siguientes es cierto?

- a. `BOD` no es *tidy*: solo tiene seis filas.
- b. `BOD` no es *tidy*: la primera columna es solo un índice.
- c. `BOD` es *tidy*: cada fila es una observación con dos valores (tiempo y demanda)
- d. `BOD` es *tidy*: todos los sets de datos pequeños son *tidy* por definición.

4. ¿Cuál de los siguientes sets de datos integrados es *tidy*? Puede elegir más de uno.

- a. `BJsales`
- b. `EuStockMarkets`
- c. `DNase`
- d. `Formaldehyde`
- e. `Orange`
- f. `UCBAdmissions`

4.3 Cómo manipular los *data frames*

El paquete `dplyr` del *tidyverse* ofrece funciones que realizan algunas de las operaciones más comunes cuando se trabaja con *data frames* y usa nombres para estas funciones que son relativamente fáciles de recordar. Por ejemplo, para cambiar la tabla de datos agregando una nueva columna, utilizamos `mutate`. Para filtrar la tabla de datos a un subconjunto de filas, utilizamos `filter`. Finalmente, para subdividir los datos seleccionando columnas específicas, usamos `select`.

4.3.1 Cómo añadir una columna con `mutate`

Queremos que toda la información necesaria para nuestro análisis se incluya en la tabla de datos. Entonces, la primera tarea es añadir las tasas de asesinatos a nuestro *data frame* de asesinatos. La función `mutate` toma el *data frame* como primer argumento y el nombre y los valores de la variable como segundo argumento usando la convención `name = values`. Entonces, para añadir tasas de asesinatos, usamos:

```
library(dslabs)
data("murders")
murders <- mutate(murders, rate = total/ population * 100000)
```

Recuerden que aquí usamos `total` y `population` dentro de la función, que son objetos **no** definidos en nuestro espacio de trabajo. Pero, ¿por qué no recibimos un error?

Esta es una de las principales características de `dplyr`. Las funciones en este paquete, como `mutate`, saben buscar variables en el *data frame* que el primer argumento les provee. En la llamada a `mutate` que vemos arriba, `total` tendrá los valores de `murders$total`. Este enfoque hace que el código sea mucho más legible.

Podemos ver que se agrega la nueva columna:

```
head(murders)
#> #>   state abb region population total rate
#> 1  Alabama AL  South    4779736  135 2.82
#> 2  Alaska AK  West     710231   19 2.68
#> 3 Arizona AZ  West    6392017  232 3.63
#> 4 Arkansas AR  South   2915918   93 3.19
#> 5 California CA  West  37253956 1257 3.37
#> 6 Colorado CO  West   5029196   65 1.29
```

Aunque hemos sobreescrito el objeto original `murders`, esto no cambia el objeto que se cargó con `data(murders)`. Si cargamos los datos `murders` nuevamente, el original sobreescribirá nuestra versión mutada.

4.3.2 Cómo crear subconjuntos con `filter`

Ahora supongan que queremos filtrar la tabla de datos para mostrar solo las entradas para las cuales la tasa de asesinatos es inferior a 0.71. Para hacer esto, usamos la función `filter`, que toma la tabla de datos como primer argumento y luego la declaración condicional como el segundo. Igual que con `mutate`, podemos usar los nombres de variables sin comillas de `murders` dentro de la función y esta sabrá que nos referimos a las columnas y no a los objetos en el espacio de trabajo.

```
filter(murders, rate <= 0.71)
#> #>   state abb      region population total rate
#> 1  Hawaii HI  West    1360301    7 0.515
#> 2  Iowa IA North Central  3046355   21 0.689
#> 3 New Hampshire NH Northeast 1316470    5 0.380
#> 4 North Dakota ND North Central 672591    4 0.595
#> 5 Vermont VT Northeast  625741    2 0.320
```

4.3.3 Cómo seleccionar columnas con `select`

Aunque nuestra tabla de datos solo tiene seis columnas, algunas tablas de datos incluyen cientos. Si queremos ver solo algunas columnas, podemos usar la función `select` de `dplyr`. En el siguiente código, seleccionamos tres columnas, asignamos el resultado a un nuevo objeto y luego filtramos este nuevo objeto:

```
new_table <- select(murders, state, region, rate)
filter(new_table, rate <= 0.71)
```

```
#>      state      region    rate
#> 1      Hawaii      West 0.515
#> 2      Iowa North Central 0.689
#> 3 New Hampshire      Northeast 0.380
#> 4 North Dakota North Central 0.595
#> 5 Vermont      Northeast 0.320
```

En la llamada a `select`, el primer argumento `murders` es un objeto, pero `state`, `region` y `rate` son nombres de variables.

4.4 Ejercicios

1. Cargue el paquete `dplyr` y el set de datos de asesinatos de EE.UU.

```
library(dplyr)
library(dslabs)
data(murders)
```

Puede añadir columnas usando la función `mutate` de `dplyr`. Esta función reconoce los nombres de la columnas y dentro de la función puede llamarlos sin comillas:

```
murders <- mutate(murders, population_in_millions = population/ 10^6)
```

Podemos escribir `population` en vez de `murders$population`. La función `mutate` sabe que estamos agarrando columnas de `murders`.

Use la función `mutate` para añadir una columna de asesinatos llamada `rate` con la tasa de asesinatos por 100,000 como en el código del ejemplo anterior. Asegúrese de redefinir `murders` como se hizo en el código del ejemplo anterior (`murders <- [su código]`) para que podamos seguir usando esta variable.

2. Si `rank(x)` le da el rango de las entradas de `x` de menor a mayor, `rank(-x)` le da los rangos de mayor a menor. Use la función `mutate` para añadir una columna `rank` que contiene el rango de la tasa de asesinatos de mayor a menor. Asegúrese de redefinir `murders` para poder seguir usando esta variable.

3. Con `dplyr`, podemos usar `select` para mostrar solo ciertas columnas. Por ejemplo, con este código solo mostrariamos los estados y los tamaños de población:

```
select(murders, state, population) %>% head()
```

Utilice `select` para mostrar los nombres de los estados y las abreviaturas en `murders`. No redefina `murders`, solo muestre los resultados.

4. La función `filter` de `dplyr` se utiliza para elegir filas específicas del *data frame* para guardar. A diferencia de `select` que es para columnas, `filter` es para filas. Por ejemplo, puede mostrar solo la fila de Nueva York así:

```
filter(murders, state == "New York")
```

Puede usar otros vectores lógicos para filtrar filas.

Utilice `filter` para mostrar los cinco estados con las tasas de asesinatos más altas. Después de añadir la tasa y el rango de asesinatos, no cambie el set de datos de asesinatos de EE. UU., solo muestre el resultado. Recuerde que puede filtrar basándose en la columna `rank`.

5. Podemos eliminar filas usando el operador `!=`. Por ejemplo, para eliminar Florida, haríamos esto:

```
no_florida <- filter(murders, state != "Florida")
```

Cree un nuevo *data frame* con el nombre `no_south` que elimina los estados del sur. ¿Cuántos estados hay en esta categoría? Puede usar la función `nrow` para esto.

6. También podemos usar `%in%` para filtrar con `dplyr`. Por lo tanto, puede ver los datos de Nueva York y Texas de esta manera:

```
filter(murders, state %in% c("New York", "Texas"))
```

Cree un nuevo *data frame* llamado `murders_nw` con solo los estados del noreste y oeste. ¿Cuántos estados hay en esta categoría?

7. Suponga que desea vivir en el noreste u oeste y desea que la tasa de asesinatos sea inferior a 1. Queremos ver los datos de los estados que satisfacen estas opciones. Tenga en cuenta que puede usar operadores lógicos con `filter`. Aquí hay un ejemplo en el que filtramos para mantener solo estados pequeños en la región noreste.

```
filter(murders, population < 5000000 & region == "Northeast")
```

Asegúrese que `murders` ha sido definido con `rate` y `rank` y todavía tiene todos los estados. Cree una tabla llamada `my_states` que contiene filas para los estados que satisfacen ambas condiciones: está localizado en el noreste u oeste y la tasa de asesinatos es inferior a 1. Use `select` para mostrar solo el nombre del estado, la tasa y el rango.

4.5 El *pipe*: `%>%`

Con `dplyr`, podemos realizar una serie de operaciones, por ejemplo `select` y entonces `filter`, enviando los resultados de una función a otra usando lo que se llama el *pipe operator*: `%>%`. Algunos detalles se incluyen a continuación.

Escribimos el código anterior para mostrar tres variables (estado, región, tasa) para los estados que tienen tasas de asesinatos por debajo de 0.71. Para hacer esto, definimos el objeto intermedio `new_table`. En `dplyr`, podemos escribir código que se parece más a una descripción de lo que queremos hacer sin objetos intermedios:

original data → select → filter

Para tal operación, podemos usar el *pipe* `%>%`. El código se ve así:

```
murders %>% select(state, region, rate) %>% filter(rate <= 0.71)
#>           state      region    rate
#> 1        Hawaii       West  0.515
#> 2        Iowa North Central 0.689
#> 3 New Hampshire     Northeast 0.380
#> 4 North Dakota North Central 0.595
#> 5   Vermont     Northeast 0.320
```

Esta línea de código es equivalente a las dos líneas de código anteriores. ¿Qué está pasando aquí?

En general, el *pipe* envía el resultado que se encuentra en el lado izquierdo del *pipe* para ser el primer argumento de la función en el lado derecho del *pipe*. Aquí vemos un ejemplo sencillo:

```
16 %>% sqrt()
#> [1] 4
```

Podemos continuar canalizando (*piping* en inglés) valores a lo largo de:

```
16 %>% sqrt() %>% log2()
#> [1] 2
```

La declaración anterior es equivalente a `log2(sqrt(16))`.

Recuerden que el *pipe* envía valores al primer argumento, por lo que podemos definir otros argumentos como si el primer argumento ya estuviera definido:

```
16 %>% sqrt() %>% log(base = 2)
#> [1] 2
```

Por lo tanto, al usar el *pipe* con *data frames* y **dplyr**, ya no necesitamos especificar el primer argumento requerido puesto que las funciones **dplyr** que hemos descrito toman todos los datos como el primer argumento. En el código que escribimos:

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

`murders` es el primer argumento de la función `select` y el nuevo *data frame* (anteriormente `new_table`) es el primer argumento de la función `filter`.

Tengan en cuenta que el *pipe* funciona bien con las funciones donde el primer argumento son los datos de entrada. Las funciones en los paquetes **tidyverse** y **dplyr** tienen este formato y se pueden usar fácilmente con el *pipe*.

4.6 Ejercicios

1. El *pipe* `%>%` se puede usar para realizar operaciones secuencialmente sin tener que definir objetos intermedios. Comience redefiniendo `murders` para incluir la tasa y el rango.

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

En la solución al ejercicio anterior, hicimos lo siguiente:

```
my_states <- filter(murders, region %in% c("Northeast", "West") &
                     rate < 1)

select(my_states, state, rate, rank)
```

El *pipe* `%>%` nos permite realizar ambas operaciones secuencialmente sin tener que definir una variable intermedia `my_states`. Por lo tanto, podríamos haber mutado y seleccionado en la misma línea de esta manera:

```
mutate(murders, rate = total/ population * 100000,
       rank = rank(-rate)) %>%
  select(state, rate, rank)
```

Note que `select` ya no tiene un *data frame* como primer argumento. Se supone que el primer argumento sea el resultado de la operación realizada justo antes de `%>%`.

Repita el ejercicio anterior, pero ahora, en lugar de crear un nuevo objeto, muestre el resultado y solo incluya las columnas de estado, velocidad y rango. Use un *pipe* `%>%` para hacer esto en una sola línea.

2. Reinicie `murders` a la tabla original usando `data(murders)`. Use un *pipe* para crear un nuevo *data frame* llamado `my_states` que considera solo los estados del noreste u oeste que tienen una tasa de asesinatos inferior a 1 y contiene solo las columnas de estado, tasa y rango. El *pipe* también debe tener cuatro componentes separados por tres `%>%`. El código debería verse algo similar a lo siguiente:

```
my_states <- murders %>%
  mutate SOMETHING %>%
  filter SOMETHING %>%
  select SOMETHING
```

4.7 Cómo resumir datos

Una parte importante del análisis exploratorio de datos es resumir los datos. El promedio y la desviación estándar son dos ejemplos de estadísticas de resumen ampliamente utilizadas. A menudo se pueden obtener resúmenes más informativos dividiendo primero los datos en grupos. En esta sección, cubrimos dos nuevos verbos de `dplyr` que facilitan estos cálculos: `summarize` y `group_by`. Aprendemos a acceder a los valores resultantes utilizando la función `pull`.

4.7.1 `summarize`

La función `summarize` de `dplyr` ofrece una forma de calcular estadísticas de resumen con código intuitivo y legible. Comenzamos con un ejemplo sencillo basado en alturas. El set de datos `heights` incluye las alturas y el sexo reportado por los estudiantes en una encuesta en clase.

```
library(dplyr)
library(dslabs)
data(heights)
```

El siguiente código calcula el promedio y la desviación estándar para las hembras:

```
s <- heights %>%
  filter(sex == "Female") %>%
  summarize(average = mean(height), standard_deviation = sd(height))
s
#>   average standard deviation
#> 1     64.9           3.76
```

Esto toma nuestra tabla de datos original como entrada, la filtra para incluir solo a las filas representando hembras y luego produce una nueva tabla resumida con solo el promedio y la desviación estándar de las alturas. Podemos elegir los nombres de las columnas de la tabla resultante. Por ejemplo, arriba decidimos usar `average` y `standard_deviation`, pero podríamos haber usado otros nombres de la misma manera.

Como la tabla resultante almacenada en `s` es un *data frame*, podemos acceder a los componentes con el operador de acceso `$`:

```
s$average
#> [1] 64.9
s$standard_deviation
#> [1] 3.76
```

Igual que con la mayoría de las otras funciones de `dplyr`, `summarize` conoce los nombres de las variables y podemos usarlos directamente. Entonces, cuando escribimos `mean(height)` dentro de la llamada a la función `summarize`, la función accede a la columna con el nombre “`height`”, o altura, y luego calcula el promedio del vector numérico resultante. Podemos calcular cualquier otro resumen que opera en vectores y devuelve un solo valor.

Para otro ejemplo de cómo podemos usar la función `summarize`, calculemos la tasa promedio de asesinatos en Estados Unidos. Recuerden que nuestra tabla de datos incluye los asesinatos totales y el tamaño de la población para cada estado y ya hemos usado `dplyr` para añadir una columna de índice de asesinatos:

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

Recuerden que la tasa de asesinatos en EE. UU. `no` es el promedio de las tasas de asesinatos estatales:

```
summarize(murders, mean(rate))
#>   mean(rate)
#> 1      2.78
```

Esto se debe a que en el cálculo anterior, los estados pequeños tienen el mismo peso que los grandes. La tasa de asesinatos de Estados Unidos es el número total de asesinatos en Estados Unidos dividido por la población total. Entonces el cálculo correcto es:

```
us_murder_rate <- murders %>%
  summarize(rate = sum(total) / sum(population) * 100000)
us_murder_rate
#>   rate
#> 1 3.03
```

Este cálculo cuenta estados más grandes proporcionalmente a su tamaño, lo que da como resultado un valor mayor.

4.7.2 Resúmenes múltiples

Supongan que queremos tres resúmenes de una variable, como por ejemplo la mediana, el mínimo y el máximo. Podemos usar `summarize` así:

```
heights %>%
  filter(sex == "Female") %>%
  summarize(median = median(height), minimum = min(height),
            maximum = max(height))
#>   median minimum maximum
#> 1      65       51       79
```

Pero podemos obtener estos tres valores con una sola línea de código usando la función `quantile`: `quantile(x, c(0.5, 0, 1))` devuelve la mediana (percentil 50), el mínimo (percentil 0), y el máximo (percentil 100). Podemos usar la función con `summarize` así:

```
heights %>%
  filter(sex == "Female") %>%
  summarize(median_min_max = quantile(height, c(0.5, 0, 1)))
#>   median_min_max
#> 1              65
#> 2              51
#> 3              79
```

Ahora noten que los resúmenes se devuelven en filas separadas. Para obtenerlos en columnas, tenemos que definir una función que devuelve un *data frame* así:

```
median_min_max <- function(x){
  qs <- quantile(x, c(0.5, 0, 1))
  data.frame(median = qs[1], minimum = qs[2], maximum = qs[3])
}
```

```
heights %>%
  filter(sex == "Female") %>%
  summarize(median_min_max(height))
#>   median minimum maximum
#> 1      65      51      79
```

En la próxima sección veremos lo útil que esto puede ser cuando resumimos por grupo.

4.7.3 Cómo agrupar y luego resumir con `group_by`

Una operación común en la exploración de datos es dividir primero los datos en grupos y luego calcular resúmenes para cada grupo. Por ejemplo, podemos querer calcular el promedio y la desviación estándar para las alturas de hombres y mujeres por separado. La función `group_by` nos ayuda a hacer esto.

Si escribimos esto:

```
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
#> # ... with 1,045 more rows
```

El resultado no se ve muy diferente de `heights`, excepto que vemos `Groups: sex [2]` cuando imprimimos el objeto. Aunque no es inmediatamente obvio por su apariencia, esto ahora es un *data frame* especial llamado un *grouped data frame* y las funciones de `dplyr`, en particular `summarize`, se comportarán de manera diferente cuando actúan sobre este objeto. Conceptualmente, pueden pensar en esta tabla como muchas tablas, con las mismas columnas pero no necesariamente el mismo número de filas, apiladas juntas en un objeto. Cuando resumimos los datos después de la agrupación, esto es lo que sucede:

```
heights %>%
  group_by(sex) %>%
  summarize(average = mean(height), standard_deviation = sd(height))
#> # A tibble: 2 x 3
#>   sex     average standard deviation
#>   <fct>    <dbl>          <dbl>
#> 1 Female    64.9           3.76
#> 2 Male      69.3           3.61
```

La función `summarize` aplica el resumen a cada grupo por separado.

Para ver otro ejemplo, calculemos la mediana, el mínimo y máximo de la tasa de asesinatos en las cuatro regiones del país usando la función `median_min_max` definida anteriormente:

```
murders %>%
  group_by(region) %>%
  summarize(median_min_max(rate))
#> # A tibble: 4 x 4
#>   region      median  minimum maximum
#>   <fct>       <dbl>    <dbl>    <dbl>
#> 1 Northeast    1.80    0.320    3.60
#> 2 South        3.40    1.46     16.5
#> 3 North Central 1.97    0.595    5.36
#> 4 West         1.29    0.515    3.63
```

4.8 pull

El objeto `us_murder_rate` definido anteriormente representa solo un número. Sin embargo, lo estamos almacenando en un *data frame*:

```
class(us_murder_rate)
#> [1] "data.frame"
```

ya que, como la mayoría de las funciones de `dplyr`, `summarize` siempre devuelve un *data frame*.

Esto podría ser problemático si queremos usar este resultado con funciones que requieren un valor numérico. Aquí mostramos un truco útil para acceder a los valores almacenados en los datos cuando usamos *pipes*: cuando un objeto de datos se canaliza (*is piped* en inglés), ese objeto y sus columnas se pueden acceder usando la función `pull`. Para entender lo que queremos decir, miren esta línea de código:

```
us_murder_rate %>% pull(rate)
#> [1] 3.03
```

Esto devuelve el valor en la columna `rate` de `us_murder_rate` haciéndolo equivalente a `us_murder_rate$rate`.

Para obtener un número de la tabla de datos original con una línea de código, podemos escribir:

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

us_murder_rate
#> [1] 3.03
```

que ahora es numérico:

```
class(us_murder_rate)
#> [1] "numeric"
```

4.9 Cómo ordenar los *data frames*

Al examinar un set de datos, a menudo es conveniente ordenar, numérica o alfabéticamente, basado en una o más de las columnas de la tabla. Conocemos las funciones `order` y `sort`, pero para ordenar tablas enteras, la función `arrange` de `dplyr` es útil. Por ejemplo, aquí ordenamos los estados según el tamaño de la población:

```
murders %>%
  arrange(population) %>%
  head()
#> #> state abb region population total 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
```

Con `arrange` podemos decidir cuál columna usar para ordenar. Para ver los estados por tasa de asesinatos, desde menor a mayor, organizamos por el `rate`:

```
murders %>%
  arrange(rate) %>%
  head()
#> #> state abb region population total rate
#> 1 Vermont VT Northeast 625741 2 0.320
#> 2 New Hampshire NH Northeast 1316470 5 0.380
#> 3 Hawaii HI West 1360301 7 0.515
#> 4 North Dakota ND North Central 672591 4 0.595
#> 5 Iowa IA North Central 3046355 21 0.689
#> 6 Idaho ID West 1567582 12 0.766
```

Tengan en cuenta que el comportamiento por defecto es ordenar en orden ascendente. En `dplyr`, la función `desc` transforma un vector para que esté en orden descendente. Para ordenar la tabla en orden descendente, podemos escribir:

```
murders %>%
  arrange(desc(rate))
```

4.9.1 Cómo ordenar anidadamente

Si estamos ordenando una columna cuando hay empates, podemos usar una segunda columna para romper el empate. Del mismo modo, se puede usar una tercera columna

para romper empates entre la primera y la segunda, y así sucesivamente. Aquí ordenamos por `region` y entonces, dentro de la región, ordenamos por tasa de asesinatos:

```
murders %>%
  arrange(region, rate) %>%
  head()
#> #> state abb region population total rate
#> 1 Vermont VT Northeast 625741 2 0.320
#> 2 New Hampshire NH Northeast 1316470 5 0.380
#> 3 Maine ME Northeast 1328361 11 0.828
#> 4 Rhode Island RI Northeast 1052567 16 1.520
#> 5 Massachusetts MA Northeast 6547629 118 1.802
#> 6 New York NY Northeast 19378102 517 2.668
```

4.9.2 Los primeros n

En el código anterior, usamos la función `head` para evitar que la página se llene con todo el set de datos. Si queremos ver una mayor proporción, podemos usar la función `top_n`. Esta función toma un *data frame* como primer argumento, el número de filas para mostrar en el segundo y la variable para filtrar en el tercero. Aquí hay un ejemplo de cómo ver las 5 filas superiores:

```
murders %>% top_n(5, rate)
#> #> state abb region population total rate
#> 1 District of Columbia DC South 601723 99 16.45
#> 2 Louisiana LA South 4533372 351 7.74
#> 3 Maryland MD South 5773552 293 5.07
#> 4 Missouri MO North Central 5988927 321 5.36
#> 5 South Carolina SC South 4625364 207 4.48
```

Tengan en cuenta que las filas no están ordenadas por `rate`, solo filtradas. Si queremos ordenar, necesitamos usar `arrange`. Recuerden que si el tercer argumento se deja en blanco, `top_n` filtra por la última columna.

4.10 Ejercicios

Para estos ejercicios, utilizaremos los datos de la encuesta recopilada por el Centro Nacional de Estadísticas de Salud de Estados Unidos (NCHS por sus siglas en inglés). Este centro ha realizado una serie de encuestas de salud y nutrición desde la década de 1960. A partir de 1999, alrededor de 5,000 individuos de todas las edades han sido entrevistados cada año y completan el componente de examen de salud de la encuesta. Parte de los datos está disponible a través del paquete **NHANES**. Una vez que instale el paquete **NHANES**, puede cargar los datos así:

```
library(NHANES)
data(NHANES)
```

Los datos **NHANES** tienen muchos valores faltantes. Las funciones `mean` y `sd` devolverán `NA` si alguna de las entradas del vector de entrada es un `NA`. Aquí hay un ejemplo:

```
library(dslabs)
data(na_example)
mean(na_example)
#> [1] NA
sd(na_example)
#> [1] NA
```

Para ignorar los `NAs`, podemos usar el argumento `na.rm`:

```
mean(na_example, na.rm = TRUE)
#> [1] 2.3
sd(na_example, na.rm = TRUE)
#> [1] 1.22
```

Exploraremos ahora los datos de **NHANES**.

1. Le ofrecemos algunos datos básicos sobre la presión arterial. Primero, seleccionemos un grupo para establecer el estándar. Utilizaremos hembras de 20 a 29 años. `AgeDecade` es una variable categórica con estas edades. Tenga en cuenta que la categoría está codificada como " 20-29", ¡con un espacio al frente! ¿Cuál es el promedio y la desviación estándar de la presión arterial sistólica según se guarda en la variable `BPSysAve`? Guárdela en una variable llamada `ref`.

Sugerencia: use `filter` y `summarize` y use el argumento `na.rm = TRUE` al calcular el promedio y la desviación estándar. También puede filtrar los valores de `NA` utilizando `filter`.

2. Usando un `pipe`, asigne el promedio a una variable numérica `ref_avg`. Sugerencia: use el código similar al anterior y luego `pull`.

3. Ahora indique los valores mínimo y máximo para el mismo grupo.

4. Calcule el promedio y la desviación estándar para las hembras, pero para cada grupo de edad por separado en lugar de una década seleccionada como en la pregunta 1. Tenga en cuenta que los grupos de edad se definen por `AgeDecade`. Sugerencia: en lugar de filtrar por edad y género, filtre por `Gender` y luego use `group_by`.

5. Repita el ejercicio 4 para los varones.

6. Podemos combinar ambos resúmenes para los ejercicios 4 y 5 en una línea de código. Esto es porque `group_by` nos permite agrupar por más de una variable. Obtenga una gran tabla de resumen usando `group_by(AgeDecade, Gender)`.

7. Para los varones entre las edades de 40-49, compare la presión arterial sistólica según raza, como aparece en la variable `Race1`. Ordene la tabla resultante según la presión arterial sistólica promedio de más baja a más alta.

4.11 Tibbles

Los datos *tidy* deben almacenarse en *data frames*. Discutimos el *data frame* en la Sección 2.4.1 y hemos estado usando el *data frame* `murders` en todo el libro. En la sección 4.7.3, presentamos la función `group_by`, que permite estratificar los datos antes de calcular las estadísticas de resumen. Pero, ¿dónde se almacena la información del grupo en el *data frame*?

```
murders %>% group_by(region)
#> # A tibble: 51 x 6
#> # Groups:   region [4]
#>   state     abb   region population total    rate
#>   <chr>    <chr> <fct>      <dbl> <dbl>   <dbl>
#> 1 Alabama   AL    South       4779736   135   2.82
#> 2 Alaska    AK    West        710231    19   2.68
#> 3 Arizona   AZ    West        6392017   232   3.63
#> 4 Arkansas AR    South       2915918    93   3.19
#> 5 California CA    West       37253956  1257   3.37
#> # ... with 46 more rows
```

Observen que no hay columnas con esta información. Pero si miran el *output* anterior, verán la línea `A tibble` seguida por unas dimensiones. Podemos aprender la clase del objeto devuelto usando:

```
murders %>% group_by(region) %>% class()
#> [1] "grouped_df" "tbl_df"      "tbl"        "data.frame"
```

El `tbl` es un tipo especial de *data frame*. Las funciones `group_by` y `summarize` siempre devuelven este tipo de *data frame*. La función `group_by` devuelve un tipo especial de `tbl`, el `grouped_df`. Discutiremos esto más adelante. Por coherencia, los verbos de manipulación `dplyr` (`select`, `filter`, `mutate` y `arrange`) preservan la clase del *input*: si reciben un *data frame* regular, devuelven un *data frame* regular, mientras que si reciben un *tibble*, devuelven un *tibble*. Pero los *tibbles* son el formato preferido por el *tidyverse* y, como resultado, las funciones *tidyverse* que producen un *data frame* desde cero devuelven un *tibble*. Por ejemplo, en el Capítulo 5, veremos que las funciones del *tidyverse* que se usan para importar datos crean *tibbles*.

Los *tibbles* son muy similares a los *data frames*. De hecho, pueden pensar en ellos como una versión moderna de *data frames*. Sin embargo, hay tres diferencias importantes que describiremos a continuación.

4.11.1 Los *tibbles* se ven mejor

El método de impresión para *tibbles* es más legible que el de un *data frame*. Para ver esto, comparen el *output* de escribir `murders` y el *output* de asesinatos si los convertimos en un *tibble*. Podemos hacer esto usando `as_tibble(murders)`. Si usan RStudio, el *output* para un *tibble* se ajusta al tamaño de sus ventanas. Para ver esto, cambien el ancho de su consola R y observen cómo se muestran más/menos columnas.

4.11.2 Los subconjuntos de *tibbles* son *tibbles*

Si creamos subconjuntos de las columnas de un *data frame*, le pueden devolver un objeto que no es un *data frame*, como un vector o escalar. Por ejemplo:

```
class(murders[,4])
#> [1] "numeric"
```

no es un *data frame*. Con *tibbles*, esto no sucede:

```
class(as_tibble(murders)[,4])
#> [1] "tbl_df"     "tbl"        "data.frame"
```

Esto es útil en el *tidyverse* ya que las funciones requieren *data frames* como *input*.

Con *tibbles*, si desean acceder al vector que define una columna y no recuperar un *data frame*, deben usar el operador de acceso \$:

```
class(as_tibble(murders)$population)
#> [1] "numeric"
```

Una característica relacionada es que *tibbles* les dará una advertencia si intentan acceder a una columna que no existe. Por ejemplo, si escribimos accidentalmente `Population` en lugar de `population` vemos que:

```
murders$Population
#> NULL
```

devuelve un `NULL` sin advertencia, lo que puede dificultar la depuración. Por el contrario, si intentamos esto con un *tibble*, obtenemos una advertencia informativa:

```
as_tibble(murders)$Population
#> Warning: Unknown or uninitialized column: `Population`.
#> NULL
```

4.11.3 Los *tibbles* pueden tener entradas complejas

Si bien las columnas del *data frame* deben ser vectores de números, cadenas o valores lógicos, los *tibbles* pueden tener objetos más complejos, como listas o funciones. Además, podemos crear *tibbles* con 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>
```

4.11.4 Los *tibbles* se pueden agrupar

La función `group_by` devuelve un tipo especial de *tibble*: un *tibble* agrupado. Esta clase almacena información que les permite saber qué filas están en qué grupos. Las funciones `tidyverse`, en particular `summarize`, están al tanto de la información del grupo.

4.11.5 Cómo crear un *tibble* usando `tibble` en lugar de `data.frame`

A veces es útil para nosotros crear nuestros propios *data frames*. Para crear un *data frame* en formato *tibble*, pueden utilizar la función `tibble`.

```
grades <- tibble(names = c("John", "Juan", "Jean", "Yao"),
                  exam_1 = c(95, 80, 90, 85),
                  exam_2 = c(90, 85, 85, 90))
```

Noten que la base R (sin paquetes cargados) tiene una función con un nombre muy similar, `data.frame`, que se puede usar para crear un *data frame* regular en vez de un *tibble*.

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

Para convertir un *data frame* normal en un *tibble*, pueden usar la función `as_tibble`.

```
as_tibble(grades) %>% class()
#> [1] "tbl_df"     "tbl"        "data.frame"
```

4.12 El operador punto

Una de las ventajas de utilizar el *pipe* `%>%` es que no tenemos que seguir nombrando nuevos objetos mientras manipulamos el *data frame*. Recuerden que si queremos calcular la tasa de asesinatos promedio para los estados del sur, en lugar de escribir:

```
tab_1 <- filter(murders, region == "South")
tab_2 <- mutate(tab_1, rate = total/ population * 10^5)
rates <- tab_2$rate
median(rates)
#> [1] 3.4
```

podemos evitar definir nuevos objetos intermedios escribiendo:

```
filter(murders, region == "South") %>%
  mutate(rate = total/ population * 10^5) %>%
  summarize(median = median(rate)) %>%
  pull(median)
#> [1] 3.4
```

Podemos hacer esto porque cada una de estas funciones toma un *data frame* como primer argumento. Pero, ¿qué pasa si queremos acceder a un componente del *data frame*? Por ejemplo, ¿qué pasa si la función `pull` no está disponible y queremos acceder `tab_2$rate`? ¿Qué nombre de *data frame* usamos? La respuesta es el operador punto (*dot operator* en inglés).

Por ejemplo, para acceder al vector de velocidad sin la función `pull`, podríamos usar:

```
rates <- filter(murders, region == "South") %>%
  mutate(rate = total / population * 10^5) %>%
  .$rate
median(rates)
#> [1] 3.4
```

4.13 El paquete purrr

En la Sección 3.5, aprendimos sobre la función `sapply`, que nos permitió aplicar la misma función a cada elemento de un vector. Construimos una función y utilizamos `sapply` para calcular la suma de los primeros `n` enteros para varios valores de `n` así:

```
compute_s_n <- function(n){
  x <- 1:n
  sum(x)
}
n <- 1:25
s_n <- sapply(n, compute_s_n)
```

Este tipo de operación, que aplica la misma función o procedimiento a elementos de un objeto, es bastante común en el análisis de datos. El paquete **purrr** incluye funciones similares a `sapply`, pero que interactúan mejor con otras funciones del *tidyverse*. La principal ventaja es que podemos controlar mejor el tipo de resultado de las funciones. En cambio, `sapply` puede devolver varios tipos de objetos diferentes, convirtiéndolos cuando sea conveniente. Las funciones de **purrr** nunca harán esto: devolverán objetos de un tipo específico o devolverán un error si esto no es posible.

La primera función de **purrr** que aprenderemos es `map`, que funciona muy similar a `sapply` pero siempre, sin excepción, devuelve una lista:

```
library(purrr)
s_n <- map(n, compute_s_n)
class(s_n)
#> [1] "list"
```

Si queremos un vector numérico, podemos usar `map_dbl` que siempre devuelve un vector de valores numéricos.

```
s_n <- map_dbl(n, compute_s_n)
class(s_n)
#> [1] "numeric"
```

Esto produce los mismos resultados que la llamada `sapply` que vemos arriba.

Una función de `purrr` particularmente útil para interactuar con el resto del *tidyverse* es `map_df`, que siempre devuelve un *tibble data frame*. Sin embargo, la función que llamamos debe devolver un vector o una lista con nombres. Por esta razón, el siguiente código daría como resultado un error `Argument 1 must have names`:

```
s_n <- map_df(n, compute_s_n)
```

Necesitamos cambiar la función para arreglar esto:

```
compute_s_n <- function(n){
  x <- 1:n
  tibble(sum = sum(x))
}
s_n <- map_df(n, compute_s_n)
```

El paquete `purrr` ofrece mucha más funcionalidad no discutida aquí. Para obtener más detalles, pueden consultar recursos en línea¹.

4.14 Los condicionales de *tidyverse*

Un análisis de datos típicos frecuentemente implicará una o más operaciones condicionales. En la Sección 3.1, describimos la función `ifelse`, que utilizaremos ampliamente en este libro. Ahora presentamos dos funciones de `dplyr` que ofrecen una funcionalidad adicional para realizar operaciones condicionales.

4.14.1 `case_when`

La función `case_when` es útil para vectorizar declaraciones condicionales. Esto es similar a `ifelse`, pero puede generar cualquier cantidad de valores, en lugar de solo TRUE o FALSE. Aquí hay un ejemplo que divide los números en negativo, positivo y 0:

```
x <- c(-2, -1, 0, 1, 2)
case_when(x < 0 ~ "Negative",
          x > 0 ~ "Positive",
          TRUE ~ "Zero")
#> [1] "Negative" "Negative" "Zero"      "Positive" "Positive"
```

¹<https://jennybc.github.io/purrr-tutorial/>

Un uso común de esta función es definir unas variables categóricas basadas en variables existentes. Por ejemplo, supongan que queremos comparar las tasas de asesinatos en cuatro grupos de estados: *New England*, *West Coast*, *South* y *Other*. Para cada estado, primero preguntamos si está en *New England*. Si la respuesta es no, entonces preguntamos si está en el *West Coast*, y si no, preguntamos si está en el *South* y, si no, entonces asignamos ninguna de las anteriores (*Other*). Aquí vemos como usamos `case_when` para hacer esto:

```
murders %>%
  mutate(group = case_when(
    abb %in% c("ME", "NH", "VT", "MA", "RI", "CT") ~ "New England",
    abb %in% c("WA", "OR", "CA") ~ "West Coast",
    region == "South" ~ "South",
    TRUE ~ "Other")) %>%
  group_by(group) %>%
  summarize(rate = sum(total)/ sum(population) * 10^5)
#> # A tibble: 4 x 2
#>   group      rate
#>   <chr>     <dbl>
#> 1 New England 1.72
#> 2 Other       2.71
#> 3 South       3.63
#> 4 West Coast  2.90
```

4.14.2 between

Una operación común en el análisis de datos es determinar si un valor cae dentro de un intervalo. Podemos verificar esto usando condicionales. Por ejemplo, para verificar si los elementos de un vector `x` están entre `a` y `b`, podemos escribir:

```
x >= a & x <= b
```

Sin embargo, esto puede volverse complicado, especialmente dentro del enfoque *tidyverse*. La función `between` realiza la misma operación:

```
between(x, a, b)
```

4.15 Ejercicios

1. Cargue el set de datos `murders`. ¿Cuál de los siguientes es cierto?

- a. `murders` está en formato *tidy* y se almacena en un *tibble*.
- b. `murders` está en formato *tidy* y se almacena en un *data frame*.
- c. `murders` no está en formato *tidy* y se almacena en un *tibble*.
- d. `murders` no está en formato *tidy* y se almacena en un *data frame*.

2. Utilice `as_tibble` para convertir la tabla de datos `murders` en un *tibble* y guárdelo en un objeto llamado `murders_tibble`.
3. Utilice la función `group_by` para convertir `murders` en un *tibble* que se agrupa por región.
4. Escriba el código `tidyverse` que es equivalente a este código:

```
exp(mean(log(murders$population)))
```

Escríbalo usando el *pipe* para que cada función se llame sin argumentos. Use el operador punto para acceder a la población. Sugerencia: el código debe comenzar con `murders %>%`.

5. Utilice el `map_df` para crear un *data frame* con tres columnas que se denominan `n`, `s_n` y `s_n_2`. La primera columna debe contener los números del 1 al 100. La segunda y la tercera columna deben contener la suma del 1 al 100 `n` con `n` representando el número de fila.

5

Importando datos

Hemos estado usando sets de datos ya almacenados como objetos R. Los científicos de datos rara vez tendrán tanta suerte y frecuentemente tendrán que importar datos a R desde un archivo, una base de datos u otras fuentes. Actualmente, una de las formas más comunes de almacenar y compartir datos para el análisis es a través de hojas de cálculo electrónicas. Una hoja de cálculo almacena datos en filas y columnas. Básicamente es una versión de archivo de un *data frame*. Al guardar dicha tabla en un archivo de computadora, uno necesita una manera de definir cuándo termina una nueva fila o columna y cuando comienza la otra. Esto a su vez define las celdas en las que se almacenan los valores individuales.

Al crear hojas de cálculo con archivos de texto, como esas creadas con un editor sencillo de texto, se define una nueva fila con *return* y se separan las columnas con un carácter especial predefinido. Los caracteres más comunes son coma (,), punto y coma (;), espacio () y el *tab* (un número predeterminado de espacios o \t). Aquí tenemos un ejemplo de cómo se ve un archivo separado por comas si lo abrimos con un editor básico de texto:

state,abb,region,population,total
Alabama,AL,South,4779736,135
Alaska,AK,West,710231,19
Arizona,AZ,West,6392817,232
Arkansas,AR,South,2915500,93
Colorado,CO,West,57253956,1257
Colorado,CO,West,5029196,65
Connecticut,CT,Northeast,3574097,97
Delaware,DE,South,897934,38
District of Columbia,DC,South,601723,99
Florida,FL,South,19687653,669
Georgia,GA,South,9920000,376
Hawaii,HI,West,1360301,7
Idaho,ID,West,1567582,12
Illinois,IL,North Central,12838632,364
Indiana,IN,North Central,6498882,142
Iowa,IA,North Central,3063552,21
Kansas,KS,North Central,2853118,63
Kentucky,KY,South,4339367,116
Louisiana,LA,South,4533372,351
Maine,ME,Northeast,1328361,11
Maryland,MD,South,5773552,293
Massachusetts,MA,Northeast,6547629,118
Michigan,MI,North Central,9883640,413
Minnesota,MN,North Central,5303925,53
Mississippi,MS,South,2967297,128
Missouri,MO,North Central,5988927,321
Montana,MT,West,90015,12
Nebraska,NE,North Central,1826341,32
Nevada,NV,West,2700551,84

La primera fila contiene nombres de columnas en lugar de datos. Nos referimos a esto como un *encabezado* (*header* en inglés), y cuando leemos (*read-in* en inglés) datos de una hoja de cálculo es importante saber si el archivo tiene un encabezado o no. La mayoría de las funciones de lectura suponen que hay un encabezado. Para saber si el archivo tiene un encabezado, miren el archivo antes de intentar leerlo. Esto se puede hacer con un editor de texto o con RStudio. En RStudio, podemos hacerlo abriendo el archivo en el editor o navegando a la ubicación del archivo, haciendo doble clic en el archivo y presionando *View File*.

Sin embargo, no todos los archivos de hojas de cálculo están en formato de texto. Las hojas

de cálculo de Google (*Google Sheets* en inglés), por ejemplo, se acceden con un navegador. Otro ejemplo es el formato propietario utilizado por Microsoft Excel, que no se puede ver con un editor de texto. A pesar de esto y debido a la popularidad del software Microsoft Excel, este formato se utiliza ampliamente.

Comenzamos este capítulo describiendo las diferencias entre archivos de texto (ASCII), Unicode y binarios y cómo estas afectan la forma en que los importamos. Entonces, explicamos los conceptos de rutas de archivos y directorios de trabajo, que son esenciales para comprender cómo importar datos de manera efectiva. Luego, presentamos los paquetes **readr** y **readxl** y las funciones disponibles para importar hojas de cálculo en R. Finalmente, ofrecemos algunas recomendaciones sobre cómo almacenar y organizar datos en archivos. Los desafíos más complejos, sin embargo, como la extracción de datos de páginas web o de documentos PDF, se discutirán en la parte del libro “*Wrangling de datos*”.

5.1 Las rutas y el directorio de trabajo

El primer paso para importar datos desde una hoja de cálculo es ubicar el archivo que contiene los datos. Aunque no lo recomendamos, pueden utilizar un enfoque similar al que usan para abrir archivos en Microsoft Excel: haciendo clic en el menú de *File* de RStudio, haciendo clic en *Import Dataset* y luego haciendo clic en las carpetas hasta encontrar el archivo. Queremos poder escribir código en lugar de estar apuntando y haciendo clic. Las claves y los conceptos que necesitamos para aprender a hacer esto se describen en detalle en la parte del libro “Herramientas de productividad”. Aquí ofrecemos una descripción general de los conceptos básicos.

El principal reto de este primer paso es permitir que las funciones de R que realizan la importación sepan dónde buscar el archivo que contiene los datos. La forma más sencilla de hacer esto es tener una copia del archivo en la carpeta donde las funciones de importación buscan por defecto. Una vez que hagamos esto, solo tenemos que proveerle el nombre del archivo a la función de importación.

El paquete **dslabs** incluye una hoja de cálculo que contiene los datos de los asesinatos de EE. UU. Encontrar este archivo no es obvio, pero las siguientes líneas de código copian el archivo a la carpeta en la que R busca por defecto. A continuación explicamos cómo funcionan estas líneas.

```
filename <- "murders.csv"
dir <- system.file("extdata", package = "dslabs")
fullpath <- file.path(dir, filename)
file.copy(fullpath, "murders.csv")
```

Este código no lee los datos en R, solo copia un archivo. Pero una vez copie el archivo, podemos importar los datos con solo una línea de código. Aquí usamos la función **read_csv** del paquete **readr**, que forma parte del **tidyverse**.

```
library(tidyverse)
dat <- read_csv(filename)
```

Los datos se importan y almacenan en `dat`. El resto de esta sección define algunos conceptos importantes y ofrece una visión general de cómo escribimos código para que R pueda encontrar los archivos que queremos importar. Capítulo 38 ofrece más detalles sobre este tema.

5.1.1 El sistema de archivos

Pueden pensar en el sistema de archivos (*filesystem* en inglés) de su computadora como una serie de carpetas anidadas, cada una con otras carpetas y archivos. Los científicos de datos se refieren a las carpetas como *directorios* y a la carpeta que contiene todas las demás carpetas como el *directorio raíz* (*root directory* en inglés). El directorio en el que estamos ubicados actualmente se llama el *directorio de trabajo* (*working directory* en inglés). Por lo tanto, el directorio de trabajo cambia a medida que se muevan por las carpetas: considérenlo como su ubicación actual.

5.1.2 Las rutas relativas y completas

La *ruta* (*path* en inglés) de un archivo es una lista de nombres de directorios que se pueden considerar instrucciones sobre en qué carpetas hacer clic y en qué orden encontrar el archivo. Si estas instrucciones son para encontrar el archivo desde el directorio raíz, nos referiremos a ellas como la *ruta completa* (*full path* en inglés). Si las instrucciones son para encontrar el archivo desde el directorio de trabajo, nos referimos a ellas como una *ruta relativa* (*relative path* en inglés). Sección 38.3 ofrece más detalles sobre este tema.

Para ver un ejemplo de una ruta completa en su sistema, escriban lo siguiente:

```
system.file(package = "dslabs")
```

Las cadenas separadas por barras son los nombres de los directorios. La primera barra diagonal representa el directorio raíz y sabemos que esta es una ruta completa porque comienza con una barra diagonal. Si el primer nombre del directorio aparece sin una barra diagonal en el comienzo, entonces R supone que la ruta es relativa. Podemos usar la función `list.files` para ver ejemplos de rutas relativas:

```
dir <- system.file(package = "dslabs")
list.files(path = dir)
#> [1] "data"          "DESCRIPTION"   "extdata"      "help"
#> [5] "html"          "INDEX"        "Meta"         "NAMESPACE"
#> [9] "R"             "script"
```

Estas rutas relativas nos dan la localización de los archivos o directorios si comenzamos en el directorio con la ruta completa. Por ejemplo, la ruta completa al directorio `help` en el ejemplo anterior es: `/Library/Frameworks/R.framework/Versions/3.5/Resources/library/dslabs/help`.

Nota: Probablemente no harán mucho uso de la función `system.file` en su trabajo diario de análisis de datos. Lo presentamos en esta sección porque facilita el intercambio de hojas de cálculo al incluirlas en el paquete `dslabs`. Raras veces tendrán el lujo de tener datos incluidos en paquetes que ya han instalado. Sin embargo, con frecuencia necesitarán navegar por rutas completas y relativas e importar datos con formato de hoja de cálculo.

5.1.3 El directorio de trabajo

Recomendamos escribir solo rutas relativas en su código ya que las rutas completas son exclusivas de sus computadoras y es preferible que su código sea portátil. Pueden obtener la ruta completa de su directorio de trabajo sin escribirla explícitamente utilizando la función `getwd`:

```
wd <- getwd()
```

Si necesitan cambiar su directorio de trabajo, pueden usar la función `setwd` o pueden cambiarlo a través de RStudio haciendo clic en *Session*.

5.1.4 Cómo generar los nombres de ruta

Otro ejemplo de cómo obtener una ruta completa sin escribirla explícitamente se ofreció arriba cuando creamos el objeto `fullpath` de esta manera:

```
filename <- "murders.csv"
dir <- system.file("extdata", package = "dslabs")
fullpath <- file.path(dir, filename)
```

La función `system.file` provee la ruta completa de la carpeta que contiene todos los archivos y directorios relevantes para el paquete especificado por el argumento `package`. Al explorar los directorios en `dir`, nos encontramos con que `extdata` contiene el archivo que queremos:

```
dir <- system.file(package = "dslabs")
filename %in% list.files(file.path(dir, "extdata"))
#> [1] TRUE
```

La función `system.file` nos permite proveer un subdirectorio como primer argumento, para que podamos obtener la ruta completa del directorio `extdata` así:

```
dir <- system.file("extdata", package = "dslabs")
```

La función `file.path` se usa para combinar los nombres de directorios para producir la ruta completa del archivo que queremos importar.

```
fullpath <- file.path(dir, filename)
```

5.1.5 Cómo copiar los archivos usando rutas

La última línea de código que usamos para copiar el archivo en nuestro directorio de inicio usó la función `file.copy`. Esta toma dos argumentos: el nombre del archivo para copiar y el nombre que se usará en el nuevo directorio.

```
file.copy(fullpath, "murders.csv")
#> [1] TRUE
```

Si un archivo se copia exitosamente, la función `file.copy` devuelve `TRUE`. Tengan en cuenta que le estamos dando al archivo el mismo nombre, `murders.csv`, pero podríamos haberle dado cualquier nombre. También recuerden que al no iniciar la cadena con una barra diagonal, R supone que esta es una ruta relativa y copia el archivo al directorio de trabajo.

Deberían poder ver el archivo en su directorio de trabajo usando:

```
list.files()
```

5.2 Los paquetes `readr` y `readxl`

En esta sección presentamos las principales funciones de importación del *tidyverse*. Utilizaremos el archivo `murders.csv` del paquete `dslabs` como ejemplo. Para simplificar la ilustración, copiaremos el archivo a nuestro directorio de trabajo usando el siguiente código:

```
filename <- "murders.csv"
dir <- system.file("extdata", package = "dslabs")
fullpath <- file.path(dir, filename)
file.copy(fullpath, "murders.csv")
```

5.2.1 `readr`

El paquete `readr` incluye funciones para leer datos almacenados en hojas de cálculo. `readr` es parte del paquete `tidyverse`, o pueden cargarlo directamente así:

```
library(readr)
```

Las siguientes funciones están disponibles para leer hojas de cálculo:

Función	Formato	Sufijo típico
<code>read_table</code>	valores separados por espacios en blanco	txt
<code>read_csv</code>	valores separados por comas	csv
<code>read_csv2</code>	valores separados por punto y coma	csv
<code>read_tsv</code>	valores separados delimitados por tab	tsv
<code>read_delim</code>	formato de archivo de texto general, debe definir delimitador	txt

Aunque el sufijo generalmente nos indica qué tipo de archivo es, no hay garantía de que estos siempre coincidan. Podemos abrir el archivo para echar un vistazo o usar la función `read_lines` para ver algunas líneas:

```
read_lines("murders.csv", n_max = 3)
#> [1] "state,abb,region,population,total"
#> [2] "Alabama,AL,South,4779736,135"
#> [3] "Alaska,AK,West,710231,19"
```

Esto también muestra que hay un encabezado. Ahora estamos listos para leer los datos en R. Del sufijo .csv y del vistazo al archivo, sabemos que tenemos que usar `read_csv`:

```
dat <- read_csv(filename)
```

Tengan en cuenta que recibimos un mensaje informándonos qué tipos de datos se utilizaron para cada columna. También observen que `dat` es un `tibble`, no solo un *data frame*. Esto es porque `read_csv` es un leedor (*parser* en inglés) del *tidyverse*. Podemos confirmar que los datos se han leído de la siguiente manera:

```
View(dat)
```

Finalmente, recuerden que también podemos usar la ruta completa para el archivo:

```
dat <- read_csv(fullpath)
```

5.2.2 readxl

Pueden cargar el paquete `readxl` usando:

```
library(readxl)
```

El paquete ofrece funciones para leer formatos de Microsoft Excel:

Función	Formato	Sufijo típico
<code>read_excel</code>	detectar automáticamente el formato	xls, xlsx
<code>read_xls</code>	formato original	xls
<code>read_xlsx</code>	nuevo formato	xlsx

Los formatos de Microsoft Excel le permiten tener más de una hoja de cálculo en un archivo. Estos se conocen como *hojas* (*sheets* en inglés). Las funciones enumeradas anteriormente leen la primera hoja por defecto, pero también podemos leer las otras. La función `excel_sheets` nos da los nombres de todas las hojas en un archivo de Excel. Estos nombres entonces se pueden pasar al argumento `sheet` en las tres funciones anteriores para leer hojas distintas a la primera.

5.3 Ejercicios

1. Utilice la función `read_csv` para leer cada uno de los archivos que el siguiente código guarda en el objeto `files`:

```
path <- system.file("extdata", package = "dslabs")
files <- list.files(path)
files
```

2. Observe que el último, el archivo `olive`, nos da una advertencia. Esto se debe a que a la primera línea del archivo le falta el encabezado de la primera columna.

Lea la página de ayuda para `read_csv` para aprender cómo leer el archivo sin leer este encabezado. Si omite el encabezado, no debería recibir esta advertencia. Guarde el resultado en un objeto llamado `dat`.

3. Un problema con el enfoque anterior es que no sabemos qué representan las columnas. Escriba:

```
names(dat)
```

para confirmar que los nombres no son informativos.

Utilice la función `readLines` para leer solo la primera línea (luego aprenderemos cómo extraer valores del *output*).

5.4 Cómo descargar archivos

Otro lugar común donde residen los datos es en el internet. Cuando estos datos están en archivos, podemos descargarlos y luego importarlos, o incluso leerlos directamente de la web. Por ejemplo, notamos que como nuestro paquete `dslabs` está en GitHub, el archivo que descargamos con el paquete tiene una URL:

```
url <- "https://raw.githubusercontent.com/rafalab/dslabs/master/inst/
extdata/murders.csv"
```

El archivo `read_csv` puede leer estos archivos directamente:

```
dat <- read_csv(url)
```

Si quieren tener una copia local del archivo, pueden usar la función `download.file`:

```
download.file(url, "murders.csv")
```

Esto descargará el archivo y lo guardará en su sistema con el nombre `murders.csv`. Pueden usar cualquier nombre aquí, no necesariamente `murders.csv`. Recuerden que al usar `download.file` deben tener cuidado ya que **sobrescribirá los archivos existentes sin previo aviso**.

Dos funciones que a veces son útiles al descargar datos del internet son `tempdir` y `tempfile`. La primera crea un directorio con un nombre aleatorio que es muy probable que sea único. Igualmente, `tempfile` crea una cadena de caracteres, no un archivo, que probablemente sea un nombre de archivo único. Entonces pueden ejecutar un comando, como el siguiente, que borra el archivo temporero una vez que importe los datos:

```
tmp_filename <- tempfile()
download.file(url, tmp_filename)
dat <- read_csv(tmp_filename)
file.remove(tmp_filename)
```

5.5 Las funciones de importación de base R

La base R también provee funciones de importación. Estos tienen nombres similares a esas del *tidyverse*, por ejemplo `read.table`, `read.csv` y `read.delim`.

```
dat2 <- read.csv(filename)
```

5.5.1 `scan`

Al leer hojas de cálculo, muchas cosas pueden salir mal. El archivo puede tener un encabezado multilínea, pueden faltar celdas, o puede usar una codificación inesperada¹. Les recomendamos que lean esta publicación sobre problemas comunes².

Con experiencia, aprenderán a manejar los diferentes retos. Además, les ayudará leer detenidamente los archivos de ayuda para las funciones que discutimos aquí. Con `scan` pueden leer cada celda de un archivo, como vemos aquí:

```
path <- system.file("extdata", package = "dslabs")
filename <- "murders.csv"
x <- scan(file.path(path, filename), sep=",", what = "c")
x[1:10]
#> [1] "state"      "abb"        "region"     "population" "total"
#> [6] "Alabama"    "AL"         "South"      "4779736"    "135"
```

Noten que el *tidyverse* incluye `read_lines`, una función igualmente útil.

¹https://en.wikipedia.org/wiki/Character_encoding

²<https://www.joelonsoftware.com/2003/10/08/the-absolute-minimum-every-software-developer-absolutely-positively-must-know-about-unicode-and-character-sets-no-excuses/>

5.6 Archivos de texto versus archivos binarios

En la ciencia de datos, los archivos generalmente se pueden clasificar en dos categorías: archivos de texto (también conocidos como archivos ASCII) y archivos binarios. Ya han trabajado con archivos de texto. Todos sus *scripts* de R son archivos de texto igual que los archivos de R markdown utilizados para crear este libro. Las tablas csv que han leído también son archivos de texto. Una gran ventaja de estos archivos es que podemos “mirarlos” fácilmente sin tener que comprar ningún tipo de software especial o seguir instrucciones complicadas. Se puede usar cualquier editor de texto para examinar un archivo de texto, incluyendo los editores disponibles gratuitamente como RStudio, Notepad, textEdit, vi, emacs, nano y pico. Para ver esto, intenten abrir un archivo csv con la herramienta de RStudio *Open file*. Deberían poder ver el contenido directamente en su editor. Sin embargo, si intentan abrir, digamos, un archivo Excel xls, jpg o png, no podrán ver nada inmediatamente útil. Estos son archivos binarios. Los archivos de Excel son carpetas comprimidas con varios archivos de texto dentro de ellas. Pero la principal distinción aquí es que los archivos de texto se pueden examinar fácilmente.

Aunque R incluye herramientas para leer archivos binarios ampliamente utilizados, como archivos xls, en general es mejor encontrar sets de datos almacenados en archivos de texto. Del mismo modo, al compartir datos, es mejor que estén disponibles como archivos de texto siempre que el almacenamiento no sea un problema (los archivos binarios son mucho más eficientes para ahorrar espacio en su disco). En general, los formatos de texto facilitan el intercambio de datos, ya que no requieren software comercial para trabajar con los datos.

Extraer datos de una hoja de cálculo almacenada como un archivo de texto es quizás la forma más fácil de llevar datos de un archivo a una sesión R. Desafortunadamente, las hojas de cálculo no siempre están disponibles y el hecho de que puedan ver los archivos de texto no necesariamente implica que extraer datos de ellos sea sencillo. En la parte del libro “*Wrangling de datos*”, aprendemos a extraer datos de archivos de texto más complejos, como los archivos HTML.

5.7 Unicode versus ASCII

Un problema en la ciencia de datos es suponer que un archivo es un archivo de texto ASCII cuando en actualidad es otra cosa que puede parecerse mucho a un archivo de texto ASCII: un archivo de texto Unicode.

Para entender la diferencia entre estos, recuerden que todo en una computadora necesita convertirse eventualmente en 0s y 1s. ASCII es una *codificación* (*encoding* en inglés) que define una correspondencia entre caracteres y números. ASCII usa 7 bits (0s y 1s) que resulta en $2^7 = 128$ elementos únicos, suficientes para codificar todos los caracteres en un teclado en inglés. Sin embargo, otros idiomas, como el español, usan caracteres no incluidos en esta codificación. Por ejemplo, las tildes no están codificadas por ASCII. Por esta razón, se definió una nueva codificación que utiliza más de 7 bits: Unicode. Cuando se utiliza Unicode, se puede elegir entre 8, 16 y 32 bits abreviados UTF-8, UTF-16 y UTF-32 respectivamente. RStudio usa la codificación UTF-8 por defecto.

Aunque no entraremos en detalles sobre cómo lidiar con las diferentes codificaciones aquí, es importante que sepan que existen diferentes codificaciones para que pueden diagnosticar bien un problema si lo encuentran. Una forma en que se manifiestan los problemas es cuando surgen caracteres de “aspecto extraño” que no esperaban. Esta discusión de StackOverflow es un ejemplo: <https://stackoverflow.com/questions/18789330/r-on-windows-character-encoding-hell>.

5.8 Cómo organizar datos con hojas de cálculo

Aunque este libro se enfoca casi exclusivamente en el análisis de datos, el manejo de datos también es una parte importante de la ciencia de datos. Como explicamos en la introducción, no cubrimos este tema. Sin embargo, los analistas de datos frecuentemente necesitan recopilar datos, o trabajar con otros que recopilan datos, de manera que la forma más conveniente de almacenarlos es en una hoja de cálculo. Aunque completar una hoja de cálculo a mano es una práctica que no recomendamos y preferimos que el proceso se automatice lo más posible, a veces no queda otro remedio. Por lo tanto, en esta sección, ofrecemos recomendaciones sobre cómo organizar los datos en una hoja de cálculo. Aunque hay paquetes R diseñados para leer hojas de cálculo de Microsoft Excel, generalmente queremos evitar este formato. Recomendamos *Google Sheets* como una herramienta de software gratuita. Abajo resumimos las recomendaciones hechas en una publicación de Karl Broman y Kara Woo³. Favor de leer el artículo completo para más detalles importantes.

- **Sea coherente** - Antes de empezar a ingresar datos, tenga un plan. Una vez lo tenga, sea consistente y sígalo.
- **Elija buenos nombres para las cosas:** Los nombres que elije para los objetos, los archivos y los directorios deben ser memorables, fáciles de deletrear y descriptivos. Este es un equilibrio difícil de lograr y requiere tiempo y reflexión. Una regla importante a seguir es **no usar espacios**; en su lugar, usen guiones bajos _ o guiones -. Además, evite los símbolos; es mejor utilizar las letras y los números.
- **Escriba fechas como AAAA-MM-DD** - Para evitar confusión, recomendamos utilizar el estándar global ISO 8601.
- **Evite las celdas vacías** - Llene todas las celdas y use un código común para los datos faltantes.
- **Ponga solo una cosa en cada celda** - Es mejor añadir columnas para almacenar la información adicional en vez de tener más de una pieza de información en una celda.
- **Hazlo un rectángulo** - La hoja de cálculo debe ser un rectángulo.
- **Cree un diccionario de datos** - Si necesita explicar cosas, por ejemplo cuáles son las columnas o cuáles son las etiquetas utilizadas para las variables categóricas, hágalo en un archivo separado.
- **No haga cálculos en los archivos de datos sin procesar** - Excel le permite realizar cálculos. No haga esto parte de su hoja de cálculo. El código para los cálculos debe estar en un *script*.
- **No use color o resaltado como datos** - La mayoría de funciones de importación no pueden importar esta información. En cambio, codifique esta información como una variable.

³<https://www.tandfonline.com/doi/abs/10.1080/00031305.2017.1375989>

- **Respalde su información:** Respalde sus datos frecuentemente.
 - **Utilice la validación de datos para evitar errores** - Aproveche las herramientas en su software de hoja de cálculo para que el proceso sea lo más libre posible de errores y de lesiones por estrés repetitivo.
 - **Guarde los datos como archivos de texto** - Guarde los archivos para compartir en formato delimitado por comas o *tabs*.
-

5.9 Ejercicios

1. Elija una medida que pueda tomar regularmente. Por ejemplo, su peso diario o cuánto tiempo le toma correr 8 kilómetros. Mantenga una hoja de cálculo que incluya la fecha, la hora, la medición y cualquier otra variable informativa que considere valiosa. Haga esto por 2 semanas. Luego haga un gráfico.

Part II

Visualización de datos

6

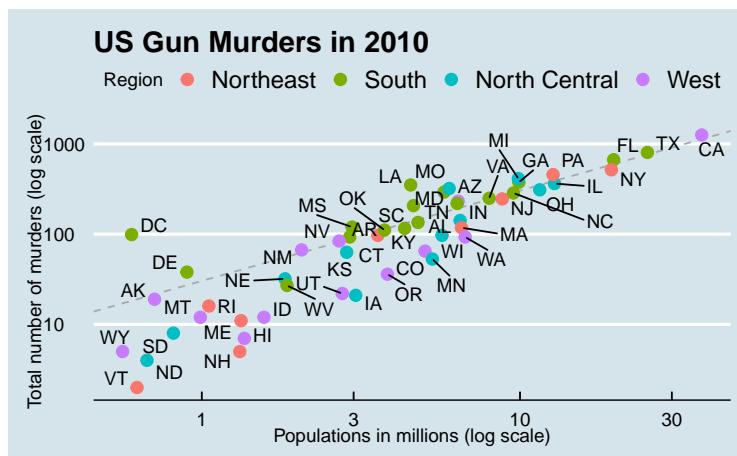
Introducción a la visualización de datos

Raras veces es útil mirar los números y las cadenas de caracteres que definen un set de datos. Para confirmar esto, impriman y miren la tabla de datos de asesinatos de Estados Unidos:

```
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

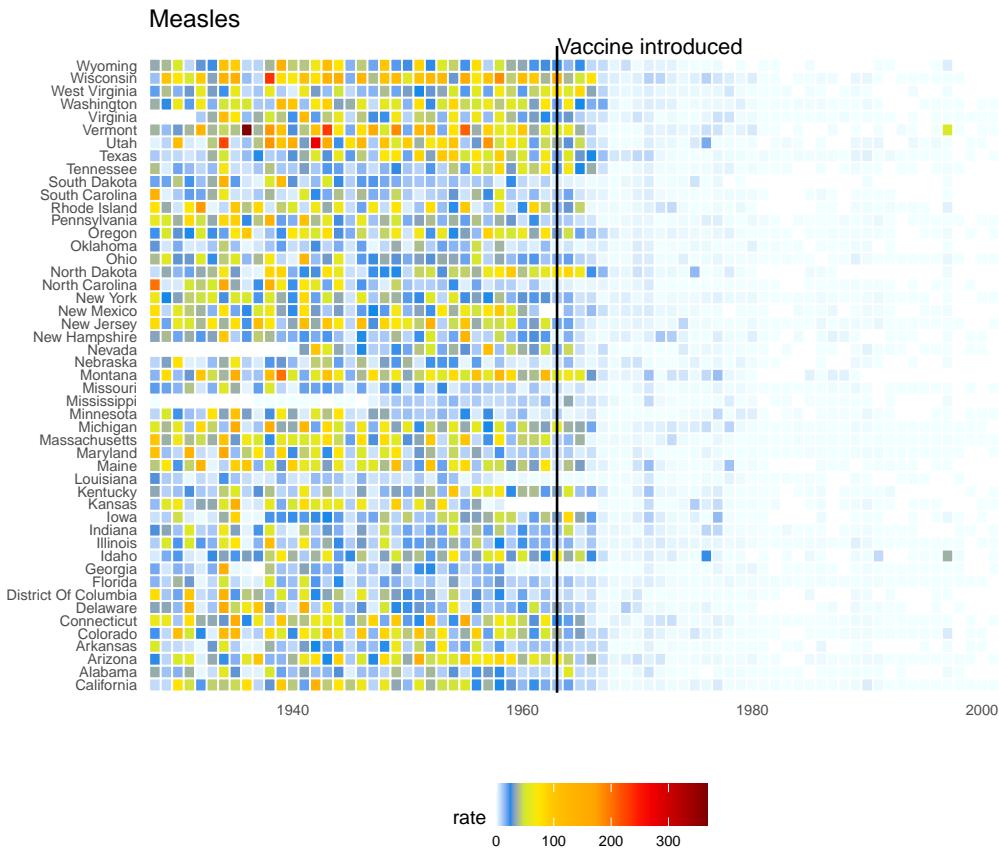
¿Qué aprenden de ver esta tabla? ¿Cuán rápido pueden determinar qué estados tienen las poblaciones más grandes? ¿Qué estados tienen las más pequeñas? ¿Cuán grande es un estado típico? ¿Existe una relación entre el tamaño de la población y el total de asesinatos? ¿Cómo varían las tasas de asesinatos entre las regiones del país? Para la mayoría de cerebros humanos, es bastante difícil extraer esta información simplemente mirando los números. En cambio, las respuestas a todas las preguntas anteriores están fácilmente disponibles al examinar este gráfico:



Esto nos recuerda del dicho “una imagen vale más que mil palabras”. La visualización de datos ofrece una forma muy efectiva de comunicar hallazgos basados en datos. En algunos casos, la visualización es tan convincente que no requiere un análisis de seguimiento.

La creciente disponibilidad de sets de datos informativos y de herramientas de software ha conducido a una mayor dependencia de la visualización de datos en muchas industrias, academias y gobiernos. Un ejemplo destacado son las organizaciones de noticias, que están adoptando cada vez más el *periodismo de datos* e incluyendo *infografías* eficaces como parte de sus informes.

Un ejemplo particularmente efectivo es un artículo del Wall Street Journal¹ que muestra datos relacionados con el impacto de las vacunas en la lucha contra las enfermedades infecciosas. Uno de los gráficos muestra los casos de sarampión por estado de EE. UU. a lo largo de los años con una línea vertical que indica cuándo se introdujo la vacuna.



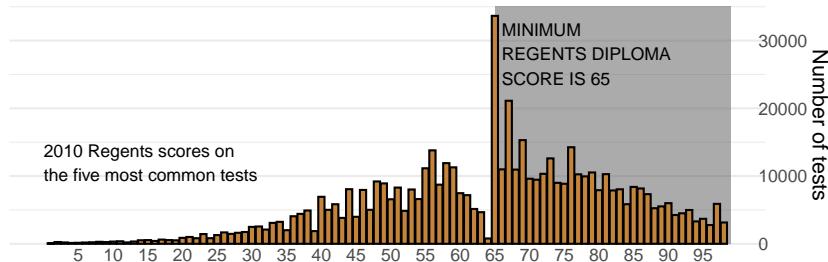
Otro ejemplo notable proviene de un gráfico del New York Times² que resume los resultados de los exámenes de los Regentes de la ciudad de Nueva York. Según el artículo³, estas puntuaciones se recopilan por varias razones, incluso para determinar si un estudiante se gradúa de escuela secundaria. En la ciudad de Nueva York, se necesita una puntuación mínima de 65 para aprobar. La distribución de las puntuaciones de las pruebas nos obliga a notar algo un poco problemático:

¹http://graphics.wsj.com/infectious-diseases-and-vaccines/?mc_cid=711ddeb86e

²<http://graphics8.nytimes.com/images/2011/02/19/nyregion/19schools/19schools-ch-popup.gif>

³<https://www.nytimes.com/2011/02/19/nyregion/19schools.html>

Scraping by



La puntuación de prueba más común es la calificación mínima para aprobar, con muy pocas puntuaciones justo por debajo del umbral. Este resultado inesperado es consistente con el aumento de la puntuación de los estudiantes cerca de aprobar, pero sin obtener el mínimo de 65.

Este es un ejemplo de cómo la visualización de datos puede conducir a descubrimientos que de otro modo se perderían si simplemente sometiéramos los datos a una serie de herramientas o procedimientos de análisis de datos. La visualización de datos es la herramienta más efectiva de lo que llamamos el *análisis exploratorio de datos* (EDA por sus siglas en inglés). John W. Tukey⁴, considerado el padre de EDA, una vez dijo:

“El mayor valor de una imagen es cuando nos obliga a notar lo que nunca esperábamos ver.”

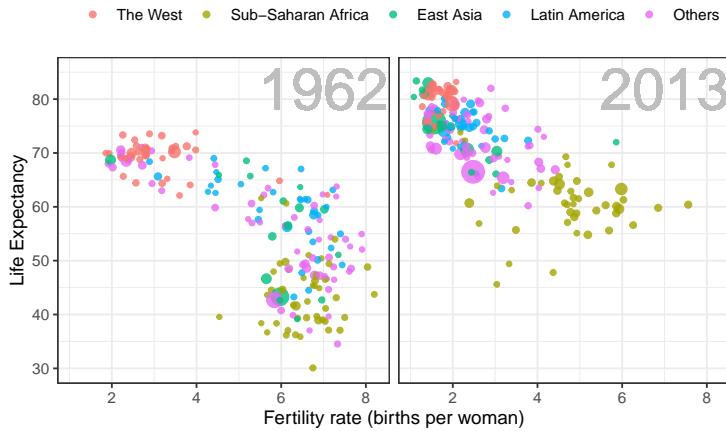
Muchas de las herramientas de análisis de datos más ampliamente utilizadas fueron inicialmente desarrolladas gracias al EDA. Este es quizás la parte más importante del análisis de datos, pero a menudo se ignora.

La visualización de datos ahora también es omnipresente en organizaciones filantrópicas y educativas. En las conferencias “New Insights on Poverty”⁵ y “The Best Stats You’ve Never Seen”⁶, Hans Rosling nos obliga a notar lo inesperado con una serie de gráficos relacionados con la salud y la economía mundial. En sus videos, Rosling usa unos gráficos animados para demostrar cómo el mundo está cambiando y cómo las viejas narrativas ya no son ciertas.

⁴https://en.wikipedia.org/wiki/John_Tukey

⁵https://www.ted.com/talks/hans_rosling_reveals_new_insights_on_poverty?language=en

⁶https://www.ted.com/talks/hans_rosling_shows_the_best_stats_you_ve_ever_seen



También es importante recordar que las equivocaciones, los prejuicios, los errores sistemáticos y otros problemas inesperados a menudo conducen a datos que se deben analizar con cuidado. No descubrir estos problemas puede dar lugar a análisis defectuosos y descubrimientos falsos. Como ejemplo, consideren que los instrumentos de medición a veces fallan y que la mayoría de los procedimientos de análisis de datos no están diseñados para detectarlos. Sin embargo, estos procedimientos aún le darán una respuesta. El hecho de que puede ser difícil, o hasta imposible, notar un error solo a partir de los resultados que se reportan hace que la visualización de datos sea particularmente importante.

En esta parte del libro aprenderemos los conceptos básicos de la visualización de datos y del análisis exploratorio de datos mediante el uso de tres ejemplos motivantes. Usaremos el paquete **ggplot2** para codificar. Para aprender los conceptos básicos, utilizaremos un ejemplo algo artificial: alturas reportadas por estudiantes. Entonces discutiremos dos ejemplos mencionados anteriormente: 1) la salud y economía mundial y 2) las tendencias de enfermedades infecciosas en Estados Unidos.

Por supuesto, la visualización de datos es mucho más de lo que cubrimos aquí. A continuación ofrecemos unas referencias para los que quieran aprender más:

- ER Tufte (1983) *The visual display of quantitative information*. Graphics Press.
- ER Tufte (1990) *Envisioning information*. Graphics Press.
- ER Tufte (1997) *Visual explanations*. Graphics Press.
- WS Cleveland (1993) *Visualizing data*. Hobart Press.
- WS Cleveland (1994) *The elements of graphing data*. CRC Press.
- A Gelman, C Pasarica, R Dodhia (2002) Let's practice what we preach: Turning tables into graphs. *The American Statistician* 56:121-130.
- NB Robbins (2004) *Creating more effective graphs*. Wiley.
- A Cairo (2013) *The functional art: An introduction to information graphics and visualization*. New Riders.
- N Yau (2013) *Data points: Visualization that means something*. Wiley.

Finalmente, no discutiremos gráficos interactivos, un tema demasiado avanzado para este libro. Abajo incluimos algunos recursos útiles para aquellos interesados en aprender más sobre ese tema:

- <https://shiny.rstudio.com>
- <https://d3js.org>

7

ggplot2

La visualización de datos exploratorios es quizás la mayor ventaja de R. Uno puede pasar rápidamente de la idea a los datos al gráfico con un equilibrio único de flexibilidad y facilidad. Por ejemplo, Excel puede ser más fácil que R para algunos gráficos, pero no es tan flexible. D3.js puede ser más flexible y poderoso que R, pero se tarda mucho más en generar una gráfica.

A lo largo del libro, crearemos gráficos usando el paquete **ggplot2**¹.

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

Hay muchas opciones para graficar disponibles en R. De hecho, las capacidades para graficar que vienen con una instalación básica de R ya son bastante útiles. También hay otros paquetes para crear gráficos como **grid** y **lattice**. En este libro decidimos usar **ggplot2** porque divide los gráficos en componentes de una manera que le permite a los principiantes crear gráficos relativamente complejos y estéticamente agradables utilizando una sintaxis intuitiva y relativamente fácil de recordar.

Una razón por la cual **ggplot2** es generalmente más intuitiva para los principiantes es porque usa una gramática de gráficos², el *gg* de **ggplot2**. Esto es análogo a la forma en que aprender gramática puede ayudar a un estudiante construir cientos de oraciones diferentes al aprender solo una pequeña cantidad de verbos, sustantivos y adjetivos, en vez de memorizar cada oración específica. Del mismo modo, al aprender una pequeña cantidad de los componentes básicos de **ggplot2** y de su gramática, podrán crear cientos de gráficos diferentes.

Otra razón por la cual **ggplot2** es fácil para los principiantes es que su comportamiento por defecto se ha elegido cuidadosamente para satisfacer la gran mayoría de los casos y, además, es visualmente agradable. Como resultado, es posible crear gráficos informativos y elegantes con un código relativamente sencillo y legible.

Una limitación de **ggplot2** es que está diseñado para trabajar exclusivamente con tablas de datos en formato *tidy* (donde las filas son observaciones y las columnas son variables). Sin embargo, un porcentaje sustancial de sets de datos con los que los principiantes trabajan están en este formato o pueden convertirse a tal. Una ventaja de este enfoque es que, con tal que nuestros datos estén *tidy*, **ggplot2** simplifica el código de graficar y el aprendizaje de gramática para una variedad de gráficos.

Para usar **ggplot2**, tendrán que aprender varias funciones y argumentos. Estos son difíciles de memorizar, por lo que les recomendamos que tengan a mano la hoja de referencia de ggplot2. Pueden obtener una copia en línea³ o simplemente realizar una búsqueda en internet de “ggplot2 cheat sheet”.

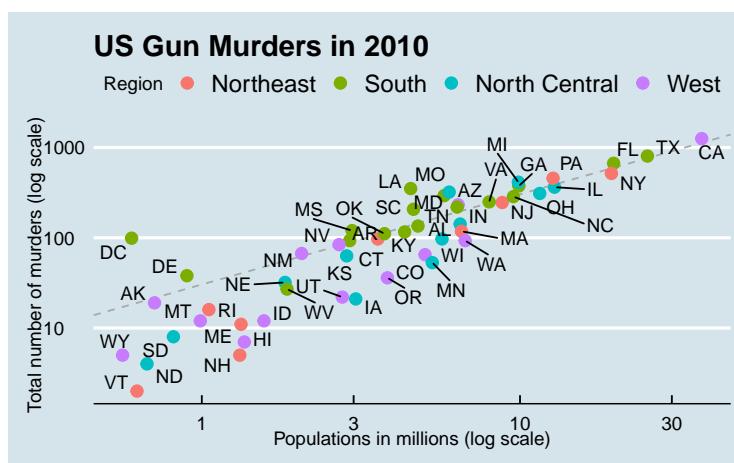
¹<https://ggplot2.tidyverse.org/>

²<http://www.springer.com/us/book/9780387245447>

³<https://github.com/rstudio/cheatsheets/raw/master/translations/spanish/ggplot2.pdf>

7.1 Los componentes de un gráfico

Construiremos un gráfico como el siguiente, que resume el set de datos de asesinatos con armas de fuego en Estados Unidos:



Podemos ver claramente cuánto varían los estados según el tamaño de la población y el número total de asesinatos. No es sorprendente que también se observe una relación clara entre los totales de asesinatos y el tamaño de la población. Un estado que cae en la línea discontinua gris tiene la misma tasa de asesinatos que el promedio de EE. UU. Las cuatro regiones geográficas se denotan con color, que señala cómo la mayoría de los estados del sur tienen tasas de asesinatos por encima del promedio.

Esta visualización de datos nos muestra prácticamente toda la información de la tabla de datos. El código necesario para hacer el gráfico es relativamente sencillo. Aprenderemos a crearlo parte por parte.

El primer paso para aprender **ggplot2** es poder separar un gráfico en componentes. Empezaremos analizando el gráfico anterior e introduciendo algo de la terminología de **ggplot2**. Los tres componentes principales para considerar son:

- **Data:** Se está resumiendo el set de datos de asesinatos con armas de Estados Unidos. Nos referimos a esto como el componente **data**.
 - **Geometría:** El gráfico anterior es un diagrama de dispersión. Esto se denomina el componente de **geometría**. Otras posibles geometrías son diagrama de barras, histograma, densidades suaves (*smooth densities* en inglés), gráfico Q-Q y diagrama de cajas.
 - **Mapeo estético:** El gráfico usa varias señales visuales para representar la información proveída por el set de datos. Las dos señales más importantes en este gráfico son las posiciones de los puntos en el eje-x y el eje-y, que representan el tamaño de la población y el número total de asesinatos, respectivamente. Cada punto representa una observación diferente, y *mapeamos* los datos de estas observaciones y las señales visuales a las escalas x e y. El color es otra señal visual que asignamos a la región. Nos referimos a esto como el componente de **mapeo estético**. La forma en que definimos el mapeo depende de qué **geometría** estamos usando.

También observamos que:

- Los puntos están etiquetados con las abreviaturas de los estados.
- El rango del eje-x y el eje-y parece estar definido por el rango de los datos. Ambos están en escalas logarítmicas.
- Hay etiquetas, un título, una leyenda y utilizamos el estilo de la revista “The Economist”.

Ahora construiremos el gráfico parte por parte. Comenzemos cargando el set de datos:

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

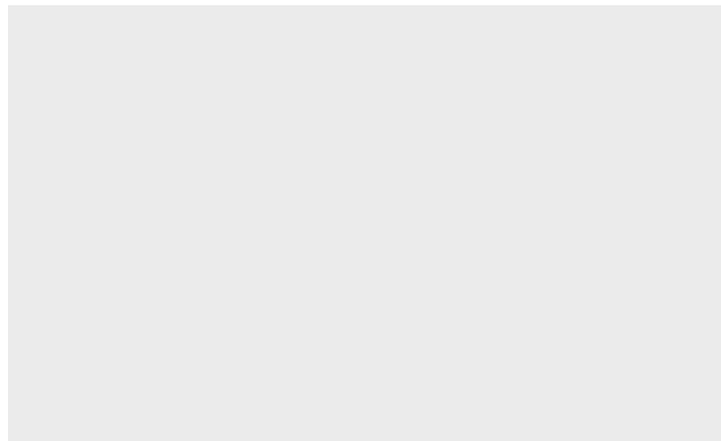
7.2 Objetos ggplot

El primer paso para crear un gráfico **ggplot2** es definir un objeto **ggplot**. Hacemos esto con la función **ggplot**, que inicializa el gráfico. Si leemos la página de ayuda para esta función, vemos que el primer argumento se usa para especificar qué datos están asociados con este objeto:

```
ggplot(data = murders)
```

También podemos *pipe* los datos como primer argumento. Entonces, esta línea de código es equivalente a la anterior:

```
murders %>% ggplot()
```



El código crea un gráfico, en este caso una pizarra en blanco ya que no se ha definido la geometría. La única opción de estilo que vemos es un fondo gris.

Lo que sucedió es que el objeto fue creado y, debido a que no fue asignado, se evaluó automáticamente. Pero podemos asignar nuestro gráfico a un objeto, por ejemplo así:

```
p <- ggplot(data = murders)
class(p)
#> [1] "gg"      "ggplot"
```

Para representar el gráfico asociado con este objeto, simplemente imprimimos el objeto p. Cada una de las siguientes dos líneas de código produce el mismo gráfico que vemos arriba:

```
print(p)
p
```

7.3 Geometrías

En ggplot2 creamos gráficos agregando *capas* (*layers* en inglés). Las capas pueden definir geometrías, calcular estadísticas de resumen, definir qué escalas (*scales* en inglés) usar o incluso cambiar estilos. Para añadir capas, usamos el símbolo +. En general, una línea de código se verá así:

```
DATOS %>% ggplot() + CAPA 1 + CAPA 2 + ... + CAPA N
```

Usualmente, la primera capa que agregamos define la geometría. Queremos hacer un diagrama de dispersión. ¿Qué geometría debemos utilizar?

Echando un vistazo rápido a la hoja de referencia, vemos que la función utilizada para crear gráficos con esta geometría es `geom_point`.



(Imagen cortesía de RStudio⁴. Licencia CC-BY-4.0⁵.)

Los nombres de las funciones de geometría siguen el patrón: `geom_X` donde X es el nombre de la geometría. Algunos ejemplos incluyen `geom_point`, `geom_bar` y `geom_histogram`.

Para que `geom_point` funcione bien, necesitamos proveer datos y una correspondencia.

⁴<https://github.com/rstudio/cheatsheets>

⁵<https://github.com/rstudio/cheatsheets/blob/master/LICENSE>

Ya hemos conectado el objeto `p` con la tabla de datos `murders` y si agregamos la capa `geom_point`, esta por defecto usa los datos de asesinatos. Para saber qué correspondencias se esperan, lean la Sección **Aesthetics** de la página de ayuda de `geom_point`:

```
> Aesthetics
>
> geom_point understands the following aesthetics (required aesthetics are in bold):
>
> x
>
> y
>
> alpha
>
> colour
```

y, como se esperaba, vemos que se requieren al menos dos argumentos: `x` e `y`.

7.4 Mapeos estéticos

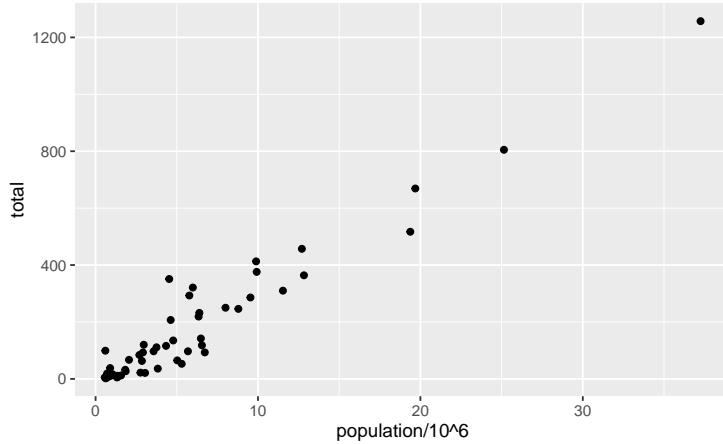
Los **mapeos estéticos** (*aesthetic mappings* en inglés) describen cómo las propiedades de los datos se conectan con las características del gráfico, como la distancia a lo largo de un eje, el tamaño o el color. La función `aes` conecta los datos con lo que vemos en el gráfico mediante la definición de asignaciones estéticas y, por eso, será una de las funciones que más utilizarán al graficar. El resultado de la función `aes` a menudo se utiliza como argumento de una función de geometría. Este ejemplo produce un diagrama de dispersión de asesinatos totales versus población en millones:

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

Podemos quitar el `x` = e `y` = si quisieramos ya que estos son el primer y el segundo argumento esperado, como se ve en la página de ayuda.

En lugar de definir nuestro gráfico desde cero, podemos añadir una capa al objeto `p` que se definió anteriormente como `p <- ggplot(data = murders)`:

```
p + geom_point(aes(population/10^6, total))
```



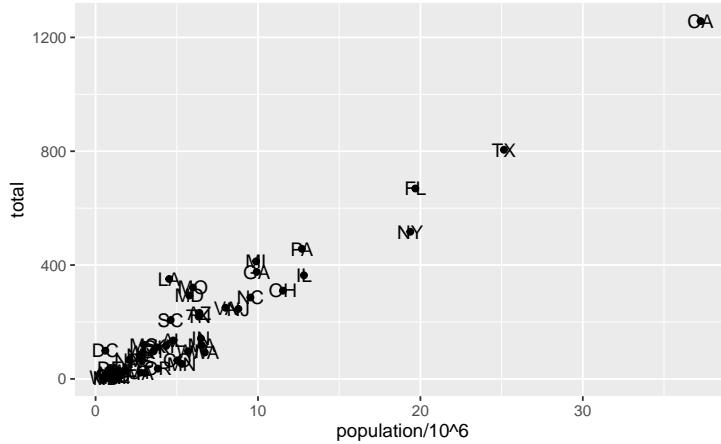
La escala y las etiquetas se definen por defecto al agregar esta capa. Al igual que las funciones de `dplyr`, `aes` también usa los nombres de variables del componente objeto: podemos usar `population` y `total` sin tener que llamarlos como `murders$population` and `murders$total`. El comportamiento de reconocer las variables del componente de datos es específico a `aes`. Con la mayoría de las funciones, si intentan acceder a los valores de `population` o `total` fuera de `aes`, recibirán un error.

7.5 Capas

Una segunda capa en el gráfico que queremos hacer implica añadir una etiqueta a cada punto para identificar el estado. Las funciones `geom_label` y `geom_text` nos permiten añadir texto al gráfico con o sin un rectángulo detrás del texto, respectivamente.

Debido a que cada punto (cada estado en este caso) tiene una etiqueta, necesitamos un mapeo estético para hacer la conexión entre los puntos y las etiquetas. Al leer la página de ayuda, aprendemos que el mapeo entre el punto y la etiqueta se provee a través del argumento `label` de `aes`. Entonces el código se ve así:

```
p + geom_point(aes(population/10^6, total)) +
  geom_text(aes(population/10^6, total, label = abb))
```



Hemos agregado exitosamente una segunda capa al gráfico.

Como ejemplo del comportamiento único de `aes` mencionado anteriormente, observen que esta llamada:

```
p_test <- p + geom_text(aes(population/10^6, total, label = abb))
```

está bien, mientras que esta llamada:

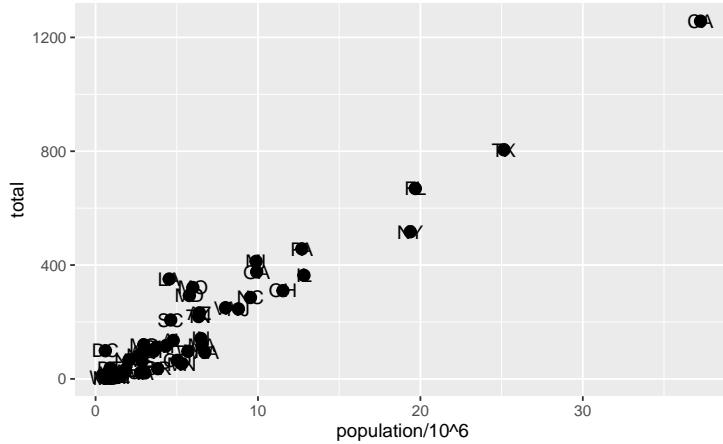
```
p_test <- p + geom_text(aes(population/10^6, total), label = abb)
```

les dará un error ya que `abb` no se encuentra porque está fuera de la función `aes`. La capa `geom_text` no sabe dónde encontrar `abb` porque es un nombre de columna y no una variable global.

7.5.1 Cómo probar varios argumentos

Cada función de geometría tiene muchos otros argumentos además de `aes` y `data`. Estos suelen ser específicos de la función. Por ejemplo, en el gráfico que queremos hacer, los puntos son más grandes que el tamaño predeterminado. En el archivo de ayuda vemos que `size` es una estética y se puede cambiar así:

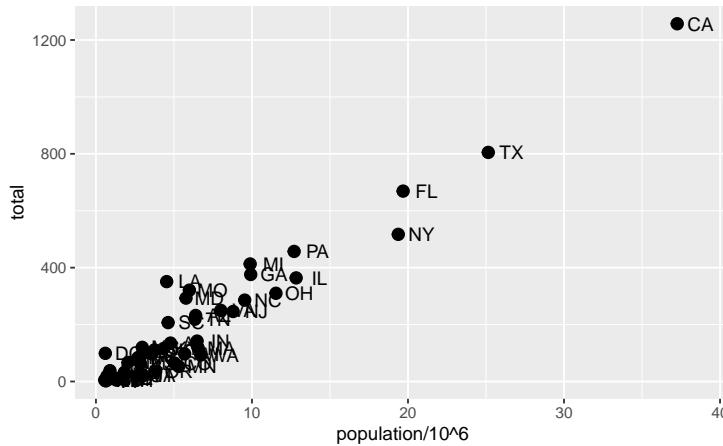
```
p + geom_point(aes(population/10^6, total), size = 3) +
  geom_text(aes(population/10^6, total, label = abb))
```



`size` no es un mapeo: mientras los mapeos usan datos de observaciones específicas y necesitan estar dentro de `aes()`, las operaciones que queremos que afecten a todos los puntos de la misma manera no necesitan ser incluidas dentro `aes`.

Ahora, debido a que los puntos son más grandes, es difícil ver las etiquetas. Si leemos la página de ayuda para `geom_text`, vemos que el argumento `nudge_x` mueve el texto ligeramente hacia la derecha o hacia la izquierda:

```
p + geom_point(aes(population/10^6, total), size = 3) +
  geom_text(aes(population/10^6, total, label = abb), nudge_x = 1.5)
```



Esto es preferible puesto que facilita la lectura del texto. En la Sección 7.11, aprenderemos una mejor manera de asegurarnos de que podemos ver los puntos y las etiquetas.

7.6 Mapeos estéticos globales versus locales

En la línea anterior de código, definimos el mapeo `aes(population/10^6, total)` dos veces, una vez en cada geometría. Podemos evitar esto usando un mapeo estético *global* cuando definimos la pizarra en blanco que nos da el objeto `ggplot`. Recuerden que la función `ggplot` contiene un argumento que nos permite definir mapeos estéticos:

```
args(ggplot)
#> function (data = NULL, mapping = aes(), ..., environment = parent.frame())
#> NULL
```

Si definimos un mapeo en `ggplot`, todas las geometrías que se agregan como capas se asignarán por defecto a este mapeo. Redefinimos `p`:

```
p <- murders %>% ggplot(aes(population/10^6, total, label = abb))
```

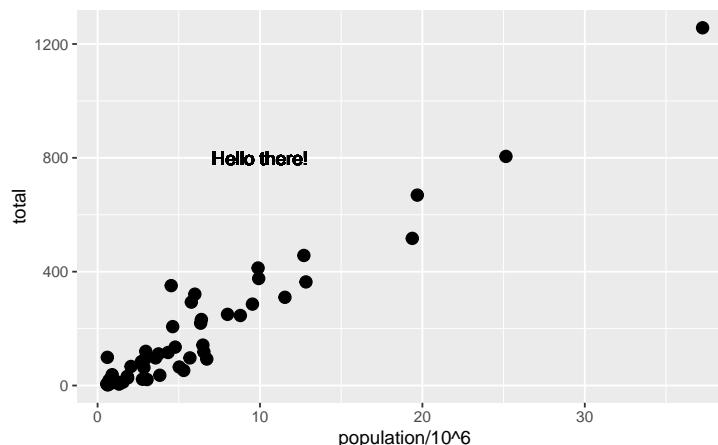
y entonces podemos simplemente escribir el siguiente código para producir el gráfico anterior:

```
p + geom_point(size = 3) +
  geom_text(nudge_x = 1.5)
```

Mantenemos los argumentos `size` y `nudge_x` en `geom_point` y `geom_text`, respectivamente, porque solo queremos aumentar el tamaño de los puntos y ajustar la posición (*nudge* en inglés) de las etiquetas. Si ponemos esos argumentos en `aes`, entonces se aplicarán a ambos gráficos. También tengan en cuenta que la función `geom_point` no necesita un argumento `label` y por lo tanto ignora esa estética.

Si es necesario, podemos anular el mapeo global definiendo un nuevo mapeo dentro de cada capa. Estas definiciones *locales* reemplazan a las *globales*. Aquí hay un ejemplo:

```
p + geom_point(size = 3) +
  geom_text(aes(x = 10, y = 800, label = "Hello there!"))
```

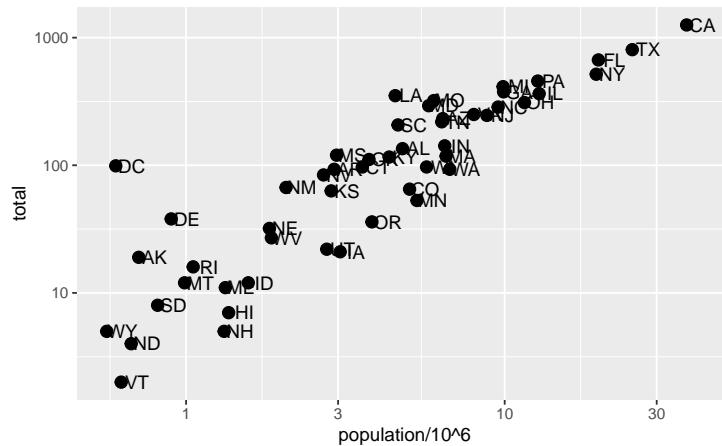


Claramente, la segunda llamada a `geom_text` no usa `population` y `total`.

7.7 Escalas

Primero, las escalas que queremos están en escala logarítmica. Este no es el valor predeterminado, por lo que este cambio debe añadirse a través de una capa de *escalas*. Una mirada rápida a la hoja de referencia revela que la función `scale_x_continuous` nos permite controlar el comportamiento de las escalas. La usamos así:

```
p + geom_point(size = 3) +
  geom_text(nudge_x = 0.05) +
  scale_x_continuous(trans = "log10") +
  scale_y_continuous(trans = "log10")
```



Debido a que ahora estamos en la escala logarítmica, el ajuste a la posición debe hacerse más pequeño.

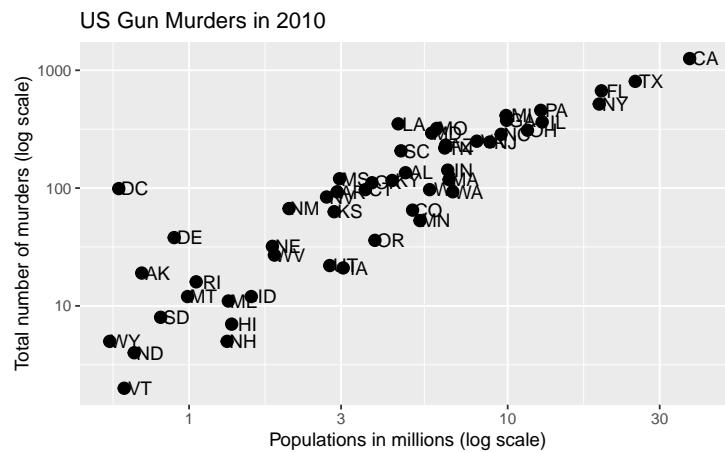
Esta transformación particular es tan común que **ggplot2** ofrece dos funciones especializadas `scale_x_log10` y `scale_y_log10`, que podemos usar para reescribir el código de esta manera:

```
p + geom_point(size = 3) +  
  geom_text(nudge_x = 0.05) +  
  scale_x_log10() +  
  scale_y_log10()
```

7.8 Etiquetas y títulos

Del mismo modo, la hoja de referencia revela que para cambiar las etiquetas y añadir un título, utilizamos las siguientes funciones:

```
p + geom_point(size = 3) +
  geom_text(nudge_x = 0.05) +
  scale_x_log10() +
  scale_y_log10() +
  xlab("Populations in millions (log scale)") +
  ylab("Total number of murders (log scale)") +
  ggtitle("US Gun Murders in 2010")
```



¡Casi terminamos! Lo único que nos falta es añadir color, leyenda y cambios opcionales al estilo.

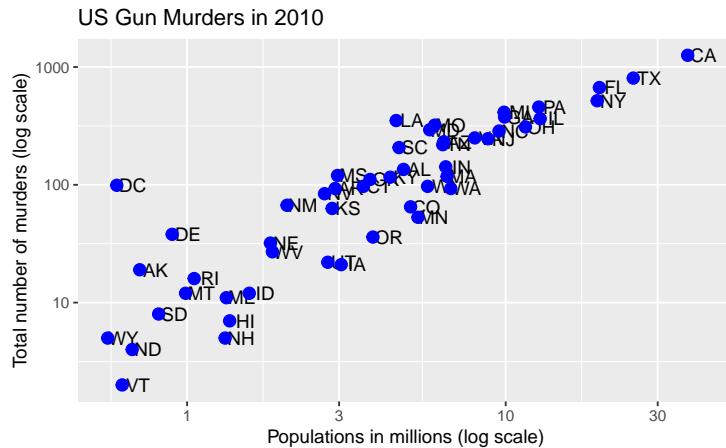
7.9 Categorías como colores

Podemos cambiar el color de los puntos usando el argumento `col` en la función `geom_point`. Para facilitar la demostración de características nuevas, redefiniremos `p` para ser todo excepto la capa de puntos:

```
p <- murders %>% ggplot(aes(population/10^6, total, label = abb)) +
  geom_text(nudge_x = 0.05) +
  scale_x_log10() +
  scale_y_log10() +
  xlab("Populations in millions (log scale)") +
  ylab("Total number of murders (log scale)") +
  ggtitle("US Gun Murders in 2010")
```

y luego probaremos lo que sucede cuando agregamos diferentes llamadas a `geom_point`. Por ejemplo, podemos hacer que todos los puntos sean azules agregando el argumento `color`:

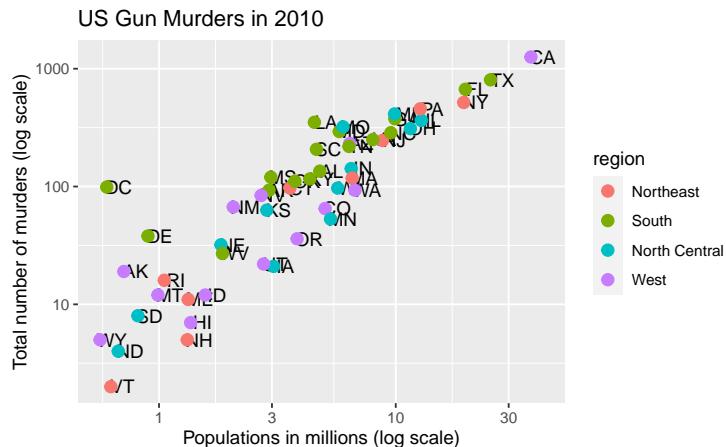
```
p + geom_point(size = 3, color = "blue")
```



Sin embargo, no queremos esto. Queremos asignar color según la región geográfica. Un buen comportamiento por defecto de **ggplot2** es que si asignamos una variable categórica al color, automáticamente asigna un color diferente a cada categoría, además de una leyenda.

Dado que la elección del color está determinada por una característica de cada observación, este es un mapeo estético. Para asignar un color a cada punto, necesitamos usar `aes`. Usamos el siguiente código:

```
p + geom_point(aes(col=region), size = 3)
```



Los mapeos `x` e `y` se heredan de esos ya definidos en `p`, así que no los redefinimos. También movemos `aes` al primer argumento, dado que ahí es donde se esperan los mapeos en esta llamada.

Aquí vemos otro comportamiento útil por defecto: **ggplot2** automáticamente agrega una leyenda que asigna el color a la región. Si no quieren añadir esta leyenda, establecemos el argumento **geom_point** como **show.legend = FALSE**.

7.10 Anotación, formas y ajustes

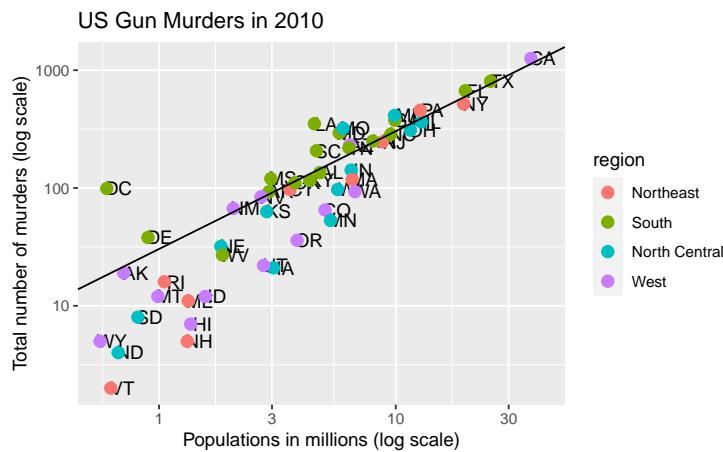
A menudo queremos añadir formas o anotaciones a las figuras que no se derivan directamente del mapeo estético; algunos ejemplos incluyen etiquetas, cuadros, áreas sombreadas y líneas.

Aquí queremos añadir una línea que represente la tasa promedio de asesinatos en todo el país. Una vez que determinemos la tasa por millón a ser r , esta línea se define por la fórmula: $y = rx$, con y y x nuestros ejes: asesinatos totales y población en millones, respectivamente. En la escala logarítmica, esta línea se convierte en: $\log(y) = \log(r) + \log(x)$. Entonces, en nuestro gráfico, es una línea con pendiente 1 e intercepto $\log(r)$. Para calcular este valor, utilizamos nuestros conocimientos de `dplyr`:

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

Para añadir una línea, usamos la función `geom_abline`. `ggplot2` utiliza `ab` en el nombre para recordarnos que estamos suministrando el intercepto (`a`) y el pendiente (`b`). La línea predeterminada tiene pendiente 1 e intercepto 0, por lo que solo tenemos que definir el intercepto:

```
p + geom_point(aes(col=region), size = 3) +
  geom_abline(intercept = log10(r))
```



Aquí `geom_abline` no utiliza ninguna información del objeto de datos.

Podemos cambiar el tipo de línea y el color de las líneas usando argumentos. Además, la dibujamos primero para que no tape nuestros puntos.

```
p <- p + geom_abline(intercept = log10(r), lty = 2, color = "darkgrey") +
  geom_point(aes(col=region), size = 3)
```

Noten que hemos redefinido `p` y usaremos esta nueva `p` a continuación y en la siguiente sección.

Los gráficos por defecto creados por **ggplot2** ya son muy útiles. Sin embargo, con frecuencia necesitamos hacer pequeños ajustes al comportamiento predeterminado. Aunque no siempre es obvio cómo hacer esto aun con la hoja de referencia, **ggplot2** es muy flexible.

Por ejemplo, podemos hacer cambios a la leyenda a través de la función `scale_color_discrete`. En nuestro gráfico original, la palabra *region* está en mayúscula y podemos cambiarla así:

```
p <- p + scale_color_discrete(name = "Region")
```

7.11 Paquetes complementarios

El poder de **ggplot2** se incrementa aún más debido a la disponibilidad de paquetes adicionales. Los cambios restantes necesarios para darle los toques finales a nuestro gráfico requieren los paquetes **ggthemes** y **ggrepel**.

El estilo de un gráfico **ggplot2** se puede cambiar usando las funciones de `theme`. Se incluyen varios temas (*themes* en inglés) como parte del paquete **ggplot2**. De hecho, para la mayoría de los gráficos de este libro, utilizamos una función del paquete **dslabs** que automáticamente establece un tema por defecto:

```
ds_theme_set()
```

El paquete **ggthemes** añade muchos otros temas, incluso el tema `theme_economist` que escogimos. Después de instalar el paquete, pueden cambiar el estilo agregando una capa como la siguiente:

```
library(ggthemes)
p + theme_economist()
```

Pueden ver cómo se ven algunos de los otros temas simplemente cambiando la función. Por ejemplo, pueden probar el tema `theme_fivethirtyeight()` en vez del anterior.

La diferencia final tiene que ver con la posición de las etiquetas. En nuestro gráfico, algunas de las etiquetas se superponen. El paquete de complementos **ggrepel** incluye una geometría que añade etiquetas a la vez que garantiza que no se superpongan entre sí. Para utilizarla, simplemente cambiamos `geom_text` a `geom_text_repel`.

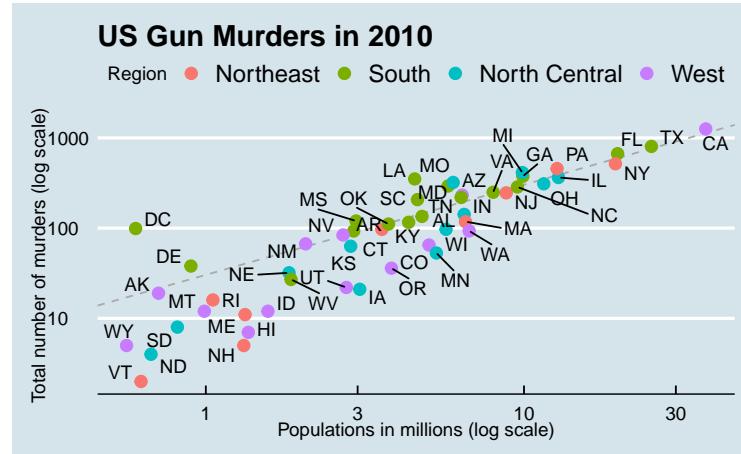
7.12 Cómo combinarlo todo

Ahora que hemos terminado las pruebas, podemos escribir un código que produzca nuestro gráfico deseado partiendo de cero.

```
library(ggthemes)
library(ggrepel)

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

murders %>% ggplot(aes(population/10^6, 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("Populations in millions (log scale)") +
  ylab("Total number of murders (log scale)") +
  ggtitle("US Gun Murders in 2010") +
  scale_color_discrete(name = "Region") +
  theme_economist()
```



7.13 Gráficos rápidos con qplot

Hemos aprendido las técnicas eficaces **ggplot2** para generar visualizaciones. Sin embargo, hay casos en que sólo necesitamos un gráfico rápido de, por ejemplo, un histograma de los valores en un vector, un diagrama de dispersión de los valores en dos vectores o un diagrama de caja usando vectores categóricos y numéricos. Ya hemos demostrado cómo generar estos gráficos con **hist**, **plot** y **boxplot**. Sin embargo, si queremos ser consistentes con el estilo de ggplot, podemos usar la función **qplot**.

Si tenemos valores en dos vectores como:

```
data(murders)
x <- log10(murders$population)
y <- murders$total
```

y queremos hacer un diagrama de dispersión con **ggplot2**, tendríamos que escribir algo como:

```
data.frame(x = x, y = y) %>%
  ggplot(aes(x, y)) +
  geom_point()
```

Esto parece ser demasiado código para una gráfico tan sencillo. La función **qplot** sacrifica la flexibilidad ofrecida por el enfoque de **ggplot2**, pero nos permite rápidamente generar un gráfico.

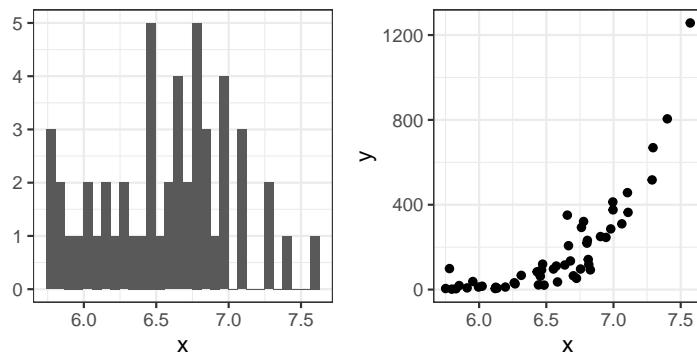
```
qplot(x, y)
```

Aprenderemos más sobre **qplot** en la Sección 8.16

7.14 Cuadrículas de gráficos

A menudo tenemos que poner gráficos uno al lado del de otro. El paquete **gridExtra** nos permite hacer eso:

```
library(gridExtra)
p1 <- qplot(x)
p2 <- qplot(x,y)
grid.arrange(p1, p2, ncol = 2)
```



7.15 Ejercicios

Comience cargando los paquetes `dplyr` y `ggplot2`, así como los datos `murders` y `heights`.

```
library(dplyr)
library(ggplot2)
library(dslabs)
data(heights)
data(murders)
```

1. Con `ggplot2`, los gráficos se pueden guardar como objetos. Por ejemplo, podemos asociar un set de datos con un objeto de gráfico así:

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

Como `data` es el primer argumento, no necesitamos explicarlo:

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

y también podemos usar el *pipe*:

```
p <- murders %>% ggplot()
```

¿Cuál es la clase del objeto `p`?

2. Recuerde que para imprimir un objeto puede usar el comando `print` o simplemente escribir el objeto. Imprima el objeto `p` definido en el ejercicio uno y describa lo que ve.

- a. No pasa nada.
- b. Una gráfica de pizarra en blanco.
- c. Un diagrama de dispersión.
- d. Un histograma.

3. Usando el *pipe* `%>%`, cree un objeto `p` pero esta vez asociado con el set de datos `heights` en lugar del set de datos `murders`.

4. ¿Cuál es la clase del objeto `p` que acaba de crear?

5. Ahora vamos a añadir una capa y las mapeos estéticos correspondientes. Para los datos de asesinatos, graficamos asesinatos totales versus tamaños de población. Explore el set de datos `murders` para recordar cuáles son los nombres de estas dos variables y escoja la respuesta correcta. **Sugerencia:** Mire `?murders`.

- a. `state` y `abb`
- b. `total_murders` y `population_size`
- c. `total` y `population`
- d. `murders` y `size`

6. Para crear el diagrama de dispersión, agregamos una capa con `geom_point`. Los mapeos estéticos requieren que definamos las variables del eje-x y del eje-y, respectivamente. Entonces el código se ve así:

```
murders %>% ggplot(aes(x = , y = )) +  
  geom_point()
```

excepto que tenemos que definir las dos variables `x` e `y`. Llene el espacio con los nombres correctos de las variables.

7. Recuerde que si no usamos nombres de argumentos, podemos obtener el mismo gráfico si ingresamos los nombres de las variables en el orden correcto de esta manera:

```
murders %>% ggplot(aes(population, total)) +  
  geom_point()
```

Vuelva a hacer el gráfico pero ahora con `total` en el eje-x y población en el eje-y.

8. Si en lugar de puntos queremos añadir texto, podemos usar las geometrías `geom_text()` o `geom_label()`. El siguiente código:

```
murders %>% ggplot(aes(population, total)) + geom_label()
```

nos dará el mensaje de error: `Error: geom_label requires the following missing aesthetics: label`

¿Por qué ocurre esto?

- Necesitamos mapear un carácter a cada punto a través del argumento de etiqueta en `aes`.
- Necesitamos dejar que `geom_label` sepa qué carácter usar en el gráfico.
- La geometría `geom_label` no requiere valores del eje-x y del eje-y.
- `geom_label` no es un comando de `ggplot2`.

9. Reescriba el código anterior para que use abreviaturas como el `label` (la etiqueta) a través de `aes`.

10. Cambie el color de las etiquetas a azul. ¿Cómo se hace?

- Agregando una columna llamada `blue` a `murders`.
- Debido a que cada etiqueta necesita un color diferente, mapeamos los colores a través de `aes`.
- Utilizando el argumento `color` en `ggplot`.
- Como queremos que todos los colores sean azules, no necesitamos asignar colores, solo usar el argumento `color` en `geom_label`.

11. Reescriba el código anterior para que las etiquetas sean azules.

12. Ahora suponga que queremos usar color para representar las diferentes regiones. En este caso, ¿cuál de los siguientes es el más apropiado?

- Añadir una columna llamada `color` a `murders` con el color que queremos usar.

- b. Como cada etiqueta necesita un color diferente, mapear los colores a través del argumento de color de `aes`.
 - c. Utilizar el argumento `color` en `ggplot`.
 - d. Como queremos que todos los colores sean azules, no necesitamos asignar colores, solo usar el argumento de color en `geom_label`.
13. Reescriba el código anterior para que el color de las etiquetas sea determinado por la región del estado.
14. Ahora vamos a cambiar el eje-x a una escala logarítmica para tomar en cuenta el hecho de que la distribución de la población es asimétrica. Comencemos definiendo un objeto `p` guardando el gráfico que hemos hecho hasta ahora:
- ```
p <- murders %>%
 ggplot(aes(population, total, label = abb, color = region)) +
 geom_label()
```
- Para cambiar el eje-y a una escala logarítmica, aprendimos sobre la función `scale_x_log10()`. Agregue esta capa al objeto `p` para cambiar la escala y crear el gráfico.
15. Repita el ejercicio anterior pero ahora cambie ambos ejes para que estén en la escala logarítmica.
  16. Ahora edite el código anterior para añadir el título “Gun murder data” al argumento. Sugerencia: use la función `ggtitle`.



# 8

---

## Cómo visualizar distribuciones de datos

---

Los datos numéricos a menudo se resumen con el valor *promedio*. Por ejemplo, la calidad de una escuela secundaria a veces se resume con un solo número: la puntuación promedio en una prueba estandarizada. Ocasionalmente, se incluye un segundo número: la *desviación estándar*. Por ejemplo, pueden leer un informe que indique que las puntuaciones fueron 680 más o menos 50 (la desviación estándar). El informe ha resumido un vector completo de puntuaciones con solo dos números. ¿Es esto apropiado? ¿Hay alguna información importante que no estamos considerando al ver este resumen en lugar de la lista completa?

Nuestro primer componente básico de visualización de datos es aprender a resumir listas de factores o vectores numéricos. Generalmente, la mejor manera de compartir o explorar este resumen es a través de la visualización de datos. El resumen estadístico más básico de una lista de objetos o números es su distribución. Una vez que un vector se haya resumido como una distribución, existen varias técnicas de visualización de datos para transmitir esta información de manera efectiva.

En este capítulo, primero discutiremos las propiedades de una variedad de distribuciones y cómo visualizar las distribuciones usando un ejemplo motivante de alturas de estudiantes. Luego, en la Sección 8.16, discutiremos las geometrías de **ggplot2** para estas visualizaciones.

---

### 8.1 Tipos de variables

Trabajaremos con dos tipos de variables: categóricas y numéricas. Cada uno puede dividirse en otros dos grupos: las variables categóricas pueden ser ordinales o no, mientras que las numéricas pueden ser discretas o continuas.

Cuando cada entrada en un vector proviene de uno de un pequeño número de grupos, nos referimos a los datos como *datos categóricos*. Dos ejemplos sencillos son el sexo (masculino o femenino) y las regiones (noreste, sur, norte central, oeste). Algunos datos categóricos se pueden ordenar aunque no sean números, por ejemplo cuán picante es una comida (poco, medio, muy). En los libros de texto de estadísticas, los datos categóricos ordenados se denominan datos *ordinales*.

Ejemplos de datos numéricos son el tamaño de la población, las tasas de asesinatos y las alturas. Algunos datos numéricos se pueden tratar como ordenados categóricos. Podemos dividir aún más los datos numéricos en continuos y discretos. Las variables continuas son aquellas que pueden tomar cualquier valor, como las alturas, si se miden con suficiente precisión. Por ejemplo, un par de gemelos pueden medir 68.12 y 68.11 pulgadas, respectivamente. Los conteos, como el tamaño de la población, son discretos porque tienen que ser números redondos.

Tengan en cuenta que los datos numéricos discretos pueden considerarse ordinales. Aunque

esto es técnicamente cierto, generalmente reservamos el término datos ordinales para variables que pertenecen a un pequeño número de grupos diferentes, y cada grupo tiene muchos miembros. En cambio, cuando tenemos muchos grupos con pocos casos en cada grupo, generalmente nos referimos a ellos como variables numéricas discretas. Entonces, por ejemplo, el número de paquetes de cigarrillos que una persona fuma al día, redondeado al paquete más cercano, se consideraría ordinal, mientras que el número real de cigarrillos se consideraría una variable numérica. Sin embargo, hay ejemplos que pueden considerarse tanto numéricos como ordinales cuando se trata de visualizar datos.

---

## 8.2 Estudio de caso: describiendo alturas de estudiantes

Aquí presentamos un nuevo problema motivante. Es artificial, pero nos ayudará ilustrar los conceptos necesarios para comprender las distribuciones.

Imaginen que tenemos que describir las alturas de nuestros compañeros de clase a ET, un extraterrestre que nunca ha visto humanos. Como primer paso, necesitamos recopilar datos. Para hacer esto, les pedimos a los estudiantes que indiquen sus alturas en pulgadas. Les pedimos que nos provean información sobre su sexo biológico porque sabemos que hay dos distribuciones diferentes por sexo. Recopilamos los datos y los guardamos en el set de datos `heights`:

```
library(tidyverse)
library(dslabs)
data(heights)
```

Una forma de transmitir las alturas a ET es simplemente enviarle esta lista de 1,050 alturas. Sin embargo, hay formas mucho más efectivas de transmitir la información y, para lograr esto, nos ayudará a entender el concepto de distribuciones. Para simplificar la explicación, primero nos enfocamos en las alturas masculinas. Examinamos los datos de altura femenina en la Sección 8.14.

---

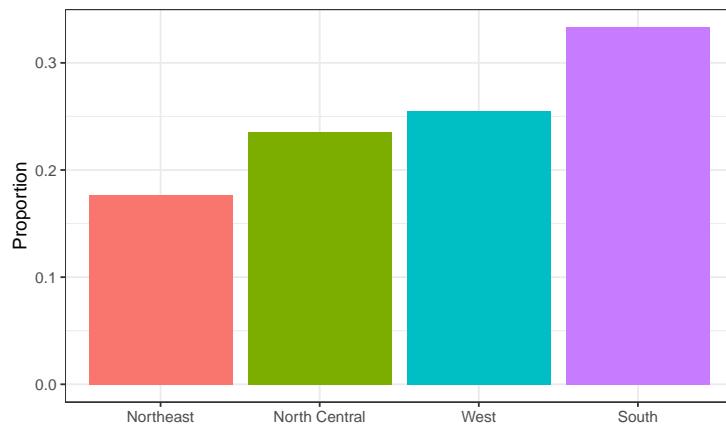
## 8.3 La función de distribución

Resulta que, en algunos casos, el promedio y la desviación estándar son prácticamente todo lo que necesitamos para comprender los datos. Aprenderemos técnicas de visualización de datos que nos ayudarán a determinar cuándo este resumen de dos números es apropiado. Estas mismas técnicas servirán como una alternativa para cuando dos números no son suficientes.

El resumen estadístico más básico de una lista de objetos o números es su distribución. La forma más sencilla de pensar en una distribución es como una descripción compacta de una lista con muchas entradas. Este concepto no debería ser nuevo para los lectores de este libro. Por ejemplo, con datos categóricos, la distribución simplemente describe la proporción de cada categoría única. El sexo representado en el set de datos de alturas es:

```
#>
#> Female Male
#> 0.227 0.773
```

Esta *tabla de frecuencia* de dos categorías es la forma más sencilla de una distribución. Realmente no necesitamos visualizarla ya que un número describe todo lo que necesitamos saber: 23% son mujeres y el resto son hombres. Cuando hay más categorías, un diagrama de barras sencillo describe la distribución. Aquí hay un ejemplo con las regiones estatales de EE. UU.:



Este gráfico simplemente nos muestra cuatro números, uno para cada categoría. Usualmente usamos diagramas de barras cuando tenemos pocos números. Aunque este gráfico en particular no ofrece mucha más información que una tabla de frecuencias en sí, es un primer ejemplo de cómo convertimos un vector en un gráfico que resume de manera sucinta toda la información en el vector. Cuando los datos son numéricos, la tarea de mostrar distribuciones es más retante.

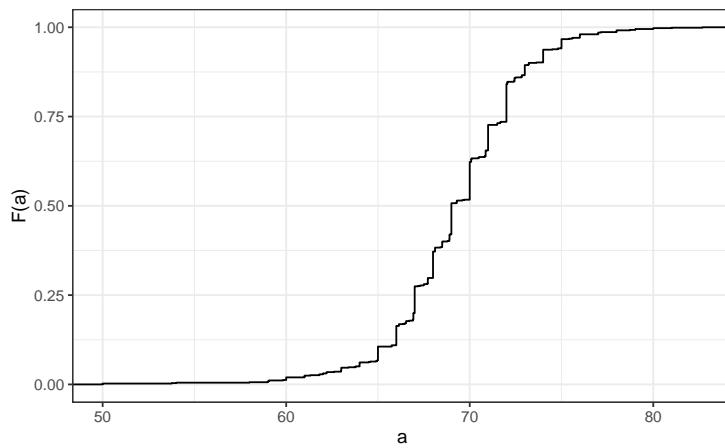
## 8.4 Funciones de distribución acumulada

Los datos numéricos que no son categóricos también tienen distribuciones. En general, cuando los datos no son categóricos, indicar la frecuencia de cada entrada no es un resumen efectivo, ya que la mayoría de las entradas son únicas. En nuestro estudio de caso, por ejemplo, mientras varios estudiantes reportaron una altura de 68 pulgadas, un estudiante indicó una altura de 68.503937007874 pulgadas y otro una altura 68.8976377952756 pulgadas. Suponemos que convirtieron sus alturas de 174 y 175 centímetros, respectivamente.

Los libros de texto de estadísticas nos enseñan que una forma más útil de definir una distribución de datos numéricos es definir una función que indique la proporción de los datos a continuación  $a$  para todos los valores posibles de  $a$ . Esta función se llama la *función de distribución acumulada* (*cumulative distribution function* o CDF por sus siglas en inglés). En estadística, se usa la siguiente notación:

$$F(a) = \Pr(x \leq a)$$

Aquí vemos un gráfico de  $F$  para los datos de altura masculina:



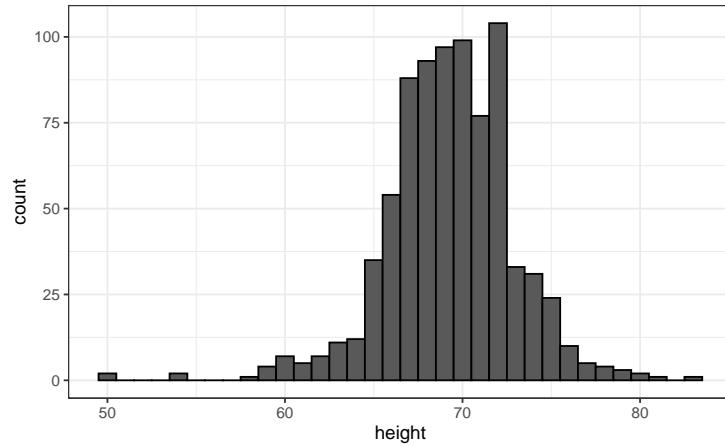
Similar a lo que hace la tabla de frecuencias para datos categóricos, la CDF define la distribución de datos numéricos. En el gráfico, podemos ver que 16% de los valores son menos de 65, ya que  $F(66) = 0.164$ , o que 84% de los valores son menos de 72, ya que  $F(72) = 0.841$ , y así. De hecho, podemos informar la proporción de valores entre dos alturas, digamos  $a$  y  $b$ , al calcular  $F(b) - F(a)$ . Esto significa que si enviamos este diagrama a ET, tendrá toda la información necesaria para reconstruir la lista completa. Parafraseando la expresión “una imagen vale más que mil palabras”, en este caso una imagen es tan informativa como 812 números.

Una nota final: debido a que las CDF pueden definirse matemáticamente, la palabra *empírica* se añade para distinguir cuando se usan los datos. Por lo tanto, utilizamos el término CDF empírico (o eCDF por sus siglas en inglés).

## 8.5 Histogramas

Aunque el concepto de CDF se discute ampliamente en los libros de texto de estadística, el gráfico no es muy popular en la práctica. La razón principal es que no transmite fácilmente características de interés como: ¿en qué valor se centra la distribución? ¿La distribución es simétrica? ¿Qué rangos contienen el 95% de los valores? Los histogramas son preferidos porque facilitan enormemente la respuesta a tales preguntas. Los histogramas sacrifican solo un poco de información para producir gráficos que son mucho más fáciles de interpretar.

La forma más sencilla de hacer un histograma es dividir el *span* de nuestros datos en *compartimientos* (*bins* en inglés) no superpuestos del mismo tamaño. Entonces para cada compartimiento, contamos el número de valores que se encuentran en ese intervalo. El histograma grafica estos conteos como barras con la base de la barra definida por los intervalos. A continuación tenemos el histograma para los datos de altura que dividen el rango de valores en intervalos de una pulgada: [49.5, 50.5], [51.5, 52.5], (53.5, 54.5], ..., (82.5, 83.5].



Como pueden ver en la figura de arriba, un histograma es similar a un diagrama de barras, pero difiere en que el eje-x es numérico, no categórico.

Si le enviamos este diagrama a ET, inmediatamente aprenderá algunos detalles importantes sobre nuestros datos. Primero, el rango de datos es de 50 a 84 con la mayoría (más del 95%) entre 63 y 75 pulgadas. Además, las alturas son casi simétricas alrededor de 69 pulgadas. Por último, al sumar conteos, ET puede obtener una muy buena aproximación de la proporción de los datos en cualquier intervalo. Por lo tanto, el histograma anterior no solo es fácil de interpretar, sino que también ofrece casi toda la información contenida en la lista cruda de 812 alturas con aproximadamente 30 conteos, uno por cada compartimiento.

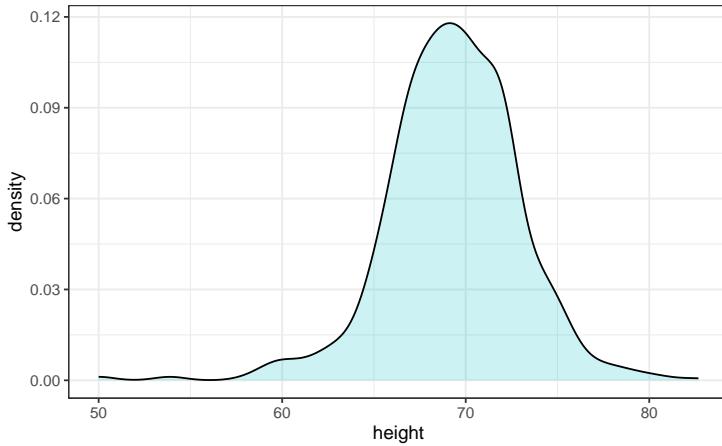
¿Qué información perdemos? Recuerden que todos los valores en cada intervalo se tratan de la misma manera cuando se calculan las alturas del compartimiento. Entonces, por ejemplo, el histograma no distingue entre 64, 64.1 y 64.2 pulgadas. Dado que estas diferencias son casi imperceptibles a la vista, las implicaciones prácticas son insignificantes y pudimos resumir los datos en solo 23 números.

Discutimos cómo codificar histogramas en la Sección 8.16.

---

## 8.6 Densidad suave

Los gráficos de densidad suave son estéticamente más atractivos que los histogramas. A continuación vemos un gráfico de densidad suave para nuestros datos de altura:

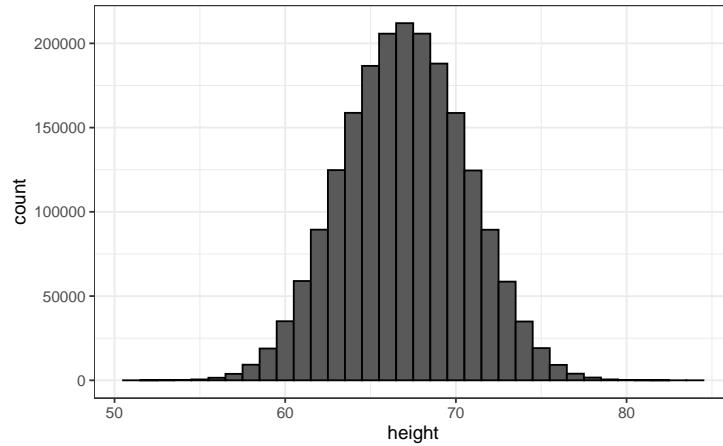


En este gráfico, ya no tenemos bordes afilados en los límites de intervalo y se han eliminado muchos de los picos locales. Además, la escala del eje-y cambió de conteos a *densidad*.

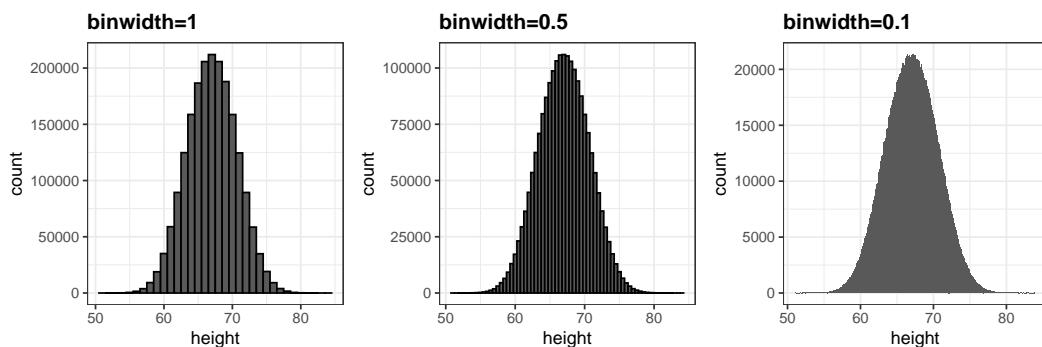
Para entender las densidades suaves, tenemos que entender *estimadores*, un tema que no abordamos hasta más tarde. Sin embargo, ofrecemos una explicación heurística para ayudarles a entender los conceptos básicos y así poder utilizar esta herramienta útil de visualización de datos.

El nuevo concepto principal que deben entender es que suponemos que nuestra lista de valores observados es un subconjunto de una lista mucho más grande de valores no observados. En el caso de las alturas, pueden imaginar que nuestra lista de 812 estudiantes varones proviene de una lista hipotética que contiene todas las alturas de todos los estudiantes varones en todo el mundo, medidos todos con mucha precisión. Digamos que hay 1,000,000 de estas medidas. Esta lista de valores tiene una distribución, como cualquier lista de valores, y esta distribución considerable es realmente lo que queremos transmitir a ET, ya que es mucho más general. Desafortunadamente, no podemos ver esa lista grandísima.

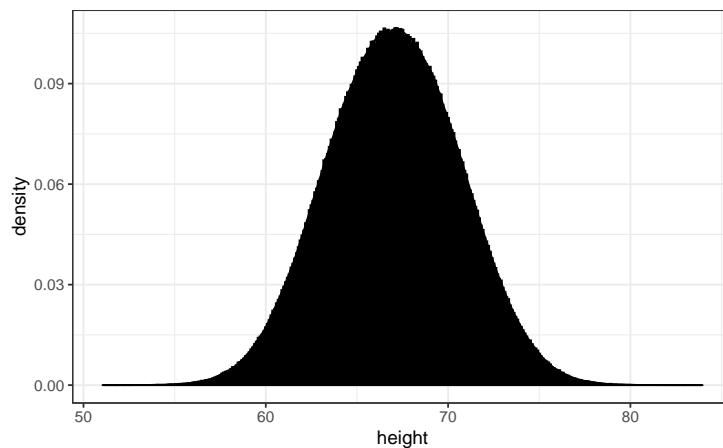
Sin embargo, podemos hacer una suposición que quizás nos ayude a aproximarla. Si tuviéramos 1,000,000 valores, medidos con mucha precisión, podríamos hacer un histograma con compartimientos muy, muy pequeños. La suposición es que si mostramos esto, la altura de los compartimientos consecutivos será similar. Esto es lo que queremos decir con suave: no tenemos grandes saltos en las alturas de los compartimientos consecutivos. A continuación tenemos un histograma hipotético con compartimientos de tamaño 1:



Entre más pequeños hacemos los compartimientos, más suave se vuelve el histograma. Aquí están los histogramas con ancho de compartimiento de 1, 0.5 y 0.1:

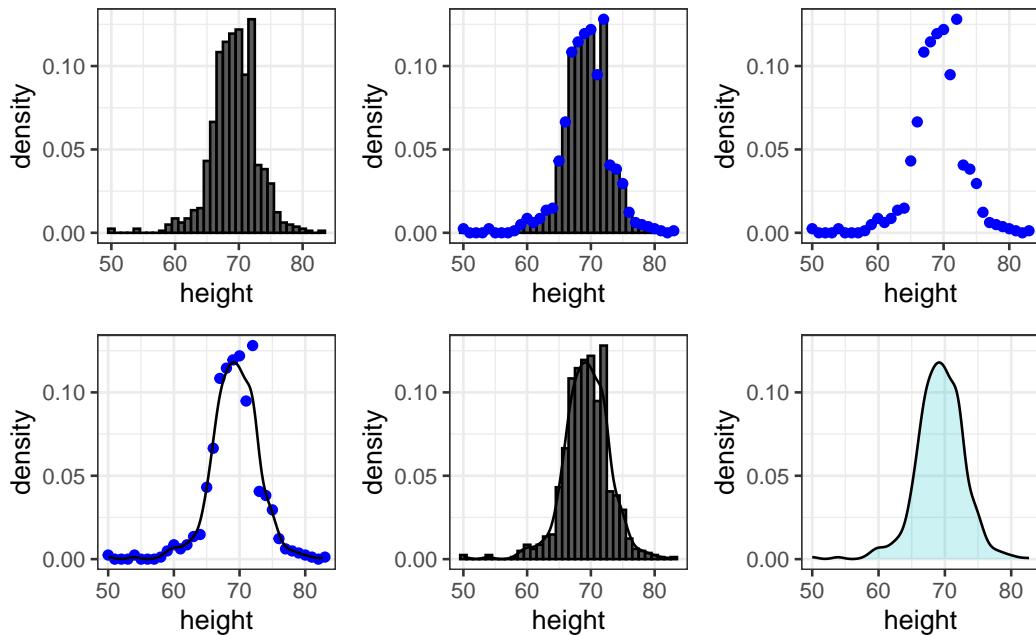


La densidad suave es básicamente la curva que atraviesa la parte superior de las barras del histograma cuando los compartimientos son muy, muy pequeños. Para que la curva no dependa del tamaño hipotético de la lista hipotética, calculamos la curva usando frecuencias en lugar de conteos:

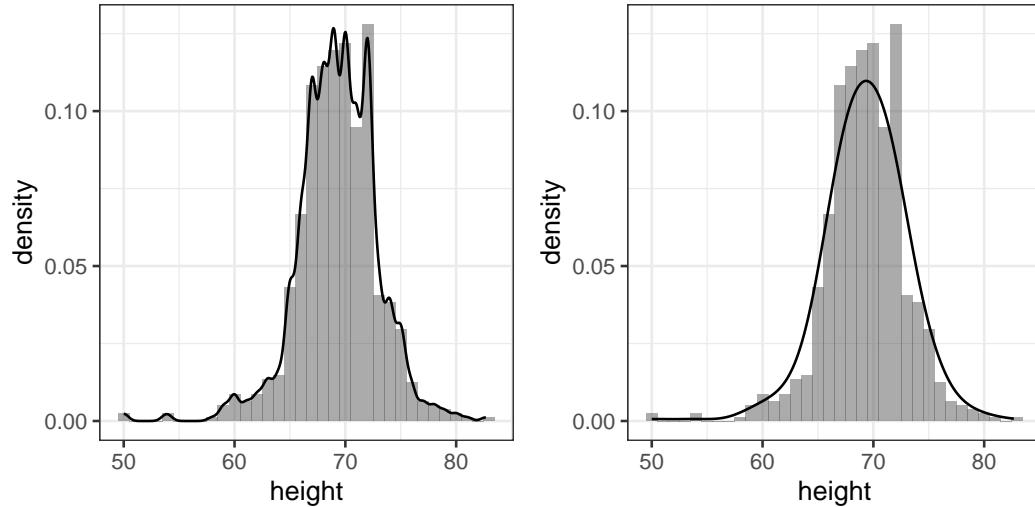


Ahora, de vuelta a la realidad. No tenemos millones de medidas. En cambio, tenemos 812 y no podemos hacer un histograma con compartimientos muy pequeños.

Por lo tanto, hacemos un histograma utilizando tamaños de compartimiento apropiados para nuestros datos y frecuencias de cómputo en lugar de conteos. Además, dibujamos una curva suave que pasa por la parte superior de las barras del histograma. Los siguientes gráficos muestran los pasos que conducen a una densidad suave:



Sin embargo, recuerden que *suave* (*smooth* en inglés) es un término relativo. De hecho, podemos controlar la *suavidad* de la curva cambiando el número de puntos en los compartimientos. Esta opción se conoce como *bandwidth*, *span*, *window size* o, en español, *ventana de suavizado* o *parámetro de suavizado*, y se puede ajustar en la función que calcula la curva de densidad suave. Aquí hay dos ejemplos que usan diferentes niveles de suavidad en el mismo histograma:

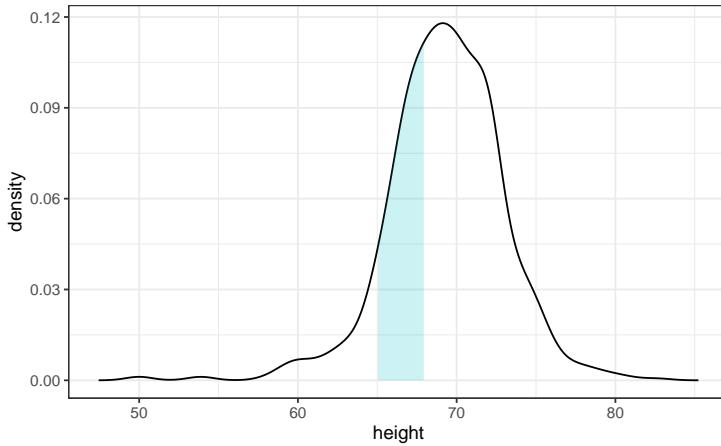


Necesitamos tomar esta decisión con cuidado ya que las visualizaciones que resultan pueden cambiar nuestra interpretación de los datos. Debemos seleccionar un grado de suavidad que podamos defender como representativo de los datos subyacentes. En el caso de la altura, realmente tenemos razones para creer que la proporción de personas con alturas similares debería ser la misma. Por ejemplo, la proporción que es 72 pulgadas debería ser más similar a la proporción que es 71 que a la proporción que es 78 o 65. Esto implica que la curva debe ser bastante suave; es decir, la curva debería parecerse más al ejemplo de la derecha que al de la izquierda.

Si bien el histograma es un resumen sin supuestos, la densidad suavizada se basa en algunos supuestos.

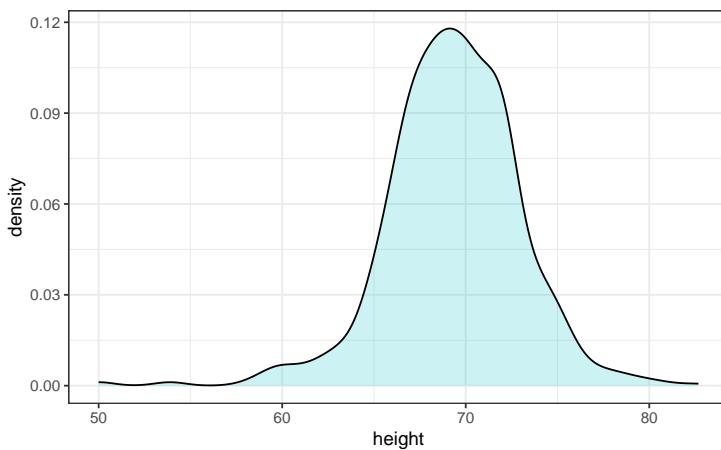
### 8.6.1 Cómo interpretar el eje-y

Tengan en cuenta que interpretar el eje-y de un gráfico de densidad suave no es obvio. Se escala para que el área bajo la curva de densidad se sume a 1. Si imaginan que formamos un compartimiento con una base de 1 unidad de longitud, el valor del eje-y nos indica la proporción de valores en ese compartimiento. Sin embargo, esto solo es cierto para compartimientos de tamaño 1. Para intervalos de otro tamaño, la mejor manera de determinar la proporción de datos en ese intervalo es calculando la proporción del área total contenida en ese intervalo. Por ejemplo, aquí vemos la proporción de valores entre 65 y 68:



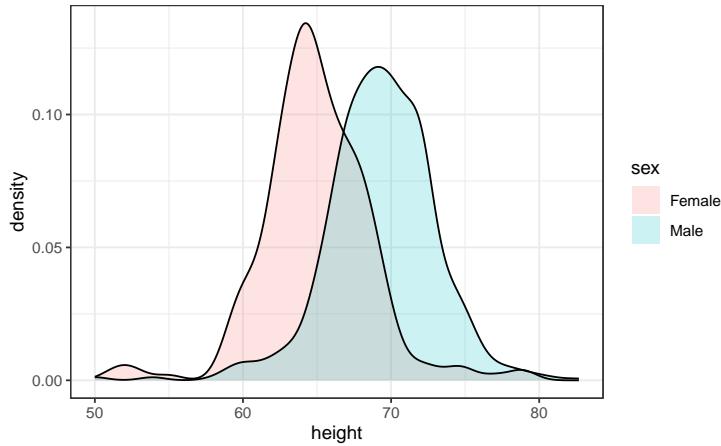
La proporción de esta área es aproximadamente 0.3, lo que significa que aproximadamente 30% de las alturas masculinas están entre 65 y 68 pulgadas.

Al comprender esto, estamos listos para usar la densidad suave como resumen. Para este set de datos, nos sentimos bastante cómodos suponiendo suavidad y, por ende, compartiendo esta figura estéticamente agradable con ET, que puede usarla para comprender nuestros datos de alturas masculinas:



### 8.6.2 Densidades permiten estratificación

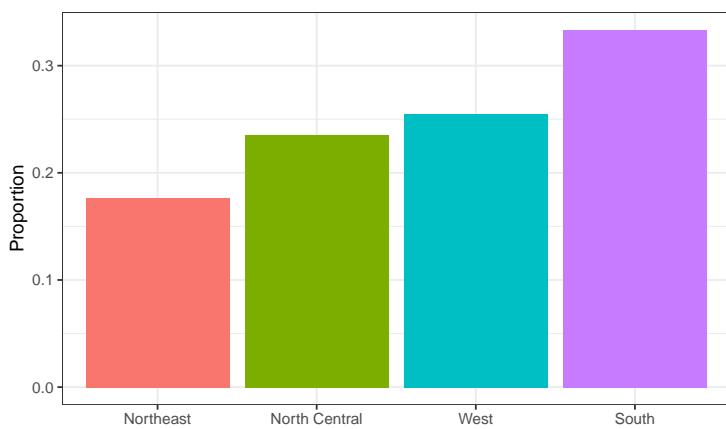
Como nota final, señalamos que una ventaja de las densidades suaves sobre los histogramas para fines de visualización es que las densidades facilitan la comparación de dos distribuciones. Esto se debe en gran parte a que los bordes irregulares del histograma añaden desorden. Aquí hay un ejemplo que compara las alturas masculinas y las femeninas:



Con el argumento correcto, `ggplot` automáticamente sombra la región de intersección con un color diferente. Mostraremos ejemplos de código de `ggplot2` para densidades en la Sección 9 así como en la Sección 8.16.

## 8.7 Ejercicios

1. En el set de datos `murders`, la región es una variable categórica y la siguiente es su distribución:

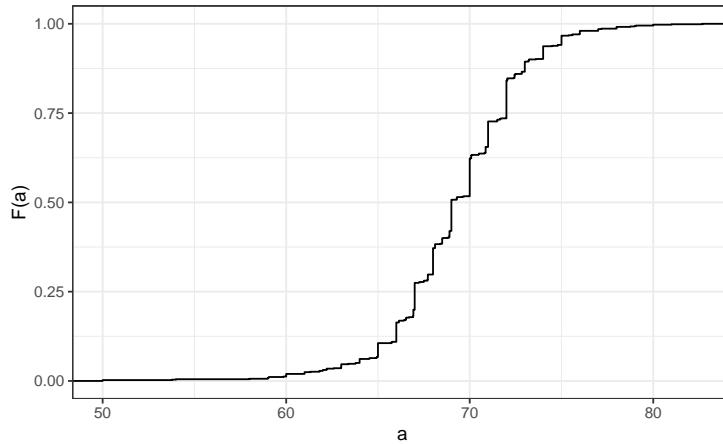


Redondeando al 5% más cercano, ¿qué proporción de los estados se encuentran en la región “North Central”?

2. ¿Cuál de los siguientes es cierto?

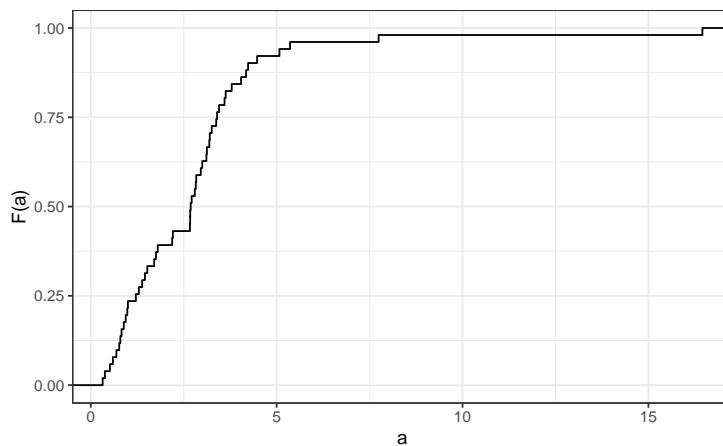
- El gráfico anterior es un histograma.
- El gráfico anterior muestra solo cuatro números con un diagrama de barras.

- c. Las categorías no son números, por lo que no tiene sentido graficar la distribución.
  - d. Los colores, no la altura de las barras, describen la distribución.
3. El siguiente gráfico muestra el eCDF para las alturas masculinas:



Según el gráfico, ¿qué porcentaje de hombres son más bajos que 75 pulgadas?

- a. 100%
  - b. 95%
  - c. 80%
  - d. 72 pulgadas
4. A la pulgada más cercana, ¿qué altura  $\bar{m}$  tiene la propiedad de que la mitad de los estudiantes varones son más altos que  $\bar{m}$  y la mitad son más bajos?
- a. 61 pulgadas
  - b. 64 pulgadas
  - c. 69 pulgadas
  - d. 74 pulgadas
5. Aquí hay un eCDF de las tasas de asesinatos en todos los estados:



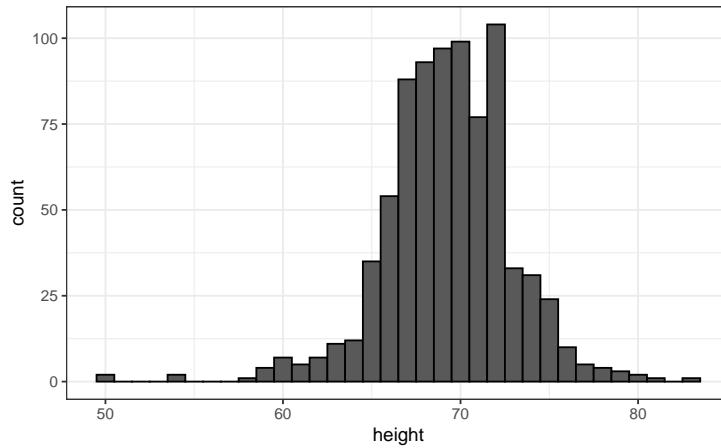
Sabiendo que hay 51 estados (contando DC) y basado en este gráfico, ¿cuántos estados tienen tasas de homicidio superiores a 10 por cada 100,000 personas?

- a. 1
- b. 5
- c. 10
- d. 50

6. Según el eCDF anterior, ¿cuál de las siguientes afirmaciones es cierta?

- a. Alrededor de la mitad de los estados tienen tasas de homicidios superiores a 7 por 100,000 y la otra mitad por debajo.
- b. La mayoría de los estados tienen tasas de homicidio de menos de 2 por 100,000.
- c. Todos los estados tienen tasas de asesinatos superiores a 2 por 100,000.
- d. Con la excepción de 4 estados, las tasas de asesinatos son inferiores a 5 por cada 100,000.

7. A continuación mostramos el histograma de alturas masculinas de nuestro set de datos `heights`:



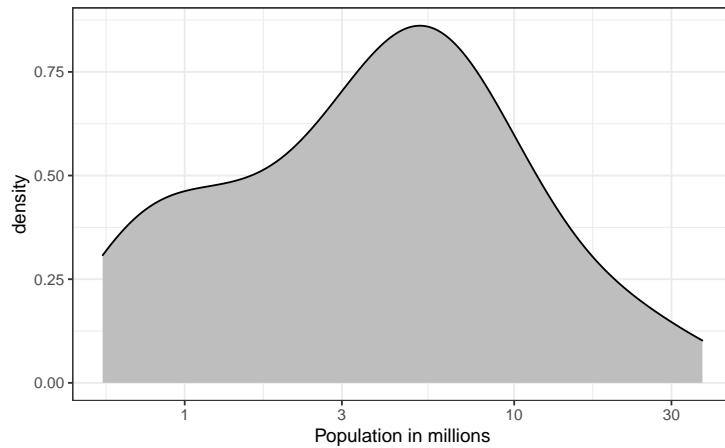
Según este gráfico, ¿cuántos hombres hay entre 63.5 y 65.5?

- a. 10
- b. 24
- c. 34
- d. 100

8. ¿Aproximadamente qué **porcentaje** son más bajos que 60 pulgadas?

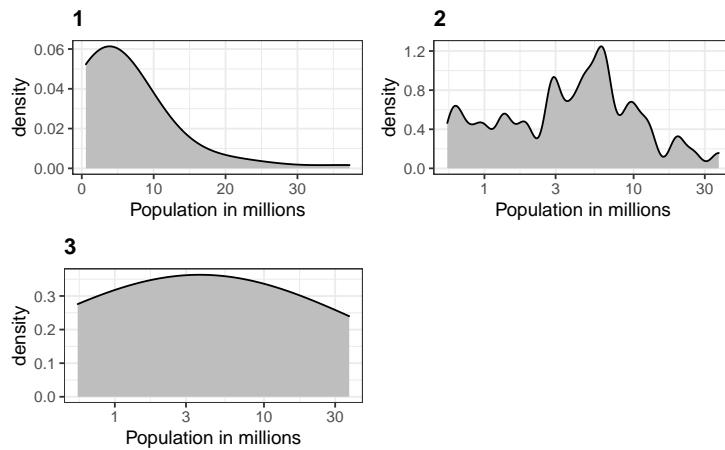
- a. 1%
- b. 10%
- c. 25%
- d. 50%

9. Según el siguiente gráfico de densidad, ¿aproximadamente qué proporción de estados de EE. UU. tiene poblaciones con más de 10 millones de personas?



- a. 0.02
- b. 0.15
- c. 0.50
- d. 0.55

10. A continuación hay tres gráficos de densidad. ¿Es posible que sean del mismo set de datos?



¿Cuál de las siguientes afirmaciones es cierta?

- a. Es imposible que sean del mismo set de datos.
- b. Son del mismo set de datos, pero los gráficos son diferentes debido a errores de código.
- c. Son del mismo set de datos, pero el primer y el segundo gráfico suavizan de menos y el tercero sobre-suaviza.
- d. Son del mismo set de datos, pero el primero no está en la escala logarítmica, el segundo suaviza de menos y el tercero sobre-suaviza.

## 8.8 La distribución normal

Los histogramas y los gráficos de densidad proveen excelentes resúmenes de una distribución. ¿Pero podemos resumir aún más? A menudo vemos el promedio y la desviación estándar utilizada como resumen estadístico: ¡un resumen de dos números! Para comprender cuáles son estos resúmenes y por qué se usan tanto, necesitamos comprender la distribución normal.

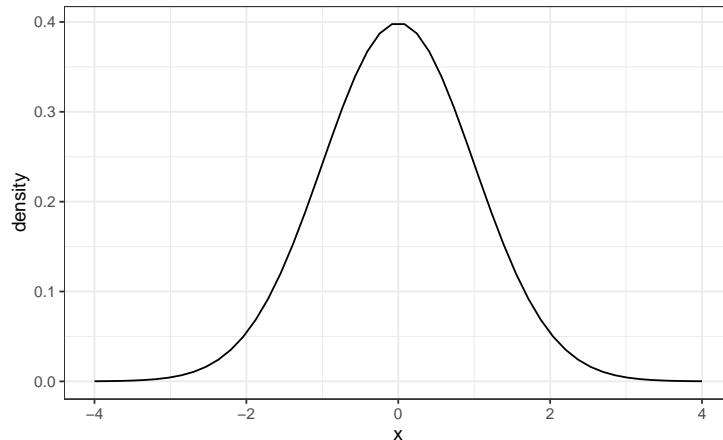
La distribución normal, también conocida como curva de campana y distribución Gausiana, es uno de los conceptos matemáticos más famosos de la historia. Una razón para esto es que se producen distribuciones aproximadamente normales en muchas situaciones, incluyendo las ganancias de juego, las alturas, los pesos, la presión arterial, las puntuaciones en las pruebas estandarizadas y los errores de medición experimentales. Hay explicaciones para esto y las describiremos más adelante. Aquí nos enfocamos en cómo la distribución normal nos ayuda a resumir los datos.

En vez de usar datos, la distribución normal se define con una fórmula matemática. Para cualquier intervalo  $(a, b)$ , la proporción de valores en ese intervalo se puede calcular utilizando esta fórmula:

$$\Pr(a < x < b) = \int_a^b \frac{1}{\sqrt{2\pi}s} e^{-\frac{1}{2}(\frac{x-m}{s})^2} dx$$

No necesitan memorizar o comprender los detalles de la fórmula. Sin embargo, recuerden que está completamente definida por solo dos parámetros:  $m$  y  $s$ . El resto de los símbolos en la fórmula representan los extremos del intervalo que determinamos  $a$  y  $b$  y las conocidas constantes matemáticas  $\pi$  y  $e$ . Estos dos parámetros,  $m$  y  $s$ , se conocen respectivamente como el *promedio* (o *la media*) y la *desviación estándar* (SD por sus siglas en inglés) de la distribución.

La distribución es simétrica, centrada en el promedio, y la mayoría de los valores (alrededor del 95%) están dentro de 2 SD del promedio. Así es como se ve la distribución normal cuando el promedio es 0 y la SD es 1:



El hecho de que la distribución está definida por solo dos parámetros implica que si la

distribución de un set de datos se puede aproximar con una distribución normal, toda la información necesaria para describir la distribución se puede codificar en solo dos números: el promedio y la desviación estándar. Ahora vamos a definir estos valores para una lista arbitraria de números.

Para una lista de números contenidos en un vector `x`, el promedio se define como:

```
m <- sum(x) / length(x)
```

y la SD se define como:

```
s <- sqrt(sum((x-mu)^2) / length(x))
```

que puede interpretarse como la distancia promedio entre los valores y su promedio.

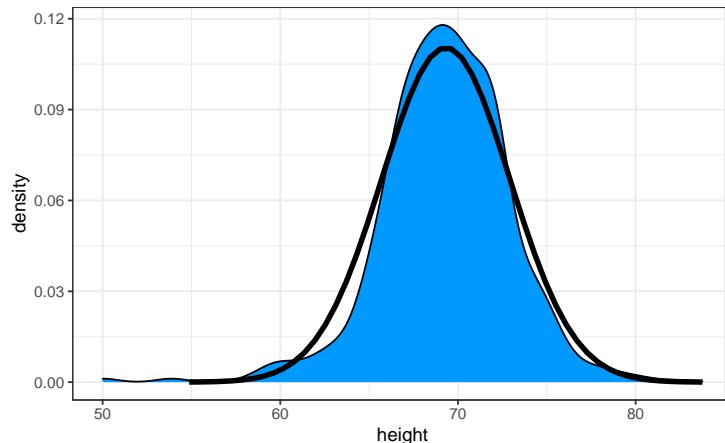
Calculemos los valores para la altura de los varones que almacenaremos en el objeto `x`:

```
index <- heights$sex == "Male"
x <- heights$height[index]
```

Se pueden usar las funciones predefinidas `mean` y `sd` (tengan en cuenta que por razones que se explican en la Sección 16.2, `sd` divide por `length(x)-1` en vez de `length(x)`) :

```
m <- mean(x)
s <- sd(x)
c(average = m, sd = s)
#> average sd
#> 69.31 3.61
```

A continuación tenemos un gráfico de la densidad suave de nuestras alturas de estudiantes en azul y la distribución normal con media = 69.3 y SD = 3.6 trazado como una línea negra:



La distribución normal parece ser una buena aproximación aquí. Ahora veremos cuán bien funciona esta aproximación para predecir la proporción de valores dentro de los intervalos.

## 8.9 Unidades estándar

Para los datos que se distribuyen aproximadamente normalmente, es conveniente pensar en términos de *unidades estándar*. La unidad estándar de un valor nos dice cuántas desviaciones estándar se alejan del promedio. Específicamente, para un valor  $x$  de un vector  $X$ , definimos el valor de  $x$  en unidades estándar como  $z = (x - \bar{m})/\bar{s}$  con  $\bar{m}$  y  $\bar{s}$  el promedio y la desviación estándar de  $X$ , respectivamente. ¿Por qué es conveniente hacer esto?

Primero, repasen la fórmula para la distribución normal y observen que lo que se está exponiendo es  $-z^2/2$  con  $z$  equivalente a  $x$  en unidades estándar. El hecho de que el máximo de  $e^{-z^2/2}$  es cuando  $z = 0$  explica por qué la distribución ocurre en el promedio. También explica la simetría ya que  $-z^2/2$  es simétrico alrededor de 0. Además, noten que si convertimos los datos distribuidos normalmente a unidades estándar, podemos saber rápidamente si, por ejemplo, una persona es aproximadamente promedio ( $z = 0$ ), entre los más altos ( $z \approx 2$ ), entre los más bajos ( $z \approx -2$ ), o una ocurrencia extremadamente rara ( $z > 3$  o  $z < -3$ ). Recuerden que no importa cuáles sean las unidades originales, estas reglas aplican a cualquier dato que es aproximadamente normal.

En R, podemos obtener unidades estándar usando la función `scale`:

```
z <- scale(x)
```

Ahora para ver cuántos hombres hay dentro de 2 SD del promedio, simplemente escribimos:

```
mean(abs(z) < 2)
#> [1] 0.95
```

¡La proporción es aproximadamente 95%, que es lo que predice la distribución normal! Para tener hasta más confirmación de que la aproximación es precisa, podemos usar gráficos Q-Q (*quantile-quantile plots* en inglés).

## 8.10 Gráficos Q-Q

Una manera sistemática de evaluar cuán bien se ajusta la distribución normal a los datos es verificar si las proporciones observadas y predecidas coinciden. En general, este es el acercamiento del gráfico Q-Q.

Primero, definimos los cuantiles teóricos para la distribución normal. En los libros de estadísticas usamos el símbolo  $\Phi(x)$  para definir la función que nos da la probabilidad de que una distribución normal estándar sea menos que  $x$ . Entonces, por ejemplo,  $\Phi(-1.96) = 0.025$  y  $\Phi(1.96) = 0.975$ . En R, podemos evaluar  $\Phi$  utilizando la función `pnorm`:

```
pnorm(-1.96)
#> [1] 0.025
```

La función inversa  $\Phi^{-1}(x)$  nos da los *cuantiles teóricos* para la distribución normal. Así, por ejemplo,  $\Phi^{-1}(0.975) = 1.96$ . En R, podemos evaluar el inverso de  $\Phi$  utilizando la función `qnorm`.

```
qnorm(0.975)
#> [1] 1.96
```

Tengan en cuenta que estos cálculos son para la distribución normal estándar por defecto (media = 0, desviación estándar = 1), pero también podemos definirlos para cualquier distribución normal. Podemos hacer esto usando los argumentos `mean` y `sd` en las funciones `pnorm` y `qnorm`. Por ejemplo, podemos usar `qnorm` para determinar cuantiles de una distribución con promedio y desviación estándar específicos:

```
qnorm(0.975, mean = 5, sd = 2)
#> [1] 8.92
```

Para la distribución normal, todos los cálculos relacionados con los cuantiles se realizan sin datos, de ahí el nombre de *cuantiles teóricos*. Pero los cuantiles se pueden definir para cualquier distribución, incluso una empírica. Entonces si tenemos datos en un vector  $x$ , podemos definir el cuantil asociado con cualquier proporción  $p$  como el  $q$  para el cual la proporción de valores por debajo de  $q$  es  $p$ . Usando el código R, podemos definir `q` como el valor para el cual `mean(x <= q) = p`. Observen que no todo  $p$  tiene un  $q$  para el cual la proporción es exactamente  $p$ . Hay varias maneras de definir el mejor  $q$  como se discute en la página de ayuda para la función `quantile`.

Como ejemplo rápido, para los datos de alturas masculinas, vemos que:

```
mean(x <= 69.5)
#> [1] 0.515
```

Alrededor del 50% son más bajos o iguales a 69 pulgadas. Esto implica que si  $p = 0.50$ , entonces  $q = 69.5$ .

La idea de un gráfico Q-Q es que si sus datos están bien aproximados por la distribución normal, los cuantiles de sus datos deberían ser similares a los cuantiles de una distribución normal. Para construir un gráfico Q-Q, hacemos lo siguiente:

1. Definimos un vector de  $m$  dimensiones  $p_1, p_2, \dots, p_m$ .
2. Definimos un vector de cuantiles  $q_1, \dots, q_m$  para las proporciones  $p_1, \dots, p_m$  usando sus datos. Nos referimos a estos como los *cuantiles muestrales* (*sample quantiles* en inglés).
3. Definimos un vector de cuantiles teóricos para las proporciones  $p_1, \dots, p_m$  para una distribución normal con el mismo promedio y desviación estándar que los datos.
4. Graficamos los cuantiles muestrales versus los cuantiles teóricos.

Construyamos un gráfico Q-Q usando el código R. Comiencen definiendo el vector de proporciones.

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

Para obtener los cuantiles de los datos, podemos usar la función `quantile` así:

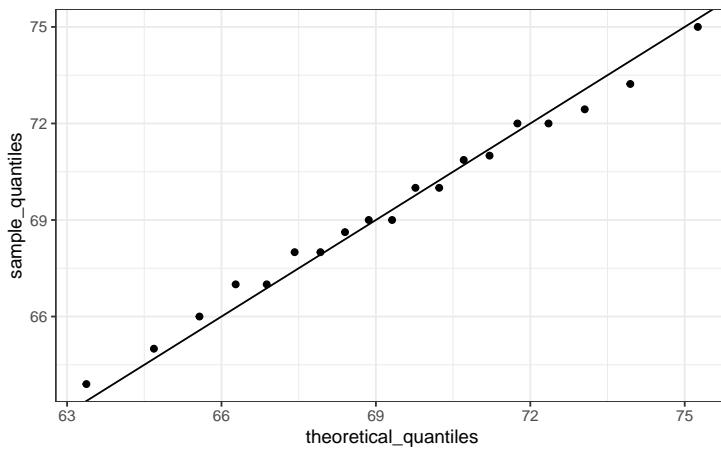
```
sample_quantiles <- quantile(x, p)
```

Para obtener los cuantiles teóricos de distribución normal con promedio y desviación estándar (SD) correspondiente, utilizamos la función `qnorm`:

```
theoretical_quantiles <- qnorm(p, mean = mean(x), sd = sd(x))
```

Para ver si coinciden o no, los graficamos uno contra el otro y dibujamos la línea de identidad:

```
qplot(theoretical_quantiles, sample_quantiles) + geom_abline()
```



Noten que este código es mucho más sencillo si usamos unidades estándar:

```
sample_quantiles <- quantile(z, p)
theoretical_quantiles <- qnorm(p)
qplot(theoretical_quantiles, sample_quantiles) + geom_abline()
```

El código anterior se incluye para ayudar a describir los gráficos Q-Q. Sin embargo, en la práctica es más fácil usar el código `ggplot2` descrito en la Sección 8.16:

```
heights %>% filter(sex == "Male") %>%
 ggplot(aes(sample = scale(height))) +
 geom_qq() +
 geom_abline()
```

Mientras que para la ilustración anterior usamos 20 cuantiles, el valor por defecto de la función `geom_qq` es utilizar la misma cantidad de cuantiles como datos.

## 8.11 Percentiles

Antes de continuar, definamos algunos términos que se usan comúnmente en el análisis exploratorio de datos.

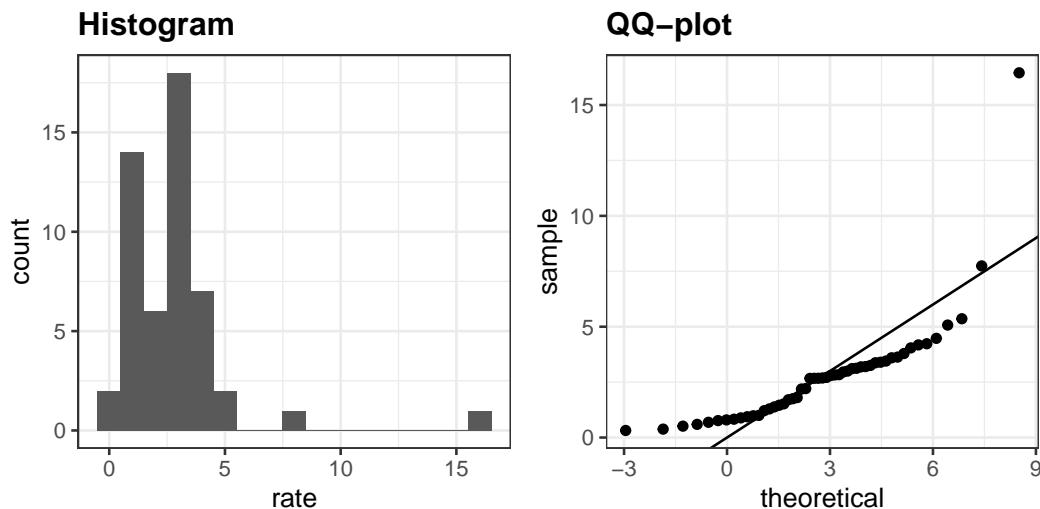
*Percentiles* son casos especiales de *cuantiles* que se usan comúnmente. Los percentiles son los cuantiles que se obtienen al configurar el  $p$  a  $0.01, 0.02, \dots, 0.99$ . Denominamos, por ejemplo, el caso de  $p = 0.25$  el cuartilo inferior, ya que nos da un número para el cual el 25% de los datos están por debajo. El percentil más famoso es el 50, también conocido como la *mediana*.

Para la distribución normal, la mediana y el promedio son los mismos, pero generalmente este no es el caso.

Otro caso especial que recibe un nombre son los *cuartiles*, que se obtienen al configurar  $p = 0.25, 0.50$  y  $0.75$ .

## 8.12 Diagramas de caja

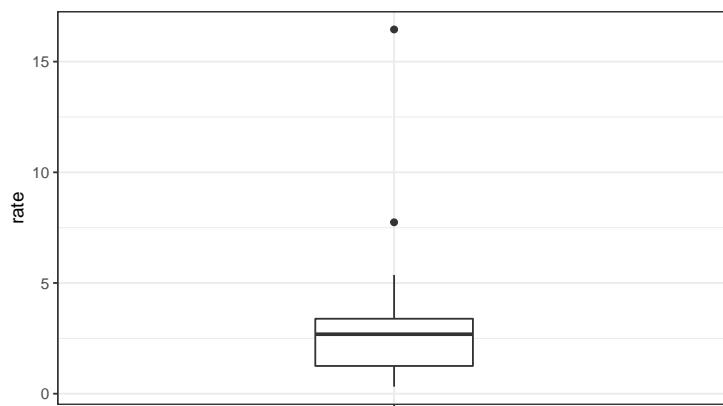
Para presentar los diagramas de caja (*boxplots* en inglés), volveremos a los datos de asesinatos de EE. UU. Supongan que queremos resumir la distribución de la tasa de asesinatos. Usando la técnica de visualización de datos que hemos aprendido, podemos ver que la aproximación normal no aplica aquí:



En este caso, el histograma anterior o un gráfico de densidad suave serviría como un resumen relativamente sucinto.

Ahora supongan que los que están acostumbrados a recibir solo dos números como resúmenes nos piden un resumen numérico más compacto.

Aquí Tukey ofreció algunos consejos. Primero, recomendó proveer un resumen de cinco números compuestos por el rango junto con los cuartiles (los percentiles 25, 50 y 75). Además, Tukey sugirió ignorar los *valores atípicos* (*outliers* en inglés) al calcular el rango y, en su lugar, graficarlos como puntos independientes. Ofreceremos una explicación detallada de los valores atípicos más adelante. Finalmente, recomendó que graficáramos estos números como una “caja” con “bigotes” así:



con el cuadro definido por los percentiles 25% y 75% y los bigotes mostrando el rango. La distancia entre estos dos se llama el rango *intercuartil*. Los dos puntos son valores atípicos según la definición de Tukey. La mediana se muestra con una línea horizontal.

A partir de este gráfico sencillo, hoy conocido como un *diagrama de caja*, sabemos que la mediana es de aproximadamente 2.5, que la distribución no es simétrica y que el rango es de 0 a 5 para la gran mayoría de los estados con dos excepciones.

Discutimos cómo crear diagramas de caja en la Sección 8.16.

---

## 8.13 Estratificación

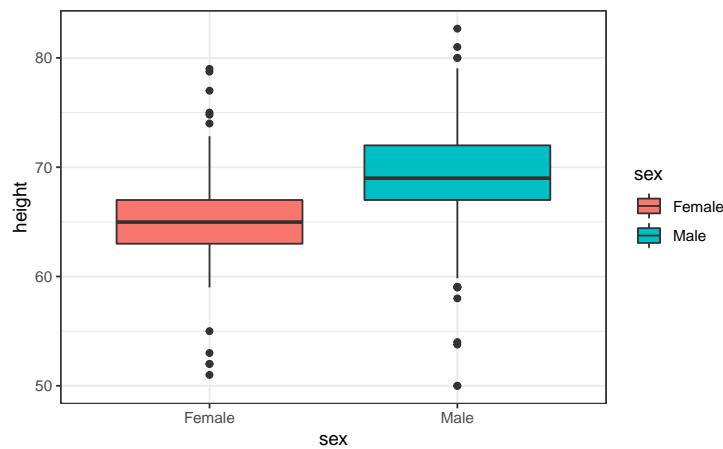
En el análisis de datos, a menudo dividimos las observaciones en grupos según los valores de una o más variables asociadas con esas observaciones. Por ejemplo, en la siguiente sección dividimos los valores de altura en grupos según una variable de sexo: hembras y varones. Llamamos a este procedimiento *estratificación* y nos referimos a los grupos resultantes como *estratos*.

La estratificación es común en la visualización de datos porque a menudo estamos interesados en cómo la distribución de variables difiere entre los diferentes subgrupos. Veremos varios ejemplos a lo largo de esta parte del libro. Revisaremos el concepto de estratificación cuando aprendamos regresión en el Capítulo 17 y en la parte de *machine learning* del libro.

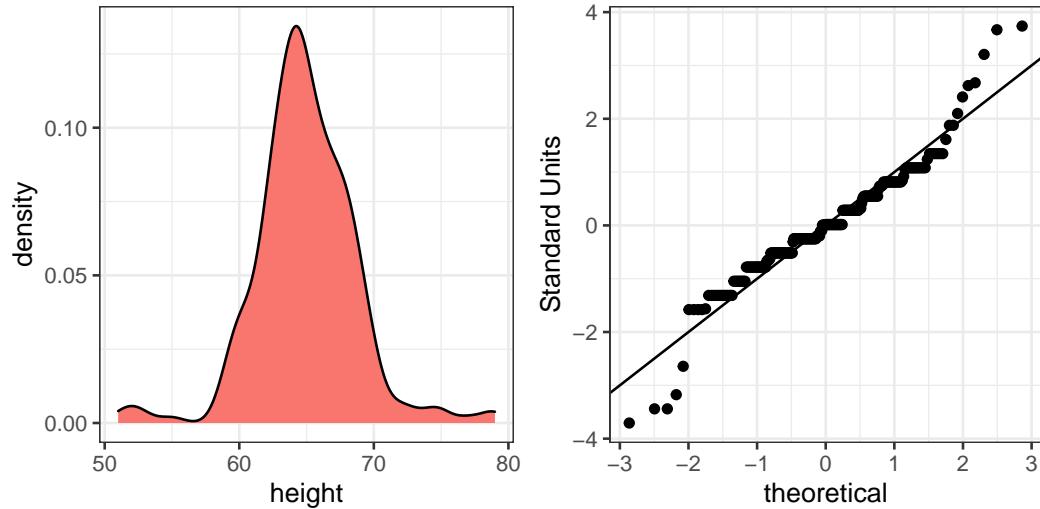
### 8.14 Estudio de caso: describiendo alturas de estudiantes (continuación)

Usando el histograma, los gráficos de densidad y los gráficos Q-Q, nos hemos convencido de que los datos de altura masculina se aproximan bien con una distribución normal. En este caso, le damos a ET un resumen muy sucinto: las alturas masculinas siguen una distribución normal con un promedio de 69.3 pulgadas y una SD de 3.6 pulgadas. Con esta información, ET tendrá una buena idea de qué esperar cuando conozca a nuestros estudiantes varones. Sin embargo, para proporcionar una imagen completa, también debemos proveer un resumen de las alturas femeninas.

Aprendimos que los diagramas de caja son útiles cuando queremos comparar rápidamente dos o más distribuciones. Aquí vemos las alturas para varones y hembras:



El diagrama inmediatamente demuestra que los varones son, en promedio, más altos que las hembras. Las desviaciones estándar parecen ser similares. Pero, ¿la aproximación normal también funciona para los datos de altura femenina recopilados por la encuesta? Esperamos que sigan una distribución normal, al igual que los varones. Sin embargo, los gráficos exploratorios revelan que la aproximación no es tan útil:



Vemos algo que no observamos para los varones: el gráfico de densidad tiene una segunda protuberancia. Además, el gráfico Q-Q muestra que los puntos más altos tienden a ser más altos de lo esperado por la distribución normal. Finalmente, también vemos cinco puntos en el gráfico Q-Q que sugieren alturas más bajas de lo esperado para una distribución normal. Al nuevamente informar a ET, es posible que necesitemos proveer un histograma en lugar de solo el promedio y la desviación estándar para las alturas femeninas.

Sin embargo, releemos la cita de Tukey y nos damos cuenta de que hemos notado lo que no esperábamos ver. Si observamos otras distribuciones de altura femenina, encontramos que están bien aproximadas con una distribución normal. Entonces, ¿por qué son diferentes nuestras alumnas? ¿Es nuestra clase un requisito para el equipo femenino de baloncesto? ¿Hay una proporción pequeña de mujeres que dicen ser más altas de lo que son? Otra explicación, quizás más probable, es que en el formulario en que los estudiantes ingresaron sus alturas, FEMALE era el sexo predeterminado y algunos varones ingresaron sus alturas, pero olvidaron cambiar la variable de sexo. En cualquier caso, la visualización de datos ha ayudado a descubrir una posible falla en nuestros datos.

Con respecto a los cinco valores más pequeños, noten que estos valores son:

```
heights %>% filter(sex == "Female") %>%
 top_n(5, desc(height)) %>%
 pull(height)
#> [1] 51 53 55 52 52
```

Debido a que estas son alturas autoreportadas, una posibilidad es que las estudiantes quisieron ingresar 5'1", 5'2", 5'3" o 5'5".

## 8.15 Ejercicios

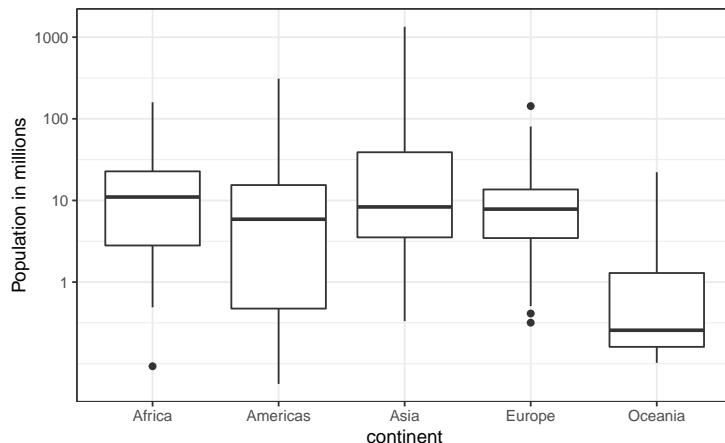
- Defina variables que contengan las alturas de varones y hembras de esta manera:

```
library(dslabs)
data(heights)
male <- heights$height[heights$sex == "Male"]
female <- heights$height[heights$sex == "Female"]
```

¿Cuántas medidas tenemos para cada una?

2. Supongan que no podemos hacer un gráfico y queremos comparar las distribuciones una al lado de otra. No podemos simplemente enumerar todos los números. En cambio, veremos los percentiles. Cree una tabla de cinco filas que muestre `female_percentiles` y `male_percentiles` con los percentiles 10, 30, 50, ..., 90 para cada sexo. Luego, cree un *data frame* con estas dos como columnas.

3. Estudie los siguientes diagramas de caja que muestran los tamaños de población por país:



¿Qué continente tiene el país con el mayor tamaño de población?

4. ¿Qué continente tiene la mediana de tamaño poblacional más grande?
5. ¿Cuál es la mediana del tamaño poblacional de África al millón más cercano?
6. ¿Qué proporción de países en Europa tiene poblaciones menores de 14 millones?

- a. 0.99
- b. 0.75
- c. 0.50
- d. 0.25

7. Si utilizamos una transformación logarítmica, ¿qué continente de los anteriores tiene el mayor rango intercuartil?

8. Cargue el set de datos de altura y cree un vector `x` con solo las alturas masculinas:

```
library(dslabs)
data(heights)
x <- heights$height[heights$sex=="Male"]
```

¿Qué proporción de los datos está entre 69 y 72 pulgadas (más alto que 69, pero más bajo o igual a 72)? Sugerencia: use un operador lógico y `mean`.

9. Supongan que lo único que sabe sobre los datos es el promedio y la desviación estándar. Use la aproximación normal para estimar la proporción que acaba de calcular. Sugerencia: comience calculando el promedio y la desviación estándar. Luego use la función `pnorm` para predecir las proporciones.

10. Note que la aproximación calculada en la pregunta nueve está muy cerca del cálculo exacto en la primera pregunta. Ahora realice la misma tarea para valores más atípicos. Compare el cálculo exacto y la aproximación normal para el intervalo (79,81]. ¿Cuántas veces mayor es la proporción real que la aproximación?

11. Aproxime la distribución de hombres adultos en el mundo como distribución normal con un promedio de 69 pulgadas y una desviación estándar de 3 pulgadas. Usando esta aproximación, calcule la proporción de hombres adultos que miden 7 pies de alto o más, conocidos como *seven footers*. Sugerencia: use la función `pnorm`.

12. Hay alrededor de mil millones de hombres entre las edades de 18 y 40 en el mundo. Use su respuesta a la pregunta anterior para estimar cuántos de estos hombres (de 18 a 40 años) miden siete pies de altura o más en el mundo.

13. Hay alrededor de 10 jugadores de la Asociación Nacional de Baloncesto (NBA) que miden 7 pies de altura o más. Usando la respuesta a las dos preguntas anteriores, ¿qué proporción de los *seven footers* del mundo, entre 18 a 40 años, están en la NBA?

14. Repita los cálculos realizados en la pregunta anterior para la altura del baloncestista Lebron James: 6 pies y 8 pulgadas. Hay alrededor de 150 jugadores que son al menos tan altos.

15. Al responder a las preguntas anteriores, descubrimos que no es raro que un jugador de siete pies se convierta en jugador de la NBA. Entonces, ¿que sería una crítica justa de nuestros cálculos?

- a. La práctica y el talento son lo que hacen a un gran jugador de baloncesto, no la altura.
- b. La aproximación normal no es apropiada para alturas.
- c. Como se observa en la pregunta 10, la aproximación normal tiende a subestimar los valores atípicos. Es posible que haya más *seven footers* de lo que predijimos.
- d. Como se observa en la pregunta 10, la aproximación normal tiende a sobreestimar los valores atípicos. Es posible que haya menos *seven footers* de lo que predijimos.

---

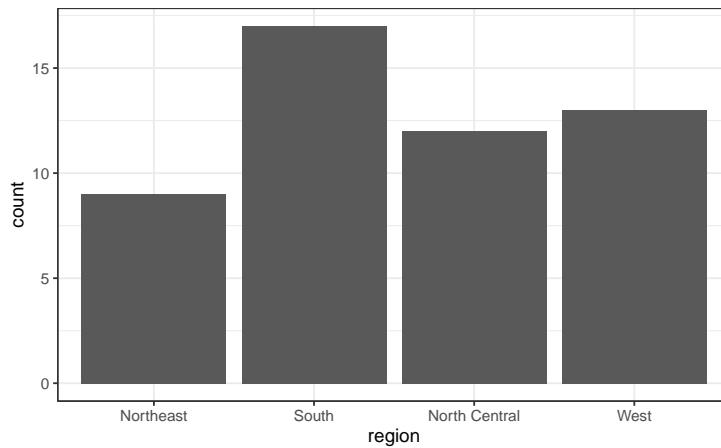
## 8.16 Geometrías ggplot2

En el Capítulo 7, presentamos el paquete **ggplot2** para la visualización de datos. Aquí demostramos cómo generar gráficos relacionados con distribuciones, específicamente los gráficos que se muestran anteriormente en este capítulo.

### 8.16.1 Diagramas de barras

Para generar un diagrama de barras (*barplots* en inglés), podemos usar la geometría `geom_bar`. Por defecto, R cuenta los casos en cada categoría y dibuja una barra. Aquí vemos el diagrama de barras para las regiones de Estados Unidos.

```
murders %>% ggplot(aes(region)) + geom_bar()
```

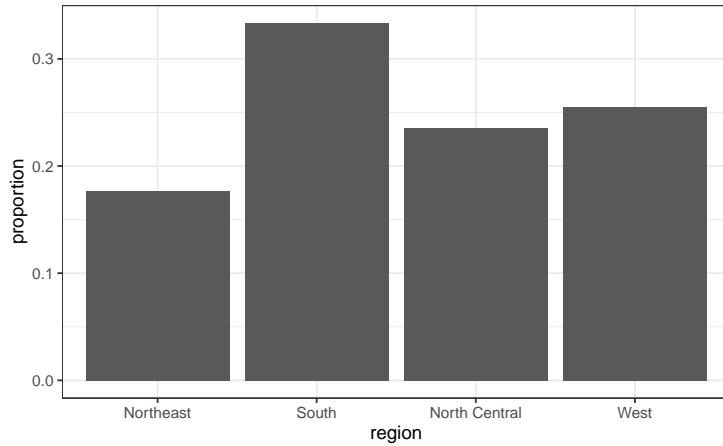


A menudo ya tenemos una tabla con una distribución que queremos presentar como diagrama de barras. Aquí tenemos un ejemplo de tal tabla:

```
data(murders)
tab <- murders %>%
 count(region) %>%
 mutate(proportion = n/sum(n))
tab
#> region n proportion
#> 1 Northeast 9 0.176
#> 2 South 17 0.333
#> 3 North Central 12 0.235
#> 4 West 13 0.255
```

Ya no queremos que `geom_bar` cuente, sino que simplemente grafique una barra a la altura proporcionada por la variable `proportion`. Para esto necesitamos proveer `x` (las categorías) e `y` (los valores) y usar la opción `stat="identity"`.

```
tab %>% ggplot(aes(region, proportion)) + geom_bar(stat = "identity")
```



### 8.16.2 Histogramas

Para generar histogramas, utilizamos `geom_histogram`. Al revisar la página de ayuda para esta función, vemos que el único argumento requerido es `x`, la variable para la cual construiremos un histograma. No usamos la `x` porque sabemos que es el primer argumento. El código se ve así:

```
heights %>%
 filter(sex == "Female") %>%
 ggplot(aes(height)) +
 geom_histogram()
```

Si ejecutamos el código anterior, nos da un mensaje:

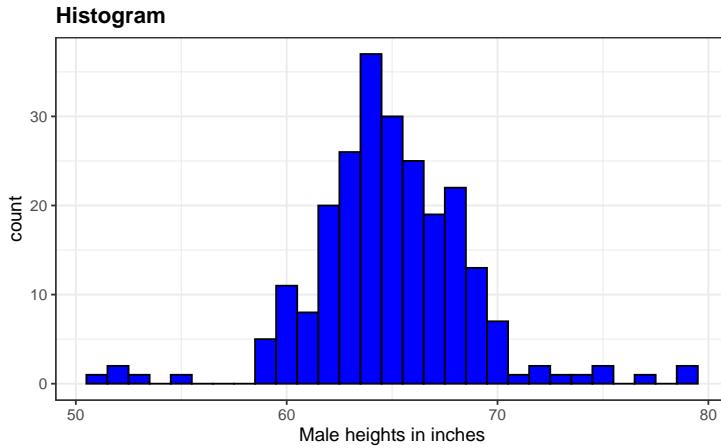
`stat_bin()` utilizando `bins = 30`. Elija un mejor valor con `binwidth`.

Anteriormente utilizamos un tamaño de compartimiento de 1 pulgada, por lo que el código se ve así:

```
heights %>%
 filter(sex == "Female") %>%
 ggplot(aes(height)) +
 geom_histogram(binwidth = 1)
```

Finalmente, si por razones estéticas queremos añadir color, usamos los argumentos descritos en la página de ayuda. También añadimos etiquetas y un título:

```
heights %>%
 filter(sex == "Female") %>%
 ggplot(aes(height)) +
 geom_histogram(binwidth = 1, fill = "blue", col = "black") +
 xlab("Male heights in inches") +
 ggtitle("Histogram")
```



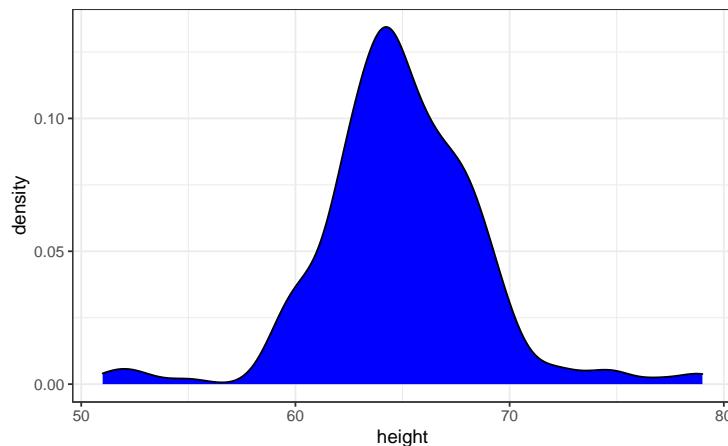
### 8.16.3 Gráficos de densidad

Para crear una densidad suave, usamos `geom_density`. Para hacer un gráfico de densidad suave con los datos que anteriormente visualizamos como un histograma, podemos usar este código:

```
heights %>%
 filter(sex == "Female") %>%
 ggplot(aes(height)) +
 geom_density()
```

Para llenar con color, podemos usar el argumento `fill`.

```
heights %>%
 filter(sex == "Female") %>%
 ggplot(aes(height)) +
 geom_density(fill="blue")
```

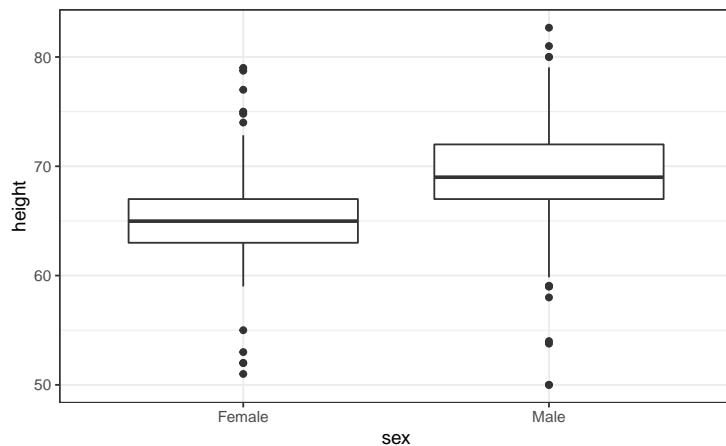


Para cambiar la suavidad de la densidad, utilizamos el argumento `adjust` para multiplicar el valor por defecto por ese `adjust`. Por ejemplo, si queremos que el parámetro de suavizado sea el doble de grande, usamos:

```
heights %>%
 filter(sex == "Female") +
 geom_density(fill="blue", adjust = 2)
```

#### 8.16.4 Diagramas de caja

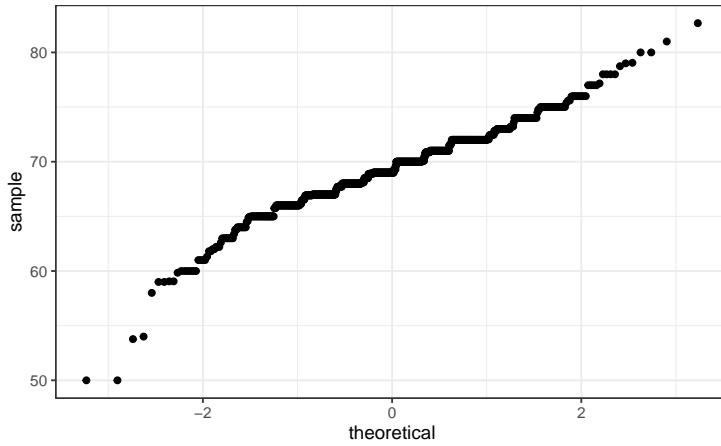
La geometría para crear diagramas de caja es `geom_boxplot`. Como ya hemos discutido, los diagramas de caja son útiles para comparar distribuciones. Por ejemplo, a continuación vemos las alturas mostradas anteriormente para las mujeres, pero aquí en comparación con los hombres. Para esta geometría, necesitamos los argumentos `x` como las categorías y los argumentos `y` como los valores:



#### 8.16.5 Gráficos Q-Q

Para gráficos Q-Q, usamos la geometría `geom_qq`. De la página de ayuda, aprendemos que necesitamos especificar el `sample` (aprenderemos sobre `samples` en un capítulo posterior). Aquí tenemos el gráfico Q-Q para alturas de varones:

```
heights %>% filter(sex=="Male") %>%
 ggplot(aes(sample = height)) +
 geom_qq()
```



Por defecto, la variable muestral se compara con una distribución normal con un promedio de 0 y una desviación estándar de 1. Para cambiar esto, utilizamos el argumento `dparams` según la página de ayuda. Para añadir una línea de identidad, simplemente asignen otra capa. Para líneas rectas, usamos la función `geom_abline`. La línea por defecto es la línea de identidad (pendiente = 1, intercepto = 0).

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

heights %>% filter(sex=="Male") %>%
 ggplot(aes(sample = height)) +
 geom_qq(dparams = params) +
 geom_abline()
```

Otra opción aquí es escalar los datos primero y luego hacer un gráfico Q-Q contra la distribución normal estándar.

```
heights %>%
 filter(sex=="Male") %>%
 ggplot(aes(sample = scale(height))) +
 geom_qq() +
 geom_abline()
```

### 8.16.6 Imágenes

No hemos tenido que usar imágenes para los conceptos descritos en este capítulo, pero los usaremos en la Sección 10.14, así que presentamos las dos geometrías utilizadas para crear imágenes: `geom_tile` y `geom_raster`. Se comportan de manera similar; para ver cómo difieren, consulten la página de ayuda. Para crear una imagen en `ggplot2`, necesitamos un *data frame* con las coordenadas x e y, así como los valores asociados con cada uno de estos. Aquí tenemos un *data frame*:

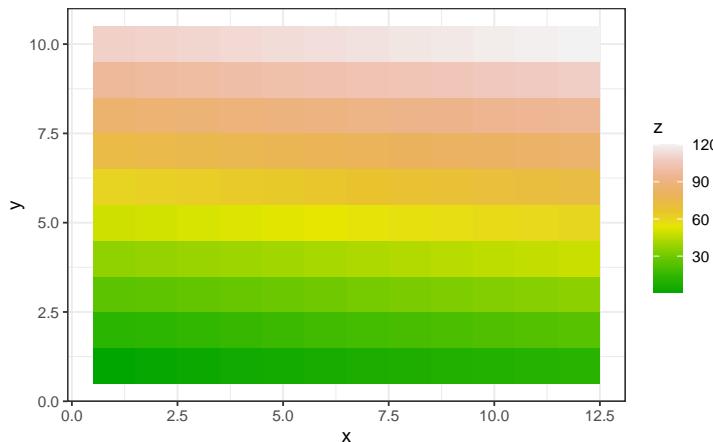
```
x <- expand.grid(x = 1:12, y = 1:10) %>%
 mutate(z = 1:120)
```

Tengan en cuenta que esta es la versión *tidy* de una matriz, `matrix(1:120, 12, 10)`. Para graficar la imagen, usamos el siguiente código:

```
x %>% ggplot(aes(x, y, fill = z)) +
 geom_raster()
```

Con estas imágenes, a menudo querrán cambiar la escala de color. Esto se puede hacer a través de la capa `scale_fill_gradientn`.

```
x %>% ggplot(aes(x, y, fill = z)) +
 geom_raster() +
 scale_fill_gradientn(colors = terrain.colors(10))
```



### 8.16.7 Gráficos rápidos

En la Sección 7.13, presentamos `qplot` como una función útil cuando necesitamos hacer un diagrama de dispersión rápido. También podemos usar `qplot` para hacer histogramas, diagramas de densidad, diagramas de caja, gráficos Q-Q y más. Aunque no provee el nivel de control de `ggplot`, `qplot` es definitivamente útil, ya que nos permite hacer un gráfico con un pequeño fragmento de código.

Supongan que tenemos las alturas femeninas en un objeto `x`:

```
x <- heights %>%
 filter(sex=="Male") %>%
 pull(height)
```

Para hacer un histograma rápido podemos usar:

```
qplot(x)
```

La función adivina que queremos hacer un histograma porque solo proveemos una variable. En la Sección 7.13, vimos que si le damos dos variables a `qplot`, automáticamente crea un diagrama de dispersión.

Para hacer un gráfico Q-Q rápido, tienen que usar el argumento `sample`. Recuerden que podemos añadir capas tal como lo hacemos con `ggplot`.

```
qplot(sample = scale(x)) + geom_abline()
```

Si proveemos un factor y un vector numérico, obtenemos un gráfico como el que vemos abajo. Tengan en cuenta que en el código estamos utilizando el argumento `data`. Como el *data frame* no es el primer argumento en `qplot`, tenemos que usar el operador de punto.

```
heights %>% qplot(sex, height, data = .)
```

También podemos seleccionar una geometría específica mediante el uso del argumento `geom`. Entonces, para convertir el diagrama anterior en un diagrama de caja, usamos el siguiente código:

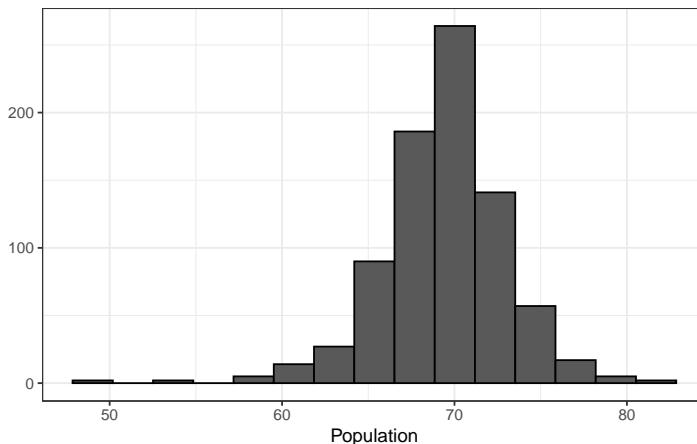
```
heights %>% qplot(sex, height, data = ., geom = "boxplot")
```

También podemos usar el argumento `geom` para generar un gráfico de densidad en lugar de un histograma:

```
qplot(x, geom = "density")
```

Aunque no tanto como con `ggplot`, tenemos cierta flexibilidad para mejorar los resultados de `qplot`. Mirando la página de ayuda, vemos varias formas en que podemos mejorar el aspecto del histograma anterior. Por ejemplo:

```
qplot(x, bins=15, color = I("black"), xlab = "Population")
```



**Nota técnica:** La razón por la que usamos `I("black")` es porque queremos que `qplot` trate "black" como un carácter en lugar de convertirlo en un factor. Este es el comportamiento por defecto dentro de `aes`, que se llama internamente aquí. En general, la función `I` se usa en R para decir "manténgalo como está".

## 8.17 Ejercicios

1. Ahora vamos a usar la función `geom_histogram` para hacer un histograma de las alturas en el set de datos `height`. Al leer la documentación para esta función, vemos que requiere solo una asignación, los valores que se utilizarán para el histograma. Haga un histograma de todos los gráficos.

¿Cuál es la variable que contiene las alturas?

- a. `sex`
- b. `heights`
- c. `height`
- d. `heights$height`

2. Ahora cree un objeto `ggplot` usando el *pipe* para asignar los datos de altura a un objeto `ggplot`. Asigne `height` a los valores de `x` a través de la función `aes`.

3. Ahora estamos listos para añadir una capa para hacer el histograma. Utilice el objeto creado en el ejercicio anterior y la función `geom_histogram` para hacer el histograma.

4. Cuando ejecutamos el código en el ejercicio anterior, recibimos la advertencia: `stat_bin()` utilizando `bins = 30`. Elija un mejor valor con `binwidth`.

Utilice el argumento `binwidth` para cambiar el histograma creado en el ejercicio anterior para usar compartimientos de tamaño de 1 pulgada.

5. En lugar de un histograma, vamos a hacer un gráfico de densidad suave. En este caso, no crearemos un objeto, sino haremos y mostraremos el gráfico con una línea de código. Cambie la geometría en el código utilizado anteriormente para hacer una densidad suave en lugar de un histograma.

6. Ahora vamos a hacer un gráfico de densidad para varones y hembras por separado. Podemos hacer esto usando el argumento `group`. Asignamos grupos a través del mapeo estético, ya que cada punto necesita un grupo antes de hacer los cálculos necesarios para estimar una densidad.

7. También podemos asignar grupos a través del argumento `color`. Esto tiene el beneficio adicional de que utiliza el color para distinguir los grupos. Cambie el código anterior para usar `color`.

8. Además, podemos asignar grupos a través del argumento `fill`. Esto tiene el beneficio adicional de que usa colores para distinguir los grupos así:

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

Sin embargo, aquí la segunda densidad se traza sobre la primera. Podemos hacer que las curvas sean más visibles mediante el uso de *alpha blending* para añadir transparencia. Establezca el parámetro alfa a 0.2 en la función `geom_density` para hacer este cambio.

# 9

---

## Visualización de datos en la práctica

---

En este capítulo, demostraremos cómo el código relativamente sencillo de `ggplot2` puede crear gráficos esclarecedores y estéticamente agradables. Como motivación, crearemos gráficos que nos ayudarán a comprender mejor las tendencias de la salud y la economía mundial. Implementaremos lo que aprendimos en los Capítulos 7 y 8.16 y aprenderemos a expandir el código para perfeccionar los gráficos. A medida que avancemos en nuestro estudio de caso, describiremos los principios generales más relevantes a la visualización de datos y aprenderemos conceptos como *facetas*, *gráficos de series de tiempo*, *transformaciones* y *gráficos ridge*.

---

### 9.1 Estudio de caso: nuevas ideas sobre la pobreza

Hans Rosling<sup>1</sup> era el cofundador de la Fundación Gapminder<sup>2</sup>, una organización dedicada a educar al público mediante datos para disipar mitos comunes sobre el llamado mundo en desarrollo. La organización utiliza datos para mostrar cómo las tendencias actuales en los campos de salud y economía contradicen las narrativas que emanen de la cobertura sensacionalista de los medios de catástrofes, tragedias y otros eventos desafortunados. Como se indica en el sitio web de la Fundación Gapminder:

Los periodistas y cabilderos cuentan historias dramáticas. Ese es su trabajo. Cuentan historias sobre eventos extraordinarios y personas inusuales. Las historias dramáticas se acumulan en las mentes de las personas en una visión del mundo demasiado dramática y con fuertes sentimientos de estrés negativo: “¡El mundo está empeorando!”, “¡Somos nosotros contra ellos!”, “¡Las demás personas son extrañas!”, “¡La población sigue creciendo!” y “¡A nadie le importa!”

Hans Rosling se dedicó, en su propia manera dramática, a educar al público sobre tendencias basadas en datos utilizando visualizaciones de datos eficaces. Esta sección se basa en dos charlas que ejemplifican esta perspectiva educativa: *New Insights on Poverty*<sup>3</sup> y *The Best Stats You've Ever Seen*<sup>4</sup>. Específicamente, en esta sección usamos datos para intentar responder a las siguientes dos preguntas:

1. ¿Es una caracterización justa del mundo actual decir que está dividido en naciones

<sup>1</sup>[https://en.wikipedia.org/wiki/Hans\\_Rosling](https://en.wikipedia.org/wiki/Hans_Rosling)

<sup>2</sup><http://www.gapminder.org/>

<sup>3</sup>[https://www.ted.com/talks/hans\\_rosling\\_reveals\\_new\\_insights\\_on\\_poverty?language=en](https://www.ted.com/talks/hans_rosling_reveals_new_insights_on_poverty?language=en)

<sup>4</sup>[https://www.ted.com/talks/hans\\_rosling\\_shows\\_the\\_best\\_stats\\_you\\_ve\\_ever\\_seen](https://www.ted.com/talks/hans_rosling_shows_the_best_stats_you_ve_ever_seen)

- ricas occidentales y el mundo en desarrollo compuesto por África, Asia y América Latina?
2. ¿Ha empeorado la desigualdad de ingresos en todos los países durante los últimos 40 años?

Para responder a estas preguntas, utilizaremos el set de datos `gapminder` proveído por `dslabs`. Este set de datos se creó utilizando varias hojas de cálculo disponibles de la Fundación Gapminder. Pueden acceder a la tabla de esta manera:

```
library(tidyverse)
library(dslabs)
data(gapminder)
gapminder %>% as_tibble()
#> # A tibble: 10,545 x 9
#> country year infant_mortality life_expectancy fertility population
#> <fct> <int> <dbl> <dbl> <dbl> <dbl>
#> 1 Albania 1960 115. 62.9 6.19 1636054
#> 2 Algeria 1960 148. 47.5 7.65 11124892
#> 3 Angola 1960 208 36.0 7.32 5270844
#> 4 Antigua a~ 1960 NA 63.0 4.43 54681
#> 5 Argentina 1960 59.9 65.4 3.11 20619075
#> # ... with 10,540 more rows, and 3 more variables: gdp <dbl>,
#> # continent <fct>, region <fct>
```

### 9.1.1 La prueba de Hans Rosling

Igual que en el video *New Insights on Poverty*, comenzamos examinando nuestros conocimientos sobre las diferencias en la mortalidad infantil en diferentes países. Para cada uno de los cinco pares de países a continuación, ¿qué países creen que tuvieron las tasas de mortalidad infantil más altas en 2015? ¿Qué pares creen que son más similares?

1. Sri Lanka o Turquía
2. Polonia o Corea del Sur
3. Malasia o Rusia
4. Pakistán o Vietnam
5. Tailandia o Sudáfrica

Al responder a estas preguntas sin datos, los países no europeos suelen ser elegidos como los que tienen tasas de mortalidad infantil más altas: Sri Lanka sobre Turquía, Corea del Sur sobre Polonia y Malasia sobre Rusia. También es común suponer que los países considerados como parte del mundo en desarrollo: Pakistán, Vietnam, Tailandia y Sudáfrica, tienen tasas de mortalidad igualmente altas.

Para responder a estas preguntas **con datos**, podemos usar `dplyr`. Por ejemplo, para la primera comparación vemos que:

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

```
#> 1 Sri Lanka 8.4
#> 2 Turkey 11.6
```

Turquía tiene la mayor tasa de mortalidad infantil.

Podemos usar este código en todas las comparaciones y descubrimos lo siguiente:

```
#> New names:
#> * country -> country...1
#> * infant_mortality -> infant_mortality...2
#> * country -> country...3
#> * infant_mortality -> infant_mortality...4
```

| country   | infant mortality | country      | infant mortality |
|-----------|------------------|--------------|------------------|
| Sri Lanka | 8.4              | Turkey       | 11.6             |
| Poland    | 4.5              | South Korea  | 2.9              |
| Malaysia  | 6.0              | Russia       | 8.2              |
| Pakistan  | 65.8             | Vietnam      | 17.3             |
| Thailand  | 10.5             | South Africa | 33.6             |

Vemos que los países europeos en esta lista tienen tasas de mortalidad infantil más altas: Polonia tiene una tasa más alta que Corea del Sur y Rusia tiene una tasa más alta que Malasia. También vemos que Pakistán tiene una tasa mucho más alta que Vietnam y Sudáfrica tiene una tasa mucho más alta que Tailandia. Resulta que cuando Hans Rosling le dio este cuestionario a grupos de personas educadas, la puntuación promedio fue menos de 2.5 de 5, peor de lo que hubieran obtenido si hubieran adivinado. Esto implica que más que ignorantes, estamos mal informados. En este capítulo, vemos cómo la visualización de datos nos ayuda a informarnos.

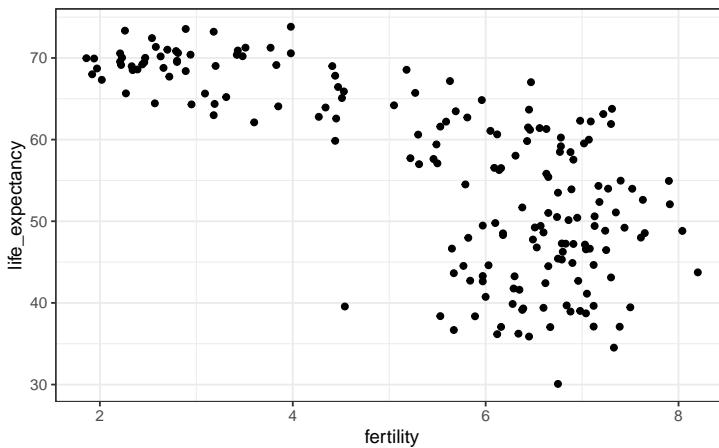
## 9.2 Diagramas de dispersión

La razón por las bajas puntuaciones se deriva de la noción preconcebida de que el mundo está dividido en dos grupos: el mundo occidental (Europa occidental y América del Norte), caracterizado por una larga vida y familias pequeñas, versus el mundo en desarrollo (África, Asia y América Latina), caracterizados por cortos períodos de vida y familias numerosas. ¿Pero los datos respaldan esta visión dicotómica?

Los datos necesarios para responder a esta pregunta también están disponibles en nuestra tabla `gapminder`. Usando nuestras recién aprendidas habilidades de visualización de datos, podemos enfrentar este desafío.

Para analizar esta visión del mundo, nuestro primer gráfico es un diagrama de dispersión de la esperanza de vida versus las tasas de fertilidad (número promedio de hijos por mujer). Comenzamos mirando los datos de hace unos 50 años, cuando quizás esta visión se consolidó por primera vez en nuestras mentes.

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

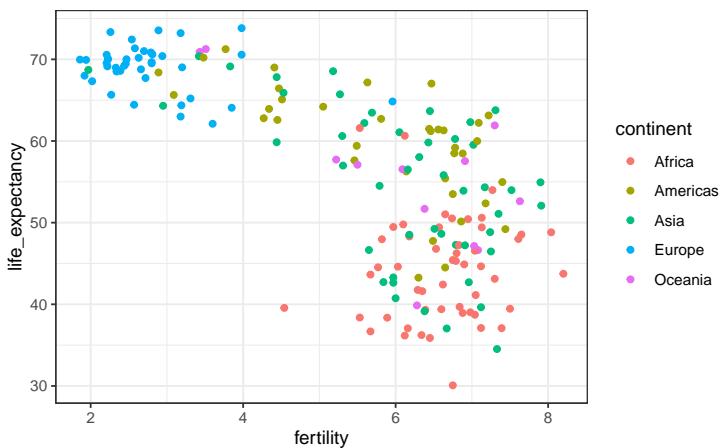


La mayoría de puntos se dividen en dos categorías distintas:

1. Esperanza de vida alrededor de 70 años y 3 o menos hijos por familia.
2. Esperanza de vida inferior a 65 años y más de 5 niños por familia.

Para confirmar que estos países son de las regiones que esperamos, podemos usar un color para representar un continente.

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



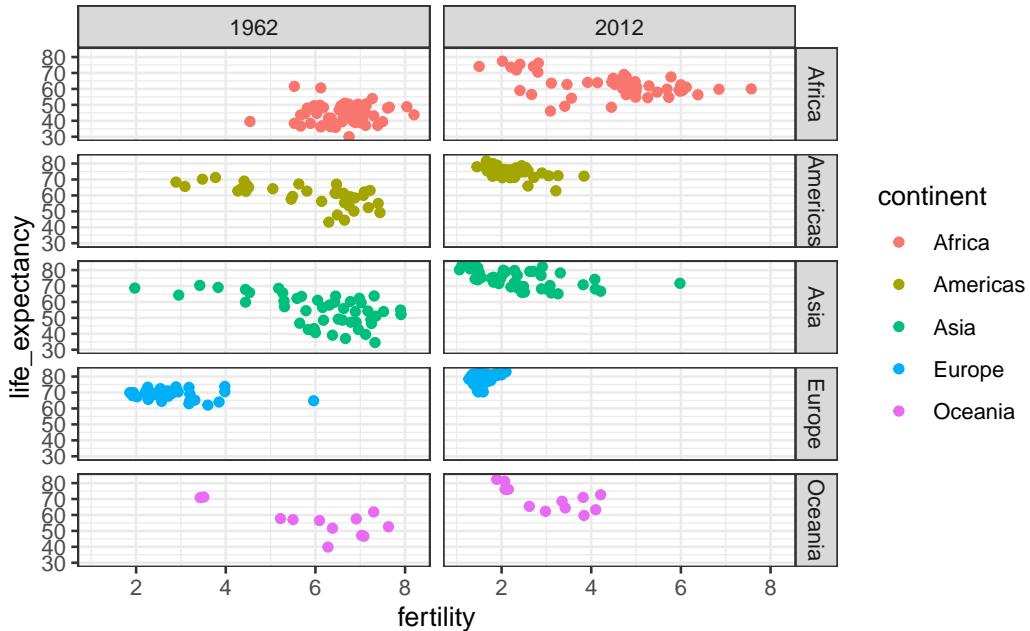
En 1962, la visión del “Oeste versus el mundo en desarrollo” se basaba en cierta realidad. ¿Sigue siendo así 50 años después?

### 9.3 Separar en facetas

Podemos graficar fácilmente los datos de 2012 de la misma manera que lo hicimos para 1962. Sin embargo, para hacer comparaciones, es preferible graficar lado a lado. En **ggplot2**, logramos esto separando las variables en *facetas* (*faceting* en inglés): estratificamos los datos por alguna variable y hacemos el mismo gráfico para cada estrato.

Para separar en facetas, añadimos una capa con la función `facet_grid`, que automáticamente separa los gráficos. Esta función les permite separar hasta dos variables en facetas usando columnas para representar una variable y filas para representar la otra. La función espera que las variables de fila y de columna estén separadas por un `~`. Aquí vemos un ejemplo de un diagrama de dispersión donde añadimos `facet_grid` como la última capa:

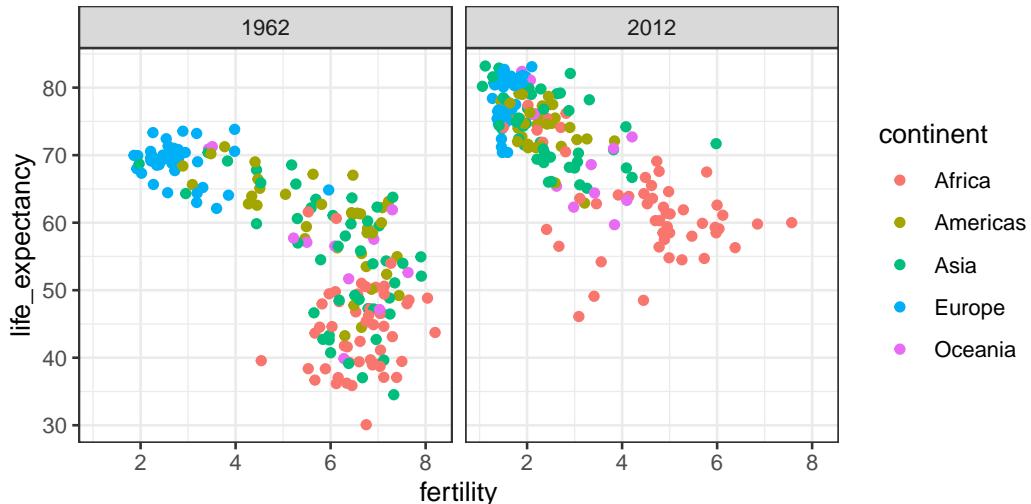
```
filter(gapminder, year%in%c(1962, 2012)) %>%
 ggplot(aes(fertility, life_expectancy, col = continent)) +
 geom_point() +
 facet_grid(continent~year)
```



Arriba vemos un gráfico para cada combinación de continente/año. Sin embargo, esto es solo un ejemplo y más de lo que queremos, que es simplemente comparar dos años: 1962 y 2012. En este caso, solo hay una variable y usamos `.~.` para que `facet_grid` sepa que no estamos usando una de las variables:

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

```
geom_point() +
facet_grid(. ~ year)
```

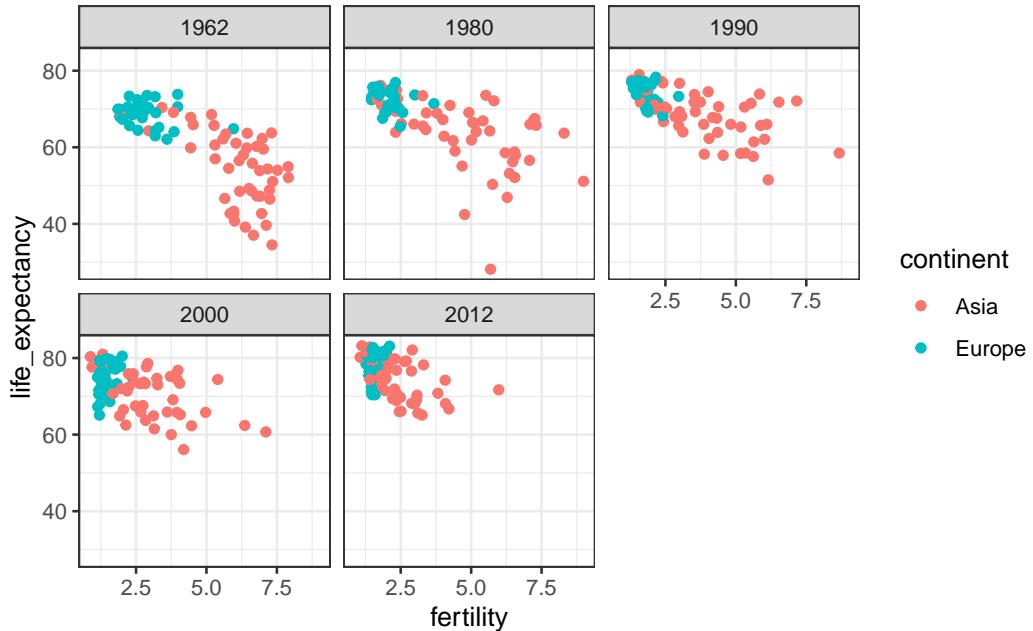


Este gráfico muestra claramente que la mayoría de los países se han mudado del conjunto *mundo en desarrollo* al conjunto *mundo occidental*. En 2012, la visión del mundo occidental versus el mundo en desarrollo ya no tiene sentido. Esto es particularmente evidente cuando se compara Europa con Asia, este último ahora con varios países que han realizado grandes mejoras.

### 9.3.1 facet\_wrap

Para explorar cómo sucedió esta transformación a través de los años, podemos hacer el gráfico para varios años. Por ejemplo, podemos añadir los años 1970, 1980, 1990 y 2000. Sin embargo, si hacemos esto, no queremos a todos los gráficos en la misma fila, que es lo que hace `facet_grid` por defecto, ya que aparecerán demasiado estrechos para mostrar los datos. En cambio, queremos usar múltiples filas y columnas. La función `facet_wrap` nos permite hacer esto automáticamente acomodando la serie de gráficos para que cada imagen tenga dimensiones visibles:

```
years <- c(1962, 1980, 1990, 2000, 2012)
continents <- c("Europe", "Asia")
gapminder %>%
 filter(year %in% years & continent %in% continents) %>%
 ggplot(aes(fertility, life_expectancy, col = continent)) +
 geom_point() +
 facet_wrap(~year)
```

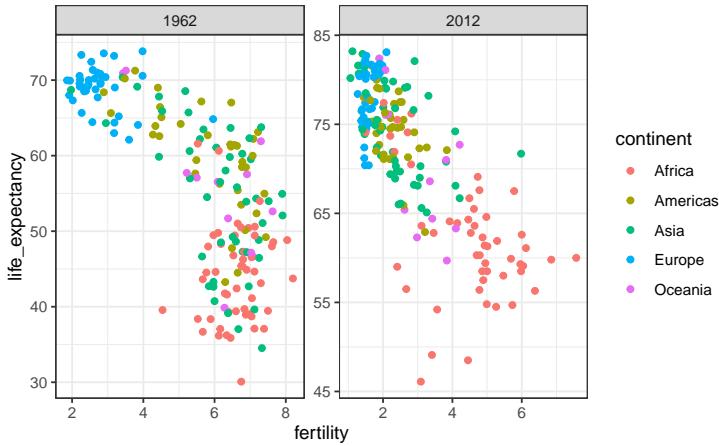


Este gráfico muestra claramente cómo la mayoría de los países asiáticos han mejorado a un ritmo mucho más rápido que los europeos.

### 9.3.2 Escalas fijas para mejores comparaciones

La elección por defecto del rango de los ejes es importante. Cuando no se usa `facet`, este rango está determinado por los datos que se muestran en el gráfico. Cuando usan `facet`, este rango está determinado por los datos que se muestran en todos los gráficos y, por lo tanto, se mantienen fijos en todos los gráficos. Esto hace que las comparaciones entre gráficos sean mucho más fáciles. Por ejemplo, en el gráfico anterior, podemos ver que la esperanza de vida ha aumentado y la fertilidad ha disminuido en la mayoría de los países. Vemos esto porque la nube de puntos se mueve. Este no es el caso si ajustamos las escalas:

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



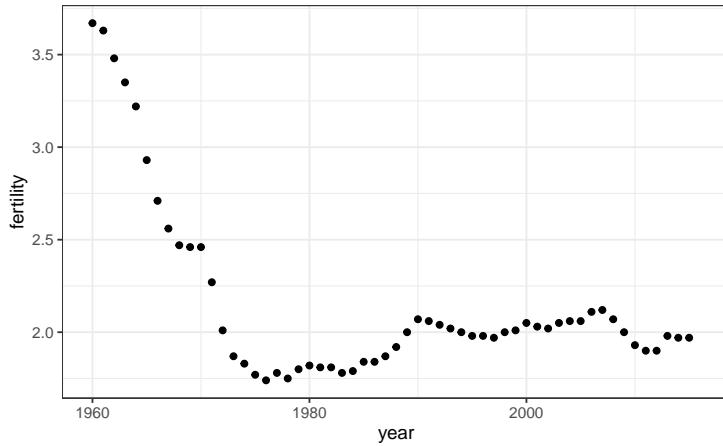
En el gráfico anterior, debemos prestar atención particular al rango para notar que el gráfico de la derecha tiene una mayor esperanza de vida.

## 9.4 Gráficos de series de tiempo

Las visualizaciones anteriores ilustran efectivamente que los datos ya no son compatibles con la visión del mundo occidental frente al mundo en desarrollo. Al ver estos gráficos, surgen nuevas preguntas. Por ejemplo, ¿qué países están mejorando más y cuáles menos? ¿La mejora fue constante durante los últimos 50 años o se aceleró más durante ciertos períodos? Para una mirada más detenida que pueda ayudar a responder a estas preguntas, presentamos *gráficos de series de tiempo* (*time series plots* en inglés).

Los gráficos de series de tiempo tienen tiempo en el eje-x y un resultado o medida de interés en el eje-y. Por ejemplo, aquí vemos un gráfico de la tendencia de las tasas de fertilidad de Estados Unidos:

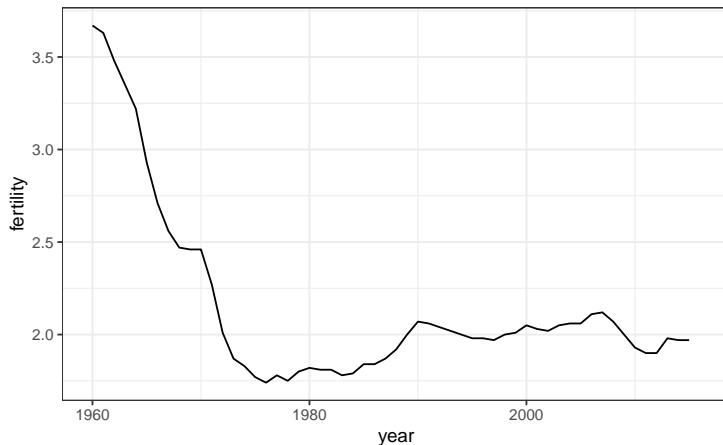
```
gapminder %>%
 filter(country == "United States") %>%
 ggplot(aes(year, fertility)) +
 geom_point()
```



Observamos que la tendencia no es lineal en absoluto, sino que durante los años sesenta y setenta se produce una fuerte caída por debajo de 2. Entonces la tendencia vuelve a 2 y se estabiliza durante los años noventa.

Cuando los puntos están regular y densamente espaciados, como vemos arriba, creamos una curva que une los puntos con líneas, para transmitir que estos datos provienen de una sola serie, aquí un país. Para hacer esto, usamos la función `geom_line` en vez de `geom_point`.

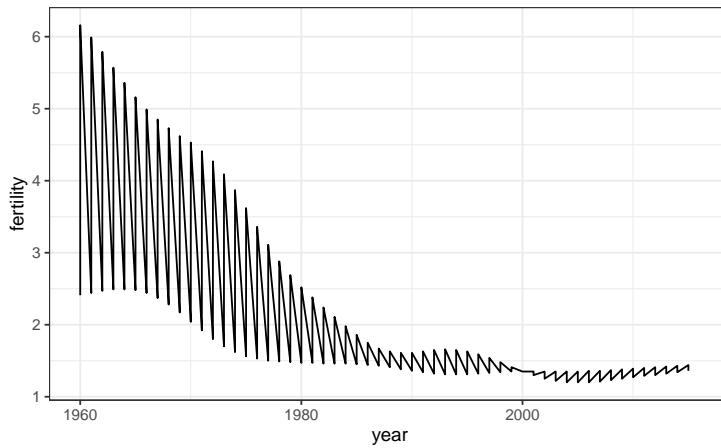
```
gapminder %>%
 filter(country == "United States") %>%
 ggplot(aes(year, fertility)) +
 geom_line()
```



Esto es particularmente útil cuando comparamos dos países. Si creamos un subconjunto de los datos para incluir dos países, uno de Europa y uno de Asia, entonces adaptamos el código anterior:

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

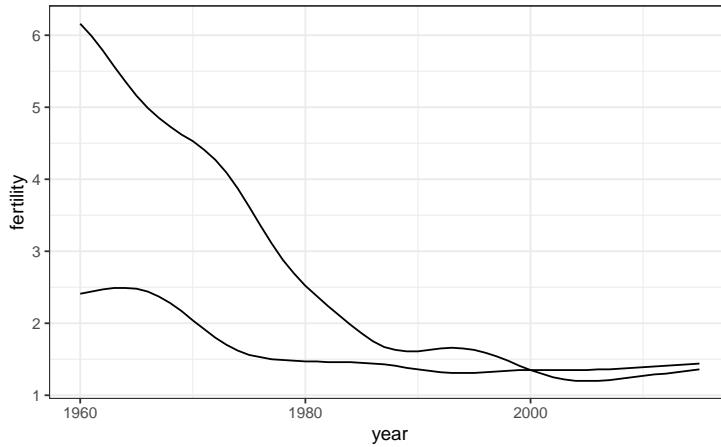
```
gapminder %>% filter(country %in% countries) %>%
 ggplot(aes(year,fertility)) +
 geom_line()
```



Claramente, este **no** es el gráfico que queremos. En lugar de una línea para cada país, se unen los puntos para ambos países porque no le hemos dicho a `ggplot` que queremos dos líneas independientes. Para que `ggplot` entienda que hay dos curvas que se deben hacer por separado, asignamos cada punto a un `group`, uno para cada país:

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

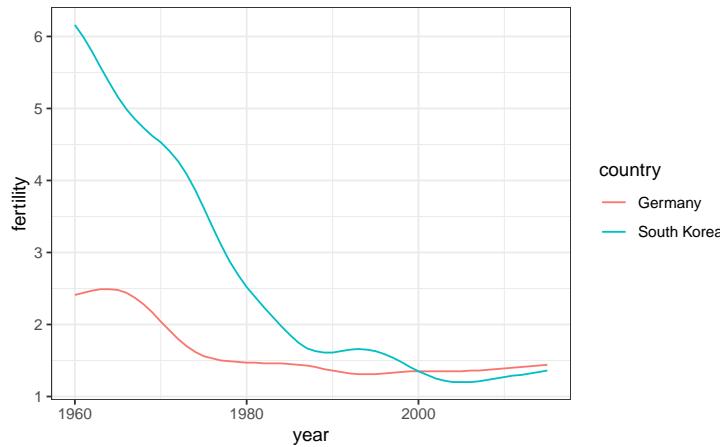
gapminder %>% filter(country %in% countries & !is.na(fertility)) %>%
 ggplot(aes(year, fertility, group = country)) +
 geom_line()
```



¿Pero qué línea va con qué país? Podemos asignar colores para hacer esta distinción. Un efecto secundario útil de usar el argumento `color` para asignar diferentes colores a los diferentes países es que los datos se agrupan automáticamente:

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

gapminder %>% filter(country %in% countries & !is.na(fertility)) %>%
 ggplot(aes(year, fertility, col = country)) +
 geom_line()
```



El gráfico muestra claramente cómo la tasa de fertilidad de Corea del Sur cayó drásticamente durante los años sesenta y setenta, y en 1990 tuvo una tasa similar a la de Alemania.

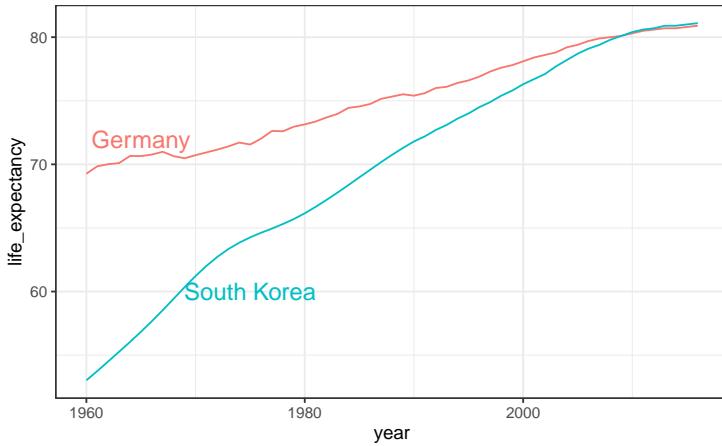
#### 9.4.1 Etiquetas en lugar de leyendas

Para los gráficos de tendencias, recomendamos etiquetar las líneas en lugar de usar leyendas, ya que el espectador puede ver rápidamente qué línea representa qué país. Esta sugerencia aplica a la mayoría de los gráficos: las etiquetas generalmente se prefieren a las leyendas.

Demostramos cómo hacer esto usando los datos de esperanza de vida. Definimos una tabla de datos con las ubicaciones de las etiquetas y luego usamos una segunda asignación solo para estas etiquetas:

```
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")
```



El gráfico muestra claramente cómo una mejora en la esperanza de vida siguió a caídas en las tasas de fertilidad. En 1960, los alemanes vivieron 15 años más que los surcoreanos, aunque para 2010 la brecha está completamente cerrada. Ejemplifica la mejora que muchos países no occidentales han logrado en los últimos 40 años.

## 9.5 Transformaciones de datos

Ahora cambiamos nuestra atención a la segunda pregunta relacionada con la idea común de que la distribución de la riqueza en todo el mundo ha empeorado durante las últimas décadas. Cuando se le pregunta al público en general si los países pobres se han vuelto más pobres y los países ricos se han vuelto más ricos, la mayoría responde que sí. Mediante el uso de estratificación, histogramas, densidades suaves y diagramas de caja, podremos ver si este realmente es el caso. Primero, aprenderemos cómo las transformaciones a veces pueden ayudar a proporcionar resúmenes y gráficos más informativos.

La tabla de datos `gapminder` incluye una columna con el producto interno bruto de los países (GDP por sus siglas en inglés). El GDP mide el valor de mercado de los bienes y servicios producidos por un país en un año. El GDP por persona a menudo se usa como un resumen aproximado de la riqueza de un país. Aquí dividimos esta cantidad por 365 para obtener la medida más interpretable de *dólares por día*. Utilizando los dólares estadounidenses actuales como una unidad, una persona que sobrevive con un ingreso de menos de \$2 por día se define como viviendo en la “pobreza absoluta”. Añadimos esta variable a la tabla de datos:

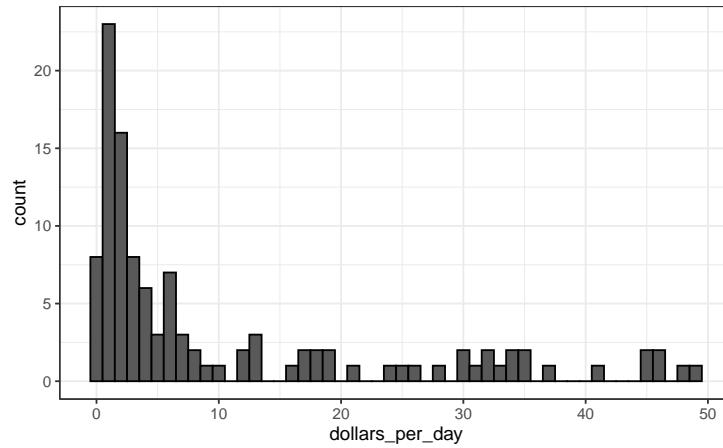
```
gapminder <- gapminder %>% mutate(dollars_per_day = gdp/population/365)
```

Los valores del GDP se ajustan a la inflación y representan dólares estadounidenses actuales, por lo que estos valores deben ser comparables a lo largo de los años. Por supuesto, estos son promedios de país y dentro de cada país hay mucha variabilidad. Todos los gráficos y las ideas que se describen a continuación se refieren a los promedios de los países y no a los individuos dentro de estos.

### 9.5.1 Transformación logarítmica

Abajo tenemos un histograma de ingresos diarios desde 1970:

```
past_year <- 1970
gapminder %>%
 filter(year == past_year & !is.na(gdp)) %>%
 ggplot(aes(dollars_per_day)) +
 geom_histogram(binwidth = 1, color = "black")
```



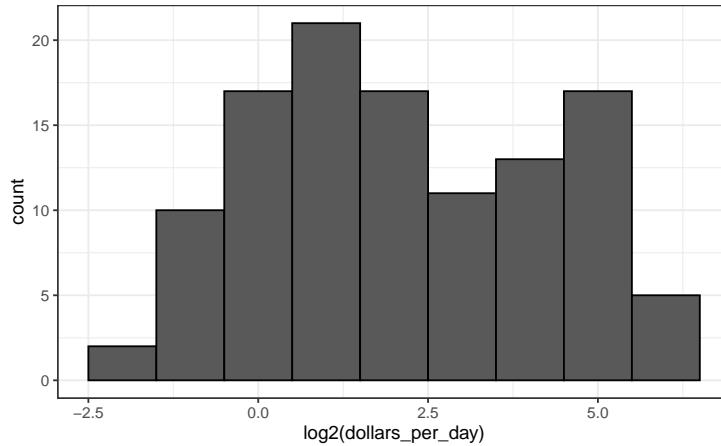
Utilizamos el argumento `color = "black"` para dibujar un límite y distinguir claramente los compartimientos.

En este gráfico, vemos que para la mayoría de los países, los promedios están por debajo de \$10 por día. Sin embargo, la mayoría del eje-x está dedicado a 35 países con promedio de menos de \$10. Por lo tanto, el gráfico no es muy informativo con respecto a países con valores inferiores a \$10 por día.

Sería más informativo poder ver rápidamente cuántos países tienen ingresos diarios promedio de aproximadamente \$1 (extremadamente pobre), \$2 (muy pobre), \$4 (pobre), \$8 (promedio), \$16 (acomodado), \$32 (rico), \$64 (muy rico) por día. Estos cambios son multiplicativos y las transformaciones logarítmicas convierten los cambios multiplicativos en aditivos: cuando se usa la base 2, la duplicación de un valor se convierte en un aumento de 1.

Aquí tenemos la distribución si aplicamos una transformación logarítmica base 2:

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



Así logramos ver de cerca a los países de ingresos medios a ingresos bajos.

### 9.5.2 ¿Qué base?

En el caso anterior, utilizamos la base 2 en las transformaciones logarítmicas. Otras opciones comunes son base e (el logaritmo natural) y base 10.

En general, no recomendamos utilizar el logaritmo natural para la exploración y visualización de datos. La razón es porque mientras  $2^2, 2^3, 2^4, \dots$  o  $10^2, 10^3, \dots$  son fáciles de calcular en nuestras cabezas, lo mismo no es cierto para  $e^2, e^3, \dots$ , por lo que la escala no es intuitiva ni fácil de interpretar.

En el ejemplo de dólares por día, utilizamos la base 2 en lugar de la base 10 porque el rango resultante es más fácil de interpretar. El rango de los valores que se trazan es 0.327, 48.885.

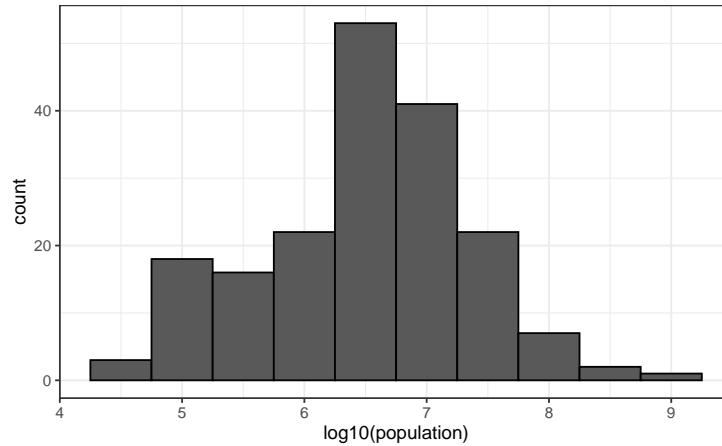
En la base 10, esto se convierte en un rango que incluye muy pocos enteros: solo 0 y 1. Con la base dos, nuestro rango incluye -2, -1, 0, 1, 2, 3, 4 y 5. Es más fácil calcular  $2^x$  y  $10^x$  cuando  $x$  es un entero y entre -10 y 10, por lo que preferimos tener enteros más pequeños en la escala. Otra consecuencia de un rango limitado es que elegir el ancho del compartimiento (*binwidth* en inglés) es más difícil. Con logaritmo base 2, sabemos que un ancho de compartimiento de 1 se convertirá en un compartimiento con rango  $x$  a  $2x$ .

Para un ejemplo en el que la base 10 tiene más sentido, consideren los tamaños de poblaciones. Un logaritmo base 10 es preferible ya que el rango para estos es:

```
filter(gapminder, year == past_year) %>%
 summarize(min = min(population), max = max(population))
#> min max
#> 1 46075 8.09e+08
```

Abajo tenemos el histograma de los valores transformados:

```
gapminder %>%
 filter(year == past_year) %>%
 ggplot(aes(log10(population))) +
 geom_histogram(binwidth = 0.5, color = "black")
```



En el gráfico anterior, rápidamente vemos que las poblaciones de los países oscilan entre diez mil y diez mil millones.

### 9.5.3 ¿Transformar los valores o la escala?

Hay dos formas en que podemos usar las transformaciones logarítmicas en los gráficos. Podemos tomar el logaritmo de los valores antes de graficarlos o usar escalas logarítmicas en los ejes. Ambos enfoques son útiles y tienen diferentes ventajas. Si tomamos el logaritmo de los datos, podemos interpretar más fácilmente los valores intermedios en la escala. Por ejemplo, si vemos:

----1----x----2-----3----

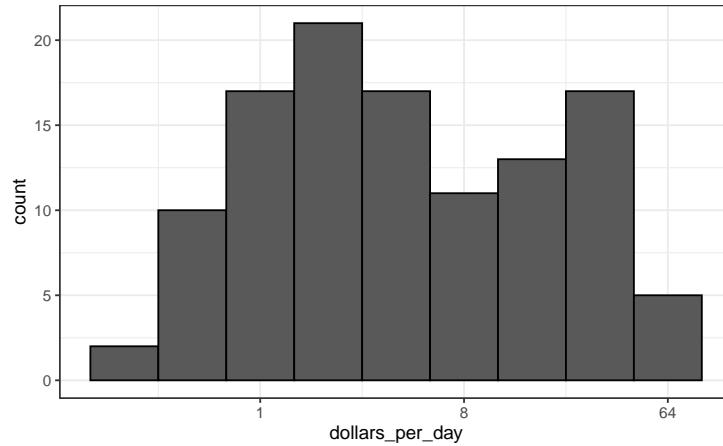
para datos transformados con el logaritmo, sabemos que el valor de  $x$  es de aproximadamente 1.5. Si usamos escalas logarítmicas:

----10----x---100-----1000----

entonces, para determinar  $x$ , necesitamos calcular  $10^{1.5}$ , que no es fácil de hacer mentalmente. La ventaja de usar escalas logarítmicas es que vemos los valores originales en los ejes. Sin embargo, la ventaja de mostrar escalas logarítmicas es que los valores originales se muestran en el gráfico y son más fáciles de interpretar. Por ejemplo, veríamos “32 dólares por día” en lugar de “5 log base 2 dólares por día”.

Como aprendimos anteriormente, si queremos escalar el eje con logaritmos, podemos usar la función `scale_x_continuous`. En vez de primero tomar el logaritmo de los valores, aplicamos esta capa:

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



Tengan en cuenta que la transformación logarítmica base 10 tiene su propia función: `scale_x_log10()`, pero actualmente la base 2 no tiene, aunque fácilmente podríamos definir una.

Hay otras transformaciones disponibles a través del argumento `trans`. Como aprenderemos más adelante, la transformación de raíz cuadrada (`sqrt`) es útil cuando se consideran conteos. La transformación logística (`logit`) es útil cuando se grafican proporciones entre 0 y 1. La transformación `reverse` es útil cuando queremos que los valores más pequeños estén a la derecha o arriba.

## 9.6 Cómo visualizar distribuciones multimodales

En el histograma anterior vemos dos protuberancias: una aproximadamente en 4 y otra aproximadamente en 32. En estadística, estas protuberancias a veces se denominan *modas* (*modes* en inglés). La moda de una distribución es el valor con la frecuencia más alta. La moda de distribución normal es el promedio. Cuando una distribución, como la anterior, no disminuye monotónicamente de la moda, llamamos a los lugares donde sube y baja de nuevo *modas locales* y decimos que la distribución tiene *modas múltiples*.

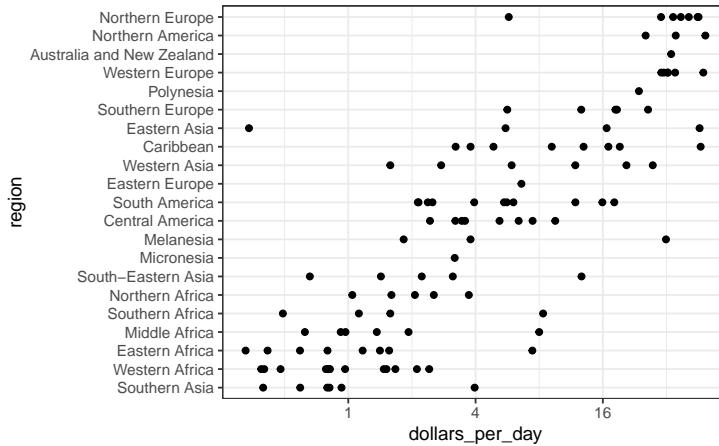
El histograma anterior sugiere que la distribución de ingreso de los países en 1970 tiene dos modas: una de aproximadamente 2 dólares por día (1 en la escala log 2) y la otra de aproximadamente 32 dólares por día (5 en la escala log 2). Esta *bimodalidad* es consistente con un mundo dicotómico compuesto por países con ingresos promedio inferiores a \$8 (3 en la escala log 2) por día y países por encima de eso.

## 9.7 Cómo comparar múltiples distribuciones con diagramas de caja y gráficos *ridge*

De acuerdo con el histograma, los valores de distribución del ingreso de 1970 muestran una dicotomía. Sin embargo, el histograma no nos muestra si los dos grupos de países están en el *oeste* o forman parte del mundo *en desarrollo*.

Comencemos examinando rápidamente los datos por región. Reordenamos las regiones por la mediana y usamos una escala logarítmica.

```
gapminder %>%
 filter(year == past_year & !is.na(gdp)) %>%
 mutate(region = reorder(region, dollars_per_day, FUN = median)) %>%
 ggplot(aes(dollars_per_day, region)) +
 geom_point() +
 scale_x_continuous(trans = "log2")
```



Ya podemos ver que efectivamente existe una dicotomía “el Oeste versus el Resto”: hay dos grupos claros, con el grupo rico compuesto por Norteamérica, Europa del Norte y Occidental, Nueva Zelanda y Australia. Definimos grupos basados en esta observación:

```
gapminder <- gapminder %>%
 mutate(group = case_when(
 region %in% c("Western Europe", "Northern Europe", "Southern Europe",
 "Northern America",
 "Australia and New Zealand") ~ "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",
 TRUE ~ "Others"))
```

Convertimos esta variable `group` en un factor para controlar el orden de los niveles:

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

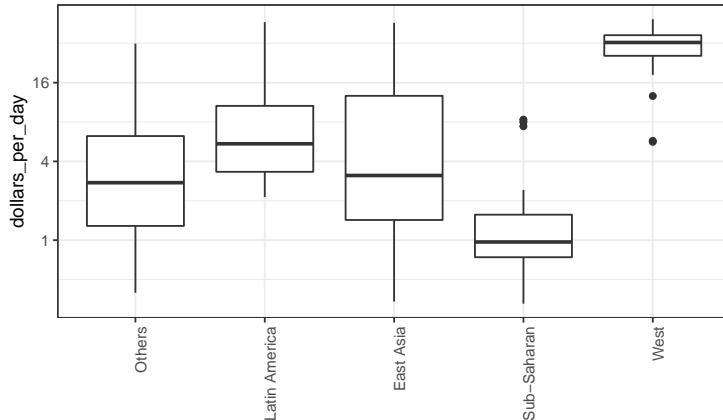
En la siguiente sección mostramos cómo visualizar y comparar distribuciones entre grupos.

### 9.7.1 Diagramas de caja

El anterior análisis exploratorio de datos reveló dos características sobre la distribución de ingreso promedio en 1970. Usando un histograma, encontramos una distribución bimodal con los modos relacionados con los países pobres y ricos. Ahora queremos comparar la distribución entre estos cinco grupos para confirmar la dicotomía “el Oeste versus el Resto”. El número de puntos en cada categoría es lo suficientemente grande como para que un gráfico de resumen pueda ser útil. Podríamos generar cinco histogramas o cinco gráficos de densidad, pero puede ser más práctico tener todos los resúmenes visuales en un gráfico. Por lo tanto, comenzamos apilando diagramas de caja uno al lado del otro. Tengan en cuenta que añadimos la capa `theme(axis.text.x = element_text(angle = 90, hjust = 1))` para que las etiquetas de grupo sean verticales, ya que no encajan si las mostramos horizontalmente, y para quitar la etiqueta del eje a fin de hacer espacio.

```
p <- gapminder %>%
 filter(year == past_year & !is.na(gdp)) %>%
 ggplot(aes(group, dollars_per_day)) +
 geom_boxplot() +
 scale_y_continuous(trans = "log2") +
 xlab("") +
 theme(axis.text.x = element_text(angle = 90, hjust = 1))
```

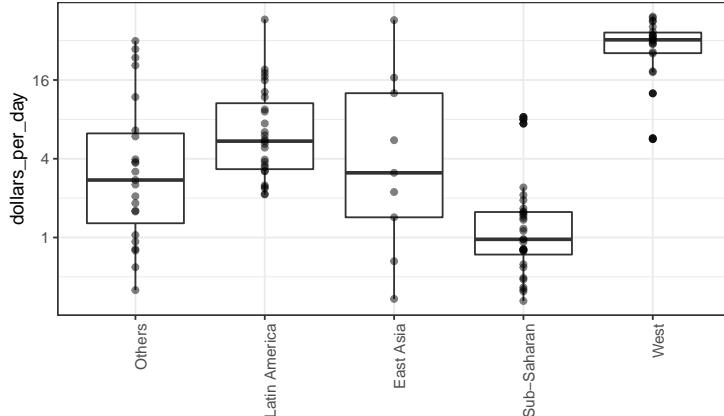
p



Los diagramas de caja tienen la limitación de que al resumir los datos en cinco números,

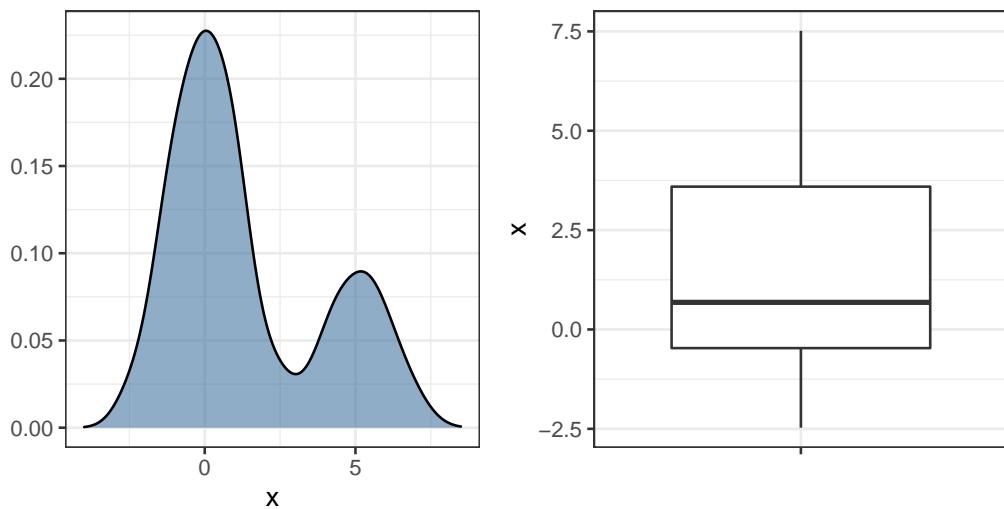
se pueden perder características importantes de los datos. Una forma de evitar esto es mostrando los datos.

```
p + geom_point(alpha = 0.5)
```



### 9.7.2 Gráficos *ridge*

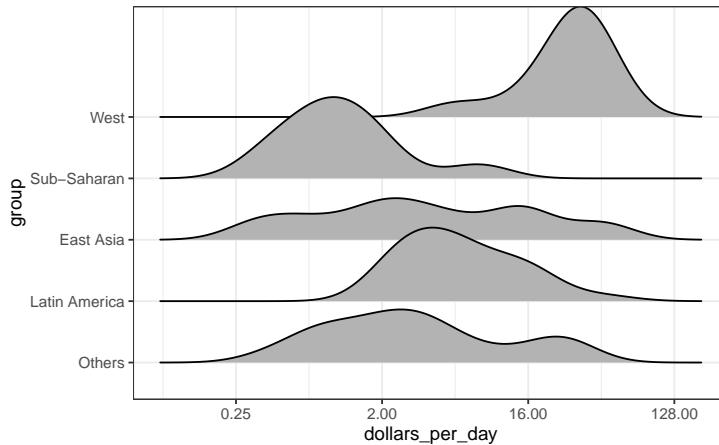
Mostrar cada punto individual no siempre revela características importantes de la distribución. Aunque no es el caso aquí, cuando el número de puntos de datos es demasiado grande acabamos sobregraficando y mostrar los datos puede ser contraproducente. Los diagramas de caja ayudan con esto al proveer un resumen de cinco números, pero esto también tiene limitaciones. Por ejemplo, los diagramas de caja no revelan distribuciones bimodales. Para ver esto, miren los dos gráficos abajo que resumen el mismo set de datos:



En los casos en que nos preocupa que el resumen del diagrama de caja sea demasiado

simplista, podemos mostrar densidades suaves o histogramas apilados utilizando *gráficos ridge*. Como estamos acostumbrados a visualizar densidades con valores en el eje-x, las apilamos verticalmente. Además, debido a que necesitamos más espacio en este enfoque, es conveniente superponerlos. El paquete **ggridges** incluye una función conveniente para hacer esto. Abajo vemos los datos de ingresos, que mostramos arriba con diagramas de caja, pero ahora visualizados con un *gráfico ridge*.

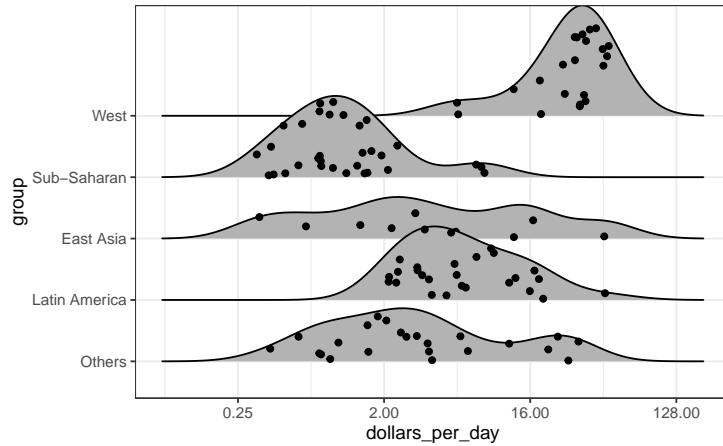
```
library(ggridges)
p <- gapminder %>%
 filter(year == past_year & !is.na(dollars_per_day)) %>%
 ggplot(aes(dollars_per_day, group)) +
 scale_x_continuous(trans = "log2")
p + geom_density_ridges()
```



Tengan en cuenta que tenemos que invertir el x e y que se usaron para el diagrama de caja. Un parámetro útil de `geom_density_ridges` es `scale`, que les permite determinar cuánto superponer; por ejemplo, `scale = 1` significa que no hay superposición. Valores mayores que 1 resultan en mayor superposición.

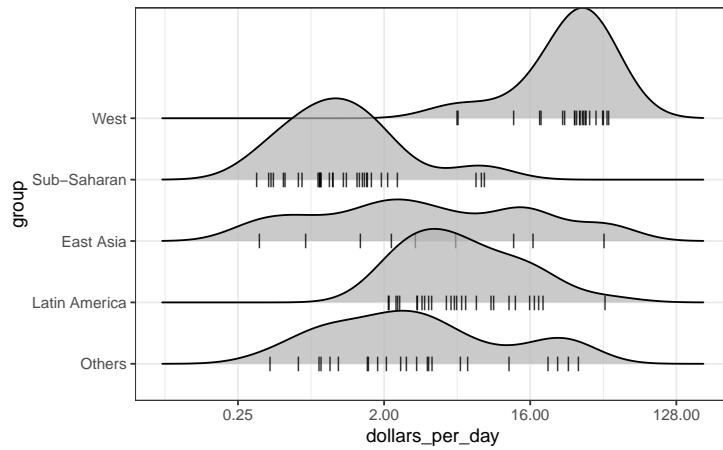
Si el número de puntos de datos es lo suficientemente pequeño, podemos añadirlos al gráfico *ridge* usando el siguiente código:

```
p + geom_density_ridges(jittered_points = TRUE)
```



Por defecto, la altura de los puntos está *jittered* y no se debe interpretar de ninguna manera. Para mostrar puntos de datos, pero sin usar *jitter*, podemos usar el siguiente código para agregar lo que se conoce como una representación *rug* de los datos.

```
p + geom_density_ridges(jittered_points = TRUE,
 position = position_points_jitter(height = 0),
 point_shape = '|', point_size = 3,
 point_alpha = 1, alpha = 0.7)
```

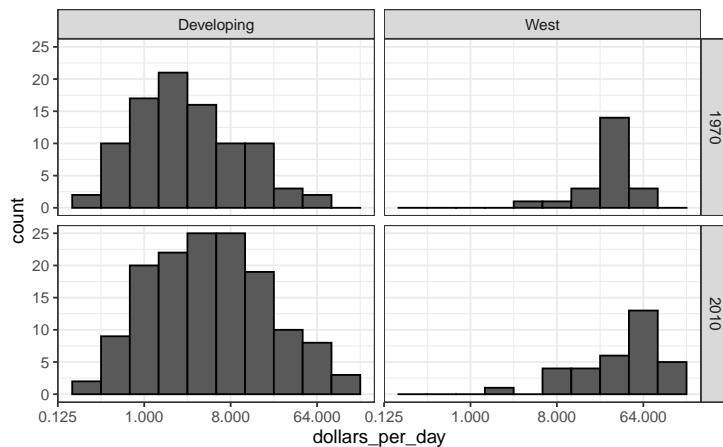


### 9.7.3 Ejemplo: distribuciones de ingresos de 1970 versus 2010

La exploración de datos muestra claramente que en 1970 hubo una dicotomía del “Oeste versus el Resto”. ¿Pero persiste esta dicotomía? Vamos a usar `facet_grid` para ver cómo han cambiado las distribuciones. Para comenzar, nos enfocamos en dos grupos: el Oeste y el Resto. Hacemos cuatro histogramas.

```
past_year <- 1970
present_year <- 2010
```

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



Antes de interpretar los hallazgos de este gráfico, notamos que hay más países representados en los histogramas de 2010 que en 1970: los conteos totales son mayores. Una razón para esto es que varios países se fundaron después de 1970. Por ejemplo, la Unión Soviética se dividió en diferentes países durante la década de 1990. Otra razón es que hay mas datos disponibles para más países en 2010.

Rehacemos los gráficos utilizando solo países con datos disponibles para ambos años. En la parte sobre *wrangling* de datos de este libro, aprenderemos a usar herramientas de **tidyverse** que nos permitirá escribir código eficiente para esto, pero aquí podemos usar un código sencillo usando la función **intersect**:

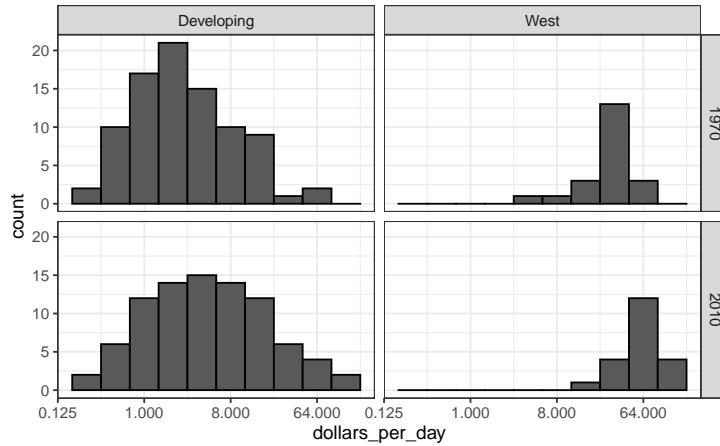
```
country_list_1 <- gapminder %>%
 filter(year == past_year & !is.na(dollars_per_day)) %>%
 pull(country)

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

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

Estos 108 constituyen 86% de la población mundial, por lo que este subconjunto debe ser representativo.

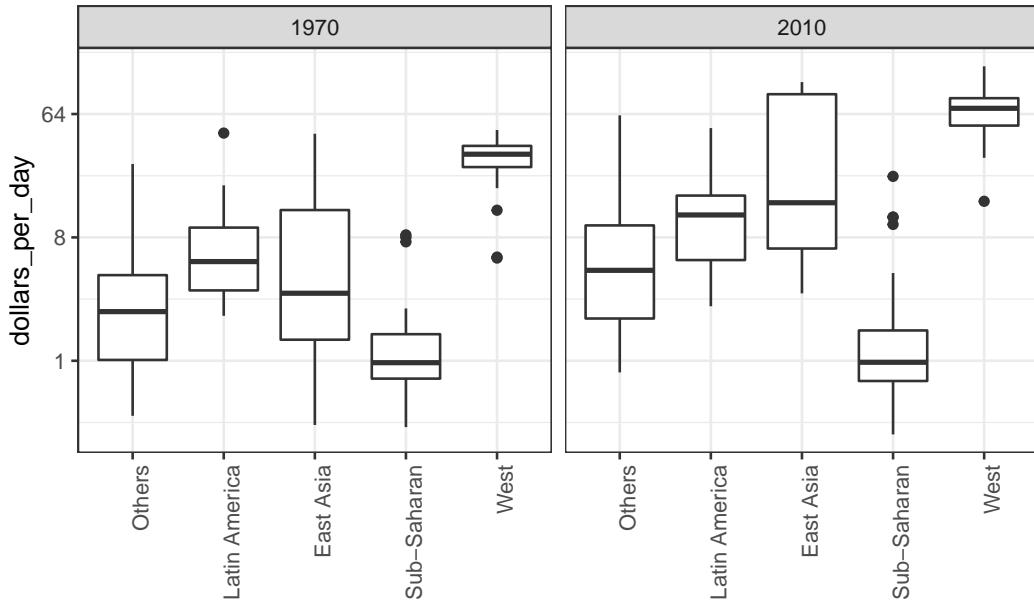
Vamos a rehacer el gráfico, pero solo para este subconjunto simplemente agregando **country %in% country\_list** a la función **filter**:



Ahora vemos que los países ricos se han vuelto un poco más ricos, pero en términos de porcentaje, los países pobres parecen haber mejorado más. En particular, vemos que la proporción de países *en desarrollo* que ganan más de \$16 por día aumentó sustancialmente.

Para ver qué regiones específicas mejoraron más, podemos rehacer los diagramas de caja que hicimos anteriormente, pero ahora añadimos el año 2010 y luego usamos *facet* para comparar los dos años.

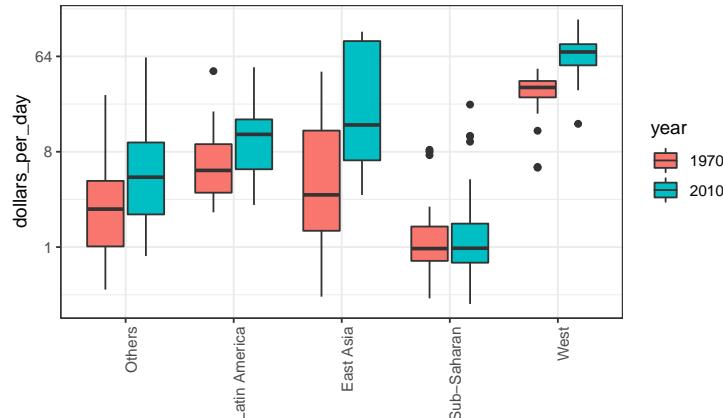
```
gapminder %>%
 filter(year %in% years & country %in% country_list) %>%
 ggplot(aes(group, dollars_per_day)) +
 geom_boxplot() +
 theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
 scale_y_continuous(trans = "log2") +
 xlab("") +
 facet_grid(. ~ year)
```



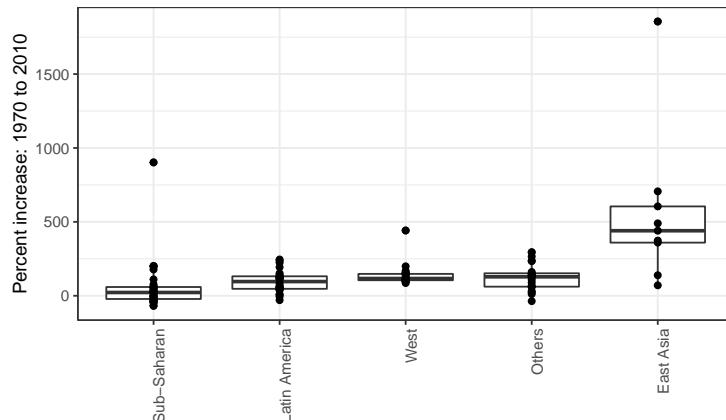
Aquí pausamos para presentar otra importante característica de **ggplot2**. Como queremos comparar cada región antes y después, sería conveniente tener el diagrama de caja de 1970 al lado del de 2010 para cada región. En general, las comparaciones son más fáciles cuando los datos se grafican uno al lado del otro.

Entonces, en lugar de separar en facetas, mantenemos los datos de cada año juntos y pedimos colorearlos (o rellenarlos) según el año. Tengan en cuenta que los grupos se separan automáticamente por año y cada par de diagramas de caja se dibujan uno al lado del otro. Como el año es un número, lo convertimos en un factor ya que **ggplot2** asigna automáticamente un color a cada categoría de un factor. Recuerden que tenemos que convertir la columnas `year` de numérica a factor.

```
gapminder %>%
 filter(year %in% years & country %in% country_list) %>%
 mutate(year = factor(year)) %>%
 ggplot(aes(group, dollars_per_day, fill = year)) +
 geom_boxplot() +
 theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
 scale_y_continuous(trans = "log2") +
 xlab("")
```



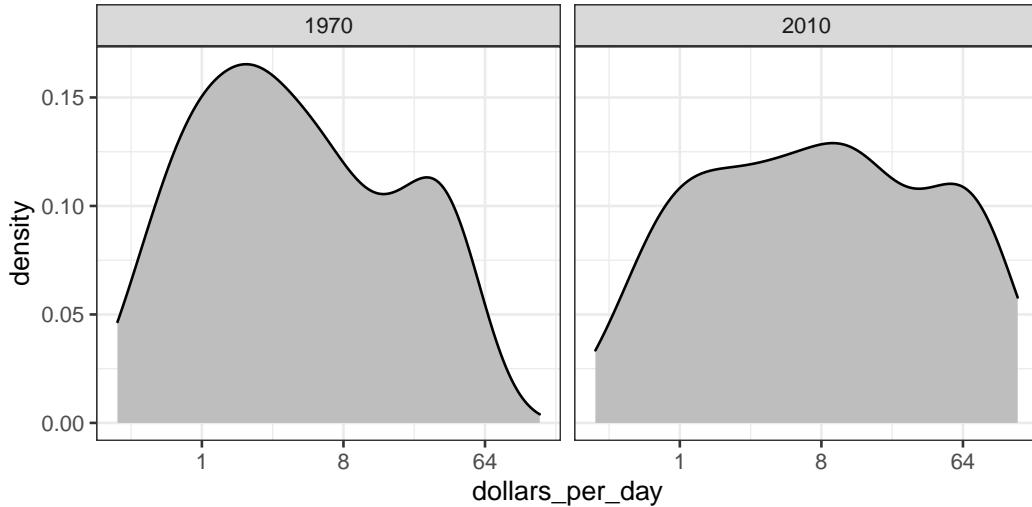
Finalmente, señalamos que si lo más que nos interesa es comparar los valores de antes y después, podría tener más sentido graficar los aumentos porcentuales. Todavía no estamos listos para aprender a codificar esto, pero así es como se vería el gráfico:



La exploración de datos previa sugiere que la brecha de ingresos entre países ricos y pobres se ha reducido considerablemente durante los últimos 40 años. Usamos una serie de histogramas y diagramas de caja para ver esto. Sugerimos una forma sucinta de transmitir este mensaje con solo un gráfico.

Empecemos observando que los gráficos de densidad para la distribución del ingreso en 1970 y 2010 transmiten el mensaje de que la brecha se está cerrando:

```
gapminder %>%
 filter(year %in% years & country %in% country_list) %>%
 ggplot(aes(dollars_per_day)) +
 geom_density(fill = "grey") +
 scale_x_continuous(trans = "log2") +
 facet_grid(. ~ year)
```



En el gráfico de 1970, vemos dos modas claras: países pobres y ricos. En 2010, parece que algunos de los países pobres se han desplazado hacia la derecha, cerrando la brecha.

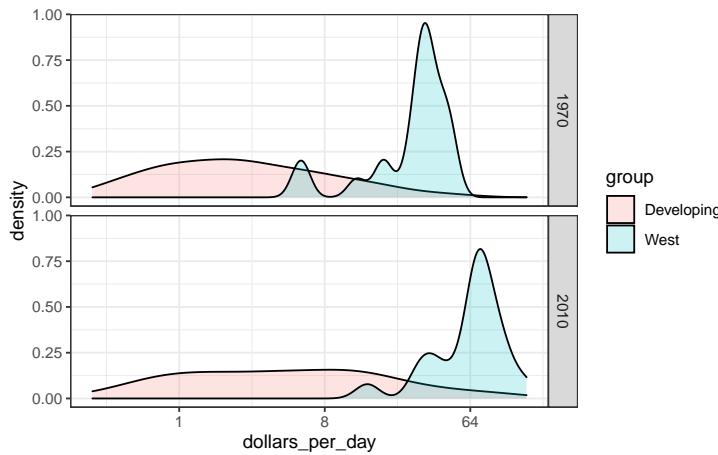
El próximo mensaje que debemos transmitir es que la razón de este cambio en distribución es que varios países pobres se hicieron más ricos, en lugar de que algunos países ricos se hicieron más pobres. Para hacer esto, podemos asignar un color a los grupos que identificamos durante la exploración de datos.

Sin embargo, primero tenemos que aprender a hacer estas densidades suaves de una manera que conserve la información sobre el número de países en cada grupo. Para entender por qué necesitamos esto, recuerden la discrepancia en el tamaño de cada grupo:

| Developing | West |
|------------|------|
| 87         | 21   |

Pero cuando superponemos dos densidades, el comportamiento por defecto es que el área representada por cada distribución sume a 1, independientemente del tamaño de cada grupo:

```
gapminder %>%
 filter(year %in% years & country %in% country_list) %>%
 mutate(group = ifelse(group == "West", "West", "Developing")) %>%
 ggplot(aes(dollars_per_day, fill = group)) +
 scale_x_continuous(trans = "log2") +
 geom_density(alpha = 0.2) +
 facet_grid(year ~ .)
```



El gráfico de arriba hace que parezca que hay el mismo número de países en cada grupo. Para cambiar esto, necesitaremos aprender a acceder a las variables calculadas con la función `geom_density`.

#### 9.7.4 Cómo obtener acceso a variables calculadas

Para que las áreas de estas densidades sean proporcionales al tamaño de los grupos, simplemente multiplicamos los valores del eje-y por el tamaño del grupo. En el archivo de ayuda de `geom_density`, vemos que las funciones calculan una variable denominada `count` que hace exactamente esto. Queremos que esta variable, y no la densidad, esté en el eje-y.

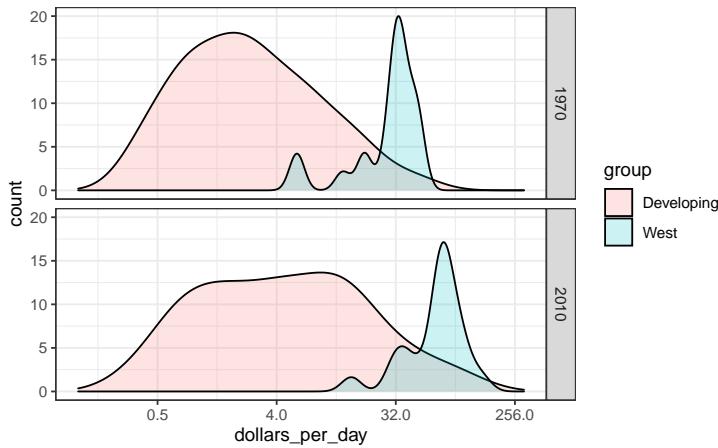
En `ggplot2`, obtenemos acceso a estas variables rodeando el nombre con dos puntos. Por lo tanto, utilizaremos el siguiente mapeo:

```
aes(x = dollars_per_day, y = ..count..)
```

Ahora podemos crear el diagrama deseado simplemente cambiando el mapeo en el fragmento del código anterior. También ampliaremos los límites del eje-x.

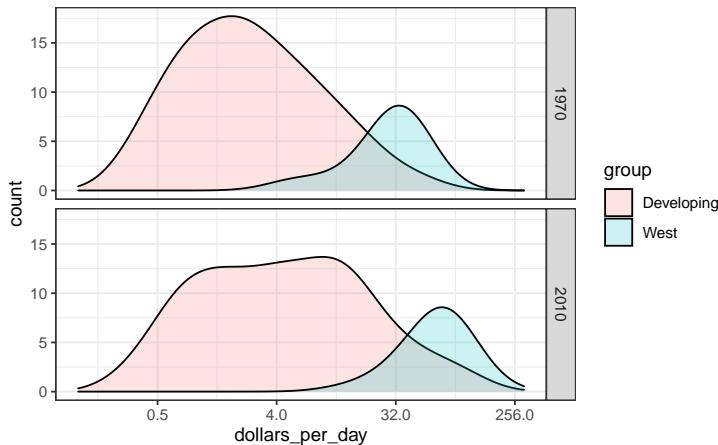
```
p <- gapminder %>%
 filter(year %in% years & country %in% country_list) %>%
 mutate(group = ifelse(group == "West", "West", "Developing")) %>%
 ggplot(aes(dollars_per_day, y = ..count.., fill = group)) +
 scale_x_continuous(trans = "log2", limit = c(0.125, 300))

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



Si queremos que las densidades sean más suaves, usamos el argumento `bw` para que se use el mismo parámetro de suavizado en cada densidad. Seleccionamos 0.75 después de probar varios valores.

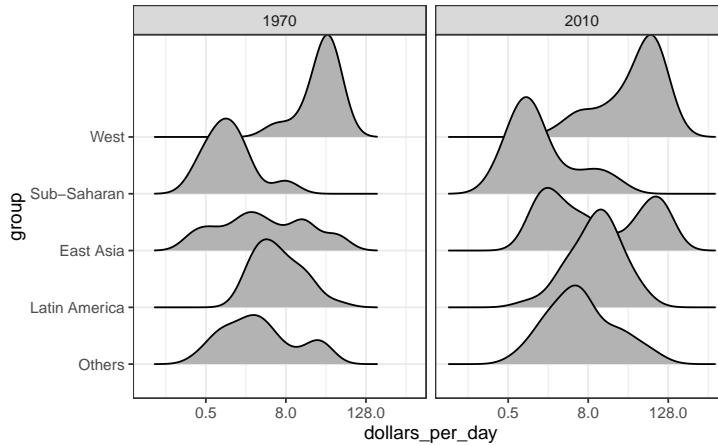
```
p + geom_density(alpha = 0.2, bw = 0.75) + facet_grid(year ~ .)
```



Este gráfico ahora muestra lo que está sucediendo muy claramente. La distribución del mundo en desarrollo está cambiando. Aparece una tercera moda formada por los países que más redujeron la brecha.

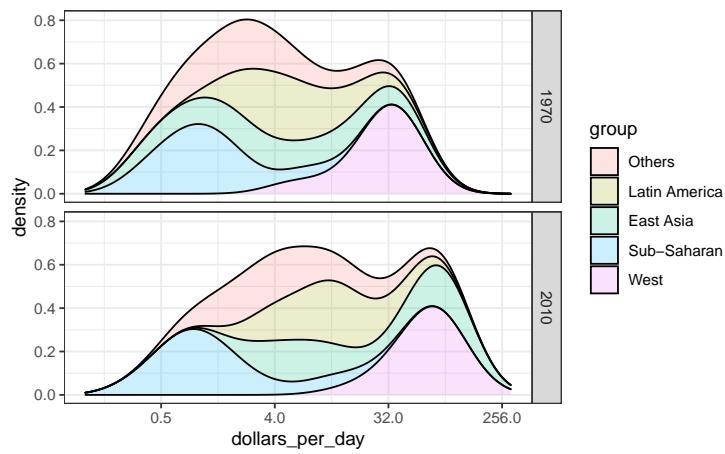
Para visualizar si alguno de los grupos definidos anteriormente son la causa principal de estos cambios, rápidamente podemos hacer un gráfico *ridge*:

```
gapminder %>%
 filter(year %in% years & !is.na(dollars_per_day)) %>%
 ggplot(aes(dollars_per_day, group)) +
 scale_x_continuous(trans = "log2") +
 geom_density_ridges(adjust = 1.5) +
 facet_grid(. ~ year)
```



Otra forma de lograr esto es apilando las densidades una encima de otra:

```
gapminder %>%
 filter(year %in% years & country %in% country_list) %>%
 group_by(year) %>%
 mutate(weight = population/sum(population)*2) %>%
 ungroup() %>%
 ggplot(aes(dollars_per_day, fill = group)) +
 scale_x_continuous(trans = "log2", limit = c(0.125, 300)) +
 geom_density(alpha = 0.2, bw = 0.75, position = "stack") +
 facet_grid(year ~ .)
```

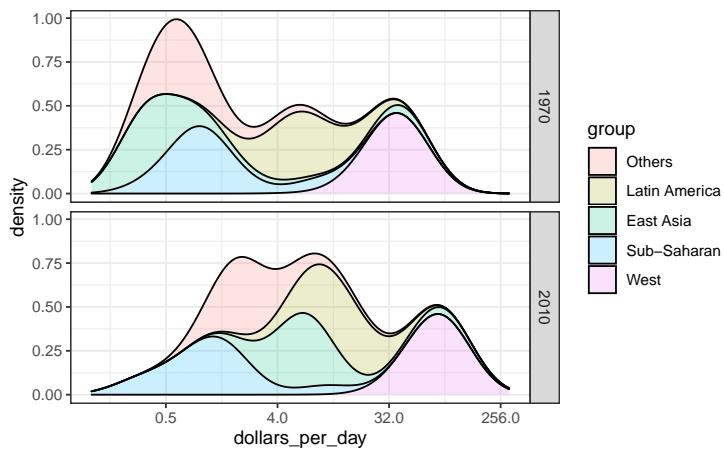


Aquí podemos ver claramente cómo las distribuciones para Asia Oriental, América Latina y otros se desplazan notablemente hacia la derecha. Mientras que África subsahariana permanece estancada.

Noten que ordenamos los niveles del grupo para que la densidad del Occidente se grafique primero, luego África subsahariana. Tener los dos extremos graficados primero nos permite ver mejor la bimodalidad restante.

### 9.7.5 Densidades ponderadas

Como punto final, notamos que estas distribuciones ponderan cada país igual. Entonces si la mayoría de la población está mejorando, pero viviendo en un país muy grande, como China, podríamos no apreciar esto. De hecho, podemos ponderar las densidades suaves usando el argumento de mapeo `weight`. El gráfico entonces se ve así:

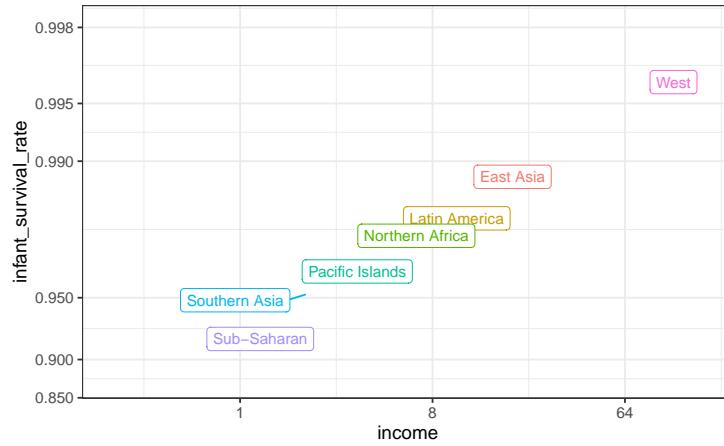


Esta figura en particular muestra muy claramente cómo se está cerrando la brecha de distribución de ingresos y que la mayoría de países que siguen en la pobreza están en África subsahariana.

## 9.8 La falacia ecológica y la importancia de mostrar los datos

A lo largo de esta sección, hemos estado comparando regiones del mundo. Hemos visto que, en promedio, algunas regiones obtienen mejores resultados que otras. En esta sección, nos enfocamos en describir la importancia de la variabilidad dentro de los grupos al examinar la relación entre las tasas de mortalidad infantil de un país y el ingreso promedio.

Definimos algunas regiones más y comparamos los promedios entre regiones:



La relación entre estas dos variables es casi perfectamente lineal y el gráfico muestra una diferencia dramática. Mientras que en el Occidente mueren menos del 0.5% de los bebés, ¡en África subsahariana la tasa es superior al 6%!

Observen que el gráfico utiliza una nueva transformación, la transformación logística.

### 9.8.1 Transformación logística

La transformación logística o *logit* para una proporción o tasa  $p$  se define como:

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

Cuando  $p$  es una proporción o probabilidad, la cantidad que transformamos con el logaritmo,  $p/(1-p)$ , se llama *odds*. En este caso  $p$  es la proporción de bebés que sobrevivieron. Los *odds* nos dicen cuántos más bebés se espera que sobrevivan a que mueran. La transformación logarítmica lo hace simétrico. Si las tasas son iguales, entonces el *log odds* es 0. Los aumentos multiplicativos se convierten en incrementos positivos o negativos, respectivamente.

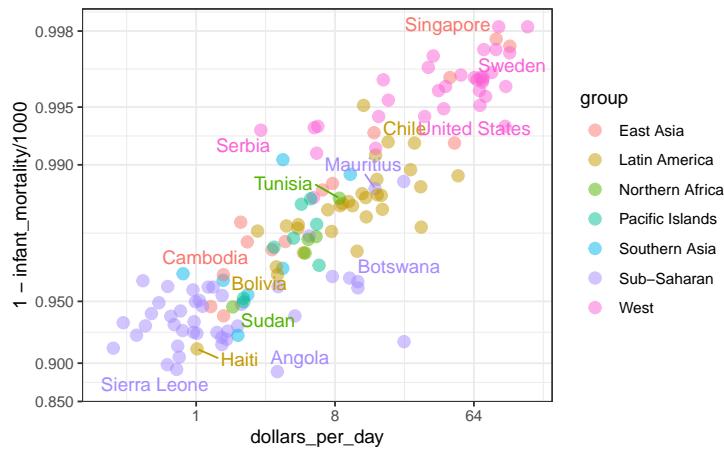
Esta escala es útil cuando queremos resaltar diferencias cercanas a 0 o 1. Para las tasas de supervivencia, esto es importante porque una tasa de supervivencia del 90% es inaceptable, mientras que una supervivencia del 99% es relativamente buena. Preferiríamos mucho una tasa de supervivencia más cercana al 99.9%. Queremos que nuestra escala resalte estas diferencias y el *logit* lo hace. Recuerden que  $99.9/0.1$  es aproximadamente 10 veces más grande que  $99/1$ , que es aproximadamente 10 veces más grande que  $90/10$ . Al usar el logaritmo, estos incrementos multiplicativos se convierten en aumentos constantes.

### 9.8.2 Mostrar los datos

Ahora, regresamos a nuestro gráfico. Basado en el gráfico anterior, ¿concluimos que un país con ingresos bajos está destinado a tener una tasa de supervivencia baja? Además, ¿concluimos que las tasas de supervivencia en el África subsahariana son más bajas que en el sur de Asia, que a su vez son más bajas que en las islas del Pacífico y así sucesivamente?

Llegar a esta conclusión basada en un gráfico que muestra promedios se denomina una

*fallacia ecológica.* La relación casi perfecta entre las tasas de supervivencia y los ingresos solo se observa para los promedios a nivel regional. Cuando mostramos todos los datos, vemos una historia más complicada:



Específicamente, vemos que hay una gran cantidad de variabilidad. Vemos que los países de las mismas regiones pueden ser bastante diferentes y que los países con los mismos ingresos pueden tener diferentes tasas de supervivencia. Por ejemplo, mientras que, en promedio, África subsahariana tuvo los peores resultados económicos y de salud, existe una gran variabilidad dentro de ese grupo. Mauricio y Botswana están mejores que Angola y Sierra Leona, con Mauricio comparable a países occidentales.

# 10

---

## *Principios de visualización de datos*

---

Ya hemos presentado algunas reglas a seguir mientras creamos gráficos para nuestros ejemplos. Aquí nuestro objetivo es ofrecer algunos principios generales que podemos usar como guía para una visualización de datos efectiva. Gran parte de esta sección se basa en una charla de Karl Broman<sup>1</sup> titulada “Creating Effective Figures and Tables”<sup>2</sup> e incluye algunas de las figuras que se hicieron con el código que Karl pone a disposición en su repositorio de GitHub<sup>3</sup>, así como las notas de la clase “Introduction to Data Visualization” de Peter Aldhous<sup>4</sup>. Siguiendo el enfoque de Karl, mostramos algunos ejemplos de estilos de gráficos que debemos evitar, explicamos cómo mejorarlos y entonces los usamos como motivación para una lista de principios. Además, comparamos y contrastamos los gráficos que siguen estos principios con otros que los ignoran.

Los principios se basan principalmente en investigaciones relacionadas a la manera en que los humanos detectan patrones y hacen comparaciones visuales. Los enfoques preferidos son aquellos que mejor se adaptan a la forma en que nuestros cerebros procesan la información visual. Al escoger las herramientas de visualización, es importante tener en cuenta nuestro objetivo. Podemos estar comparando una cantidad de números suficientemente pequeña que se pueden distinguir, describiendo distribuciones de datos categóricos o valores numéricos, comparando los datos de dos grupos o describiendo la relación entre dos variables. Todo esto afecta la presentación que escojemos. Como nota final, queremos enfatizar que para los científicos de datos es importante adaptar y optimizar los gráficos para la audiencia. Por ejemplo, un gráfico exploratorio hecho para nosotros será diferente a una tabla destinada a comunicar un hallazgo a una audiencia general.

Utilizaremos estos paquetes:

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

---

### 10.1 Cómo codificar datos utilizando señales visuales

Comenzamos describiendo algunos principios para codificar datos. Hay varios acercamientos a nuestra disposición que incluyen posición, largo, ángulos, área, brillo y tono de color.

Para ilustrar cómo se comparan algunas de estas estrategias, supongan que queremos in-

---

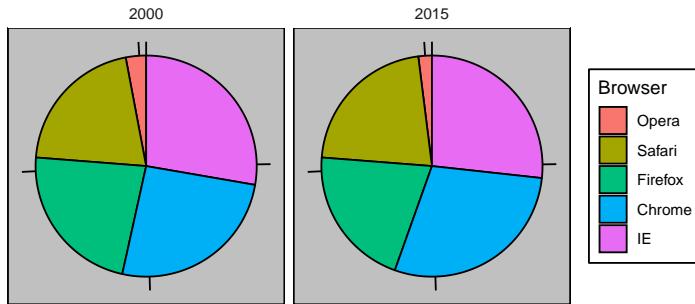
<sup>1</sup><http://kbroman.org/>

<sup>2</sup><https://www.biostat.wisc.edu/~kbroman/presentations/graphs2017.pdf>

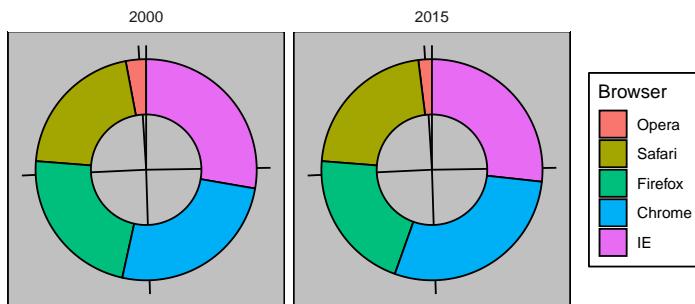
<sup>3</sup>[https://github.com/kbroman/Talk\\_Graphs](https://github.com/kbroman/Talk_Graphs)

<sup>4</sup><http://paldhous.github.io/ucb/2016/dataviz/index.html>

formar los resultados de dos encuestas hipotéticas, tomadas en 2000 y luego en 2015, con respecto a la preferencia de navegador. Para cada año, simplemente estamos comparando cinco cantidades: los cinco porcentajes. Una representación gráfica de porcentajes ampliamente utilizada y popularizada por Microsoft Excel, es el gráfico circular:



Aquí estamos representando cantidades con áreas y ángulos, ya que tanto el ángulo como el área de cada sección del gráfico son proporcionales a la cantidad que representa el sector. Esto resulta ser una opción subóptima dado que, como lo demuestran los estudios de percepción, los humanos no son buenos para cuantificar ángulos con precisión y son aún peores cuando el área es la única señal visual disponible. El gráfico de anillo es un ejemplo de un gráfico que usa solo área:



Para ver cuán difícil es cuantificar los ángulos y el área, recuerden que las clasificaciones y todos los porcentajes en los gráficos anteriores cambiaron de 2000 a 2015. ¿Pueden determinar los porcentajes reales y clasificar la popularidad de los navegadores? ¿Pueden ver cómo los porcentajes cambiaron de 2000 a 2015? No es fácil distinguirlo del gráfico. De hecho, la función `pie` de la página de ayuda de R señala que:

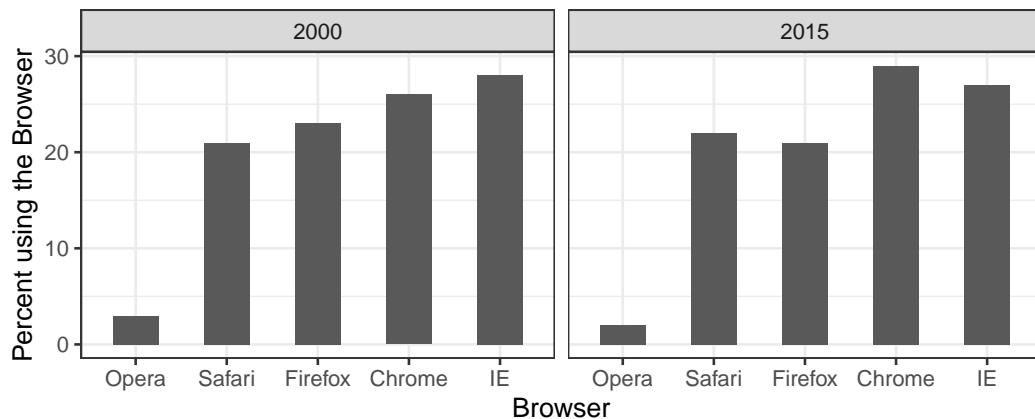
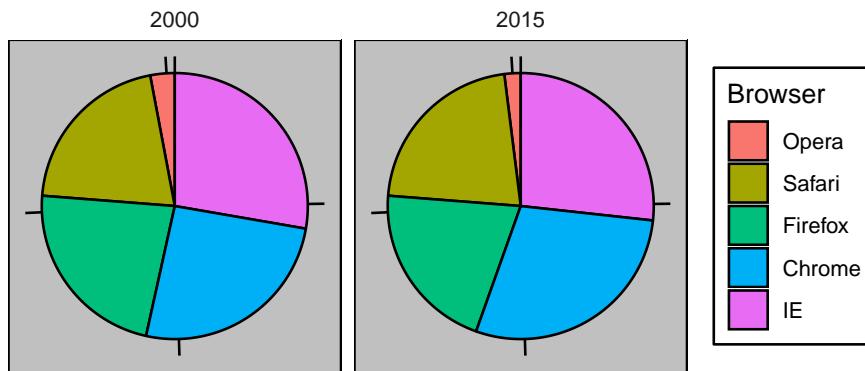
Los gráficos circulares son una forma muy mala de mostrar información. El ojo es bueno

juzgando medidas lineales y malo juzgando áreas relativas. Un diagrama de barras o de puntos es una forma preferible de mostrar este tipo de datos.

En este caso, simplemente mostrar los números no solo es más claro, sino que también ahorraría costos de impresión si imprimen una copia en papel:

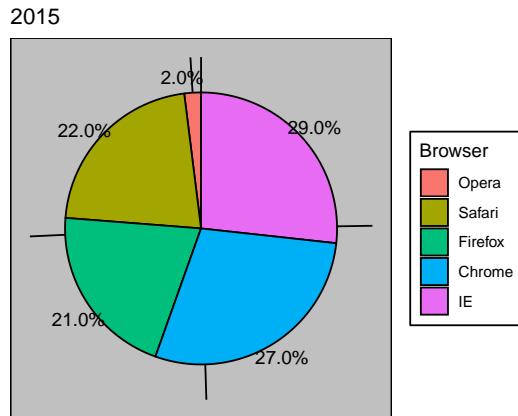
| Browser | 2000 | 2015 |
|---------|------|------|
| Opera   | 3    | 2    |
| Safari  | 21   | 22   |
| Firefox | 23   | 21   |
| Chrome  | 26   | 29   |
| IE      | 28   | 27   |

La forma preferida de graficar estas cantidades es usar la longitud y la posición como señales visuales, ya que los humanos son mucho mejores juzgando medidas lineales. El diagrama de barras usa este enfoque al usar barras de longitud proporcional a las cantidades de interés. Al añadir líneas horizontales a valores estratégicamente elegidos, en este caso en cada múltiplo de 10, aliviamos la carga visual de cuantificar a través de la posición de la parte superior de las barras. Comparen y contrasten la información que podemos extraer de los siguientes dos pares de gráficos.



Observen lo fácil que es ver las diferencias en el diagrama de barras. De hecho, ahora podemos determinar los porcentajes reales siguiendo una línea horizontal hasta el eje-x.

Si por alguna razón tienen que hacer un gráfico circular, etiqueten cada sección del círculo con su porcentaje respectivo para que la audiencia no tenga que inferirlos de los ángulos o del área:

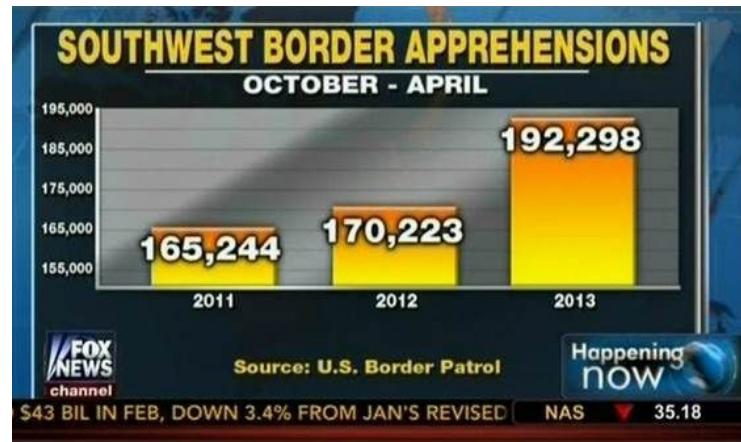


En general, cuando se muestran cantidades, se prefieren la posición y la longitud sobre los ángulos y/o el área. El brillo y el color son aún más difíciles de cuantificar que los ángulos. Pero, como veremos más adelante, a veces son útiles cuando se deben mostrar más de dos dimensiones a la vez.

## 10.2 Cuándo incluir 0

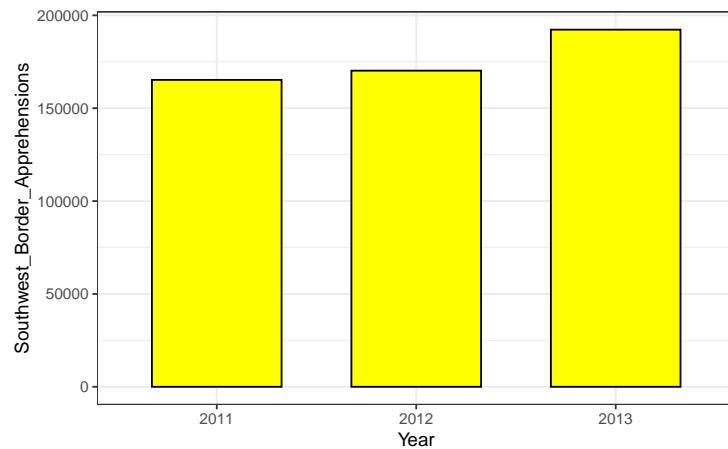
Cuando se usan diagramas de barras, es erróneo no comenzar las barras en 0. Esto se debe a que, al usar un diagrama de barras, estamos implicando que la longitud es proporcional a las cantidades que se muestran. Al evitar 0, se pueden hacer diferencias relativamente pequeñas verse mucho más grandes de lo que realmente son. Este acercamiento a menudo es utilizado por políticos o medios de comunicación que intentan exagerar la diferencia. A continuación se muestra un ejemplo ilustrativo utilizado por Peter Aldhous en su conferencia<sup>5</sup>.

<sup>5</sup><http://paldhous.github.io/ucb/2016/dataviz/week2.html>



(Fuente: Fox News, vía Media Matters<sup>6</sup>.)

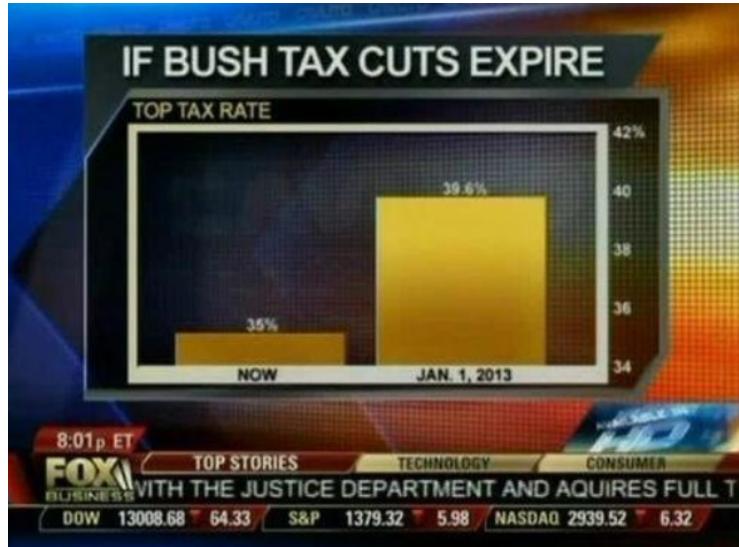
En el gráfico anterior, las detenciones parecen haber casi triplicado cuando, de hecho, solo han aumentado aproximadamente un 16%. Comenzar el gráfico en 0 ilustra esto claramente:



Abajo vemos otro ejemplo, que se describe en detalle en un artículo del blog “Flowing Data”:

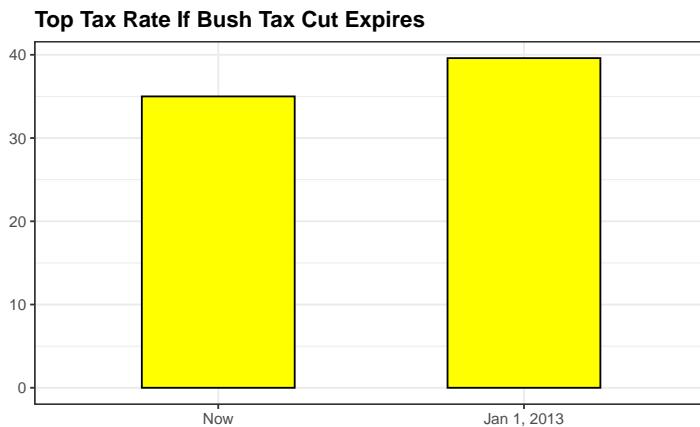
---

<sup>6</sup><http://mediamatters.org/blog/2013/04/05/fox-news-newest-dishonest-chart-immigration-enf/193507>



(Fuente: Fox News, a través de Flowing Data<sup>7</sup>.)

Este gráfico hace que un aumento del 13% parezca cinco veces más grande. Aquí tenemos el gráfico apropiado:



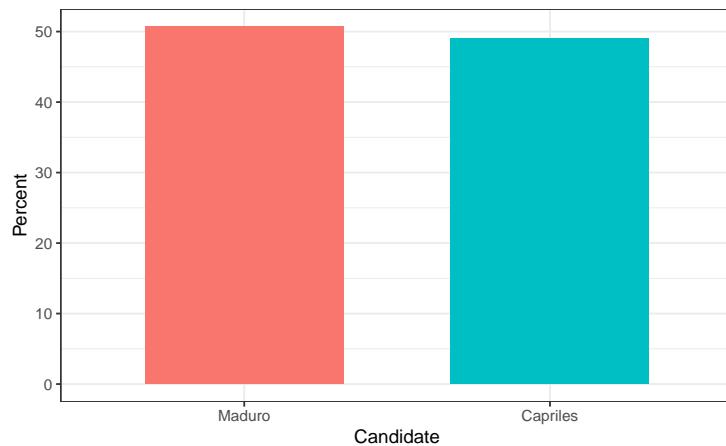
Finalmente, aquí tenemos un ejemplo extremo que hace que una diferencia muy pequeña, de menos de 2%, parezca 10 a 100 veces más grande:

<sup>7</sup><http://flowingdata.com/2012/08/06/fox-news-continues-charting-excellence/>



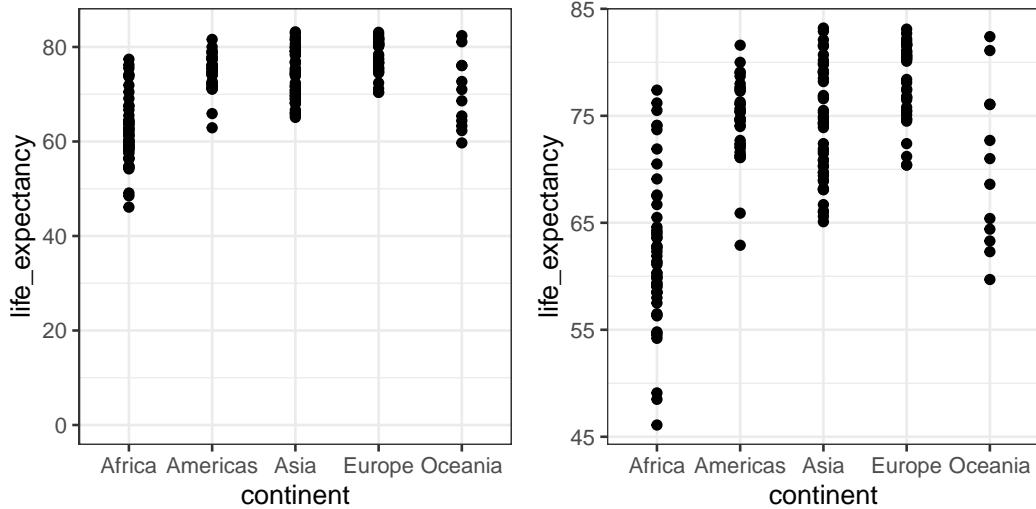
(Fuente: El Mundo<sup>8</sup> y Diego Mariano.)

Aquí está el gráfico apropiado:



Cuando se usa posición en lugar de longitud, no es necesario incluir 0. Este es el caso en particular cuando queremos comparar las diferencias entre los grupos en relación con la variabilidad dentro de un grupo. Aquí tenemos un ejemplo ilustrativo que muestra la esperanza de vida promedio de cada país estratificado por continente en 2012:

<sup>8</sup><https://www.elmundo.es/america/2013/04/15/venezuela/1366029653.html>



Tengan en cuenta que en el gráfico de la izquierda, que incluye 0, el espacio entre 0 y 43 no añade información y hace que sea más difícil comparar la variabilidad entre y dentro del grupo.

### 10.3 No distorsionar cantidades

Durante el discurso del Estado de la Unión de 2011 del Presidente Barack Obama, se utilizó el siguiente gráfico para comparar el PIB de EE. UU. con el PIB de cuatro naciones competidoras:

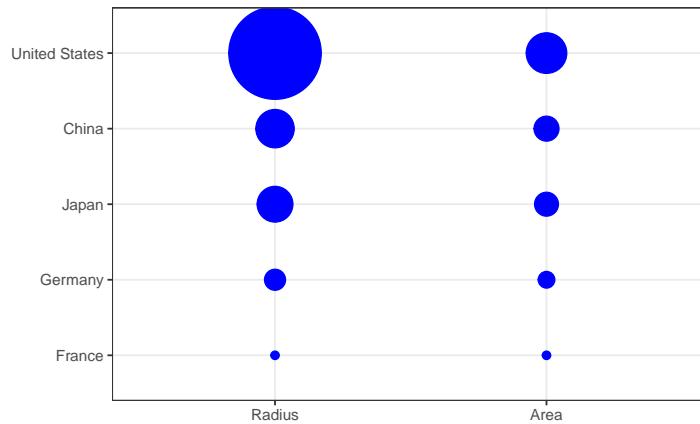


(Fuente: The 2011 State of the Union Address<sup>9</sup>.)

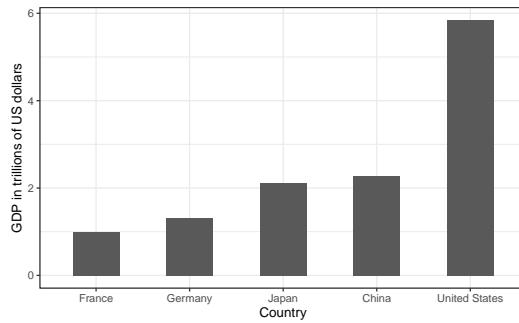
Si juzgamos por el área de los círculos, Estados Unidos parece tener una economía cinco veces

<sup>9</sup><https://www.youtube.com/watch?v=kl2g40GoRwg>

más grande que la de China y más de 30 veces más grande que la de Francia. Sin embargo, si nos fijamos en los números actuales, vemos que este no es el caso. Las proporciones son 2.6 y 5.8 veces mayores que las de China y Francia, respectivamente. La razón de esta distorsión es que el radio del círculo, en lugar del área, se hizo proporcional a la cantidad, lo que implica que la proporción entre las áreas es cuadrada: 2.6 se convierte en 6.5 y 5.8 se convierte en 34.1. Aquí hay una comparación de los círculos que obtenemos si hacemos que el valor sea proporcional al radio y al área:



No sorprende entonces que por defecto **ggplot2** use el área en lugar del radio. Sin embargo, en este caso, realmente no deberíamos usar el área, ya que podemos usar la posición y la longitud:

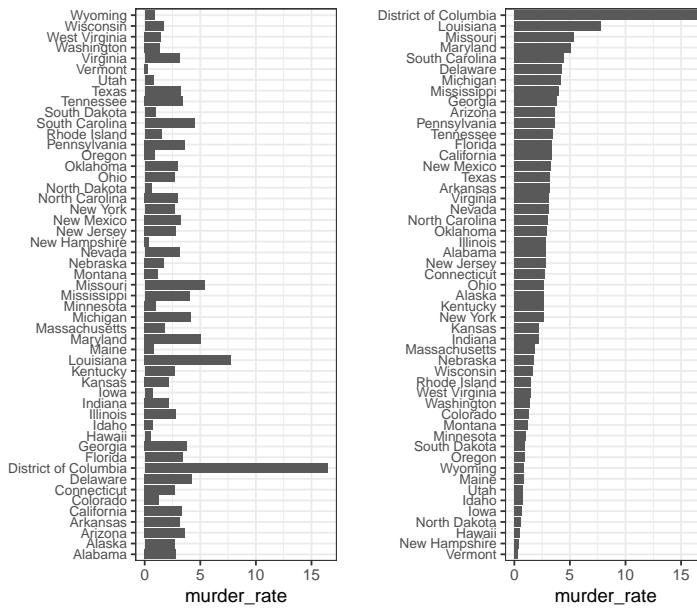


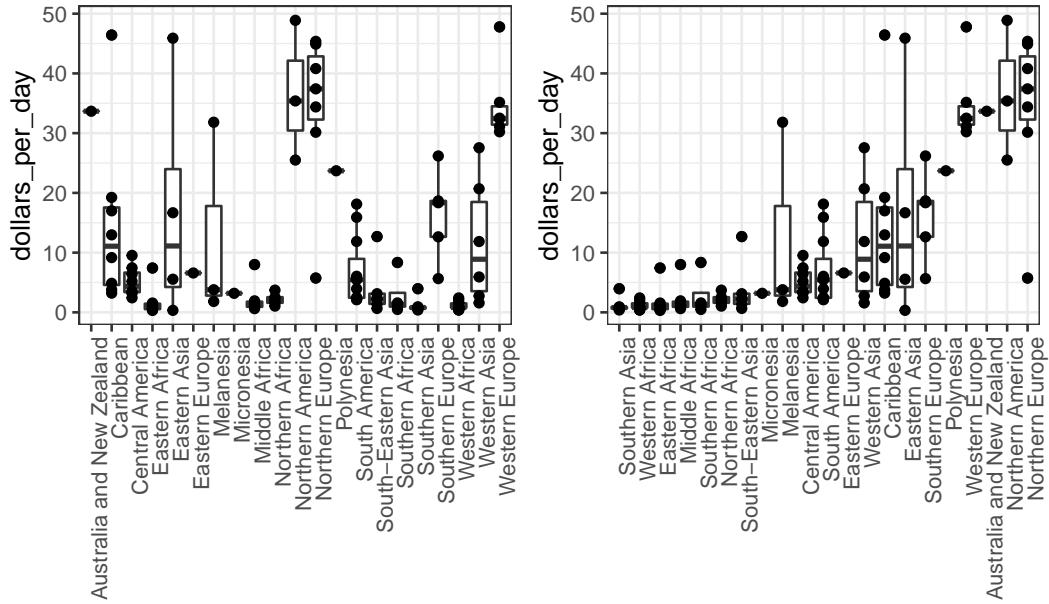
## 10.4 Ordenar categorías por un valor significativo

Cuando uno de los ejes se usa para mostrar categorías, como se hace en los diagramas de barras, el comportamiento por defecto de **ggplot2** es ordenar las categorías alfabéticamente cuando se definen por cadenas de caracteres. Si están definidas por factores, se ordenan según los niveles de factores. Raras veces queremos usar el orden alfabético. En cambio, debemos ordenar por una cantidad significativa. En todos los casos anteriores, los diagramas de barras

se ordenaron según los valores que mostraban. La excepción fueron los diagramas de barras comparando navegadores. En ese caso, mantuvimos el orden igual en todos los diagramas de barras para facilitar la comparación. Específicamente, en vez de ordenar los navegadores por separado en los dos años, ordenamos ambos años por el valor promedio de 2000 y 2015.

Anteriormente aprendimos a usar la función `reorder`, que nos ayuda a lograr este objetivo. Para apreciar cómo el orden correcto puede ayudar a transmitir un mensaje, supongan que queremos crear un gráfico para comparar la tasa de homicidios en todos los estados de EE.UU. Estamos particularmente interesados en los estados más peligrosos y los más seguros. Tengan en cuenta la diferencia cuando ordenamos alfabéticamente, la acción por defecto, versus cuando ordenamos por la tasa real:



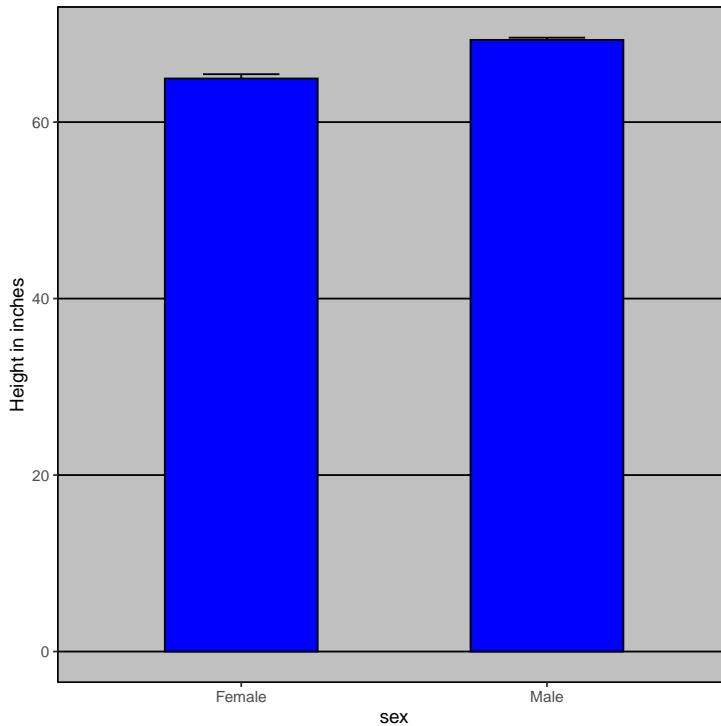


El primer gráfico ordena las regiones alfabéticamente, mientras que el segundo las ordena por la mediana del grupo.

## 10.5 Mostrar los datos

Nos hemos enfocado en mostrar cantidades únicas en todas las categorías. Ahora cambiamos nuestra atención a la visualización de datos con un enfoque en la comparación de grupos.

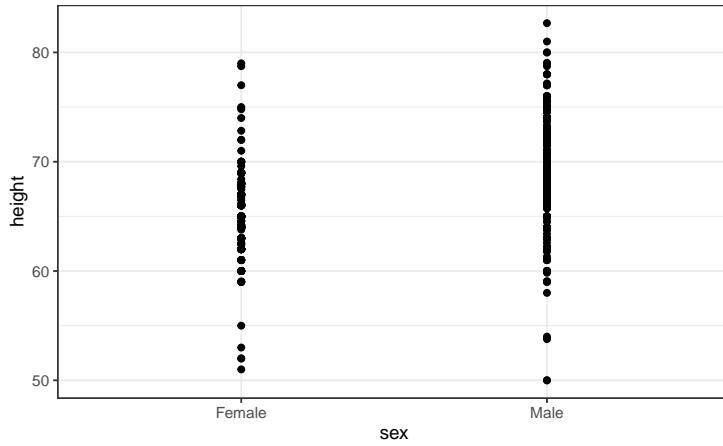
Para motivar nuestro primer principio, “mostrar los datos”, volvemos a nuestro ejemplo artificial de describir alturas a ET, un extraterrestre. Esta vez supongan que ET está interesado en la diferencia de alturas entre hombres y mujeres. Un gráfico comúnmente utilizado para comparaciones entre grupos y popularizado por software como Microsoft Excel, es el *dynamic plot*, que muestra el promedio y los errores estándar (los errores estándar se definen en un capítulo posterior, pero no los confundan con la desviación estándar de los datos). El gráfico se ve así:



El promedio de cada grupo está representado por la parte superior de cada barra y las antenas se extienden desde el promedio al promedio más dos errores estándar. Si todo lo que ET recibe es este gráfico, tendrá poca información sobre qué esperar si se encuentra con un grupo de hombres y mujeres. Las barras van a 0: ¿esto significa que hay humanos pequeños que miden menos de un pie? ¿Todos los varones son más altos que las hembras más altas? ¿Hay una rango de alturas? ET no puede responder a estas preguntas ya que casi no le hemos dado información sobre la distribución de altura.

Esto nos lleva a nuestro primer principio: mostrar los datos. Este código sencillo de **ggplot2** ya genera un gráfico más informativo que el diagrama de barras al simplemente mostrar todos los puntos de datos:

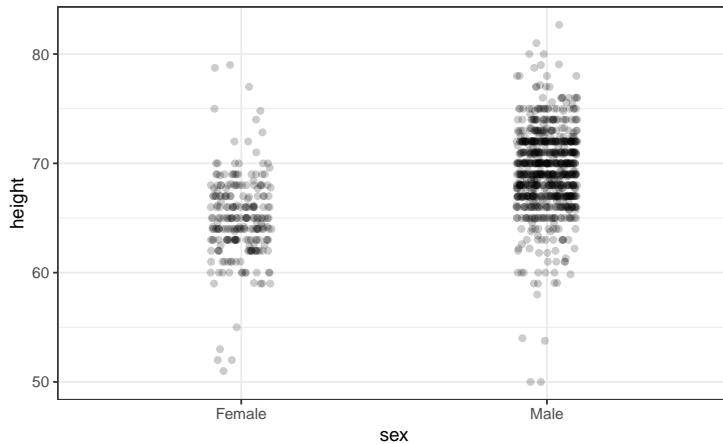
```
heights %>%
 ggplot(aes(sex, height)) +
 geom_point()
```



El gráfico anterior nos da una idea del rango de los datos. Sin embargo, este gráfico también tiene limitaciones, ya que realmente no podemos ver todos los 238 y 812 puntos graficados para hembras y varones, respectivamente, y muchos puntos están graficados uno encima del otro. Como hemos descrito anteriormente, visualizar la distribución es mucho más informativo. Pero antes de hacer esto, señalamos dos formas en que podemos mejorar un gráfico que muestra todos los puntos.

El primero es agregar *jitter*, que añade un pequeño desplazamiento aleatorio a cada punto. En este caso, agregar *jitter* horizontal no cambia la interpretación, ya que las alturas de los puntos no cambian, pero minimizamos el número de puntos que se superponen y, por lo tanto, tenemos una mejor idea visual de cómo se distribuyen los datos. Una segunda mejora proviene del uso de *alpha blending*, que hace que los puntos sean algo transparentes. Entre más puntos se superponen, más oscuro será el gráfico, que también nos ayuda tener una idea de cómo se distribuyen los puntos. Aquí está el mismo gráfico con *jitter* y *alpha blending*:

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



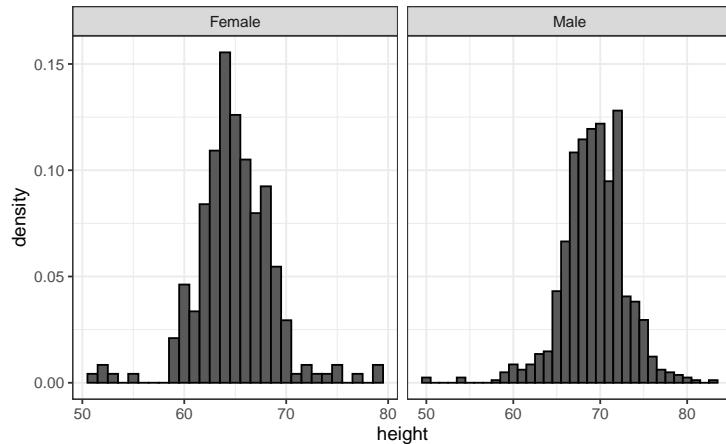
Ahora comenzamos a sentir que, en promedio, los varones son más altos que las hembras.

También observamos bandas horizontales oscuras de puntos, que demuestra que muchos estudiantes indican valores que se redondean al entero más cercano.

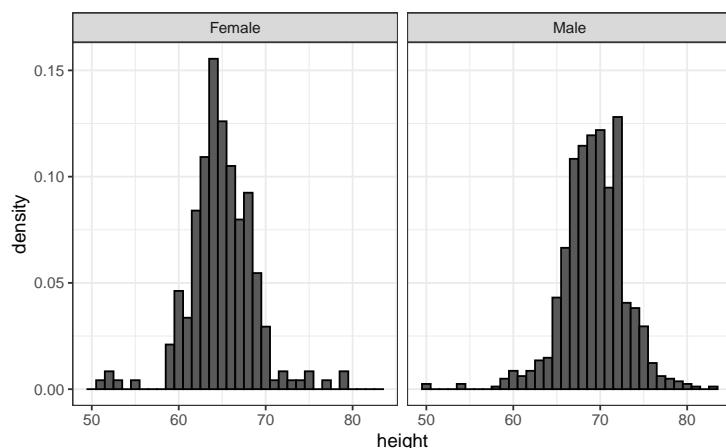
## 10.6 Cómo facilitar comparaciones

### 10.6.1 Usen ejes comunes

Como hay tantos puntos, es más efectivo mostrar distribuciones que puntos individuales. Por lo tanto, mostramos histogramas para cada grupo:

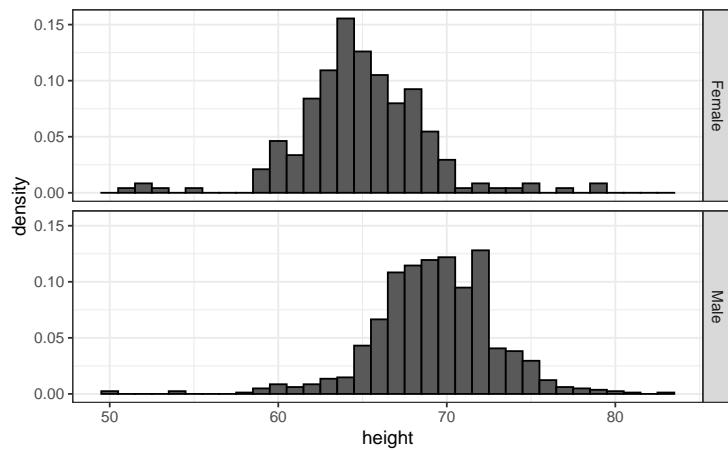


Sin embargo, mirando el gráfico arriba, no es inmediatamente obvio que los varones son, en promedio, más altos que las hembras. Tenemos que mirar cuidadosamente para notar que el eje-x tiene un rango más alto de valores en el histograma masculino. Un principio importante aquí es **mantener los ejes iguales** cuando se comparan datos en dos gráficos. A continuación, vemos cómo la comparación se vuelve más fácil:



### 10.6.2 Alineen gráficos verticalmente para ver cambios horizontales y horizontalmente para ver cambios verticales

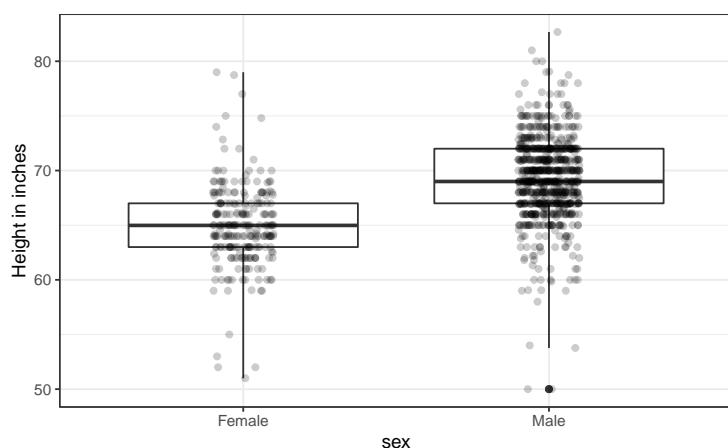
En estos histogramas, la señal visual relacionada con las disminuciones o los aumentos de altura son los cambios hacia la izquierda o hacia la derecha, respectivamente: los cambios horizontales. Alinear los gráficos verticalmente nos ayuda a ver este cambio cuando los ejes son fijos:



```
heights %>%
 ggplot(aes(height, ..density..)) +
 geom_histogram(binwidth = 1, color="black") +
 facet_grid(sex~.)
```

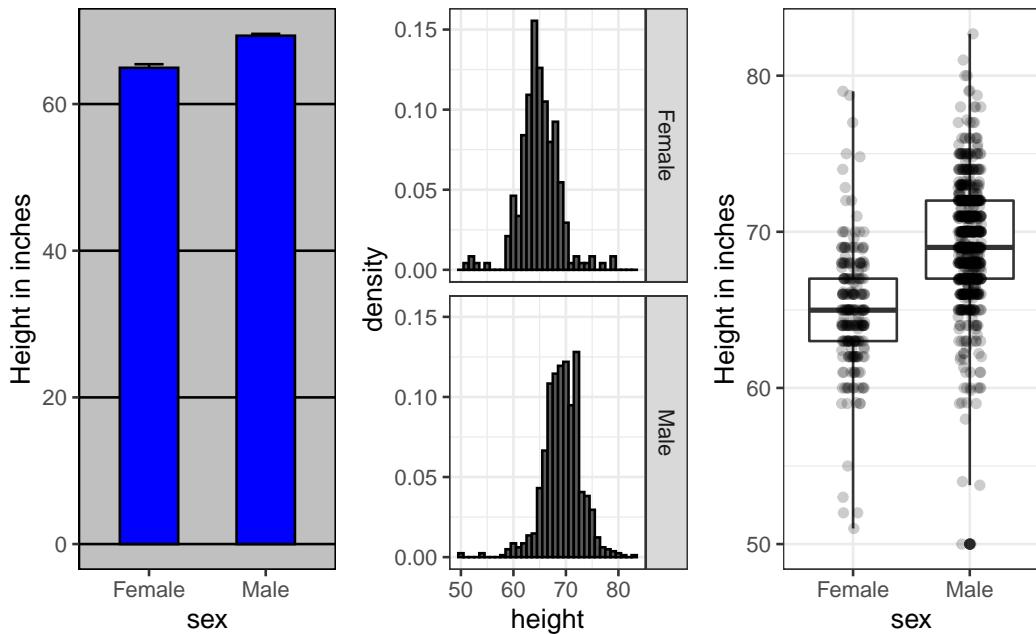
El gráfico anterior hace que sea mucho más fácil notar que los varones son, en promedio, más altos.

Si queremos obtener el resumen compacto que ofrecen los diagramas de caja, tenemos que alinearlos horizontalmente ya que, por defecto, los diagramas de caja se mueven hacia arriba y hacia abajo según los cambios de altura. Siguiendo nuestro principio de “mostrar los datos”, superponemos todos los puntos de datos:



```
heights %>%
 ggplot(aes(sex, height)) +
 geom_boxplot(coef=3) +
 geom_jitter(width = 0.1, alpha = 0.2) +
 ylab("Height in inches")
```

Ahora comparén y contrasten estos tres gráficos, basados en exactamente los mismos datos:

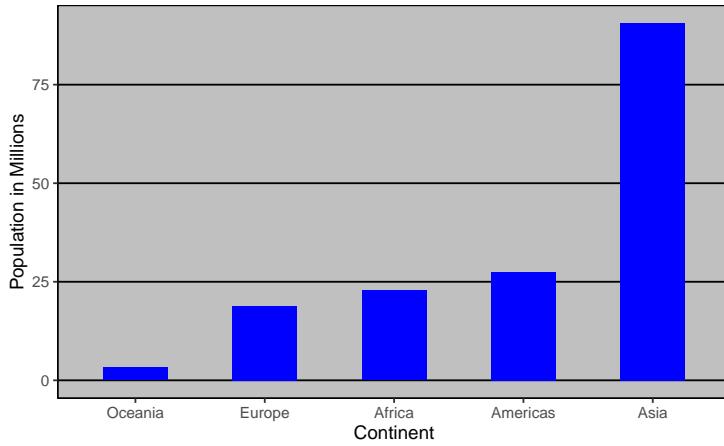


Observen cuánto más aprendemos de los dos gráficos a la derecha. Los diagramas de barras son útiles para mostrar un número, pero no son muy útiles cuando queremos describir distribuciones.

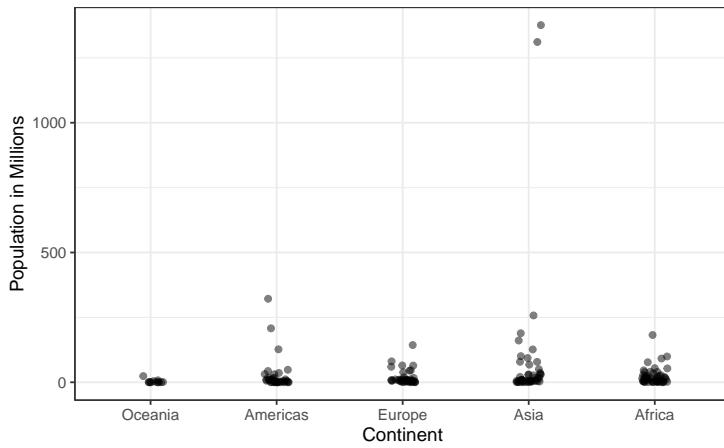
### 10.6.3 Consideren transformaciones

Hemos motivado el uso de la transformación logarítmica en los casos en que los cambios son multiplicativos. El tamaño de la población fue un ejemplo en el que encontramos que una transformación logarítmica produjo una transformación más informativa.

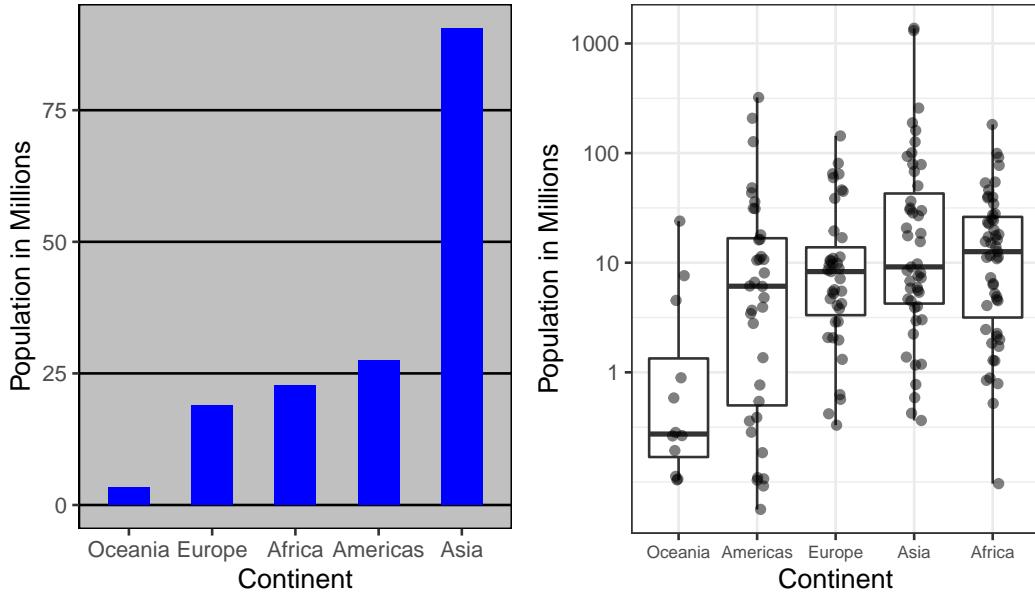
La combinación de un diagrama de barras elegido incorrectamente y no usar una transformación logarítmica cuando sea necesario puede ser particularmente distorsionante. Como ejemplo, consideren este diagrama de barras que muestra los tamaños de población promedio para cada continente en 2015:



Mirando el gráfico anterior, uno concluiría que los países de Asia son mucho más poblados que los de otros continentes. Siguiendo el principio de “mostrar los datos”, notamos rápidamente que esto se debe a dos países muy grandes, que suponemos son India y China:



El uso de una transformación logarítmica aquí produce un gráfico mucho más informativo. Comparamos el diagrama de barras original con un diagrama de caja usando la transformación de escala logarítmica para el eje-y:

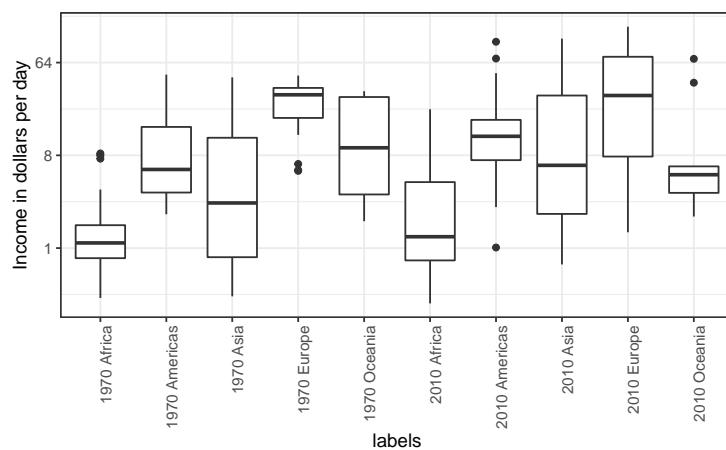


Con el nuevo gráfico, nos damos cuenta de que los países de África tienen una población mediana mayor que los de Asia.

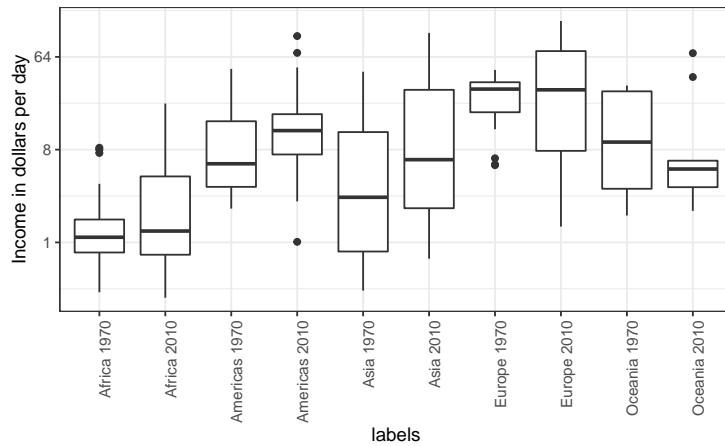
Otras transformaciones que deben considerar son la transformación logística (`logit`), que es útil para ver mejor los cambios en las probabilidades, y la transformación de la raíz cuadrada (`sqrt`), que es útil para conteos.

#### 10.6.4 Señales visuales comparadas deben estar adyacentes

Para cada continente, comparemos los ingresos en 1970 versus 2010. Al comparar los datos de ingresos entre regiones entre 1970 y 2010, hicimos un gráfico similar al siguiente, pero esta vez investigamos continentes en lugar de regiones.

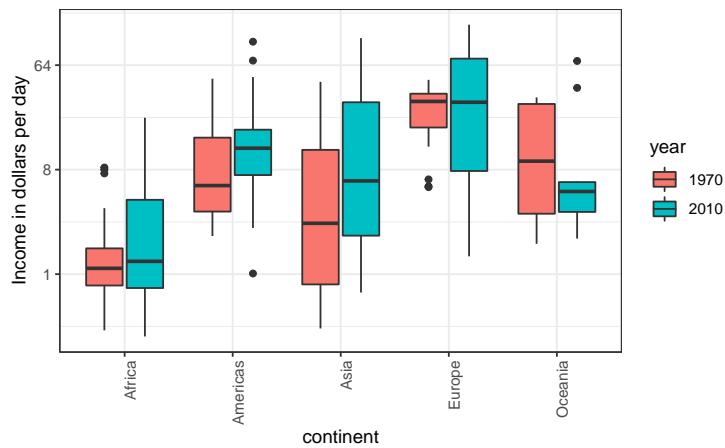


El comportamiento por defecto de **ggplot2** es ordenar las etiquetas alfabéticamente para que las etiquetas con 1970 aparezcan antes que las etiquetas con 2010. Esto dificulta las comparaciones porque la distribución de un continente en 1970 está visualmente lejos de su distribución en 2010. Es mucho más fácil hacer la comparación entre 1970 y 2010 para cada continente cuando los diagramas de caja para ese continente están uno al lado del otro:



### 10.6.5 Usen color

La comparación se hace aún más fácil si usamos color para denotar las dos cosas que queremos comparar:



## 10.7 Consideren los daltónicos

Alrededor de 10% de la población es daltónica. Desafortunadamente, los colores por defecto utilizados en **ggplot2** no son óptimos para este grupo. Sin embargo, **ggplot2** hace

fácil cambiar la paleta de colores utilizada en los gráficos. Existen ejemplos de paletas que consideran los daltónicos<sup>10</sup>.

```
color_blind_friendly_cols <-
 c("#999999", "#E69F00", "#56B4E9", "#009E73",
 "#F0E442", "#0072B2", "#D55E00", "#CC79A7")
```

Aquí están los colores:



Además, hay varios recursos en linea que pueden ayudarlos a seleccionar colores<sup>11</sup>.

## 10.8 Gráficos para dos variables

En general, deben usar diagramas de dispersión para visualizar la relación entre dos variables. Para cada caso en el que hemos examinado la relación entre dos variables, incluyendo asesinatos totales versus tamaño de población, esperanza de vida versus tasas de fertilidad y mortalidad infantil versus ingresos, hemos utilizado diagramas de dispersión y ese es el gráfico que generalmente recomendamos. Sin embargo, hay algunas excepciones y aquí describimos dos gráficos alternativos: el *slope chart* y el gráfico *Bland-Altman*.

### 10.8.1 Slope charts

Una excepción en la que otro tipo de gráfico puede ser más informativo es cuando se comparan variables del mismo tipo, pero en diferentes momentos y para un número relativamente pequeño de comparaciones. Por ejemplo, si estamos comparando la esperanza de vida entre 2010 y 2015. En ese caso, recomendaríamos un *slope chart*.

No hay geometría para los *slope charts* en **ggplot2**, pero podemos construir una usando **geom\_line**. Necesitamos hacer algunos ajustes para añadir etiquetas. A continuación, mostramos un ejemplo que compara 2010 a 2015 para los grandes países occidentales:

```
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),
```

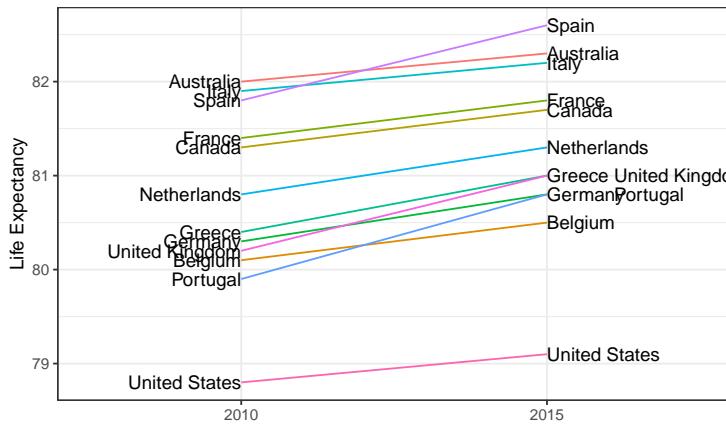
<sup>10</sup>[http://www.cookbook-r.com/Graphs/Colors\\_\(ggplot2\)/#a-colorblind-friendly-palettefont](http://www.cookbook-r.com/Graphs/Colors_(ggplot2)/#a-colorblind-friendly-palettefont)

<sup>11</sup><http://bconnelly.net/2013/10/creating-colorblind-friendly-figures/>

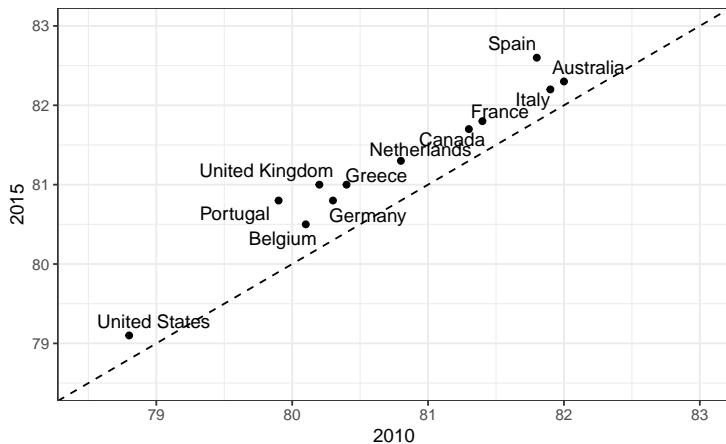
```

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")

```



Una ventaja del *slope chart* es que rápidamente nos permite tener una idea de los cambios basados en la pendiente de las líneas. Aunque estamos usando el ángulo como señal visual, también utilizamos la posición para determinar los valores exactos. Comparar las mejoras es un poco más difícil con un diagrama de dispersión:

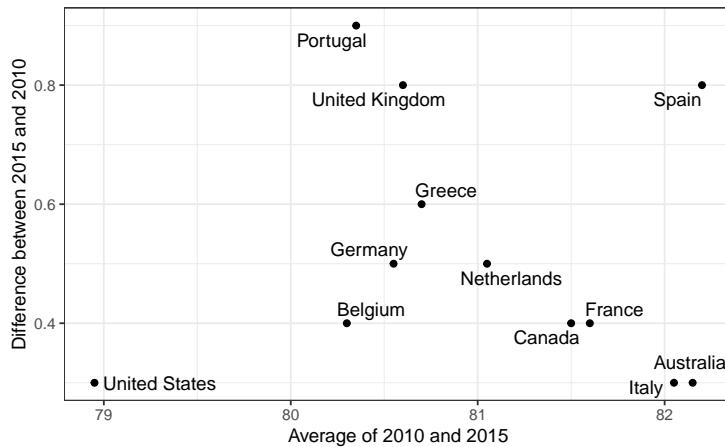


En el diagrama de dispersión, hemos seguido el principio de *usar ejes comunes* porque estamos comparando estos antes y después. Sin embargo, si tenemos muchos puntos, los *slope charts* dejan de ser útiles ya que se hace difícil ver todas las líneas.

### 10.8.2 Gráfico Bland-Altman

Como estamos interesados principalmente en la diferencia, tiene sentido dedicarle uno de nuestros ejes. El gráfico Bland-Altman, también conocido como el gráfico de diferencia de medias de Tukey y como el *MA plot*, muestra la diferencia versus el promedio:

```
library(ggplot2)
dat %>%
 mutate(year = paste0("life_expectancy_", year)) %>%
 select(country, year, life_expectancy) %>%
 pivot_wider(names_from = "year", values_from="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")
```

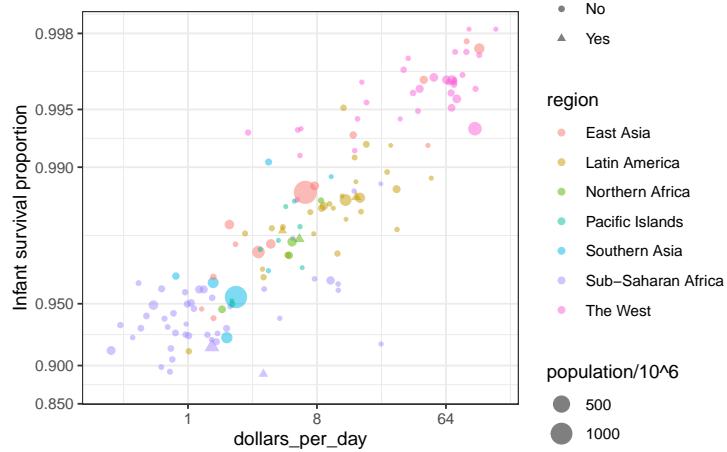


Aquí, al simplemente mirar el eje-y, rápidamente vemos qué países han mostrado la mayor mejora. También, obtenemos una idea del valor general del eje-x.

---

### 10.9 Cómo codificar una tercera variable

Un diagrama de dispersión anterior mostró la relación entre la supervivencia infantil y el ingreso promedio. A continuación se muestra una versión de este gráfico que codifica tres variables: pertenencia a OPEC, región y población.



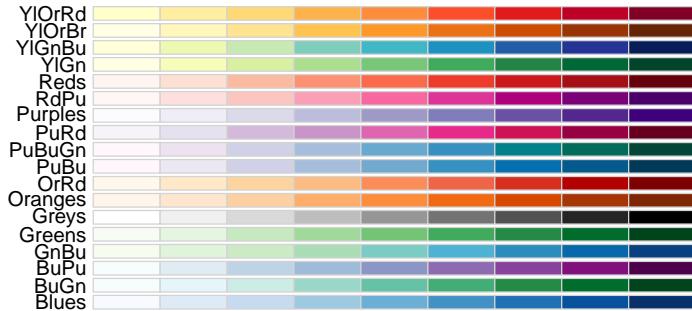
Codificamos variables categóricas con color y forma. Estas formas se pueden controlar con el argumento `shape`. Abajo mostramos las formas disponibles para su uso en R. Para los últimos cinco, el color rellena la forma.

|                 |           |             |                |                |                 |                  |                      |                       |
|-----------------|-----------|-------------|----------------|----------------|-----------------|------------------|----------------------|-----------------------|
| $\square$       | $\circ$   | $\triangle$ | $+$            | $\times$       | $\diamond$      | $\nabla$         | $\boxtimes$          | $*$                   |
| 0               | 1         | 2           | 3              | 4              | 5               | 6                | 7                    | 8                     |
| $\diamondsuit$  | $\oplus$  | $\boxtimes$ | $\blacksquare$ | $\boxotimes$   | $\boxsquare$    | $\blacksquare$   | $\bullet$            | $\blacktriangle$      |
| 9               | 10        | 11          | 12             | 13             | 14              | 15               | 16                   | 17                    |
| $\blacklozenge$ | $\bullet$ | $\bullet$   | $\bullet$      | $\blacksquare$ | $\blacklozenge$ | $\blacktriangle$ | $\blacktriangledown$ | $\blacktriangleright$ |
| 18              | 19        | 20          | 21             | 22             | 23              | 24               | 25                   |                       |

Para variables continuas, podemos usar color, intensidad o tamaño. A continuación ofrecemos un ejemplo de cómo hacer esto con un estudio de caso.

Al seleccionar colores para cuantificar una variable numérica, elegimos entre dos opciones: secuenciales y divergentes. Los colores secuenciales son adecuados para datos que van de mayor a menor. Los valores altos se distinguen claramente de los valores bajos. Aquí tenemos algunos ejemplos ofrecidos por el paquete `RColorBrewer`:

```
library(RColorBrewer)
display.brewer.all(type="seq")
```



Los colores divergentes se utilizan para representar valores que divergen de un centro. Ponemos igual énfasis en ambos extremos del rango de datos: más altos que el centro y más bajos que el centro. Un ejemplo de cuándo usaríamos un patrón divergente sería cuando mostramos la altura en cuántas desviaciones estándar están del promedio. Aquí hay algunos ejemplos de patrones divergentes:

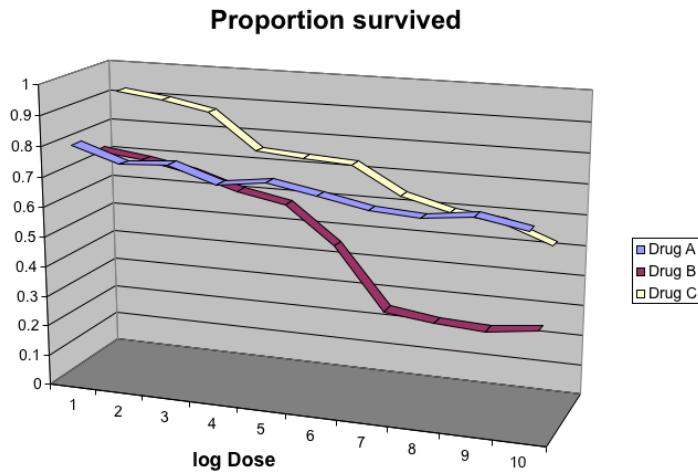
```
library(RColorBrewer)
display.brewer.all(type="div")
```



## 10.10 Eviten los gráficos pseudo-tridimensionales

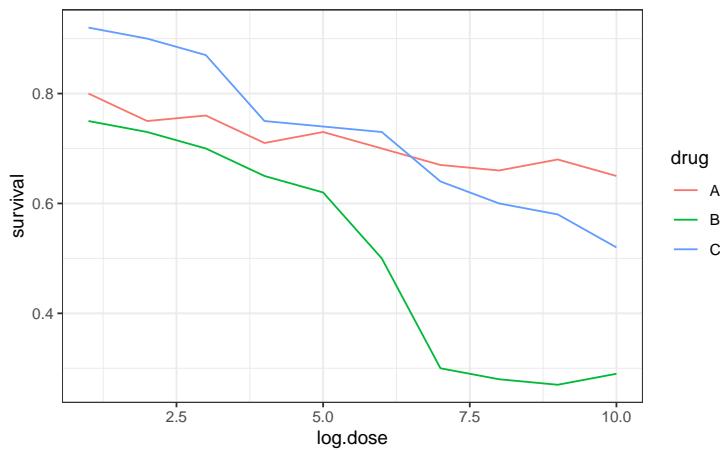
La siguiente figura, tomada de la literatura científica<sup>12</sup>, muestra tres variables: dosis, tipo de fármaco y supervivencia. Aunque sus pantallas o páginas de libros son planas y bidimensionales, el gráfico intenta imitar tres dimensiones y asigna una dimensión a cada variable.

<sup>12</sup>[https://projecteuclid.org/download/pdf\\_1/euclid.ss/1177010488](https://projecteuclid.org/download/pdf_1/euclid.ss/1177010488)



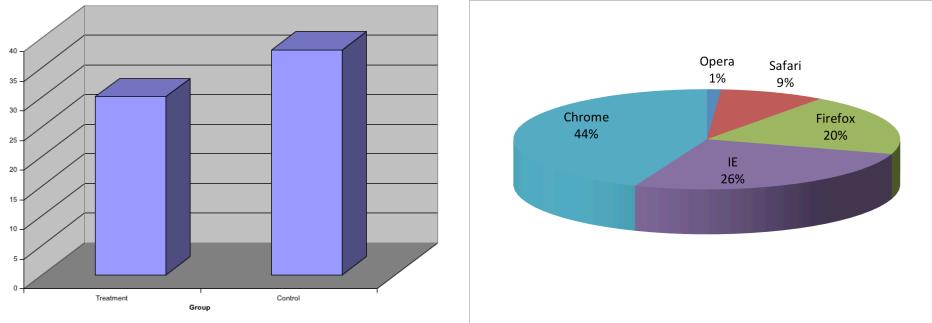
(Imagen cortesía de Karl Broman.)

Los humanos no son buenos para ver en tres dimensiones (que explica por qué es difícil estacionar en paralelo) y nuestra limitación es aún peor con respecto a las pseudo-tres dimensiones. Para ver esto, intenten determinar los valores de la variable de supervivencia en el gráfico anterior. ¿Pueden identificar cuándo la cinta púrpura se cruza con la roja? Este es un ejemplo en el que podemos fácilmente usar color para representar la variable categórica en lugar de un pseudo-3D:



Observen cuánto más fácil es determinar los valores de supervivencia.

Pseudo-3D a veces se usa de forma totalmente gratuita: los gráficos se hacen para que se vean en 3D incluso cuando la tercera dimensión no representa una cantidad. Esto solo añade confusión y hace que sea más difícil transmitir su mensaje. Aquí hay dos ejemplos:



(Imágenes cortesía de Karl Broman.)

### 10.11 Eviten demasiados dígitos significativos

Por defecto, el software estadístico como R devuelve muchos dígitos significativos. El comportamiento por defecto en R es mostrar 7 dígitos significativos. Esa cantidad de dígitos a menudo no añade información y el desorden visual agregado puede dificultar que se entienda el mensaje. Como ejemplo, aquí están las tasas de enfermedades por cada 10,000 para California en cinco décadas, calculadas de los totales y la población con R:

| state      | year | Measles    | Pertussis  | Polio     |
|------------|------|------------|------------|-----------|
| California | 1940 | 37.8826320 | 18.3397861 | 0.8266512 |
| California | 1950 | 13.9124205 | 4.7467350  | 1.9742639 |
| California | 1960 | 14.1386471 | NA         | 0.2640419 |
| California | 1970 | 0.9767889  | NA         | NA        |
| California | 1980 | 0.3743467  | 0.0515466  | NA        |

Estamos reportando precisión de hasta 0.00001 casos por 10,000, un valor muy pequeño en el contexto de los cambios que ocurren a través del tiempo. En este caso, dos cifras significativas son más que suficientes y claramente indican que las tasas están disminuyendo:

| state      | year | Measles | Pertussis | Polio |
|------------|------|---------|-----------|-------|
| California | 1940 | 37.9    | 18.3      | 0.8   |
| California | 1950 | 13.9    | 4.7       | 2.0   |
| California | 1960 | 14.1    | NA        | 0.3   |
| California | 1970 | 1.0     | NA        | NA    |
| California | 1980 | 0.4     | 0.1       | NA    |

Para cambiar el número de dígitos significativos o redondear números usamos `signif` y `round`. Pueden definir el número de dígitos significativos a nivel mundial configurando opciones como esta: `options(digits = 3)`.

Otro principio relacionado con la visualización de tablas es colocar los valores que se comparan en columnas en lugar de filas. Observen que nuestra tabla anterior es más fácil de leer que esta:

| state      | disease   | 1940 | 1950 | 1960 | 1970 | 1980 |
|------------|-----------|------|------|------|------|------|
| California | Measles   | 37.9 | 13.9 | 14.1 | 1    | 0.4  |
| California | Pertussis | 18.3 | 4.7  | NA   | NA   | 0.1  |
| California | Polio     | 0.8  | 2.0  | 0.3  | NA   | NA   |

## 10.12 Consideren a su audiencia

Los gráficos se pueden usar para 1) nuestros propios análisis exploratorios de datos, 2) para transmitir un mensaje a los expertos o 3) para ayudar a contar una historia a una audiencia general. Asegúrense de que el público destinario comprenda cada elemento del gráfico.

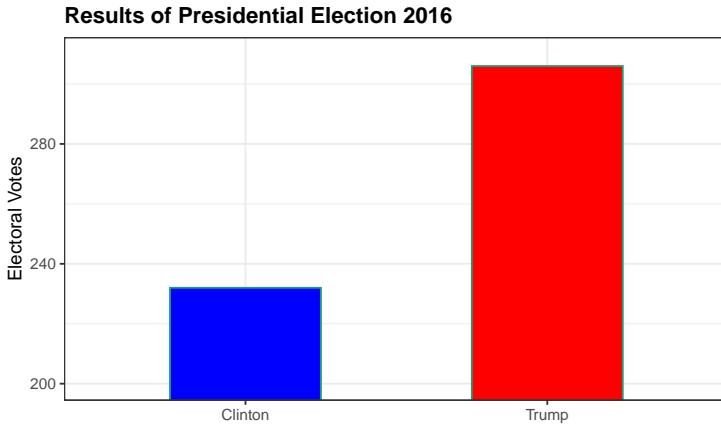
Como un ejemplo sencillo, consideren que para su propia exploración puede ser más útil transformar logarítmicamente los datos y luego graficarlos. Sin embargo, para una audiencia general que no está familiarizada con la conversión de valores logarítmicos a las medidas originales, será mucho más fácil entender el uso de una escala logarítmica para el eje en lugar de los valores transformados logarítmicamente.

## 10.13 Ejercicios

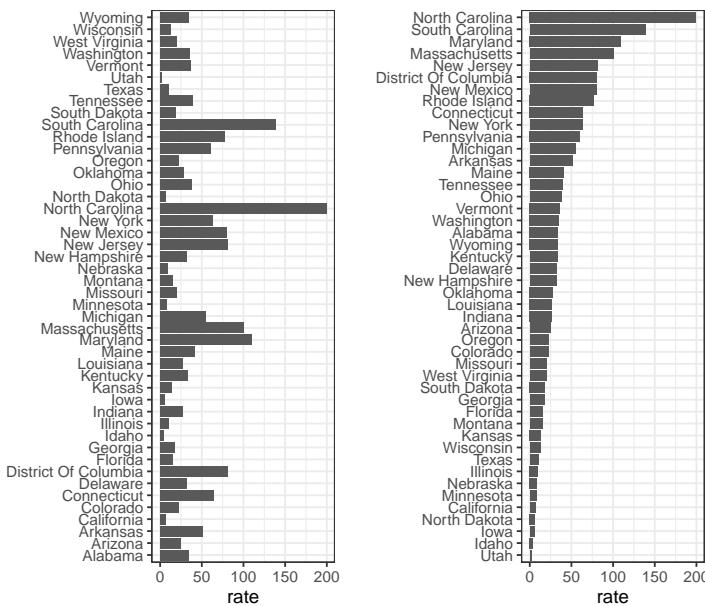
Para estos ejercicios, utilizaremos los datos de vacunas en el paquete **dslabs**:

```
library(dslabs)
data(us_contagious_diseases)
```

1. Los gráficos circulares son apropiados:
  - a. Cuando queremos mostrar porcentajes.
  - b. Cuando **ggplot2** no está disponible.
  - c. Cuando estoy jugando frisbee.
  - d. Nunca. Los diagramas de barras y las tablas siempre son mejores.
2. ¿Cuál es el problema con el siguiente gráfico?



- a. Los valores están mal. La votación final fue de 306 a 232.
  - b. El eje no comienza en 0. Juzgando por la longitud, parece que Trump recibió 3 veces más votos cuando, de hecho, fue aproximadamente 30% más.
  - c. Los colores deben ser iguales.
  - d. Los porcentajes deben mostrarse como un gráfico circular.
3. Mire a los siguientes dos gráficos. Muestran la misma información: tasas de sarampión de 1928 en los 50 estados.



¿Qué gráfico es más fácil de leer si quiere determinar cuáles son los mejores y peores estados en términos de tasas, y por qué?

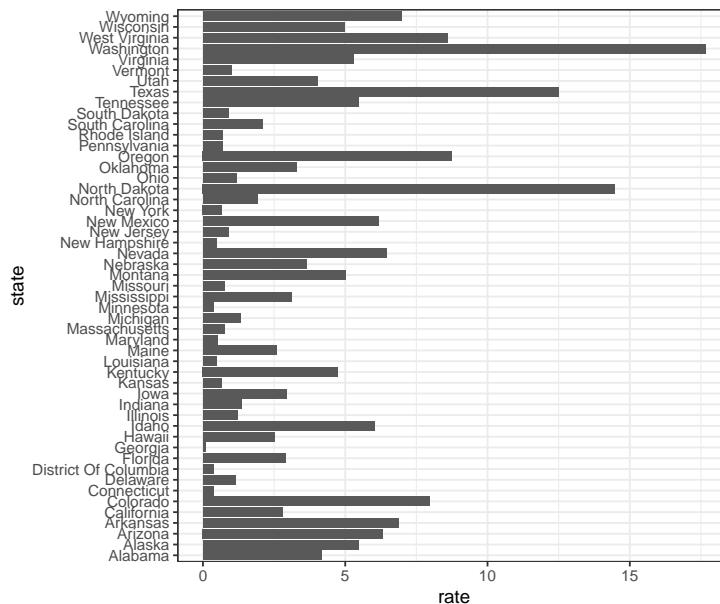
- a. Dan la misma información, por lo que ambos son igual de buenos.
- b. El gráfico de la izquierda es mejor porque ordena los estados alfabéticamente.

- c. El gráfico de la derecha es mejor porque el orden alfabético no tiene nada que ver con la enfermedad y al ordenar según la tasa real, rápidamente vemos los estados con las tasas más altas y más bajas.
- d. Ambos gráficos deben ser un gráfico circular.
4. Para hacer el gráfico de la izquierda, tenemos que reordenar los niveles de las variables de los estados.

```
dat <- us_contagious_diseases %>%
 filter(year == 1967 & disease=="Measles" & !is.na(population)) %>%
 mutate(rate = count/ population * 10000 * 52/ weeks_reporting)
```

Recuerde lo que sucede cuando hacemos un diagrama de barras:

```
dat %>% ggplot(aes(state, rate)) +
 geom_bar(stat="identity") +
 coord_flip()
```



Defina estos objetos:

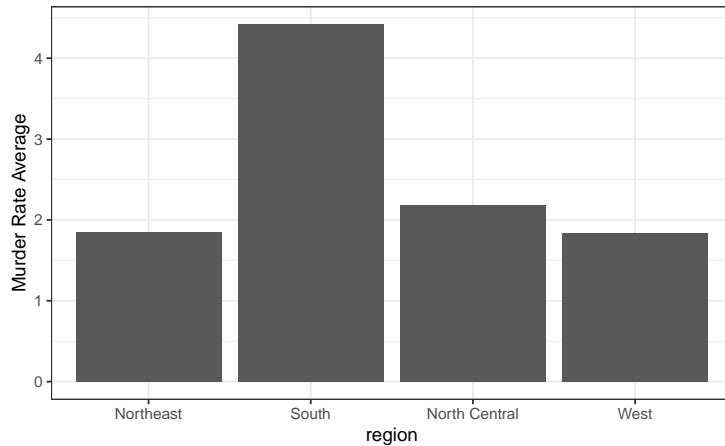
```
state <- dat$state
rate <- dat$count/dat$population*10000*52/dat$weeks_reporting
```

Redefina el objeto **state** para que los niveles se reordenen. Imprima el nuevo objeto **state** y sus niveles para que pueda ver que los niveles no reordenan el vector.

5. Ahora edite el código de arriba para redefinir **dat** para que los niveles de la variable **state** se reordenen por la variable **rate**. Entonces haga un diagrama de barras usando el código anterior, pero para este nuevo **dat**.

6. Digamos que está interesado en comparar las tasas de homicidios con armas de fuego en todas las regiones de EE. UU. Ve este gráfico:

```
library(dslabs)
data("murders")
murders %>% mutate(rate = total/population*100000) %>%
 group_by(region) %>%
 summarize(avg = mean(rate)) %>%
 mutate(region = factor(region)) %>%
 ggplot(aes(region, avg)) +
 geom_bar(stat="identity") +
 ylab("Murder Rate Average")
```



y decide mudarse a un estado en la región occidental. ¿Cuál es el problema principal con esta interpretación?

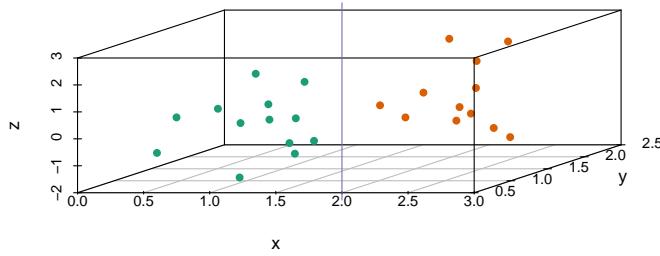
- a. Las categorías están ordenadas alfabéticamente.
- b. El gráfico no muestra errores estándar.
- c. El gráfico no muestra todos los datos. No vemos la variabilidad dentro de una región y es posible que los estados más seguros no estén en el occidente.
- d. El noreste tiene el promedio más bajo.

7. Haga un diagrama de caja de las tasas de asesinatos que se definen como:

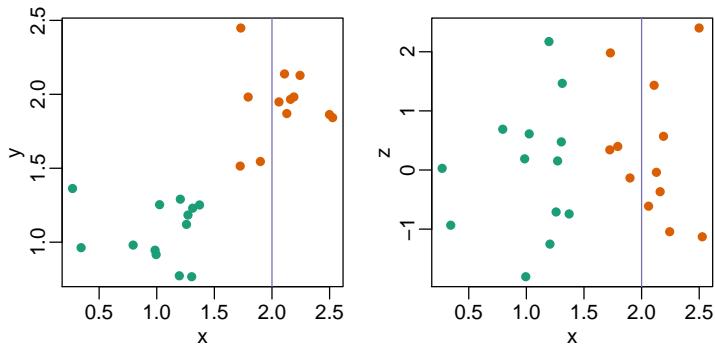
```
data("murders")
murders %>% mutate(rate = total/population*100000)
```

por región, mostrando todos los puntos y ordenando las regiones por su tasa mediana.

8. Los gráficos a continuación muestran tres variables continuas.



La línea  $x = 2$  parece separar los puntos. Pero realmente no es el caso, como vemos cuando graficamos los datos en un par de puntos bidimensionales.



¿Por qué pasa esto?

- a. Los humanos no son buenos leyendo gráficos pseudo-3D.
  - b. Debe haber un error en el código.
  - c. Los colores nos confunden.
  - d. Los diagramas de dispersión no se deben usar para comparar dos variables cuando tenemos acceso a tres variables.
9. Reproduzca el gráfico de imagen que hicimos anteriormente pero para la viruela. Para este gráfico, no incluya años en los que no se indicaron casos en 10 o más semanas.
10. Ahora reproduzca el gráfico de series de tiempo que hicimos anteriormente, pero esta vez siguiendo las instrucciones de la pregunta anterior.
11. Para el estado de California, haga gráficos de series de tiempo que muestren las tasas de todas las enfermedades. Incluya solo años en los cuales se proveen datos en 10 o más semanas. Use un color diferente para cada enfermedad.
12. Ahora haga lo mismo con las tasas para EE. UU. Sugerencia: calcule la tasa de EE. UU. usando `summarize`, el total de casos dividido por la población total.

### 10.14 Estudio de caso: las vacunas y las enfermedades infecciosas

Las vacunas han ayudado a salvar millones de vidas. En el siglo XIX, antes de que se lograra la inmunización de grupo a través de programas de vacunación, las muertes por enfermedades infecciosas, como la viruela y la poliomielitis, eran comunes. Sin embargo, hoy los programas de vacunación se han vuelto algo controversiales a pesar de toda la evidencia científica de su importancia.

La controversia comenzó con un artículo<sup>13</sup> publicado en 1988 y liderado por Andrew Wakefield, que afirmó la existencia de un vínculo entre la administración de la vacuna contra el sarampión, las paperas y la rubéola, y el autismo y las enfermedades intestinales. A pesar de la gran cantidad de evidencia científica que contradice este hallazgo, los informes sensacionalistas de los medios de comunicación y el alarmismo de los que creen en teorías de conspiración llevaron a partes del público a creer que las vacunas eran perjudiciales. Como resultado, muchos padres dejaron de vacunar a sus hijos. Esta práctica peligrosa puede ser potencialmente desastrosa dado que los Centros para el Control de Enfermedades de EE.UU. (CDC por sus siglas en inglés) estiman que las vacunas evitarán más de 21 millones de hospitalizaciones y 732,000 muertes entre los niños nacidos en los últimos 20 años (ver *Benefits from Immunization during the Vaccines for Children Program Era — United States, 1994–2013, MMWR*<sup>14</sup>). Desde entonces, “The Lancet” ha retractado el artículo y Andrew Wakefield finalmente fue “excluido del registro médico en mayo de 2010 con una observación que indica la falsificación fraudulenta en que incurrió y se le revocó la licencia para ejercer la medicina en el Reino Unido” (Fuente: Wikipedia<sup>15</sup>). Sin embargo, los conceptos erróneos perduran, en parte debido a activistas autoproclamados que continúan diseminando información errónea sobre las vacunas.

La comunicación efectiva de datos es un fuerte antídoto contra la información errónea y el miedo. Anteriormente enseñamos un ejemplo de un artículo del Wall Street Journal<sup>16</sup> que muestra datos relacionados con el impacto de las vacunas en la lucha contra las enfermedades infecciosas. A continuación reconstruiremos ese ejemplo.

Los datos utilizados para estos gráficos fueron recopilados, organizados y distribuidos por el Proyecto Tycho<sup>17</sup> e incluyen conteos reportados semanalmente para siete enfermedades desde 1928 hasta 2011 en los cincuenta estados de EE.UU. Incluimos los totales anuales en el paquete **dslabs**:

```
library(tidyverse)
library(RColorBrewer)
library(dslabs)
data(us_contagious_diseases)
names(us_contagious_diseases)
#> [1] "disease" "state" "year"
#> [4] "weeks_reporting" "count" "population"
```

Creamos un objeto temporero **dat** que almacena solo los datos de sarampión, incluye la tasa

<sup>13</sup>[http://www.thelancet.com/journals/lancet/article/PIIS0140-6736\(97\)11096-0/abstract](http://www.thelancet.com/journals/lancet/article/PIIS0140-6736(97)11096-0/abstract)

<sup>14</sup><https://www.cdc.gov/mmwr/preview/mmwrhtml/mm6316a4.htm>

<sup>15</sup>[https://es.wikipedia.org/wiki/Andrew\\_Wakefield](https://es.wikipedia.org/wiki/Andrew_Wakefield)

<sup>16</sup><http://graphics.wsj.com/infectious-diseases-and-vaccines/>

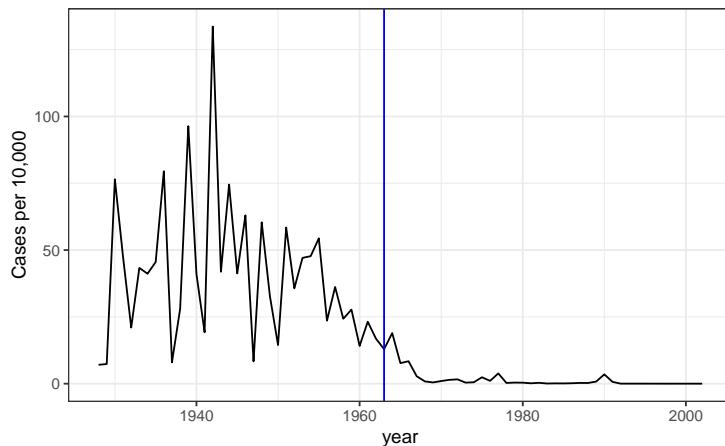
<sup>17</sup><http://www.tycho.pitt.edu/>

por 100,000, ordena a los estados según el valor promedio de enfermedad y elimina Alaska y Hawái ya que estos dos se convirtieron en estados a fines de la década de 1950. Tengan en cuenta que hay una columna `weeks_reporting` que nos dice cuántas semanas del año se reportaron datos. Tenemos que ajustar ese valor al calcular la tasa:

```
the_disease <- "Measles"
dat <- us_contagious_diseases %>%
 filter(!state%in%c("Hawaii", "Alaska") & disease == the_disease) %>%
 mutate(rate = count/ population * 10000 * 52/ weeks_reporting) %>%
 mutate(state = reorder(state, rate))
```

Ahora podemos graficar fácilmente las tasas de enfermedad por año. Aquí están los datos de sarampión de California:

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



Añadimos una línea vertical en 1963, ya que es cuando se introdujo la vacuna<sup>18</sup>.

¿Ahora podemos mostrar datos para todos los estados en un gráfico? Tenemos tres variables para incluir: año, estado y tasa. En la figura del WSJ, usan el eje-x para el año, el eje-y para el estado y el tono de color para representar las tasas. Sin embargo, la escala de colores que utilizan, que va del amarillo al azul al verde al naranja al rojo, se puede mejorar.

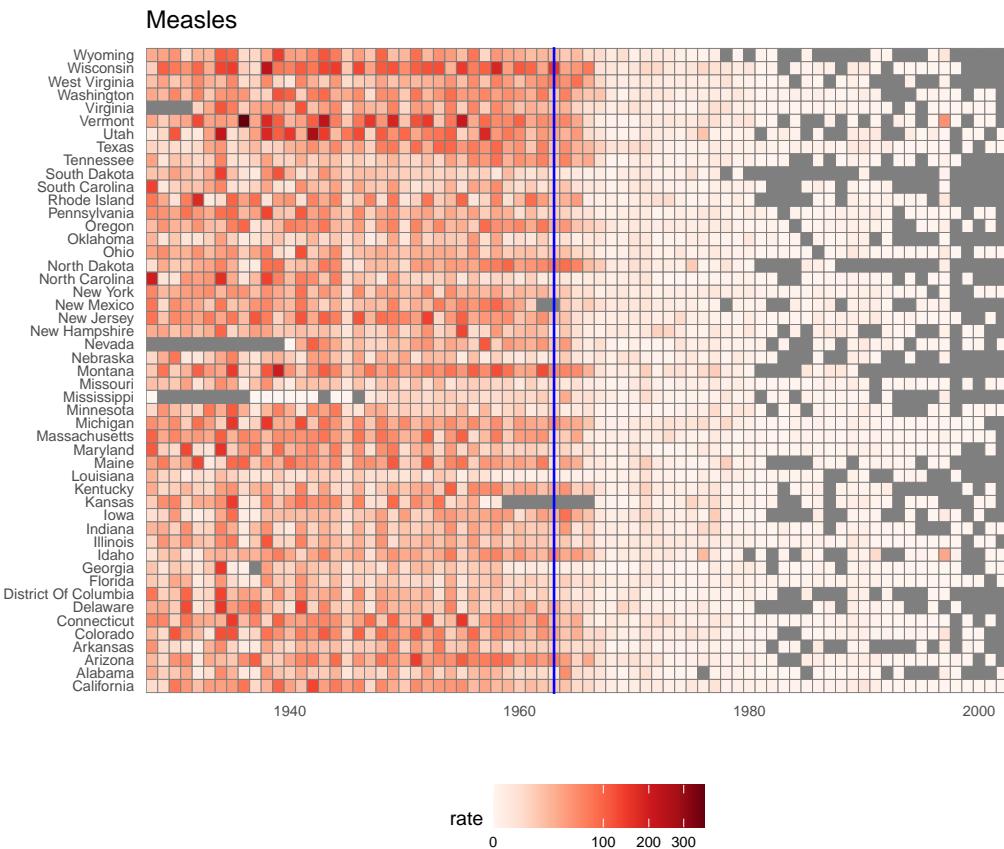
En nuestro ejemplo queremos usar una paleta secuencial ya que no hay un centro significativo, solo tasas bajas y altas.

Usamos la geometría `geom_tile` para tejer la región con colores que representan las tasas de enfermedad. Usamos una transformación de raíz cuadrada para evitar que los conteos particularmente altos dominen el gráfico. Observen que los valores faltantes se muestran en

<sup>18</sup>Control, Centers for Disease; Prevention (2014). CDC health information for international travel 2014 (the yellow book). p. 250. ISBN 9780199948505

gris. Además, noten que tan pronto una enfermedad fue prácticamente erradicada, algunos estados dejaron de informar casos por completo. Es por esa razón que vemos tanto gris después de 1980.

```
dat %>% ggplot(aes(year, state, fill = rate)) +
 geom_tile(color = "grey50") +
 scale_x_continuous(expand=c(0,0)) +
 scale_fill_gradientn(colors = brewer.pal(9, "Reds"), trans = "sqrt") +
 geom_vline(xintercept=1963, col = "blue") +
 theme_minimal() +
 theme(panel.grid = element_blank(),
 legend.position="bottom",
 text = element_text(size = 8)) +
 ggtitle(the_disease) +
 ylab("") + xlab("")
```



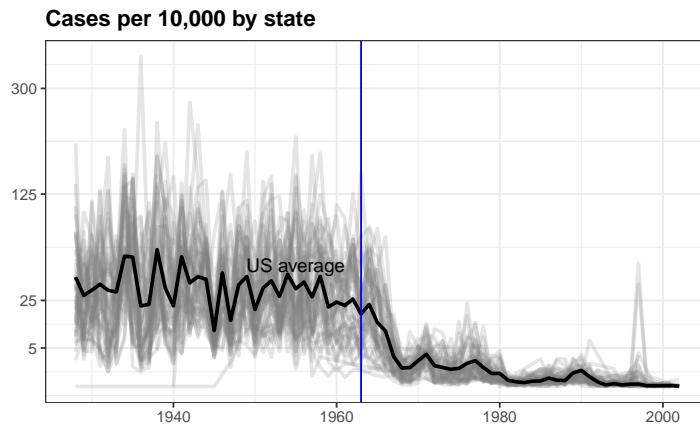
Este gráfico ofrece evidencia preponderante a favor de la contribución de las vacunas. Sin embargo, una limitación es que usa el color para representar la cantidad, que, como explicamos anteriormente, dificulta saber exactamente cuán altos llegan los valores. La posición y la longitud son mejores señales. Si estamos dispuestos a perder información de estado, pode-

mos hacer una versión del gráfico que muestre los valores con posición. También podemos mostrar el promedio de EE. UU., que calculamos así:

```
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)
```

Ahora para hacer el gráfico simplemente usamos la geometría `geom_line`:

```
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) +
 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")
```



En teoría, podríamos usar el color para representar los estados, que son una variable categórica, pero es difícil elegir 50 colores distintos.

## 10.15 Ejercicios

1. Reproduzca el mapa de matriz que hicimos anteriormente pero para la viruela. Para este gráfico, no incluya los años en que no se reportaron casos por 10 o más semanas.

2. Ahora reproduzca el gráfico de series de tiempo que hicimos anteriormente, pero esta vez siguiendo las instrucciones de la pregunta anterior para la viruela.
3. Para el estado de California, haga un gráfico de series de tiempo que muestre las tasas de todas las enfermedades. Incluya solo años con informes de 10 o más semanas. Use un color diferente para cada enfermedad.
4. Ahora haga lo mismo con las tasas para Estados Unidos. Sugerencia: calcule la tasa de EE. UU. usando `summarize`: total dividido por población total.

# 11

---

## Resúmenes robustos

---

### 11.1 Valores atípicos

Anteriormente describimos cómo los diagramas de caja muestran *valores atípicos* (*outliers* en inglés), pero no ofrecemos una definición precisa. Aquí discutimos los valores atípicos, los acercamientos que pueden ayudar para detectarlos y los resúmenes que toman en cuenta su presencia.

Los valores atípicos son muy comunes en la ciencia de datos. La recopilación de datos puede ser compleja y es común observar puntos de datos generados por error. Por ejemplo, un viejo dispositivo de monitoreo puede leer mediciones sin sentido antes de fallar por completo. El error humano también es una fuente de valores atípicos, en particular cuando la entrada de datos se realiza manualmente. Un individuo, por ejemplo, puede ingresar erróneamente su altura en centímetros en lugar de pulgadas o colocar el decimal en el lugar equivocado.

¿Cómo distinguimos un valor atípico de mediciones que son demasiado grandes o pequeñas simplemente debido a la variabilidad esperada? Esta no siempre es una pregunta fácil de contestar, pero intentaremos ofrecer alguna orientación. Comencemos con un caso sencillo.

Supongan que un colega se encarga de recopilar datos demográficos para un grupo de varones. Los datos indican la altura en pies y se almacenan en el objeto:

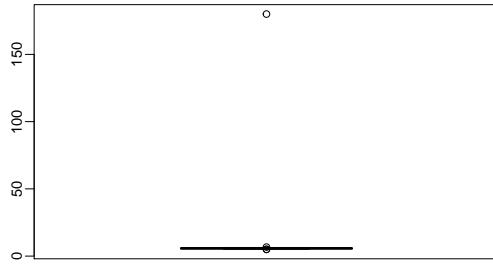
```
library(tidyverse)
library(dslabs)
data(outlier_example)
str(outlier_example)
#> num [1:500] 5.59 5.8 5.54 6.15 5.83 5.54 5.87 5.93 5.89 5.67 ...
```

Nuestro colega utiliza el hecho de que las alturas suelen estar bien aproximadas por una distribución normal y resume los datos con el promedio y la desviación estándar:

```
mean(outlier_example)
#> [1] 6.1
sd(outlier_example)
#> [1] 7.8
```

y escribe un informe sobre el hecho interesante de que este grupo de varones es mucho más alto de lo normal. ¡La altura promedio es más de seis pies! Sin embargo, al usar sus conocimientos de ciencia de datos, notan algo más que es inesperado: la desviación estándar es de más de 7 pies. Al sumar y restar dos desviaciones estándar, observan que el 95% de esta población parece tener alturas entre -9.489, 21.697 pies, que no tiene sentido. Un gráfico rápido muestra el problema:

```
boxplot(outlier_example)
```



Parece que hay al menos un valor que no tiene sentido, ya que sabemos que una altura de 180 pies es imposible. El diagrama de caja detecta este punto como un valor atípico.

## 11.2 Mediana

Cuando tenemos un valor atípico como este, el promedio puede llegar a ser muy grande. Matemáticamente, podemos hacer que el promedio sea tan grande como queramos simplemente cambiando un número: con 500 puntos de datos, podemos aumentar el promedio en cualquier cantidad  $\Delta$  añadiendo  $\Delta \times 500$  a un solo número. La mediana, definida como el valor para el cual la mitad de los valores son más pequeños y la otra mitad son más grandes, es robusta para tales valores atípicos. No importa cuán grande hagamos el punto más grande, la mediana sigue siendo la misma.

Con estos datos, la mediana es:

```
median(outlier_example)
#> [1] 5.74
```

lo cual es aproximadamente 5 pies y 9 pulgadas.

La mediana es lo que los diagramas de caja muestran como una línea horizontal.

## 11.3 El rango intercuartil (IQR)

La caja en un diagrama de caja se define por el primer y tercer cuartil. Estos están destinados a proveer una idea de la variabilidad en los datos: el 50% de los datos están dentro de este rango. La diferencia entre el 3er y 1er cuartil (o los percentiles 75 y 25) se conoce como el rango intercuartil (IQR por sus siglas en inglés). Como sucede con la mediana, esta cantidad será robusta para los valores atípicos ya que los valores grandes no la afectan. Podemos hacer

algunos cálculos y ver que para los datos que siguen la distribución normal, el  $IQR/1.349$  se aproxima a la desviación estándar de los datos si un valor atípico no hubiera estado presente. Podemos ver que esto funciona bien en nuestro ejemplo, ya que obtenemos un estimado de la desviación estándar de:

```
IQR(outlier_example) / 1.349
#> [1] 0.245
```

lo cual es cerca de 3 pulgadas.

## 11.4 La definición de Tukey de un valor atípico

En R, los puntos que caen fuera de los bigotes del diagrama de caja se denominan *valores atípicos*, una definición que Tukey introdujo. El bigote superior termina en el percentil 75 más  $1.5 \times IQR$ , mientras que el bigote inferior termina en el percentil 25 menos  $1.5 \times IQR$ . Si definimos el primer y el tercer cuartil como  $Q_1$  y  $Q_3$ , respectivamente, entonces un valor atípico es cualquier valor fuera del rango:

$$[Q_1 - 1.5 \times (Q_3 - Q_1), Q_3 + 1.5 \times (Q_3 - Q_1)].$$

Cuando los datos se distribuyen normalmente, las unidades estándar de estos valores son:

```
q3 <- qnorm(0.75)
q1 <- qnorm(0.25)
iqr <- q3 - q1
r <- c(q1 - 1.5*iqr, q3 + 1.5*iqr)
r
#> [1] -2.7 2.7
```

Utilizando la función `pnorm`, vemos que 99.3% de los datos caen en este intervalo.

Tengan en cuenta que este no es un evento tan extremo: si tenemos 1000 puntos de datos que se distribuyen normalmente, esperamos ver unos 7 fuera de este rango. Pero estos no serían valores atípicos ya que esperamos verlos bajo la variación típica.

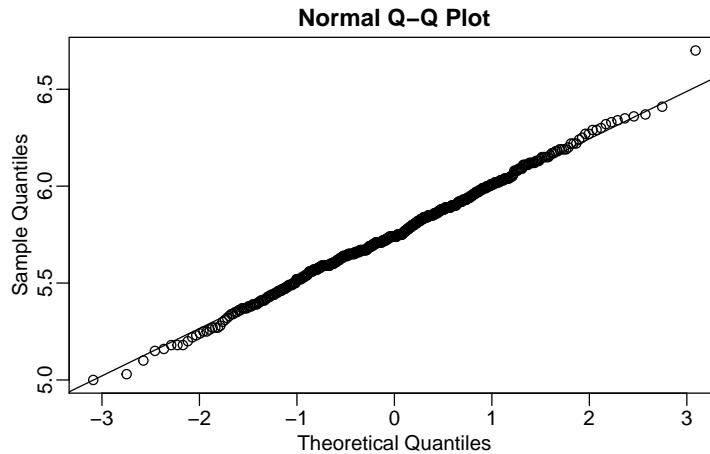
Si queremos que un valor atípico sea más raro, podemos cambiar el 1.5 a un número más grande. Tukey también usó 3 y los denominó *far out outliers* o *valores atípicos extremos*. Con una distribución normal, 100% de los datos caen en este intervalo. Esto se traduce en aproximadamente 2 en un millón de posibilidades de estar fuera del rango. En la función `geom_boxplot`, esto se puede controlar usando el argumento `outlier.size`, que por defecto es 1.5.

La medida de 180 pulgadas está más allá del rango de los datos de altura:

```
max_height <- quantile(outlier_example, 0.75) + 3*IQR(outlier_example)
max_height
#> 75%
#> 6.91
```

Si sacamos este valor, podemos ver que los datos se distribuyen normalmente como se espera:

```
x <- outlier_example[outlier_example < max_height]
qqnorm(x)
qqline(x)
```



## 11.5 Desviación absoluta mediana

Otra opción para estimar la desviación estándar de manera robusta en presencia de valores atípicos es usar la desviación absoluta mediana (*median absolute deviation* o MAD por sus siglas en inglés). Para calcular el MAD, primero calculamos la mediana y luego, para cada valor, calculamos la distancia entre ese valor y la mediana. El MAD se define como la mediana de estas distancias. Por razones técnicas que no discutimos aquí, esta cantidad debe multiplicarse por 1.4826 para asegurar que se aproxime a la desviación estándar real. La función `mad` ya incorpora esta corrección. Para los datos de altura, obtenemos una MAD de:

```
mad(outlier_example)
#> [1] 0.237
```

lo cual es cerca de 3 pulgadas.

## 11.6 Ejercicios

Vamos a usar el paquete **HistData**. Si no lo ha instalando, puede hacerlo así:

```
install.packages("HistData")
```

Cargue el set de datos de altura y cree un vector `x` que contiene solo las alturas masculinas de los datos de Galton de los padres y sus hijos de su investigación histórica sobre la herencia.

```
library(HistData)
data(Galton)
x <- Galton$child
```

1. Calcule el promedio y la mediana de estos datos.
2. Calcule la mediana y el MAD de estos datos.
3. Ahora supongan que Galton cometió un error al ingresar el primer valor y olvidó usar el punto decimal. Puede imitar este error escribiendo:

```
x_with_error <- x
x_with_error[1] <- x_with_error[1]*10
```

¿Cuántas pulgadas crece el promedio como resultado de este error?

4. ¿Cuántas pulgadas crece la SD como resultado de este error?
5. ¿Cuántas pulgadas crece la mediana como resultado de este error?
6. ¿Cuántas pulgadas crece el MAD como resultado de este error?
7. ¿Cómo podríamos utilizar el análisis exploratorio de datos para detectar que se cometió un error?
  - a. Dado que es solo un valor entre muchos, no se puede detectar esto.
  - b. Veríamos un cambio obvio en la distribución.
  - c. Un diagrama de caja, histograma o gráfico Q-Q revelarían un valor atípico obvio.
  - d. Un diagrama de dispersión mostraría altos niveles de error de medición.
8. ¿Cuánto puede crecer accidentalmente el promedio con errores como este? Escribe una función llamada `error_avg` que toma un valor `k` y devuelve el promedio del vector `x` después de que la primera entrada cambie a `k`. Muestre los resultados para `k=10000` y `k=-10000`.

---

## 11.7 Estudio de caso: alturas autoreportadas de estudiantes

Las alturas que hemos estado estudiando no son las alturas originales reportadas por los estudiantes. Las alturas originales también se incluyen en el paquete `dslabs` y se pueden cargar así:

```
library(dslabs)
data("reported_heights")
```

La altura es un vector de caracteres, por lo que creamos una nueva columna con la versión numérica:

```
reported_heights <- reported_heights %>%
 mutate(original_heights = height, height = as.numeric(height))
#> Warning in mask$eval_all_mutate(quo): NAs introducidos por coerción
```

Tengan en cuenta que recibimos una advertencia sobre los NAs. Esto se debe a que algunas de las alturas autoreportadas no eran números. Podemos ver por qué obtenemos estos NAs:

```
reported_heights %>% filter(is.na(height)) %>% head()
#> time_stamp sex height original_heights
#> 1 2014-09-02 15:16:28 Male NA 5' 4"
#> 2 2014-09-02 15:16:37 Female NA 165cm
#> 3 2014-09-02 15:16:52 Male NA 5'7
#> 4 2014-09-02 15:16:56 Male NA >9000
#> 5 2014-09-02 15:16:56 Male NA 5'7"
#> 6 2014-09-02 15:17:09 Female NA 5'3"
```

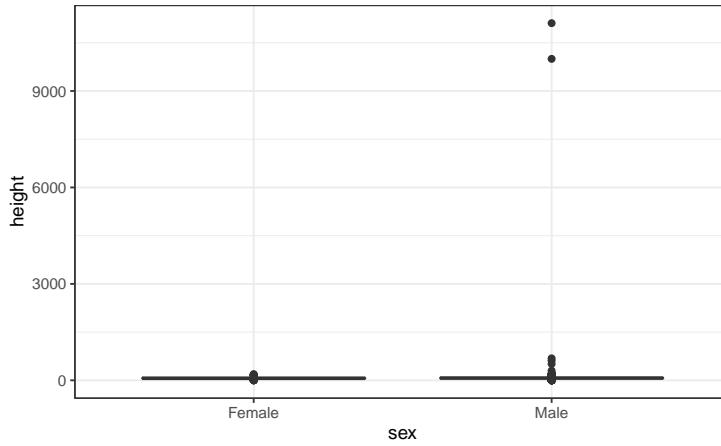
Algunos estudiantes indicaron sus alturas usando pies y pulgadas en lugar de solo pulgadas. Otros usaron centímetros y otros solo estaban “trolleando”. Por ahora eliminaremos estas entradas:

```
reported_heights <- filter(reported_heights, !is.na(height))
```

Si calculamos el promedio y la desviación estándar, observamos que obtenemos resultados extraños. El promedio y la desviación estándar son diferentes de la mediana y del MAD:

```
reported_heights %>%
 group_by(sex) %>%
 summarize(average = mean(height), sd = sd(height),
 median = median(height), MAD = mad(height))
#> # A tibble: 2 x 5
#> sex average sd median MAD
#> <chr> <dbl> <dbl> <dbl> <dbl>
#> 1 Female 63.4 27.9 64.2 4.05
#> 2 Male 103. 530. 70 4.45
```

Esto sugiere que tenemos valores atípicos, lo que se confirma creando un diagrama de caja:



Vemos algunos valores bastante extremos. Para ver cuáles son estos valores, podemos rápidamente mirar los valores más grandes utilizando la función `arrange`:

```
reported_heights %>% arrange(desc(height)) %>% top_n(10, height)
#> #> time_stamp sex height original_heights
#> 1 2014-09-03 23:55:37 Male 11111 11111
#> 2 2016-04-10 22:45:49 Male 10000 10000
#> 3 2015-08-10 03:10:01 Male 684 684
#> 4 2015-02-27 18:05:06 Male 612 612
#> 5 2014-09-02 15:16:41 Male 511 511
#> 6 2014-09-07 20:53:43 Male 300 300
#> 7 2014-11-28 12:18:40 Male 214 214
#> 8 2017-04-03 16:16:57 Male 210 210
#> 9 2015-11-24 10:39:45 Male 192 192
#> 10 2014-12-26 10:00:12 Male 190 190
#> 11 2016-11-06 10:21:02 Female 190 190
```

Las primeras siete entradas parecen errores extraños. Sin embargo, las siguientes entradas parecen haber sido ingresadas en centímetros en lugar de pulgadas. Dado que 184cm es equivalente a seis pies de altura, sospechamos que 184 realmente significa 72 pulgadas.

Podemos revisar todas las respuestas sin sentido examinando los datos que Tukey considera *far out* o *extremos*:

```
whisker <- 3*IQR(reported_heights$height)
max_height <- quantile(reported_heights$height, .75) + whisker
min_height <- quantile(reported_heights$height, .25) - whisker
reported_heights %>%
 filter(!between(height, min_height, max_height)) %>%
 select(original_heights) %>%
 head(n=10) %>% pull(original_heights)
#> [1] "6" "5.3" "511" "6" "2" "5.25" "5.5" "11111"
#> [9] "6" "6.5"
```

Al revisar estas alturas cuidadosamente, vemos dos errores comunes: entradas en centímetros, que resultan ser demasiado grandes, y entradas del tipo `x.y` con `x` e `y` representando pies

y pulgadas respectivamente, que resultan ser demasiado pequeñas. Algunos de los valores aún más pequeños, como 1.6, podrían ser entradas en metros.

En la parte de *wrangling* de datos de este libro, aprenderemos técnicas para corregir estos valores y convertirlos en pulgadas. Aquí pudimos detectar este problema mediante una cuidadosa exploración de los datos para descubrir problemas con ellos: el primer paso en la gran mayoría de los proyectos de ciencia de datos.

---

---

## Part III

# Estadísticas con R



# 12

---

## *Introducción a las estadísticas con R*

---

El análisis de datos es uno de los enfoques principales de este libro. Si bien las herramientas informáticas que hemos presentado son desarrollos relativamente recientes, el análisis de datos ha existido durante más de un siglo. A lo largo de los años, los analistas de datos que trabajan en proyectos específicos han presentado ideas y conceptos que se generalizan a muchas otras aplicaciones. También han identificado maneras comunes en que nos pueden engañar patrones aparentes en los datos y realidades matemáticas importantes que no son inmediatamente obvias. La acumulación de estas ideas y perspectivas ha dado lugar a la disciplina de la estadística, que ofrece un marco matemático para facilitar la descripción y evaluación formal de estas ideas.

Para evitar repetir errores comunes y perder el tiempo reinventando la rueda, es importante que los analistas de datos tengan una comprensión profunda de las estadísticas. Debido a la madurez de la disciplina, hay docenas de libros excelentes ya publicados sobre este tema y, por lo tanto, no nos enfocamos en describir el marco matemático aquí. En cambio, presentamos conceptos brevemente y luego ofrecemos estudios de caso que demuestran cómo se utilizan las estadísticas en el análisis de datos junto con el código R que implementa estas ideas. También usamos el código R para ayudar a aclarar algunos de los principales conceptos estadísticos que generalmente se describen usando las matemáticas. Recomendamos complementar este capítulo con un libro de texto de estadísticas. Dos ejemplos son *Statistics* de Freedman, Pisani y Purves y *Statistical Inference* de Casella y Berger. Los conceptos específicos que discutimos en esta parte del libro son probabilidad, inferencia estadística, modelos estadísticos, regresión y modelos lineales, que son los principales temas abarcados en un curso de estadística. Los estudios de caso que presentamos se relacionan con la crisis financiera, el pronóstico de los resultados de las elecciones, cómo entender la herencia y cómo formar un equipo de béisbol.



# 13

---

## Probabilidad

---

En los juegos de azar, la probabilidad tiene una definición muy intuitiva. Por ejemplo, sabemos lo que significa cuando decimos que la probabilidad de que un par de dados salga siete es 1 en 6. Sin embargo, este no es el caso en otros contextos. Hoy la teoría de probabilidad se utiliza de manera mucho más amplia con la palabra *probabilidad* ya parte del lenguaje cotidiano. Si escribimos “¿Cuáles son las probabilidades de” en Google, la función de auto-completar nos da: “tener mellizos”, “tener gemelos” y “ganar la lotería”. Uno de los objetivos de esta parte del libro es ayudarles a comprender cómo la probabilidad es útil para entender y describir eventos del mundo real cuando realizamos análisis de datos.

Dado que saber cómo calcular las probabilidades ofrece una ventaja en los juegos de azar, a lo largo de la historia muchas personas inteligentes, incluyendo matemáticos famosos como Cardano, Fermat y Pascal, le han dedicado tiempo y energía a pensar en las matemáticas de estos juegos. Como resultado, nació la teoría de la probabilidad. La probabilidad sigue siendo muy útil en los juegos de azar modernos. Por ejemplo, en póker, podemos calcular la probabilidad de ganar una mano basado en las cartas en la mesa. Además, los casinos se basan en la teoría de la probabilidad para desarrollar juegos que casi siempre les garantizan ganancias.

La teoría de la probabilidad es útil en muchos otros contextos y, en particular, en áreas que de alguna manera dependen de los datos afectados por el azar. Todos los otros capítulos de esta parte se basan en la teoría de la probabilidad. El conocimiento de la probabilidad es, por consiguiente, indispensable para la ciencia de datos.

---

### 13.1 Probabilidad discreta

Comenzamos explorando algunos principios básicos relacionados con datos categóricos. Esta parte de la probabilidad se conoce como *probabilidad discreta*. Luego, esto nos ayudará a comprender la teoría de la probabilidad que más tarde presentaremos para datos numéricos y continuos, los cuales son mucho más comunes en las aplicaciones de ciencia de datos. La probabilidad discreta es más útil en los juegos de cartas y, por ende, usamos estos como ejemplos.

#### 13.1.1 Frecuencia relativa

Si bien la palabra probabilidad se usa en el lenguaje cotidiano, responder a preguntas sobre la probabilidad es difícil, si no imposible porque el concepto de “probabilidad” no está bien definido. Aquí discutimos una definición matemática de *probabilidad* que nos permite dar respuestas precisas a ciertas preguntas.

Por ejemplo, si tengo 2 canicas rojas y 3 canicas azules dentro de una urna<sup>1</sup> (muchos libros de probabilidad usan este término arcaico, así que nosotros también) y escojo una al azar, ¿cuál es la probabilidad de elegir una roja? Nuestra intuición nos dice que la respuesta es  $2/5$  o 40%. Se puede dar una definición precisa al señalar que hay cinco resultados posibles de los cuales dos satisfacen la condición necesaria para el evento “escoger una canica roja”. Dado que cada uno de los cinco resultados tiene la misma probabilidad de ocurrir, concluimos que la probabilidad es .4 para rojo y .6 para azul.

Una forma más tangible de pensar en la probabilidad de un evento es la proporción de veces que ocurre el evento cuando repetimos el experimento un número infinito de veces, independientemente y bajo las mismas condiciones.

### 13.1.2 Notación

Usamos la notación  $\Pr(A)$  para denotar la probabilidad de que suceda evento  $A$ . Usamos el término general *evento* para referirnos a cosas que pueden suceder cuando algo ocurre por casualidad. En nuestro ejemplo anterior, el evento fue “escoger una canica roja”. En una encuesta política en la que llamamos al azar a 100 probables votantes estadounidenses, un ejemplo de un evento es “llamar a 48 demócratas y 52 republicanos”.

En las aplicaciones de ciencia de datos, frecuentemente trabajaremos con variables continuas. Estos eventos a menudo serán cosas como “es esta persona más alta que 6 pies”. En ese caso, escribimos eventos en una forma más matemática:  $X \geq 6$ . Veremos más de estos ejemplos a continuación. Aquí nos enfocamos en datos categóricos.

### 13.1.3 Distribuciones de probabilidad

Si conocemos la frecuencia relativa de las diferentes categorías, definir una distribución para resultados categóricos es relativamente sencillo. Simplemente asignamos una probabilidad a cada categoría. En los casos que pueden considerarse como canicas en una urna, para cada tipo de canica, su proporción define la distribución.

Si estamos llamando al azar a votantes probables de una población que es 44% demócratas, 44% republicanos, 10% indecisos y 2% del partido verde, estas proporciones definen la probabilidad para cada grupo. La distribución de probabilidad es:

---

|                                     |   |      |
|-------------------------------------|---|------|
| $\Pr(\text{elegir un republicano})$ | = | 0.44 |
| $\Pr(\text{elegir un demócrata})$   | = | 0.44 |
| $\Pr(\text{elegir un indeciso})$    | = | 0.10 |
| $\Pr(\text{elegir un verde})$       | = | 0.02 |

---

<sup>1</sup>[https://en.wikipedia.org/wiki/Urn\\_problem](https://en.wikipedia.org/wiki/Urn_problem)

## 13.2 Simulaciones Monte Carlo para datos categóricos

Las computadoras ofrecen una forma de realizar el experimento aleatorio sencillo descrito anteriormente: elegir una canica al azar de una urna que contiene tres canicas azules y dos rojas. Los generadores de números aleatorios nos permiten imitar el proceso de escoger al azar.

Un ejemplo es la función `sample` en R. Demostramos su uso en el código a continuación. Primero, usamos la función `rep` para generar la urna:

```
beads <- rep(c("red", "blue"), times = c(2,3))
beads
#> [1] "red" "red" "blue" "blue" "blue"
```

y luego usamos `sample` para escoger una canica al azar:

```
sample(beads, 1)
#> [1] "blue"
```

Esta línea de código produce un resultado aleatorio. Queremos repetir este experimento un número infinito de veces, pero es imposible repetirlo para siempre. Sin embargo, podemos repetir el experimento un número suficientemente grande de veces para que los resultados sean prácticamente equivalentes a repetirlo para siempre. **Este es un ejemplo de una simulación Monte Carlo.**

Gran parte de lo que estudian los estadísticos matemáticos y teóricos, que no discutimos en este libro, se relaciona con proveer definiciones rigurosas de “prácticamente equivalente”, así como estudiar cuán cerca nos llevan un gran número de experimentos a lo que sucede en el límite. Más adelante en esta sección, ofrecemos un acercamiento práctico para determinar qué es “lo suficientemente grande”.

Para realizar nuestra primera simulación Monte Carlo, utilizamos la función `replicate`, que nos permite repetir la misma tarea varias veces. Aquí, repetimos el evento aleatorio  $B = 10,000$  veces:

```
B <- 10000
events <- replicate(B, sample(beads, 1))
```

Ahora podemos ver si nuestra definición realmente está de acuerdo con esta aproximación de simulación Monte Carlo. Nosotros podemos usar `table` para ver la distribución:

```
tab <- table(events)
tab
#> events
#> blue red
#> 5976 4024
```

y `prop.table` nos da las proporciones:

```
prop.table(tab)
#> events
#> blue red
#> 0.598 0.402
```

Los números anteriores son probabilidades estimadas proveídas por una simulación Monte Carlo. La teoría estadística, que no discutimos aquí, nos dice que en lo que  $B$  se hace más grande, las estimaciones se acercan a  $3/5 = .6$  y  $2/5 = .4$ .

Aunque este es un ejemplo sencillo y no muy útil, luego utilizaremos simulaciones Monte Carlo para estimar probabilidades en casos en los cuales es difícil calcular cantidades exactas. Antes de profundizar en ejemplos más complejos, usaremos algunos sencillos para demostrar las herramientas informáticas disponibles en R.

### 13.2.1 Fijar la semilla aleatoria

Antes de continuar, explicaremos brevemente la siguiente línea importante de código:

```
set.seed(1986)
```

A lo largo de este libro, utilizamos generadores de números aleatorios. Esto implica que muchos de los resultados que presentamos pueden cambiar por casualidad y una versión congelada del libro puede mostrar un resultado diferente al que obtienen cuando intenten codificar como observan en el libro. Esto no es un problema ya que los resultados son aleatorios y pueden cambiar. Sin embargo, si quieren asegurarse de que los resultados son exactamente los mismos cada vez que los ejecuten, pueden fijar la semilla (*seed* en inglés) de generación de números aleatorios de R en un número específico. Arriba la fijamos en 1986. Queremos evitar usar la misma semilla cada vez. Una forma popular de escoger la semilla es restando el mes y el día del año. Por ejemplo, para el 20 de diciembre de 2018 fijamos la semilla en 1986:  $2018 - 12 - 20 = 1986$ .

Pueden obtener más información sobre cómo fijar la semilla mirando la documentación:

```
?set.seed
```

En los ejercicios, es posible que les pidamos que fijen la semilla para asegurar que sus resultados sean exactamente lo que esperamos.

### 13.2.2 Con y sin reemplazo

La función `sample` tiene un argumento que nos permite elegir más de un elemento de la urna. Sin embargo, por defecto, esta selección ocurre *sin reemplazo*; es decir, después de seleccionar una canica, no se vuelve a colocar en la urna. Observen lo que sucede cuando pedimos seleccionar cinco canicas al azar:

```
sample(beads, 5)
#> [1] "red" "blue" "blue" "blue" "red"
sample(beads, 5)
```

```
#> [1] "red" "red" "blue" "blue" "blue"
sample(beads, 5)
#> [1] "blue" "red" "blue" "red" "blue"
```

Esto resulta en reordenamientos que siempre tienen tres canicas azules y dos rojas. Si pedimos que se seleccionen seis canicas, obtenemos un error:

```
sample(beads, 6)
```

```
Error in sample.int(length(x), size, replace, prob) : cannot take a sample
larger than the population when 'replace = FALSE'
```

Sin embargo, la función `sample` se puede usar directamente, sin el uso de `replicate`, para repetir el mismo experimento de elegir 1 de las 5 canicas, continuamente, en las mismas condiciones. Para hacer esto, muestreamos *con reemplazo*; es decir, se devuelve la canica a la urna después de seleccionarla. Podemos decirle a `sample` que haga esto cambiando el argumento `replace`, que por defecto es `FALSE`, a `replace = TRUE`:

```
events <- sample(beads, B, replace = TRUE)
prop.table(table(events))
#> events
#> blue red
#> 0.602 0.398
```

No sorprende que obtengamos resultados muy similares a los obtenidos previamente con `replicate`.

### 13.3 Independencia

Decimos que dos eventos son independientes si el resultado de uno no afecta al otro. El ejemplo clásico es el lanzamiento de monedas. Cada vez que lanzamos una moneda, la probabilidad de ver cara es  $1/2$ , independientemente de los resultados de lanzamientos anteriores. Lo mismo es cierto cuando recogemos canicas de una urna con reemplazo. En el ejemplo anterior, la probabilidad de rojo es 0.40 independientemente de las selecciones anteriores.

Muchos ejemplos de eventos que no son independientes provienen de juegos de cartas. Cuando repartimos la primera carta, la probabilidad de obtener una K es  $1/13$  ya que hay trece posibilidades: Dos, Tres, ..., Diez, J, Q, K y As. Pero si repartimos una K como la primera carta y no la reemplazamos en la baraja, la probabilidad de que una segunda carta sea K es menor porque solo quedan tres Ks: la probabilidad es 3 de 51. Estos eventos, por lo tanto, **no son independientes**: el primer resultado afecta al siguiente.

Para ver un caso extremo de eventos no independientes, consideren nuestro ejemplo de escoger cinco canicas al azar **sin reemplazo**:

```
x <- sample(beads, 5)
```

Si tienen que adivinar el color de la primera canica, predecirán azul ya que azul tiene un 60% de probabilidad. Pero si les mostramos el resultado de los últimos cuatro resultados:

```
x[2:5]
#> [1] "blue" "blue" "blue" "red"
```

¿aún adivinarían azul? Por supuesto que no. Ahora saben que la probabilidad de rojo es 1 ya que la única canica que queda es roja. Los eventos no son independientes, por lo que las probabilidades cambian.

## 13.4 Probabilidades condicionales

Cuando los eventos no son independientes, las *probabilidades condicionales* son útiles. Ya vimos un ejemplo de una probabilidad condicional: calculamos la probabilidad de que una segunda carta repartida sea K dado que la primera fue K. En la probabilidad, usamos la siguiente notación:

$$\Pr(\text{Card 2 is a king} \mid \text{Card 1 is a king}) = 3/51$$

Utilizamos el  $\mid$  como abreviatura de “dado eso” o “condicional en”.

Cuando dos eventos, digamos  $A$  y  $B$ , son independientes, tenemos:

$$\Pr(A \mid B) = \Pr(A)$$

Esta es la forma matemática de decir: el hecho de que  $B$  sucedió no afecta la probabilidad de que  $A$  suceda. De hecho, esto puede considerarse la definición matemática de independencia.

## 13.5 Reglas de la adición y de la multiplicación

### 13.5.1 Regla de la multiplicación

Si queremos saber la probabilidad de que ocurran dos eventos, digamos  $A$  y  $B$ , podemos usar la regla de la multiplicación:

$$\Pr(A \text{ and } B) = \Pr(A)\Pr(B \mid A)$$

Usemos el juego de cartas Blackjack como ejemplo. En Blackjack, se les asignan dos cartas al azar. Despues de ver lo que tienen, pueden pedir más cartas. El objetivo es acercarse más

a 21 que el croupier, sin pasar. Las *cartas con figuras* (*face cards* en inglés) valen 10 puntos y las Ases valen 11 o 1 (uno elige).

Entonces, en Blackjack, para calcular las probabilidades de obtener un 21 recibiendo un As y luego una carta de figura, calculamos la probabilidad de que la primera carta sea un As y multiplicamos por la probabilidad de sacar una carta de figura o un 10 dado que la primera fue un As:  $1/13 \times 16/51 \approx 0.025$ .

La regla de la multiplicación también se aplica a más de dos eventos. Podemos usar la inducción para incluir más eventos:

$$\Pr(A \text{ and } B \text{ and } C) = \Pr(A)\Pr(B | A)\Pr(C | A \text{ and } B)$$

### 13.5.2 Regla de la multiplicación bajo independencia

Cuando tenemos eventos independientes, la regla de la multiplicación se hace más sencilla:

$$\Pr(A \text{ and } B \text{ and } C) = \Pr(A)\Pr(B)\Pr(C)$$

Pero debemos tener mucho cuidado antes de usar esto ya que suponer independencia cuando realmente no existe puede resultar en cálculos de probabilidad muy diferentes e incorrectos.

Como ejemplo, imaginen un caso judicial en el que se describe al sospechoso como teniendo bigote y barba. El acusado tiene bigote y barba y la fiscalía trae a un “experto” que testimonia que  $1/10$  hombres tienen barba y  $1/5$  tienen bigote, así que usando la regla de la multiplicación concluimos que solo  $1/10 \times 1/5$  o  $0.02$  tienen ambos.

¡Pero para multiplicar así necesitamos suponer independencia! Digamos que la probabilidad condicional de que un hombre tenga un bigote condicionado en que tenga barba es .95. Entonces el cálculo correcto de la probabilidad resulta en un número mucho mayor:  $1/10 \times 95/100 = 0.095$ .

La regla de la multiplicación también nos da una fórmula general para calcular probabilidades condicionales:

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

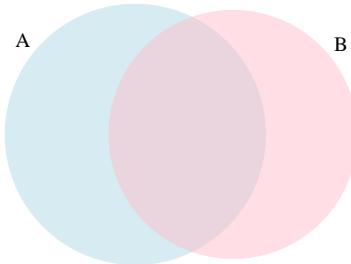
Para ilustrar cómo usamos estas fórmulas y conceptos en la práctica, utilizaremos varios ejemplos relacionados con los juegos de cartas.

### 13.5.3 Regla de la adición

La regla de la adición nos dice que:

$$\Pr(A \text{ or } B) = \Pr(A) + \Pr(B) - \Pr(A \text{ and } B)$$

Esta regla es intuitiva: piense en un diagrama de Venn. Si simplemente sumamos las probabilidades, contamos la intersección dos veces, por lo que debemos restar una instancia.



### 13.6 Combinaciones y permutaciones

En nuestro primer ejemplo, imaginamos una urna con cinco canicas. Recuerden que para calcular la distribución de probabilidad de un sorteo, simplemente enumeramos todas las probabilidades. Hubo 5 y entonces para cada evento contamos cuántas de estas probabilidades estaban asociadas con el evento. La probabilidad de elegir una canica azul es  $3/5$  porque de los cinco resultados posibles, tres fueron azules.

Para casos más complicados, los cálculos no son tan sencillos. Por ejemplo, ¿cuál es la probabilidad de que siescojo cinco cartas sin reemplazo, obtenga todas cartas del mismo palo (*suit* en inglés), lo que se conoce como “flush” en el póker? En un curso de probabilidad discreta, se aprende la teoría sobre cómo hacer estos cálculos. Aquí nos enfocamos en cómo usar el código R para calcular las respuestas.

Primero, construyamos una baraja de cartas. Para esto, usaremos las funciones `expand.grid` y `paste`. Usamos `paste` para crear cadenas uniendo cadenas más pequeñas. Para hacer esto, tomamos el número y el palo de una carta y creamos el nombre de la carta de esta manera:

```
number <- "Three"
suit <- "Hearts"
paste(number, suit)
#> [1] "Three Hearts"
```

`paste` también funciona en pares de vectores que realizan la operación elemento por elemento:

```
paste(letters[1:5], as.character(1:5))
#> [1] "a 1" "b 2" "c 3" "d 4" "e 5"
```

La función `expand.grid` nos da todas las combinaciones de entradas de dos vectores. Por ejemplo, si tienen pantalones azules y negros y camisas blancas, grises y a cuadros (*plaide* en inglés), todas sus combinaciones son:

```
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
```

Aquí es como generamos una baraja de cartas:

```
suits <- c("Diamonds", "Clubs", "Hearts", "Spades")
numbers <- c("Ace", "Deuce", "Three", "Four", "Five", "Six", "Seven",
 "Eight", "Nine", "Ten", "Jack", "Queen", "King")
deck <- expand.grid(number=numbers, suit=suits)
deck <- paste(deck$number, deck$suit)
```

Con la baraja construida, podemos verificar que la probabilidad de que una K sea la primera carta es 1/13 calculando la proporción de posibles resultados que satisfagan nuestra condición:

```
kings <- paste("King", suits)
mean(deck %in% kings)
#> [1] 0.0769
```

Ahora, ¿qué tal la probabilidad condicional de que la segunda carta sea una K dado que la primera era una K? Anteriormente, dedujimos que si una K ya está fuera de la baraja y quedan 51 cartas, entonces la probabilidad es 3/51. Confirmemos enumerando todos los resultados posibles.

Para hacer esto, podemos usar la función `permutations` del paquete **gtools**. Para cualquier lista de tamaño `n`, esta función calcula todas las diferentes combinaciones que podemos obtener cuando seleccionamos `r` artículos. Aquí están todas las formas en que podemos elegir dos números de una lista que consiste en 1,2,3:

```
library(gtools)
permutations(3, 2)
#> [,1] [,2]
#> [1,] 1 2
#> [2,] 1 3
#> [3,] 2 1
#> [4,] 2 3
#> [5,] 3 1
#> [6,] 3 2
```

Observen que el orden importa aquí: 3,1 es diferente de 1,3. Además, tengan en cuenta que (1,1), (2,2) y (3,3) no aparecen porque una vez que elegimos un número, no puede volver a aparecer.

Opcionalmente, podemos añadir un vector. Si desean ver cinco números de teléfono aleatorios (de siete dígitos) de todos los números de teléfono posibles (sin repeticiones), pueden escribir:

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

En lugar de usar los números del 1 al 10, el valor por defecto, R usa lo que proveemos a través de `v`: los dígitos de 0 a 9.

Para calcular todas las formas posibles en que podemos elegir dos cartas cuando el orden importa, escribimos:

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

Esta es una matriz con dos columnas y 2652 filas. Con una matriz podemos obtener la primera y segunda carta así:

```
first_card <- hands[,1]
second_card <- hands[,2]
```

Ahora los casos para los cuales la primera carta es una K se pueden calcular así:

```
kings <- paste("King", suits)
sum(first_card %in% kings)
#> [1] 204
```

Para obtener la probabilidad condicional, calculamos qué fracción de estos tiene una K como la segunda carta:

```
sum(first_card %in% kings & second_card %in% kings) / sum(first_card %in% kings)
#> [1] 0.0588
```

que es exactamente 3/51, como ya habíamos deducido. Tengan en cuenta que el código anterior es equivalente a:

```
mean(first_card %in% kings & second_card %in% kings) / mean(first_card %in% kings)
#> [1] 0.0588
```

que usa `mean` en lugar de `sum` y es una versión R de:

$$\frac{\Pr(A \text{ and } B)}{\Pr(A)}$$

¿Y qué tal si el orden no importa? Por ejemplo, en Blackjack, si le dan un As y una carta

de figura como su primera mano, se llama un *Natural 21* y ganan automáticamente. Si quisieramos calcular la probabilidad de que esto suceda, enumeraríamos las *combinaciones*, no las permutaciones, ya que el orden no importa.

```
combinations(3,2)
#> [,1] [,2]
#> [1,] 1 2
#> [2,] 1 3
#> [3,] 2 3
```

En la segunda línea, el resultado no incluye (2,1) porque (1,2) ya se enumeró. Lo mismo aplica a (3,1) y (3,2).

Entonces, para calcular la probabilidad de un *Natural 21*, podemos hacer esto:

```
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
```

En la última línea, suponemos que el As es la primera carta que recibimos. Esto lo sabemos porque, sabiendo como `combination` enumera las probabilidades, entendemos que enumerará este caso primero. Pero para estar seguros, podríamos haber producido la misma respuesta al escribir lo siguiente:

```
mean((hands[,1] %in% aces & hands[,2] %in% facecard) |
 (hands[,2] %in% aces & hands[,1] %in% facecard))
#> [1] 0.0483
```

### 13.6.1 Ejemplo Monte Carlo

En lugar de usar `combinations` para deducir la probabilidad exacta de un *Natural 21*, podemos usar una simulación Monte Carlo para estimar esta probabilidad. En este caso, escogemos dos cartas una y otra vez y notamos cuántos 21s tenemos. Podemos usar la función `sample` para escoger dos cartas sin reemplazos:

```
hand <- sample(deck, 2)
hand
#> [1] "Queen Clubs" "Seven Spades"
```

Y luego verificar si una carta es un As y la otra una carta de figura o un 10. De ahora en adelante, incluimos 10 cuando decimos *carta de figura* o *figura*. Ahora necesitamos verificar ambas probabilidades:

```
(hands[1] %in% aces & hands[2] %in% facecard) |
 (hands[2] %in% aces & hands[1] %in% facecard)
#> [1] FALSE
```

Si repetimos esto 10,000 veces, obtenemos una muy buena aproximación de la probabilidad de un *Natural 21*.

Comencemos escribiendo una función que escoje una mano y devuelve TRUE si obtenemos un 21. La función no necesita argumentos porque usa objetos definidos en el entorno global.

```
blackjack <- function(){
 hand <- sample(deck, 2)
 (hand[1] %in% aces & hand[2] %in% facecard) |
 (hand[2] %in% aces & hand[1] %in% facecard)
}
```

Aquí tenemos que verificar ambas probabilidades: As primero o As segundo porque no estamos usando la función `combinations`. La función devuelve TRUE si obtenemos un 21 y FALSE de otra manera:

```
blackjack()
#> [1] FALSE
```

Ahora podemos jugar este juego, digamos, 10,000 veces:

```
B <- 10000
results <- replicate(B, blackjack())
mean(results)
#> [1] 0.0475
```

## 13.7 Ejemplos

En esta sección, describimos dos ejemplos populares de probabilidad discreta: el problema Monty Hall y el problema del cumpleaños. Usamos R para ayudar a ilustrar los conceptos matemáticos.

### 13.7.1 Problema Monty Hall

En la década de 1970 en EE.UU, hubo un programa de concursos llamado “Let’s Make a Deal” y Monty Hall era el anfitrión. En algún momento del juego, se le pedía al concursante que eligiera una de tres puertas. Detrás de una puerta había un premio, mientras que detrás de las otras puertas tenían una cabra que señalaba que el concursante había perdido. Después de que el concursante eligiera una puerta y antes de revelar si esa puerta contenía un premio, Monty Hall abría una de las otras dos puertas y le mostraba al concursante que no había

ningún premio detrás de esa puerta. Luego le preguntaba al concursante: “¿Quiere cambiar de puerta?” ¿Qué harían Uds.?

Podemos usar la probabilidad para mostrar que si se quedan con la opción de la puerta original, sus probabilidades de ganar un premio siguen siendo 1 en 3. Sin embargo, si cambian a la otra puerta, sus probabilidades de ganar duplican a 2 en 3! Esto parece contradictorio. Muchas personas piensan incorrectamente que ambas probabilidades son 1 en 2 ya que uno elige entre 2 opciones. Pueden ver una explicación matemática detallada en Khan Academy<sup>2</sup> o leer una en Wikipedia<sup>3</sup>. A continuación, usamos una simulación Monte Carlo para ver qué estrategia es mejor. Tengan en cuenta que este código se escribe con más detalle de lo necesario para fines pedagógicos.

Comencemos con la estrategia de no cambiar de puerta:

```
B <- 10000
monty_hall <- function(strategy){
 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
 switch <- doors[!doors %in% c(my_pick, show)]
 choice <- ifelse(strategy == "stick", stick, switch)
 choice == prize_door
}
stick <- replicate(B, monty_hall("stick"))
mean(stick)
#> [1] 0.342
switch <- replicate(B, monty_hall("switch"))
mean(switch)
#> [1] 0.668
```

Mientras escribimos el código, notamos que las líneas que comienzan con `my_pick` y `show` no afectan la última operación lógica cuando nos atenemos a nuestra elección original. De esto, debemos darnos cuenta de que la probabilidad es de 1 en 3, la misma con la cual comenzamos. Cuando cambiamos, la estimación Monte Carlo confirma el cálculo de 2/3. Esto nos ayuda entender el problema mejor al mostrar que estamos quitando una puerta, `show`, que definitivamente no esconde un premio de nuestras opciones. También vemos que, a menos que lo hagamos bien cuando elegimos por primera vez, ustedes ganan:  $1 - 1/3 = 2/3$ .

### 13.7.2 Problema de cumpleaños

Imagínense que están en un salón de clase con 50 personas. Si suponemos que este es un grupo de 50 personas seleccionadas al azar, ¿cuál es la probabilidad de que al menos dos

<sup>2</sup><https://www.khanacademy.org/math/precalculus/prob-comb/dependent-events-precalc/v/monty-hall-problem>

<sup>3</sup>[https://en.wikipedia.org/wiki/Monty\\_Hall\\_problem](https://en.wikipedia.org/wiki/Monty_Hall_problem)

personas tengan el mismo cumpleaños? Aunque es algo avanzado, podemos deducir esto matemáticamente. Haremos esto más tarde, pero aquí usamos una simulación Monte Carlo. Para simplificar, suponemos que nadie nació el 29 de febrero. Esto realmente no cambia mucho la respuesta.

Primero, tengan en cuenta que los cumpleaños se pueden representar como números entre 1 y 365, por lo que se puede obtener una muestra de 50 cumpleaños de esta manera:

```
n <- 50
bdays <- sample(1:365, n, replace = TRUE)
```

Para verificar si en este set particular de 50 personas tenemos al menos dos con el mismo cumpleaños, podemos usar la función `duplicated`, que devuelve TRUE siempre que un elemento de un vector sea un duplicado. Aquí hay un ejemplo:

```
duplicated(c(1,2,3,1,4,3,5))
#> [1] FALSE FALSE FALSE TRUE FALSE TRUE FALSE
```

La segunda vez que aparecen 1 y 3, obtenemos un TRUE. Entonces, para verificar si dos cumpleaños son iguales, simplemente usamos las funciones `any` y `duplicated` así:

```
any(duplicated(bdays))
#> [1] TRUE
```

En este caso, vemos que sucedió. Al menos dos personas tuvieron el mismo cumpleaños.

Para estimar la probabilidad de un cumpleaños compartido en el grupo, repetimos este experimento muestreando sets de 50 cumpleaños una y otra vez:

```
B <- 10000
same_birthday <- function(n){
 bdays <- sample(1:365, n, replace=TRUE)
 any(duplicated(bdays))
}
results <- replicate(B, same_birthday(50))
mean(results)
#> [1] 0.969
```

¿Esperaban que la probabilidad fuera tan alta?

Las personas tienden a subestimar estas probabilidades. Para tener una idea de por qué es tan alta, piensen en lo que sucede cuando el tamaño del grupo se acerca a 365. Con 365, se nos acaban los días y la probabilidad es una.

Digamos que queremos usar este conocimiento para apostar con amigos sobre si dos personas en un grupo tienen el mismo cumpleaños. ¿Con un grupo de qué tamaño son las probabilidades superiores a 50%? ¿Superiores a 75%?

Empecemos creando una tabla de consulta. Rápidamente podemos crear una función para calcular esto para cualquier tamaño de grupo:

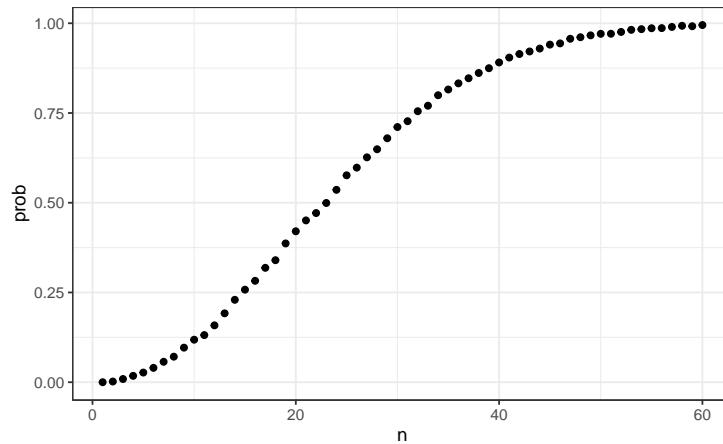
```
compute_prob <- function(n, B=10000){
 results <- replicate(B, same_birthday(n))
 mean(results)
}
```

Usando la función `sapply`, podemos realizar operaciones elemento por elemento en cualquier función:

```
n <- seq(1,60)
prob <- sapply(n, compute_prob)
```

Ahora podemos hacer un gráfico de las probabilidades estimadas de dos personas tener el mismo cumpleaños en un grupo de tamaño  $n$ :

```
library(tidyverse)
prob <- sapply(n, compute_prob)
qplot(n, prob)
```



Ahora calculemos las probabilidades exactas en lugar de usar simulaciones Monte Carlo. No solo obtenemos la respuesta exacta usando matemáticas, sino que los cálculos son mucho más rápidos ya que no tenemos que generar experimentos.

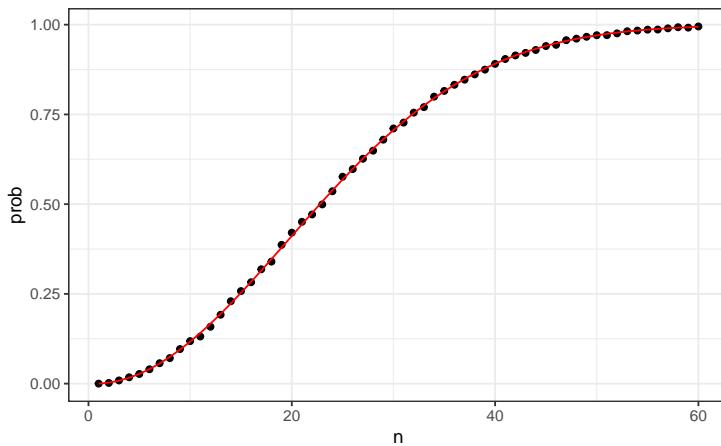
Para simplificar las matemáticas, en lugar de calcular la probabilidad de que ocurra, calcularemos la probabilidad de que no ocurra. Para esto, usamos la regla de la multiplicación.

Comencemos con la primera persona. La probabilidad de que persona 1 tenga un cumpleaños único es 1. La probabilidad de que persona 2 tenga un cumpleaños único, dado que ya se le asignó un día a persona 1, es  $364/365$ . Luego, dado que las dos primeras personas tienen cumpleaños únicos, persona 3 tiene 363 días para elegir. Continuamos de esta manera y encontramos que las probabilidades de que todas las 50 personas tengan un cumpleaños único son:

$$1 \times \frac{364}{365} \times \frac{363}{365} \cdots \frac{365-n+1}{365}$$

Podemos escribir una función que haga esto para cualquier número:

```
exact_prob <- function(n){
 prob_unique <- seq(365, 365-n+1)/365
 1 - prod(prob_unique)
}
eprob <- sapply(n, exact_prob)
qplot(n, prob) + geom_line(aes(n, eprob), col = "red")
```



Este gráfico muestra que la simulación Monte Carlo ofrece una muy buena estimación de la probabilidad exacta. Si no hubiera sido posible calcular las probabilidades exactas, aún habríamos podido estimar con precisión las probabilidades.

### 13.8 Infinito en la práctica

La teoría descrita aquí requiere repetir experimentos una y otra vez para siempre. En la práctica no podemos hacer esto. En los ejemplos anteriores, utilizamos  $B = 10,000$  experimentos Monte Carlo y resultó que esto nos dio estimados precisos. Cuanto mayor sea este número, más preciso será el estimado hasta que la aproximación sea tan buena que sus computadoras no podrán notar la diferencia. Pero en cálculos más complejos, 10,000 puede ser insuficiente. Además, para algunos cálculos, 10,000 experimentos podrían no ser computacionalmente factibles. En la práctica, no sabremos cuál es la respuesta, por lo que no sabremos si nuestra estimación Monte Carlo es precisa. Sabemos que entre más grande sea  $B$ , mejor será la aproximación. ¿Pero cuán grande necesitamos que sea? Esta es realmente una pregunta desafiante y frecuentemente contestarla requiere una formación avanzada en estadística teórica.

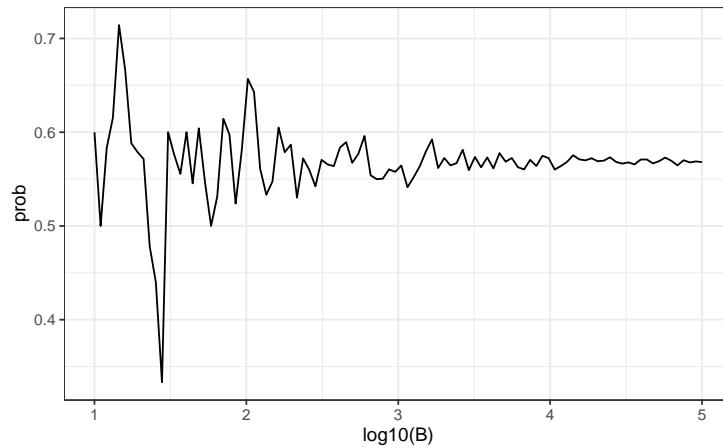
Un enfoque práctico que describiremos aquí es verificar la estabilidad del estimado. A continuación ofrecemos un ejemplo con el problema de cumpleaños para un grupo de 25 personas.

```
B <- 10^seq(1, 5, len = 100)
compute_prob <- function(B, n=25){
 same_day <- replicate(B, same_birthday(n))
```

```

 mean(same_day)
}
prob <- sapply(B, compute_prob)
qplot(log10(B), prob, geom = "line")

```



En este gráfico, podemos ver que los valores comienzan a estabilizarse (es decir, varían menos de .01) alrededor de 1000. Noten que la probabilidad exacta, que en este caso sabemos, es 0.569.

## 13.9 Ejercicios

1. Se escoge una canica al azar de una caja que contiene: 3 canicas cian, 5 canicas magenta y 7 canicas amarillas. ¿Cuál es la probabilidad de que la canica sea cian?
2. ¿Cuál es la probabilidad de que la canica no sea cian?
3. En lugar de escoger solo una canica, escoja dos canicas. Saque la primera canica sin devolverla a la caja. Este es un muestreo **sin** reemplazo. ¿Cuál es la probabilidad de que la primera canica sea cian y la segunda no sea cian?
4. Ahora repita el experimento, pero esta vez, después de sacar la primera canica y anotar el color, devuélvala a la caja y agite la caja. Este es un muestreo **con** reemplazo. ¿Cuál es la probabilidad de que la primera canica sea cian y la segunda canica no sea cian?
5. Dos eventos  $A$  y  $B$  son independientes si  $\Pr(A \text{ and } B) = \Pr(A)\Pr(B)$ . ¿Bajo qué situación son independientes la selección?
  - a. No reemplaza el artículo seleccionado.
  - b. Reemplaza el artículo seleccionado.
  - c. Ninguna.
  - d. Ambas.

6. Digamos que ha sacado 5 canicas de la caja, con reemplazo, y todas han sido amarillas. ¿Cuál es la probabilidad de que la próxima sea amarilla?
7. Si lanza un dado de 6 lados seis veces, ¿cuál es la probabilidad de no ver un 6?
8. Dos equipos de baloncesto, digamos los Celtics y los Cavs, están jugando una serie de siete juegos. Los Cavs son un mejor equipo y tienen un 60% de probabilidad de ganar cada juego. ¿Cuál es la probabilidad de que los Celtics ganen al menos un juego?
9. Cree una simulación Monte Carlo para confirmar su respuesta al problema anterior. Utilizar  $B <- 10000$  simulaciones. Sugerencia: use el siguiente código para generar los resultados de los primeros cuatro juegos:

```
celtic_wins <- sample(c(0,1), 4, replace = TRUE, prob = c(0.6, 0.4))
```

Los Celtics deben ganar uno de estos 4 juegos.

10. Dos equipos de baloncesto, digamos los Cavs y los Warriors, están jugando una serie de campeonato de siete juegos. El primero en ganar cuatro juegos, por consiguiente, gana la serie. Los equipos son igualmente buenos, por lo que cada uno tiene una probabilidad de 50-50 de ganar cada juego. Si los Cavs pierden el primer juego, ¿cuál es la probabilidad de que ganen la serie?
11. Confirme los resultados de la pregunta anterior con una simulación Monte Carlo.
12. Dos equipos,  $A$  y  $B$ , están jugando una serie de siete juegos. Equipo  $A$  es mejor que equipo  $B$  y tiene un  $p > 0.5$  probabilidad de ganar cada juego. Dado un valor- $p$ , la probabilidad de que el equipo no favorito  $B$  gane la serie se puede calcular con la siguiente función basada en una simulación Monte Carlo:

```
prob_win <- function(p){
 B <- 10000
 result <- replicate(B, {
 b_win <- sample(c(1,0), 7, replace = TRUE, prob = c(1-p, p))
 sum(b_win)>=4
 })
 mean(result)
}
```

Use la función `sapply` para calcular la probabilidad, llámela `Pr`, de ganar para  $p <- \text{seq}(0.5, 0.95, 0.025)$ . Luego grafique el resultado.

13. Repita el ejercicio anterior, pero ahora mantenga la probabilidad fija en  $p <- 0.75$  y calcule la probabilidad para diferentes números de juegos necesarios para acabar la serie: ganar 1 juego, ganar 2 de 3 juegos, ganar 3 de 5 juegos, ... Específicamente,  $N <- \text{seq}(1, 25, 2)$ . Sugerencia: use esta función:

```
prob_win <- function(N, p=0.75){
 B <- 10000
 result <- replicate(B, {
 b_win <- sample(c(1,0), N, replace = TRUE, prob = c(1-p, p))
 sum(b_win)>=(N+1)/2
 })
 mean(result)
}
```

## 13.10 Probabilidad continua

En la Sección 8.4, explicamos por qué al resumir una lista de valores numéricos, como las alturas, no es útil construir una distribución que defina una proporción para cada resultado posible. Por ejemplo, imaginemos que medimos a cada persona en una población grande, digamos de tamaño  $n$ , con una precisión extremadamente alta. Como no hay dos personas con exactamente la misma altura, debemos asignar la proporción  $1/n$  a cada valor observado y como consecuencia no se obtiene ningún resumen útil. Del mismo modo, al definir distribuciones de probabilidad, no es útil asignar una probabilidad muy pequeña a cada altura.

Al igual que cuando se usan distribuciones para resumir datos numéricos, es mucho más práctico definir una función que opere en intervalos en lugar de valores individuales. La forma estándar de hacerlo es utilizando la *función de distribución acumulada* (CDF por sus siglas en inglés).

Describimos la función de distribución acumulada empírica (eCDF) en la Sección 8.4 como un resumen básico de una lista de valores numéricos. Como ejemplo, anteriormente definimos la distribución de altura para los estudiantes varones adultos. Aquí definimos el vector  $x$  para contener estas alturas:

```
library(tidyverse)
library(dslabs)
data(heights)
x <- heights %>% filter(sex=="Male") %>% pull(height)
```

Definimos la función de distribución acumulada empírica como:

```
F <- function(a) mean(x<=a)
```

que, por cualquier valor  $a$ , da la proporción de valores en la lista  $x$  que son más pequeños o iguales que  $a$ .

Tengan en cuenta que todavía no hemos discutido la probabilidad en el contexto de la CDF. Hagamos esto preguntando lo siguiente: ¿si elijo a uno de los estudiantes varones al azar, cuál es la probabilidad de que sea más alto que 70.5 pulgadas? Debido a que cada estudiante tiene la misma probabilidad de ser elegido, la respuesta a esto es equivalente a la proporción de estudiantes que son más altos que 70.5 pulgadas. Usando la CDF obtenemos una respuesta escribiendo:

```
1 - F(70)
#> [1] 0.377
```

Una vez que se define una CDF, podemos usar esto para calcular la probabilidad de cualquier subconjunto. Por ejemplo, la probabilidad de que un estudiante esté entre altura  $a$  y altura  $b$  es:

**F(b)-F(a)**

Como podemos calcular la probabilidad de cualquier evento posible de esta manera, la función de probabilidad acumulada define la distribución de probabilidad para elegir una altura al azar de nuestro vector de alturas  $x$ .

### 13.11 Distribuciones teóricas continuas

En la Sección 8.8, presentamos la distribución normal como una aproximación útil a muchas distribuciones naturales, incluyendo la altura. La distribución acumulada para la distribución normal se define mediante una fórmula matemática que en R se puede obtener con la función `pnorm`. Decimos que una cantidad aleatoria se distribuye normalmente con promedio  $m$  y desviación estándar  $s$  si su distribución de probabilidad se define por:

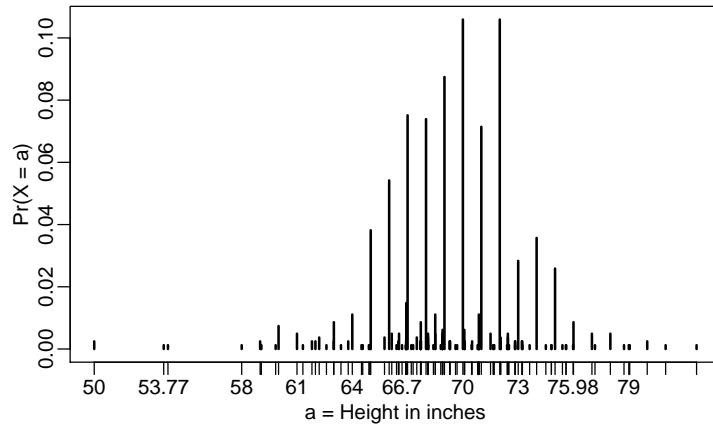
**F(a) = pnorm(a, m, s)**

Esto es útil porque si estamos dispuestos a usar la aproximación normal para, por ejemplo, la altura, no necesitamos todo el set de datos para responder a preguntas como: ¿cuál es la probabilidad de que un estudiante seleccionado al azar sea más alto que 70 pulgadas? Solo necesitamos la altura promedio y la desviación estándar:

```
m <- mean(x)
s <- sd(x)
1 - pnorm(70.5, m, s)
#> [1] 0.371
```

#### 13.11.1 Distribuciones teóricas como aproximaciones

La distribución normal se deriva matemáticamente: no necesitamos datos para definirla. Para los científicos de datos, casi todo lo que hacemos en la práctica involucra datos. Los datos son siempre, desde un punto de vista técnico, discretos. Por ejemplo, podríamos considerar nuestros datos de altura categóricos con cada altura específica como una categoría única. La distribución de probabilidad se define por la proporción de estudiantes que indican cada altura. Aquí hay un gráfico de esa distribución de probabilidad:



Mientras que la mayoría de los estudiantes redondearon sus alturas a la pulgada más cercana, otros indicaron valores con más precisión. Un estudiante indicó que su altura era 69.6850393700787 pulgadas, que equivale 177 centímetros. La probabilidad asignada a esta altura es 0.001 o 1 en 812. La probabilidad de 70 pulgadas es mucho mayor en 0.106, pero ¿tiene sentido pensar que la probabilidad de tener exactamente 70 pulgadas es diferente de 69.6850393700787? Claramente es mucho más útil para fines de análisis de datos tratar este resultado como una variable numérica continua, teniendo en cuenta que muy pocas personas, o tal vez ninguna, son exactamente 70 pulgadas y que la razón por la que obtenemos más valores en 70 es porque las personas redondean a la pulgada más cercana.

Con distribuciones continuas, la probabilidad de un valor singular no se define. Por ejemplo, no tiene sentido preguntar cuál es la probabilidad de que un valor distribuido normalmente sea 70. En cambio, definimos probabilidades para intervalos. Por lo tanto, podríamos preguntar cuál es la probabilidad de que alguien mida entre 69.5 y 70.5.

En casos como la altura, en los que los datos se redondean, la aproximación normal es particularmente útil si estamos trabajando con intervalos que incluyen exactamente un número redondo. Por ejemplo, la distribución normal es útil para aproximar la proporción de estudiantes que indican valores en intervalos como los tres siguientes:

```
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
```

Observen lo mucho que nos acercamos con la aproximación normal:

```
pnorm(68.5, m, s) - pnorm(67.5, m, s)
#> [1] 0.103
pnorm(69.5, m, s) - pnorm(68.5, m, s)
#> [1] 0.11
pnorm(70.5, m, s) - pnorm(69.5, m, s)
#> [1] 0.108
```

Sin embargo, la aproximación no es tan útil para otros intervalos. Por ejemplo, observen cómo se descompone la aproximación cuando intentamos estimar:

```
mean(x <= 70.9) - mean(x<=70.1)
#> [1] 0.0222
```

con:

```
pnorm(70.9, m, s) - pnorm(70.1, m, s)
#> [1] 0.0836
```

En general, llamamos a esta situación *discretización*. Aunque la distribución de altura real es continua, las alturas reportadas tienden a ser más comunes en valores discretos, en este caso, debido al redondeo. Con tal que sepamos cómo lidiar con esta realidad, la aproximación normal puede ser una herramienta muy útil.

### 13.11.2 La densidad de probabilidad

Para distribuciones categóricas, podemos definir la probabilidad de una categoría. Por ejemplo, un lanzamiento de dado, llamémoslo  $X$ , puede ser 1,2,3,4,5 o 6. La probabilidad de 4 se define como:

$$\Pr(X = 4) = 1/6$$

La CDF entonces se puede definir fácilmente:

$$F(4) = \Pr(X \leq 4) = \Pr(X = 4) + \Pr(X = 3) + \Pr(X = 2) + \Pr(X = 1)$$

Aunque para distribuciones continuas la probabilidad de un solo valor  $\Pr(X = x)$  no se define, hay una definición teórica que tiene una interpretación similar. La densidad de probabilidad en  $x$  se define como la función  $f(a)$  tal que:

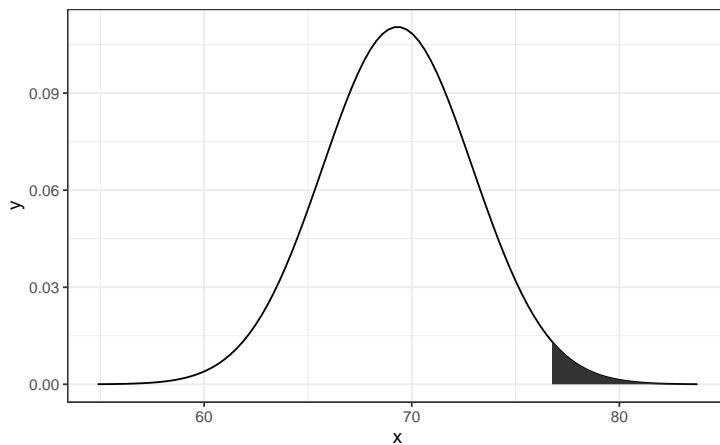
$$F(a) = \Pr(X \leq a) = \int_{-\infty}^a f(x) dx$$

Para aquellos que conocen el cálculo, recuerden que la integral está relacionada con una suma: es la suma de las barras con anchos que se aproximan a 0. Si no conocen el cálculo, pueden pensar en  $f(x)$  como una curva para la cual el área debajo de esa curva hasta el valor  $a$  les da la probabilidad  $\Pr(X \leq a)$ .

Por ejemplo, para usar la aproximación normal para estimar la probabilidad de que alguien sea más alto que 76 pulgadas, usamos:

```
1 - pnorm(76, m, s)
#> [1] 0.0321
```

que matemáticamente es el área gris a continuación:



La curva que ven es la densidad de probabilidad para la distribución normal. En R, obtenemos esto usando la función `dnorm`.

Aunque quizás no sea inmediatamente obvio por qué es útil conocer las densidades de probabilidad, comprender este concepto será esencial para aquellos que quieran ajustar modelos a datos para los que no hay funciones predefinidas disponibles.

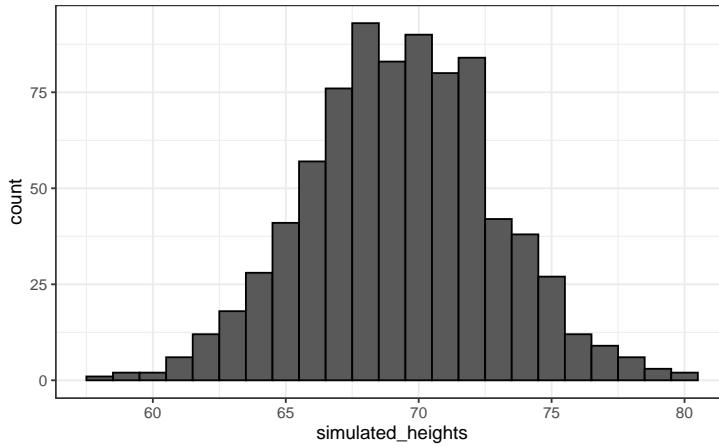
---

## 13.12 Simulaciones Monte Carlo para variables continuas

R provee funciones para generar resultados normalmente distribuidos. Específicamente, la función `rnorm` toma tres argumentos: tamaño, promedio (predeterminado a 0) y desviación estándar (predeterminada a 1) y produce números aleatorios. Aquí tenemos un ejemplo de cómo podríamos generar datos que se parezcan a nuestras alturas:

```
n <- length(x)
m <- mean(x)
s <- sd(x)
simulated_heights <- rnorm(n, m, s)
```

No sorprende que la distribución se vea normal:



Esta es una de las funciones más útiles en R, ya que nos permite generar datos que imitan eventos naturales y responder a preguntas relacionadas con lo que podría suceder por casualidad al ejecutar simulaciones Monte Carlo.

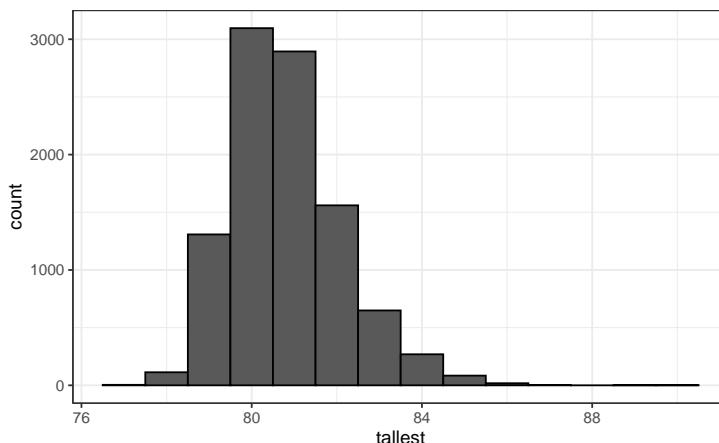
Si, por ejemplo, elegimos 800 hombres al azar, ¿cuál es la distribución de la persona más alta? ¿Cuán raro es un hombre de 7 pies, un *seven footer*, en un grupo de 800 hombres? La siguiente simulación Monte Carlo nos ayuda a responder esa pregunta:

```
B <- 10000
tallest <- replicate(B, {
 simulated_data <- rnorm(800, m, s)
 max(simulated_data)
})
```

Tener un *seven footer* es bastante raro:

```
mean(tallest >= 7*12)
#> [1] 0.0196
```

Aquí vemos la distribución resultante:



Noten que no parece normal.

---

### 13.13 Distribuciones continuas

Presentamos la distribución normal en la Sección 8.8 y se usó como ejemplo anteriormente. La distribución normal no es la única distribución teórica útil. Otras distribuciones continuas que podemos encontrar son t de Student (*Student t* en inglés), chi-cuadrada, exponencial, gamma, beta y beta-binomial. R provee funciones para calcular la densidad, los cuantiles, las funciones de distribución acumulada y para generar simulaciones Monte Carlo. R usa una convención que nos ayuda recordar los nombres: usa las letras **d**, **q**, **p** y **r** delante de una abreviatura del nombre de la distribución. Ya hemos visto las funciones **dnorm**, **pnorm** y **rnorm** para la distribución normal. La función **qnorm** nos da los cuantiles. Por lo tanto, podemos trazar una distribución así:

```
x <- seq(-4, 4, length.out = 100)
qplot(x, f, geom = "line", data = data.frame(x, f = dnorm(x)))
```

Para la distribución t de Student, descrita más adelante en la Sección 16.10, la abreviatura **t** se usa para que las funciones sean **dt** para la densidad, **qt** para los cuantiles, **pt** para la función de distribución acumulada y **rt** para la simulación Monte Carlo.

---

### 13.14 Ejercicios

1. Suponga que la distribución de las alturas femeninas se aproxima por una distribución normal con una media de 64 pulgadas y una desviación estándar de 3 pulgadas. Si elegimos una hembra al azar, ¿cuál es la probabilidad de que mida 5 pies o menos?
2. Suponga que la distribución de las alturas femeninas se aproxima por una distribución normal con una media de 64 pulgadas y una desviación estándar de 3 pulgadas. Si elegimos una hembra al azar, ¿cuál es la probabilidad de que mida 6 pies o más?
3. Suponga que la distribución de las alturas femeninas se aproxima por una distribución normal con una media de 64 pulgadas y una desviación estándar de 3 pulgadas. Si elegimos una hembra al azar, ¿cuál es la probabilidad de que mida entre 61 y 67 pulgadas?
4. Repita el ejercicio anterior, pero convierta todo a centímetros. Es decir, multiplique cada altura, incluyendo la desviación estándar, por 2.54. ¿Cuál es la respuesta ahora?
5. Noten que la respuesta a la pregunta no cambia cuando cambian las unidades. Esto tiene sentido ya que la respuesta a la pregunta no debe ser afectada por las unidades que usamos. De hecho, si se fija bien, verá que 61 y 64 están a 1 SD de distancia del promedio. Calcule la probabilidad de que una variable aleatoria distribuida normal y aleatoriamente esté dentro de 1 SD del promedio.
6. Para ver las matemáticas que explican por qué las respuestas a las preguntas 3, 4 y 5 son las mismas, suponga que tiene una variable aleatoria con promedio  $m$  y error estándar  $s$ . Suponga que quiere saber la probabilidad de que  $X$  sea más pequeña o igual a  $a$ . Recuerda que, por definición,  $a$  está a  $(a - m)/s$  desviaciones estandar  $s$  del promedio  $m$ . La probabilidad es:

$$\Pr(X \leq a)$$

Ahora reste  $\mu$  a ambos lados y luego divide ambos lados por  $\sigma$ :

$$\Pr\left(\frac{X - \mu}{\sigma} \leq \frac{a - \mu}{\sigma}\right)$$

La cantidad a la izquierda es una variable aleatoria con distribución normal unitaria. Tiene un promedio de 0 y un error estándar de 1. Lo llamaremos  $Z$ :

$$\Pr\left(Z \leq \frac{a - \mu}{\sigma}\right)$$

Entonces, sin importar las unidades, la probabilidad de  $X \leq a$  es igual a la probabilidad de que una variable normal unitaria sea menor que  $(a - \mu)/\sigma$ . Si  $m$  es el promedio y  $s$  el error estándar, ¿cuál de los siguientes códigos de R le dará la respuesta correcta en cada situación?

- a. `mean(X<=a)`
- b. `pnorm((a - m)/s)`
- c. `pnorm((a - m)/s, m, s)`
- d. `pnorm(a)`

7. Imagine que la distribución de los hombres adultos es aproximadamente normal con un valor esperado de 69 y una desviación estándar de 3. ¿Cuán alto es el hombre en el percentil 99? Sugerencia: use `qnorm`.

8. La distribución de las puntuaciones de los coeficientes intelectuales, o CI (*IQ* en inglés), se distribuye aproximadamente de manera normal. El promedio es 100 y la desviación estándar es 15. Suponga que desea conocer la distribución de los CI más altos en todas las clases graduadas de cada distrito escolar, cada uno con 10,000 personas. Ejecute una simulación Monte Carlo con  $B=1000$  generando 10,000 puntuaciones de CI y manteniendo los CI más altos. Haga un histograma.

# 14

---

## Variables aleatorias

---

En la ciencia de datos, a menudo trabajamos con datos que se ven afectados de alguna manera por el azar. Algunos ejemplos son datos que provienen de una muestra aleatoria, datos afectados por un error de medición o datos que miden algún resultado que es de naturaleza aleatoria. Ser capaz de cuantificar la incertidumbre introducida por la aleatoriedad es uno de los trabajos más importantes de los analistas de datos. La inferencia estadística ofrece un acercamiento, así como varias herramientas prácticas para hacerlo. El primer paso es aprender a describir matemáticamente variables aleatorias.

En este capítulo, presentamos variables aleatorias y sus propiedades comenzando con su aplicación a juegos de azar. Luego describimos algunos de los eventos que rodearon la crisis financiera de 2007-2008<sup>1</sup> usando la teoría de la probabilidad. Esta crisis financiera fue causada en parte por subestimar el riesgo de ciertos valores<sup>2</sup> vendidos por instituciones financieras. Específicamente, los riesgos de los valores respaldados por hipotecas (MBS o *mortgage-backed securities* en inglés) y las obligaciones de deuda garantizadas (CDO o *collateralized debt obligations* en inglés) se subestimaron enormemente. Estos activos se vendieron a precios que suponían que la mayoría de los propietarios harían sus pagos a tiempo y se calculó como baja la probabilidad de que esto no ocurriera. Una combinación de factores resultó en muchos más incumplimientos de lo esperado, lo que condujo a una caída de los precios de estos valores. Como consecuencia, los bancos perdieron tanto dinero que necesitaron rescates del gobierno para evitar cerrar por completo.

---

### 14.1 Variables aleatorias

Las variables aleatorias son los resultados numéricos de procesos aleatorios. Podemos generar fácilmente variables aleatorias utilizando algunos de los ejemplos anteriores. Por ejemplo, definir X a ser 1 si la cuenta (*bead* en inglés) es azul y 0 de lo contrario:

```
beads <- rep(c("red", "blue"), times = c(2,3))
X <- ifelse(sample(beads, 1) == "blue", 1, 0)
```

Aquí X es una variable aleatoria: cada vez que seleccionamos una nueva cuenta, el resultado cambia aleatoriamente. Vean abajo:

```
ifelse(sample(beads, 1) == "blue", 1, 0)
#> [1] 1
```

<sup>1</sup>[https://en.wikipedia.org/w/index.php?title=Financial\\_crisis\\_of\\_2007%E2%80%932008](https://en.wikipedia.org/w/index.php?title=Financial_crisis_of_2007%E2%80%932008)

<sup>2</sup>[https://en.wikipedia.org/w/index.php?title=Security\\_\(finance\)](https://en.wikipedia.org/w/index.php?title=Security_(finance))

```
ifelse(sample(beads, 1) == "blue", 1, 0)
#> [1] 0
ifelse(sample(beads, 1) == "blue", 1, 0)
#> [1] 0
```

A veces es 1 y a veces es 0.

## 14.2 Modelos de muestreo

Muchos procedimientos de generación de datos, aquellos que producen los datos que estudiamos, pueden modelarse relativamente bien como elecciones de una urna. Por ejemplo, podemos modelar el proceso de sondeo de votantes probables como sacar 0s (republicanos) y 1s (demócratas) de una urna que contiene el código 0 y 1 para todos los votantes probables. En los estudios epidemiológicos, a menudo suponemos que los sujetos de nuestro estudio son una muestra aleatoria de la población de interés. Los datos relacionados con un resultado específico se pueden modelar como una muestra aleatoria de una urna que contiene el resultado para toda la población de interés. De manera similar, en las investigaciones experimentales, a menudo suponemos que los organismos individuales que estamos estudiando, por ejemplo, gusanos, moscas o ratones, son una muestra aleatoria de una población más grande. Los experimentos aleatorios también se pueden modelar mediante selecciones de una urna dada la forma en que los individuos se asignan a grupos: cuando se les asignan, el grupo se escoge al azar. Los modelos de muestreo son, por lo tanto, omnipresentes en la ciencia de datos. Los juegos de casino ofrecen una gran cantidad de ejemplos de situaciones del mundo real en las que se utilizan modelos de muestreo para responder a preguntas específicas. Por lo tanto, comenzaremos con tales ejemplos.

Supongan que un casino muy pequeño los contratan para consultar si deben incluir las ruedas de la ruleta entre sus juegos. Para simplificar el ejemplo, suponemos que jugarán 1,000 personas y que la única apuesta que puedan hacer en la ruleta es apostar en rojo o negro. El casino quiere que predigan cuánto dinero ganarán o perderán. Quieren una gama de valores y, en particular, quieren saber cuál es la probabilidad de perder dinero. Si esta probabilidad es demasiado alta, no instalarán ruedas de ruleta.

Vamos a definir una variable aleatoria  $S$  que representará las ganancias totales del casino. Comencemos por construir la urna. Una rueda de ruleta tiene 18 casillas rojas, 18 casillas negras y 2 verdes. Entonces, jugar un color en un juego de ruleta es equivalente a escoger de la siguiente urna:

```
color <- rep(c("Black", "Red", "Green"), c(18, 18, 2))
```

Los 1,000 resultados de 1,000 personas jugando son sorteos independientes de esta urna. Si aparece el rojo, el jugador gana y el casino pierde un dólar, por lo que sacamos un  $-\$1$ . De otra manera, el casino gana un dólar y sacamos un  $\$1$ . Para construir nuestra variable aleatoria  $S$ , podemos usar este código:

```
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
```

Como sabemos las proporciones de 1s y -1s, podemos generar las elecciones con una línea de código, sin definir `color`:

```
X <- sample(c(-1,1), n, replace = TRUE, prob=c(9/19, 10/19))
```

Llamamos a esto un **modelo de muestreo** ya que estamos modelando el comportamiento aleatorio de la ruleta con el muestreo de elecciones de una urna. Las ganancias totales  $S$  son simplemente la suma de estos 1,000 sorteos independientes:

```
X <- sample(c(-1,1), n, replace = TRUE, prob=c(9/19, 10/19))
S <- sum(X)
S
#> [1] 22
```

### 14.3 La distribución de probabilidad de una variable aleatoria

Si ejecutan el código anterior, verán que  $S$  cambia cada vez. Esto es porque  $S$  es una **variable aleatoria**. La distribución de probabilidad de una variable aleatoria nos dice la probabilidad de que el valor observado caiga en cualquier intervalo dado. Entonces, por ejemplo, si queremos saber la probabilidad de que perdamos dinero, estamos preguntando la probabilidad de que  $S$  esté en el intervalo  $S < 0$ .

Tengan en cuenta que si podemos definir una función de distribución acumulativa  $F(a) = \Pr(S \leq a)$ , entonces podremos responder a cualquier pregunta relacionada con la probabilidad de eventos definidos por nuestra variable aleatoria  $S$ , incluyendo el evento  $S < 0$ . A esta  $F$  le decimos la *función de distribución* de la variable aleatoria.

Podemos estimar la función de distribución para la variable aleatoria  $S$  mediante el uso de una simulación Monte Carlo para generar muchas realizaciones de la variable aleatoria. Con este código, ejecutamos el experimento de tener 1,000 personas jugando a la ruleta, una y otra vez, específicamente  $B = 10,000$  veces:

```
n <- 1000
B <- 10000
roulette_winnings <- function(n){
 X <- sample(c(-1,1), n, replace = TRUE, prob=c(9/19, 10/19))
 sum(X)
}
S <- replicate(B, roulette_winnings(n))
```

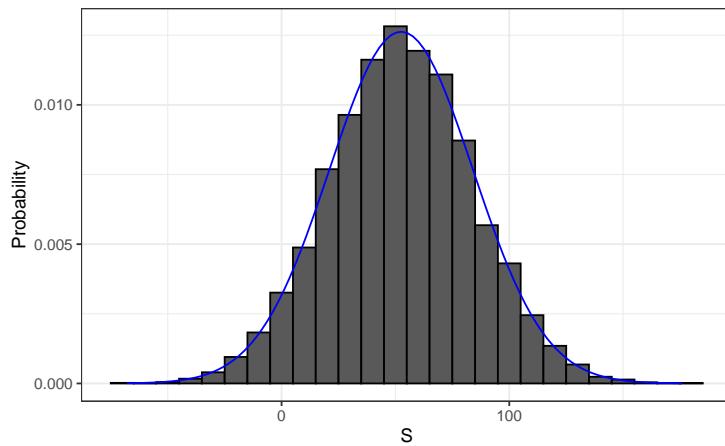
Ahora podemos preguntar lo siguiente: ¿en nuestras simulaciones, con qué frecuencia obtuvimos sumas menores o iguales a  $a$ ?

```
mean(S <= a)
```

Esta será una muy buena aproximación de  $F(a)$  y podemos responder fácilmente a la pregunta del casino: ¿cuán probable es que perdamos dinero? Podemos ver que la probabilidad es bastante baja:

```
mean(S<0)
#> [1] 0.0456
```

Podemos visualizar la distribución de  $S$  creando un histograma que muestra la probabilidad  $F(b) - F(a)$  para varios intervalos  $(a, b]$ :



Vemos que la distribución parece ser aproximadamente normal. Un gráfico Q-Q confirmará que la aproximación normal está cerca de una aproximación perfecta para esta distribución. De hecho, si la distribución es normal, entonces todo lo que necesitamos para definir la distribución es el promedio y la desviación estándar. Debido a que tenemos los valores originales a partir de los cuales se crea la distribución, podemos calcularlos fácilmente con `mean(S)` y `sd(S)`. La curva azul que se añade al histograma anterior es una densidad normal con este promedio y desviación estándar.

Este promedio y esta desviación estándar tienen nombres especiales. Se les conoce como *valor esperado* y *error estándar* de la variable aleatoria  $S$  y se discutirán más en la siguiente sección.

La teoría estadística provee una forma de derivar la distribución de variables aleatorias definidas como extracciones aleatorias independientes de una urna. Específicamente, en nuestro ejemplo anterior, podemos mostrar que  $(S + n)/2$  sigue una distribución binomial. Por lo tanto, no necesitamos ejecutar simulaciones Monte Carlo para saber la distribución de probabilidad de  $S$ . Lo hicimos con fines ilustrativos.

Podemos usar las funciones `dbinom` y `pbinom` para calcular las probabilidades exactamente. Por ejemplo, para calcular  $\Pr(S < 0)$  notamos que:

$$\Pr(S < 0) = \Pr((S + n)/2 < (0 + n)/2)$$

y podemos usar `pbinom` para calcular:

$$\Pr(S \leq 0)$$

```
n <- 1000
pbinom(n/2, size = n, prob = 10/19)
#> [1] 0.0511
```

Debido a que esta es una función de probabilidad discreta, para obtener  $\Pr(S < 0)$  en vez de  $\Pr(S \leq 0)$ , escribimos:

```
pbinom(n/2-1, size = n, prob = 10/19)
#> [1] 0.0448
```

Para más detalles sobre la distribución binomial, pueden consultar cualquier libro de probabilidad básico o incluso Wikipedia<sup>3</sup>.

Aquí no exploramos esos detalles. En cambio, discutiremos una aproximación increíblemente útil que nos da la teoría matemática que se aplica generalmente a sumas y promedios de sorteos de cualquier urna: el *teorema del límite central* (*Central Limit Theorem* o CLT por sus siglas en inglés).

## 14.4 Distribuciones versus distribuciones de probabilidad

Antes de continuar, hagamos una distinción y una conexión importante entre la distribución de una lista de números y una distribución de probabilidad. En el capítulo de visualización, describimos cómo cualquier lista de números  $x_1, \dots, x_n$  tiene una distribución. La definición es bastante sencilla. Definimos  $F(a)$  como la función que nos dice qué proporción de la lista es menor o igual a  $a$ . Debido a que son resúmenes útiles cuando la distribución es aproximadamente normal, definimos el promedio y la desviación estándar. Estos se definen con una operación sencilla del vector que contiene la lista de números  $x$ :

```
m <- sum(x)/length(x)
s <- sqrt(sum((x - m)^2) / length(x))
```

Una variable aleatoria  $X$  tiene una función de distribución. Para definir esto, no necesitamos una lista de números. Es un concepto teórico. En este caso, definimos la distribución como la  $F(a)$  que responde a la pregunta: ¿cuál es la probabilidad de que  $X$  sea menor o igual que  $a$ ? No hay una lista de números.

Sin embargo, si  $X$  se define como una selección de una urna con números en ella, entonces hay una lista: la lista de números dentro de la urna. En este caso, la distribución de esa lista es la distribución de probabilidad de  $X$  y el promedio y la desviación estándar de esa lista son el valor esperado y el error estándar de la variable aleatoria.

Otra forma de pensar en esto que no involucra una urna es ejecutar una simulación Monte

<sup>3</sup>[https://en.wikipedia.org/w/index.php?title=Binomial\\_distribution](https://en.wikipedia.org/w/index.php?title=Binomial_distribution)

Carlo y generar una lista muy grande de resultados de  $X$ . Estos resultados son una lista de números. La distribución de esta lista será una muy buena aproximación de la distribución de probabilidad de  $X$ . Cuanto más larga sea la lista, mejor será la aproximación. El promedio y la desviación estándar de esta lista se aproximarán al valor esperado y al error estándar de la variable aleatoria.

---

## 14.5 Notación para variables aleatorias

En los libros de texto estadísticos, las letras mayúsculas se usan para denotar variables aleatorias y seguimos esta convención aquí. Se usan letras minúsculas para los valores observados. Verán alguna notación que incluye ambos. Por ejemplo, verán eventos definidos como  $X \leq x$ . Aquí  $X$  es una variable aleatoria, por lo que es un evento aleatorio, y  $x$  es un valor arbitrario y no aleatorio. Así, por ejemplo,  $X$  podría representar el número en un dado y  $x$  representaría un valor real que vemos: 1, 2, 3, 4, 5 o 6. Entonces, en este caso, la probabilidad de  $X = x$  es  $1/6$  independientemente del valor observado  $x$ . Esta notación es un poco extraña porque, cuando hacemos preguntas sobre probabilidad,  $X$  no es una cantidad observada, sino una cantidad aleatoria que veremos en el futuro. Podemos describir lo que esperamos ver, qué valores son probables, pero no lo qué es. Pero tan pronto tengamos datos, vemos una realización de  $X$ . Entonces, los científicos de datos hablan de lo que pudo haber sido después de ver lo que realmente sucedió.

---

## 14.6 El valor esperado y el error estándar

Hemos descrito modelos de muestreo para sorteos. Ahora repasaremos la teoría matemática que nos permite aproximar las distribuciones de probabilidad para la suma de los sorteos. Una vez que hagamos esto, podremos ayudar al casino a predecir cuánto dinero ganarán. El mismo enfoque que usamos para la suma de los sorteos será útil para describir la distribución de promedios y la proporción que necesitaremos para entender cómo funcionan las encuestas.

El primer concepto importante para aprender es el *valor esperado* (*expected value* en inglés). En los libros de estadísticas, es común usar la letra E así:

$$\mathrm{E}[X]$$

para denotar el valor esperado de la variable aleatoria  $X$ .

Una variable aleatoria variará alrededor de su valor esperado de una manera que si toman el promedio de muchos, muchos sorteos, el promedio de los sorteos se aproximarán al valor esperado, acercándose cada vez más mientras aumentan los sorteos.

La estadística teórica ofrece técnicas que facilitan el cálculo de los valores esperados en diferentes circunstancias. Por ejemplo, una fórmula útil nos dice que el *valor esperado de una variable aleatoria definida por un sorteo es el promedio de los números en la urna*. En la urna que usamos para modelar las apuestas al rojo en la ruleta, tenemos 20 dólares y 18 dólares negativos. El valor esperado entonces es:

$$E[X] = (20 + -18)/38$$

que es como 5 centavos. Es un poco contradictorio decir que  $X$  varía alrededor de 0.05, cuando los únicos valores que toma son 1 y -1. Una manera de entender el valor esperado en este contexto es darse cuenta de que si jugamos el juego una y otra vez, el casino gana, en promedio, 5 centavos por juego. Una simulación Monte Carlo confirma esto:

```
B <- 10^6
x <- sample(c(-1,1), B, replace = TRUE, prob=c(9/19, 10/19))
mean(x)
#> [1] 0.0517
```

En general, si la urna tiene dos resultados posibles, digamos  $a$  y  $b$ , con proporciones  $p$  y  $1 - p$  respectivamente, el promedio es:

$$E[X] = ap + b(1 - p)$$

Para ver esto, observen que si hay  $n$  cuentas en la urna, entonces tenemos  $np$  as y  $n(1 - p)$  bs, y como el promedio es la suma,  $n \times a \times p + n \times b \times (1 - p)$ , dividido por el total  $n$ , obtenemos que el promedio es  $ap + b(1 - p)$ .

Ahora bien, la razón por la que definimos el valor esperado es porque esta definición matemática resulta útil para aproximar las distribuciones de probabilidad de la suma, que luego es útil para describir la distribución de promedios y proporciones. El primer hecho útil es que el *valor esperado de la suma de los sorteos* es:

number of draws  $\times$  average of the numbers in the urn

Entonces si 1,000 personas juegan a la ruleta, el casino espera ganar, en promedio, alrededor de  $1,000 \times \$0.05 = \$50$ . Pero este es un valor esperado. ¿Cuán diferente puede ser una observación del valor esperado? El casino realmente necesita saber esto. ¿Cuál es el rango de probabilidades? Si los números negativos son muy probables, no instalarán las ruedas de ruleta. La teoría estadística una vez más responde a esta pregunta. El *error estándar* (SE por sus siglas en inglés) nos da una idea del tamaño de la variación alrededor del valor esperado. En los libros de estadísticas, es común usar:

$$SE[X]$$

para denotar el error estándar de una variable aleatoria.

**Si nuestros sorteos son independientes**, entonces el *error estándar de la suma* lo da la ecuación:

$$\sqrt{\text{number of draws}} \times \text{standard deviation of the numbers in the urn}$$

Usando la definición de desviación estándar, podemos derivar, con un poco de matemática, que si una urna contiene dos valores  $a$  y  $b$  con proporciones  $p$  y  $(1 - p)$ , respectivamente, la desviación estándar es:

$$| b - a | \sqrt{p(1-p)}.$$

Entonces en nuestro ejemplo de ruleta, la desviación estándar de los valores dentro de la urna es:  $| 1 - (-1) | \sqrt{10/19 \times 9/19}$  o:

```
2 * sqrt(90)/19
#> [1] 0.999
```

El error estándar nos dice la diferencia típica entre una variable aleatoria y su expectativa. Dado que un sorteo es obviamente la suma de un solo sorteo, podemos usar la fórmula anterior para calcular que la variable aleatoria definida por un sorteo tiene un valor esperado de 0.05 y un error estándar de aproximadamente 1. Esto tiene sentido ya que obtenemos 1 o -1, con 1 ligeramente favorecido sobre -1.

Usando la fórmula anterior, la suma de 1,000 personas jugando tiene un error estándar de aproximadamente \$32:

```
n <- 1000
sqrt(n) * 2 * sqrt(90)/19
#> [1] 31.6
```

Como resultado, cuando 1,000 personas apuestan al rojo, se espera que el casino gane \$50 con un error estándar de \$32. Por lo tanto, parece una apuesta segura. Pero aún no hemos respondido a la pregunta: ¿qué probabilidades hay de perder dinero? Aquí el CLT nos ayudará.

**Nota avanzada:** Antes de continuar, debemos señalar que los cálculos exactos de probabilidad de las ganancias del casino se pueden realizar con la distribución binomial. Sin embargo, aquí nos enfocamos en el CLT, que generalmente se puede aplicar a sumas de variables aleatorias, algo que no se puede hacer con la distribución binomial.

#### 14.6.1 Población SD versus la muestra SD

La desviación estándar (SD) de una lista `x` (a continuación usamos alturas como ejemplo) se define como la raíz cuadrada del promedio de las diferencias cuadradas:

```
library(dslabs)
x <- heights$height
m <- mean(x)
s <- sqrt(mean((x-m)^2))
```

Usando notación matemática escribimos:

$$\mu = \frac{1}{n} \sum_{i=1}^n x_i \sigma = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \mu)^2}$$

Sin embargo, tengan en cuenta que la función `sd` devuelve un resultado ligeramente diferente:

```
identical(s, sd(x))
#> [1] FALSE
s-sd(x)
#> [1] -0.00194
```

Esto es porque la función `sd` en R no devuelve el `sd` de la lista, sino que utiliza una fórmula que estima las desviaciones estándar de la población usando una muestra aleatoria  $X_1, \dots, X_N$  que, por razones no discutidas aquí, divide la suma de cuadrados por  $N - 1$ .

$$\bar{X} = \frac{1}{N} \sum_{i=1}^N X_i, \quad s = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (X_i - \bar{X})^2}$$

Pueden ver que este es el caso escribiendo:

```
n <- length(x)
s-sd(x)*sqrt((n-1)/ n)
#> [1] 0
```

Para toda la teoría discutida aquí, deben calcular la desviación estándar real como se define:

```
sqrt(mean((x-m)^2))
```

Así que tengan cuidado al usar la función `sd` en R. Sin embargo, tengan en cuenta que a lo largo del libro a veces usamos la función `sd` cuando realmente queremos la SD real. Esto se debe a que cuando el tamaño de la lista es grande, estos dos son prácticamente equivalentes ya que  $\sqrt{(N-1)/N} \approx 1$ .

## 14.7 Teorema del límite central

El teorema del límite central (CLT) nos dice que cuando el número de sorteos, también llamado *tamaño de muestra*, es grande, la distribución de probabilidad de la suma de los sorteos independientes es aproximadamente normal. Debido a que los modelos de muestreo se utilizan para tantos procesos de generación de datos, el CLT se considera una de las ideas matemáticas más importantes de la historia.

Anteriormente, discutimos que si sabemos que la distribución de una lista de números se aproxima a la distribución normal, lo único que necesitamos para describir la lista es el promedio y la desviación estándar. También sabemos que lo mismo se aplica a las distribuciones de probabilidad. Si una variable aleatoria tiene una distribución de probabilidad que se aproxima a la distribución normal, entonces todo lo que necesitamos para describir la distribución de probabilidad son el promedio y la desviación estándar, lo que se conoce como el valor esperado y el error estándar.

Anteriormente ejecutamos esta simulación Monte Carlo:

```
n <- 1000
B <- 10000
roulette_winnings <- function(n){
 X <- sample(c(-1,1), n, replace = TRUE, prob=c(9/19, 10/19))
 sum(X)
}
S <- replicate(B, roulette_winnings(n))
```

El CLT nos dice que la suma  $S$  se aproxima por una distribución normal. Usando las fórmulas anteriores, sabemos que el valor esperado y el error estándar son:

```
n * (20-18)/38
#> [1] 52.6
sqrt(n) * 2 * sqrt(90)/19
#> [1] 31.6
```

Los valores teóricos anteriores coinciden con los obtenidos con la simulación Monte Carlo:

```
mean(S)
#> [1] 52.2
sd(S)
#> [1] 31.7
```

Usando el CLT, podemos omitir la simulación Monte Carlo y en su lugar calcular la probabilidad de que el casino pierda dinero usando esta aproximación:

```
mu <- n * (20-18)/38
se <- sqrt(n) * 2 * sqrt(90)/19
pnorm(0, mu, se)
#> [1] 0.0478
```

que también concuerda muy bien con nuestro resultado de la simulación Monte Carlo:

```
mean(S < 0)
#> [1] 0.0458
```

### 14.7.1 ¿Cuán grande es grande en el teorema del límite central?

El CLT funciona cuando el número de sorteos es grande. Pero grande es un término relativo. En muchas circunstancias, solo 30 sorteos son suficientes para que el CLT sea útil. En algunos casos específicos, solo 10 son suficientes. Sin embargo, estas no deben considerarse reglas generales. Por ejemplo, cuando la probabilidad de éxito es muy pequeña, necesitamos tamaños de muestra mucho más grandes.

A modo de ilustración, consideremos la lotería. En la lotería, la probabilidad de ganar son menos de 1 en un millón. Miles de personas juegan, por lo que el número de sorteos es muy grande. Sin embargo, el número de ganadores, la suma de los sorteos, oscila entre 0 y 4. La distribución normal no es una buena aproximación de la suma, por lo cual el CLT no

aplica, incluso cuando el tamaño de la muestra es muy grande. Esto es generalmente cierto cuando la probabilidad de éxito es muy baja. En estos casos, la distribución de Poisson es más apropiada.

Pueden examinar las propiedades de la distribución de Poisson usando `dpois` y `ppois`. Pueden generar variables aleatorias siguiendo esta distribución con `rpois`. Sin embargo, no discutimos la teoría aquí. Para aprender más sobre la distribución de Poisson, pueden consultar cualquier libro de texto de probabilidad e incluso Wikipedia<sup>4</sup>

## 14.8 Propiedades estadísticas de promedios

Hay varios resultados matemáticos útiles que usamos anteriormente y que a menudo empleamos al trabajar con datos. Los enumeramos a continuación.

1. El valor esperado de la suma de variables aleatorias es la suma del valor esperado de cada variable aleatoria. Podemos escribirlo así:

$$\mathrm{E}[X_1 + X_2 + \dots + X_n] = \mathrm{E}[X_1] + \mathrm{E}[X_2] + \dots + \mathrm{E}[X_n]$$

Si los  $X$  son sorteos independientes de la urna, entonces todos tienen el mismo valor esperado. Vamos a llamarlo  $\mu$  y por ende:

$$\mathrm{E}[X_1 + X_2 + \dots + X_n] = n\mu$$

que es otra forma de escribir el resultado que mostramos arriba para la suma de los sorteos.

2. El valor esperado de una constante no aleatoria multiplicada por una variable aleatoria es la constante no aleatoria multiplicada por el valor esperado de una variable aleatoria. Esto es más fácil de explicar con símbolos:

$$\mathrm{E}[aX] = a \times \mathrm{E}[X]$$

Para ver por qué esto es intuitivo, consideren el cambio de unidades. Si cambiamos las unidades de una variable aleatoria, digamos de dólares a centavos, la expectativa debería cambiar de la misma manera. Una consecuencia de los dos hechos anteriores es que el valor esperado del promedio de extracciones independientes de la misma urna es el valor esperado de la urna, llámelo  $\mu$  de nuevo:

$$\mathrm{E}[(X_1 + X_2 + \dots + X_n)/n] = \mathrm{E}[X_1 + X_2 + \dots + X_n]/n = n\mu/n = \mu$$

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. Esto es más fácil de entender en forma matemática:

$$\mathrm{SE}[X_1 + X_2 + \dots + X_n] = \sqrt{\mathrm{SE}[X_1]^2 + \mathrm{SE}[X_2]^2 + \dots + \mathrm{SE}[X_n]^2}$$

<sup>4</sup>[https://en.wikipedia.org/w/index.php?title=Poisson\\_distribution](https://en.wikipedia.org/w/index.php?title=Poisson_distribution)

El cuadrado del error estándar se denomina *varianza* en los libros de texto estadísticos. Tengan en cuenta que esta propiedad en particular no es tan intuitiva como las dos anteriores y que pueden encontrar explicaciones detalladas en los libros de texto de estadísticas.

4. El error estándar de una constante no aleatoria multiplicada por una variable aleatoria es la constante no aleatoria multiplicada por el error estándar de la variable aleatoria. Igual que para el valor esperado:

$$\text{SE}[aX] = a \times \text{SE}[X]$$

Para ver por qué esto es intuitivo, piensen nuevamente en las unidades.

Una consecuencia de 3 y 4 es que el error estándar del promedio de sorteos independientes de la misma urna es la desviación estándar de la urna dividida por la raíz cuadrada de  $n$  (el número de sorteos), llámenlo  $\sigma$ :

$$\begin{aligned}\text{SE}[(X_1 + X_2 + \dots + X_n)/n] &= \text{SE}[X_1 + X_2 + \dots + X_n]/n \\ &= \sqrt{\text{SE}[X_1]^2 + \text{SE}[X_2]^2 + \dots + \text{SE}[X_n]^2}/n \\ &= \sqrt{\sigma^2 + \sigma^2 + \dots + \sigma^2}/n \\ &= \sqrt{n\sigma^2}/n \\ &= \sigma/\sqrt{n}\end{aligned}$$

5. Si  $X$  es una variable aleatoria normalmente distribuida, entonces si  $a$  y  $b$  son constantes no aleatorias,  $aX + b$  también es una variable aleatoria normalmente distribuida. Lo único que estamos haciendo es cambiando las unidades de la variable aleatoria multiplicando por  $a$  y entonces desplazando el centro por  $b$ .

Recuerden que los libros de texto de estadísticas usan las letras griegas  $\mu$  y  $\sigma$  para denotar el valor esperado y el error estándar, respectivamente. Esto es porque  $\mu$  es la letra griega para  $m$ , la primera letra de *mean*, que es otro término utilizado para el valor esperado. Asimismo,  $\sigma$  es la letra griega para  $s$ , la primera letra de *standard error*.

## 14.9 Ley de los grandes números

Una implicación importante del resultado final es que el error estándar del promedio se vuelve más y más pequeño a medida que  $n$  se hace más grande. Cuando  $n$  es muy grande, entonces el error estándar es prácticamente 0 y el promedio de los sorteos converge al promedio de la urna. Esto se conoce en los libros de texto estadísticos como la ley de los grandes números o la ley de los promedios.

### 14.9.1 Malinterpretando la ley de promedios

La ley de los promedios a veces se malinterpreta. Por ejemplo, si tiran una moneda 5 veces y sale cara cada vez, es posible que alguien argumente que el próximo lanzamiento probablemente sea cruz debido a la ley de los promedios: en promedio, deberíamos ver 50% cara y 50% cruz. Un argumento similar sería decir que “le toca” al rojo en la rueda de la ruleta después de ver que el negro aparece cinco veces corridas. Estos eventos son independientes,

por lo que la probabilidad de que una moneda salga cara es 50% independientemente de los 5 resultados anteriores. Este también es el caso del resultado de la ruleta. La ley de promedios se aplica solo cuando el número de sorteos es muy grande y no en muestras pequeñas. Después de un millón de lanzamientos, definitivamente verán alrededor del 50% caras independientemente del resultado de los primeros cinco lanzamientos.

Otro mal uso interesante de la ley de los promedios es en los deportes cuando los presentadores de televisión predicen que un jugador está a punto de triunfar porque ha fallado varias veces seguidas.

## 14.10 Ejercicios

1. En la ruleta americana también se puede apostar al verde. Hay 18 rojos, 18 negros y 2 verdes (0 y 00). ¿Cuáles son las probabilidades de que salga un verde?

2. El pago por ganar en verde es \$17. Esto significa que si apuesta un dólar y cae en verde, obtiene \$17. Cree un modelo de muestreo usando una muestra para simular la variable aleatoria  $X$  de sus ganancias. Sugerencia: vea el ejemplo a continuación del código para apostar en rojo.

```
x <- sample(c(1,-1), 1, prob = c(9/19, 10/19))
```

3. ¿Cuál es el valor esperado de  $X$ ?

4. ¿Cuál es el error estándar de  $X$ ?

5. Ahora cree una variable aleatoria  $S$  que es la suma de sus ganancias después de apostar en verde 1,000 veces. Sugerencia: cambie los argumentos `size` y `replace` en su respuesta a la pregunta 2. Comience su código fijando la semilla en 1 con `set.seed(1)`.

6. ¿Cuál es el valor esperado de  $S$ ?

7. ¿Cuál es el error estándar de  $S$ ?

8. ¿Cuál es la probabilidad de que termine ganando dinero? Sugerencia: use el CLT.

9. Cree una simulación Monte Carlo que genere 1,000 resultados de  $S$ . Calcule el promedio y la desviación estándar de la lista resultante para confirmar los resultados de 6 y 7. Comience su código fijando la semilla en 1 con `set.seed(1)`.

10. Ahora verifique su respuesta a la pregunta 8 usando el resultado de la simulación Monte Carlo.

11. El resultado de la simulación Monte Carlo y la aproximación CLT están cerca, pero no tan cerca. ¿Qué podría explicar esto?

- a. 1,000 simulaciones no son suficientes. Si hacemos más, coinciden.
- b. El CLT no funciona tan bien cuando la probabilidad de éxito es pequeña. En este caso, fue 1/19. Si hacemos que el número de juegos de ruleta sea mayor, coincidirán mejor.
- c. La diferencia está dentro del error de redondeo.
- d. El CLT solo funciona para promedios.

12. Ahora cree una variable aleatoria  $Y$  que sea su promedio de ganancias por apuesta tras apostar al verde 1,000 veces.
  13. ¿Cuál es el valor esperado de  $Y$ ?
  14. ¿Cuál es el error estándar de  $Y$ ?
  15. ¿Cuál es la probabilidad de que cuando acabe de jugar, las ganancias por juego sean positivas? Sugerencia: use el CLT.
  16. Cree una simulación Monte Carlo que genere 2,500 resultados de  $Y$ . Calcule el promedio y la desviación estándar de la lista resultante para confirmar los resultados de 6 y 7. Comience su código fijando la semilla en 1 con `set.seed(1)`.
  17. Ahora verifique su respuesta a 8 usando el resultado de la simulación Monte Carlo.
  18. El resultado de la simulación Monte Carlo y la aproximación CLT ahora están mucho más cerca. ¿Qué podría explicar esto?
    - a. Ahora estamos calculando promedios en lugar de sumas.
    - b. 2,500 simulaciones de Monte Carlo no son mejores que 1,000.
    - c. El CLT funciona mejor cuando el tamaño de la muestra es mayor. Aumentamos de 1,000 a 2,500.
    - d. No está más cerca. La diferencia está dentro del error de redondeo.
- 

## 14.11 Estudio de caso: *The Big Short*

### 14.11.1 Tasas de interés explicadas con modelo de oportunidad

Los bancos también usan versiones más complejas de los modelos de muestreo que hemos discutido para determinar sus tasas de interés. Supongan que compran un banco pequeño que tiene un historial de identificar posibles propietarios de viviendas en los que se pueden confiar para realizar pagos. De hecho, históricamente, en un año determinado, solo el 2% de sus clientes no pagan el dinero que se les prestó. Sin embargo, el banco sabe que si simplemente le prestan dinero a todos sus clientes sin intereses, terminará perdiendo dinero debido a este 2%. Aunque el banco sabe que el 2% de sus clientes probablemente no pagarán, no sabe cuáles son esos. Pero al cobrarles a todos un poco más en intereses, pueden compensar las pérdidas incurridas debido a ese 2% y también cubrir sus costos operativos. También pueden obtener ganancias, aunque si establecen tasas de interés demasiado altas, los clientes se irán a otro banco. Utilizaremos todos estos hechos y un poco de teoría de probabilidad para determinar qué tasa de interés deben cobrar.

Supongan que su banco otorgará 1,000 préstamos de \$180,000 este año. Además, tras sumar todos los costos, supongan que su banco pierde \$200,000 por ejecución hipotecaria. Para simplificar, suponemos que esto incluye todos los costos operativos. Un modelo de muestreo para este escenario puede codificarse así:

```
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] -5800000
```

Tengan en cuenta que la pérdida total definida por la suma final es una variable aleatoria. Cada vez que ejecutan el código anterior, obtienen una respuesta diferente. Podemos construir fácilmente una simulación Monte Carlo para tener una idea de la distribución de esta variable aleatoria.

```
B <- 10000
losses <- replicate(B, {
 defaults <- sample(c(0,1), n, prob=c(1-p, p), replace = TRUE)
 sum(defaults * loss_per_foreclosure)
})
```

Realmente no necesitamos una simulación Monte Carlo. Usando lo que hemos aprendido, el CLT nos dice que, debido a que las pérdidas son una suma de sorteos independientes, su distribución es aproximadamente normal con el valor esperado y los errores estándar dados por:

```
n*(p*loss_per_foreclosure + (1-p)*0)
#> [1] -4e+06
sqrt(n)*abs(loss_per_foreclosure)*sqrt(p*(1-p))
#> [1] 885438
```

Ahora podemos establecer una tasa de interés para garantizar que, como promedio, lleguen a un punto de equilibrio. Básicamente, necesitan añadir una cantidad  $x$  a cada préstamo, que en este caso están representados por sorteos, de modo que el valor esperado sea 0. Si definen  $l$  para ser la pérdida por ejecución hipotecaria, necesitan:

$$lp + x(1 - p) = 0$$

que implica  $x$  es:

```
- loss_per_foreclosure*p/(1-p)
#> [1] 4082
```

o una tasa de interés de 0.023.

Sin embargo, todavía tenemos un problema. Aunque esta tasa de interés garantiza que, como promedio, lleguen a un punto de equilibrio, existe una probabilidad del 50% de que pierdan dinero. Si su banco pierde dinero, tendrán que cerrarlo. Por lo tanto, deben elegir una tasa de interés que los protega de esto. Al mismo tiempo, si la tasa de interés es demasiado alta, sus clientes se irán a otro banco, por lo que deben estar dispuestos a asumir algunos riesgos. Entonces, digamos que quieren que sus probabilidades de perder dinero sean de 1 en 100, entonces ¿qué cantidad debe ser  $x$  ahora? Esto es un poco más difícil. Queremos que la suma  $S$  tenga:

$$\Pr(S < 0) = 0.01$$

Sabemos que  $S$  es aproximadamente normal. El valor esperado de  $S$  es:

$$\text{E}[S] = \{lp + x(1 - p)\}n$$

con  $n$  el número de selecciones, que en este caso representa préstamos. El error estándar es:

$$\text{SD}[S] = |x - l| \sqrt{np(1 - p)}.$$

Porque  $x$  es positivo y  $l$  negativo  $|x - l| = x - l$ . Recuerden que estas son solo una aplicación de las fórmulas mostradas anteriormente, pero que usan símbolos más compactos.

Ahora vamos a utilizar un “truco” matemático que es muy común en las estadísticas. Sumamos y restamos las mismas cantidades a ambos lados del evento  $S < 0$  para que la probabilidad no cambie y terminemos con una variable aleatoria con distribución normal unitaria a la izquierda, que luego nos permitirá escribir una ecuación con solo  $x$  como un desconocido. Este “truco” es el siguiente:

Si  $\text{Pr}(S < 0) = 0.01$ , entonces:

$$\text{Pr}\left(\frac{S - \text{E}[S]}{\text{SE}[S]} < \frac{-\text{E}[S]}{\text{SE}[S]}\right)$$

Y recuerden que  $\text{E}[S]$  y  $\text{SE}[S]$  son el valor esperado y el error estándar de  $S$ , respectivamente. Lo único que hicimos arriba fue sumar y dividir por la misma cantidad en ambos lados. Hicimos esto porque ahora el término de la izquierda es una variable aleatoria con distribución normal unitaria, a la que le cambiaremos el nombre a  $Z$ . Ahora completamos los espacios en blanco con la fórmula actual para el valor esperado y el error estándar:

$$\text{Pr}\left(Z < \frac{-\{lp + x(1 - p)\}n}{(x - l)\sqrt{np(1 - p)}}\right) = 0.01$$

Puede parecer complicado, pero recuerden que  $l$ ,  $p$  y  $n$  son todas cantidades conocidas, por lo que eventualmente las reemplazaremos con números.

Ahora, como  $Z$  es una variable aleatoria normal con valor esperado 0 y error estándar 1, significa que la cantidad en el lado derecho del signo  $<$  debe ser igual a:

```
qnorm(0.01)
#> [1] -2.33
```

para que la ecuación sea cierta. Recuerden que  $z = \text{qnorm}(0.01)$  nos da el valor de  $z$  para cual:

$$\text{Pr}(Z \leq z) = 0.01$$

Esto significa que el lado derecho de la ecuación complicada debe ser  $z = \text{qnorm}(0.01)$ :

$$\frac{-\{lp + x(1 - p)\}n}{(x - l)\sqrt{np(1 - p)}} = z$$

El truco funciona porque terminamos con una expresión que contiene  $x$ , que sabemos que tiene que ser igual a una cantidad conocida  $z$ . Ahora, resolver para  $x$  es simplemente álgebra:

$$x = -l \frac{np - z\sqrt{np(1-p)}}{n(1-p) + z\sqrt{np(1-p)}}$$

que es:

```
l <- 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
```

Su tasa de interés ahora sube a 0.035. Esta sigue siendo una tasa de interés muy competitiva. Al elegir esa tasa de interés, ahora tendrán una ganancia esperada por préstamo de:

```
loss_per_foreclosure*p + x*(1-p)
#> [1] 2124
```

que es una ganancia total esperada de aproximadamente:

```
n*(loss_per_foreclosure*p + x*(1-p))
#> [1] 2124198
```

dólares!

Podemos ejecutar una simulación Monte Carlo para verificar nuestras aproximaciones teóricas:

```
B <- 100000
profit <- replicate(B, {
 draws <- sample(c(x, loss_per_foreclosure), n,
 prob=c(1-p, p), replace = TRUE)
 sum(draws)
})
mean(profit)
#> [1] 2125441
mean(profit<0)
#> [1] 0.0133
```

### 14.11.2 The Big Short

Uno de sus empleados señala que, dado que el banco está ganando 2,124 dólares por préstamo, ¡el banco debería otorgar más préstamos! ¿Por qué solo  $n$ ? Ustedes explican que encontrar esos  $n$  clientes fue difícil. Necesitan un grupo que sea predecible y que mantenga bajas las probabilidades de incumplimiento. Su empleado entonces señala que aún si la probabilidad de incumplimiento es mayor, siempre que el valor esperado sea positivo, el banco

puede minimizar sus probabilidades de pérdidas al aumentar  $n$  y confiar en la ley de grandes números.

Si empleado además afirma que incluso si la tasa predeterminada es el doble, digamos 4%, si establecen la tasa un poco más alta que este valor:

```
p <- 0.04
r <- (- loss_per_foreclosure*p/(1-p))/ 180000
r
#> [1] 0.0463
```

el banco se beneficiará. Al 5%, se garantizan un valor positivo esperado de:

```
r <- 0.05
x <- r*180000
loss_per_foreclosure*p + x * (1-p)
#> [1] 640
```

y pueden minimizar sus probabilidades de perder dinero simplemente aumentando  $n$  ya que:

$$\Pr(S < 0) = \Pr\left(Z < -\frac{\text{E}[S]}{\text{SE}[S]}\right)$$

con  $Z$  una variable aleatoria con distribución normal unitaria como se muestra anteriormente. Si definimos  $\mu$  y  $\sigma$  como el valor esperado y la desviación estándar, respectivamente, de la urna (es decir, de un solo préstamo), usando las fórmulas anteriores tenemos:  $\text{E}[S] = n\mu$  y  $\text{SE}[S] = \sqrt{n}\sigma$ . Entonces, si definimos  $z = \text{qnorm}(0.01)$ , tenemos:

$$-\frac{n\mu}{\sqrt{n}\sigma} = -\frac{\sqrt{n}\mu}{\sigma} = z$$

lo que implica que si dejamos:

$$n \geq z^2 \sigma^2 / \mu^2$$

tenemos garantizada una probabilidad de menos de 0.01. La implicación es que, siempre y cuando  $\mu$  sea positivo, podemos encontrar una  $n$  que minimiza la probabilidad de una pérdida. Esta es una versión de la ley de los grandes números: cuando  $n$  es grande, nuestras ganancias promedio por préstamo convergen a la ganancia esperada  $\mu$ .

Con  $x$  fijo, ahora podemos preguntar ¿qué  $n$  necesitamos para que la probabilidad sea 0.01? En nuestro ejemplo, si reparten:

```
z <- qnorm(0.01)
n <- ceiling((z^2*(x-1)^2*p*(1-p))/(1*p + x*(1-p))^2)
n
#> [1] 22163
```

préstamos, la probabilidad de perder es de aproximadamente 0.01 y se espera que ganen un total de:

```
n*(loss_per_foreclosure*p + x * (1-p))
#> [1] 14184320
```

dolares! Podemos confirmar esto con 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)
#> [1] 14205099
```

Entonces esto parece ser una decisión obvia. Como resultado, su empleado decide abandonar el banco y comenzar su propia compañía hipotecaria de préstamos de alto riesgo. Unos meses después, el banco de su ex-empleado se declara en quiebra. Se escribe un libro y eventualmente se hace una película relatando el error que cometió su empleado y muchos otros. ¿Qué pasó?

El esquema de su ex-empleado se basó principalmente en esta fórmula matemática:

$$\text{SE}[(X_1 + X_2 + \dots + X_n)/n] = \sigma/\sqrt{n}$$

Al hacer  $n$  grande, minimizan el error estándar de su ganancia por préstamo. Sin embargo, para que esta regla se cumpla, las  $X$ s deben ser eventos independientes: el incumplimiento de una persona debe ser independiente del incumplimiento de otros. Tengan en cuenta que en el caso de promediar el **mismo** evento una y otra vez, un ejemplo extremo de eventos que no son independientes, obtenemos un error estándar que es  $\sqrt{n}$  veces más grande:

$$\text{SE}[(X_1 + X_1 + \dots + X_1)/n] = \text{SE}[nX_1/n] = \sigma > \sigma/\sqrt{n}$$

Para construir una simulación más realista que la original que ejecutó su ex-empleado, supongan que hay un evento global que afecta a todas las personas con hipotecas de alto riesgo y cambia su probabilidad. Suponemos que con una probabilidad de 50-50, todas las probabilidades suben o bajan ligeramente a algún lugar entre 0.03 y 0.05. Pero le sucede a todos a la vez, no solo a una persona. Estos eventos ya no son independientes.

```
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)
})
```

Noten que la ganancia esperada sigue siendo grande:

```
mean(profit)
#> [1] 14353079
```

Sin embargo, la probabilidad de que el banco tenga ganancias negativas se dispara a:

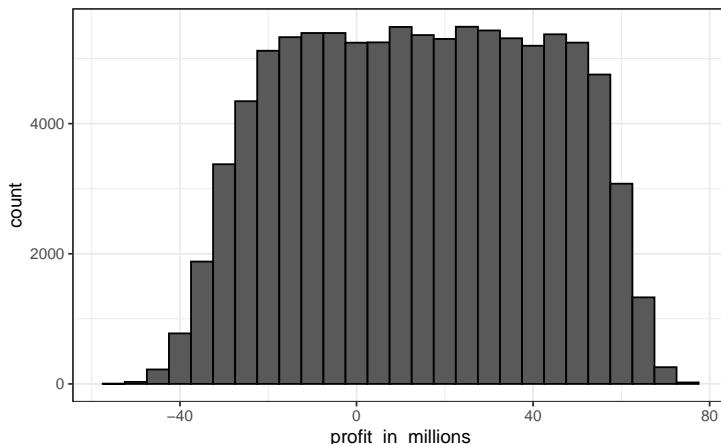
```
mean(profit < 0)
#> [1] 0.345
```

Aún más preocupante es que la probabilidad de perder más de 10 millones de dólares es:

```
mean(profit < -10000000)
#> [1] 0.238
```

Para entender cómo sucede esto, miren la distribución:

```
data.frame(profit_in_millions=profit/10^6) %>%
 ggplot(aes(profit_in_millions)) +
 geom_histogram(color="black", binwidth = 5)
```



La teoría se rompe por completo y la variable aleatoria tiene mucha más variabilidad de lo esperado. El colapso financiero de 2007 se debió, entre otras cosas, a los “expertos” financieros que presumieron independencia cuando tal no era el caso.

## 14.12 Ejercicios

1. Cree una variable aleatoria  $S$  con las ganancias de su banco si otorga 10,000 préstamos, la tasa de incumplimiento es 0.3 y pierde \$200,000 en cada ejecución hipotecaria. Sugerencia: use el código que mostramos en la sección anterior, pero cambie los parámetros.

2. Ejecute una simulación Monte Carlo con 10,000 resultados para  $S$ . Haga un histograma de los resultados.
3. ¿Cuál es el valor esperado de  $S$ ?
4. ¿Cuál es el error estándar de  $S$ ?
5. Supongan que otorgamos préstamos de \$180,000. ¿Cuál debería ser la tasa de interés para que nuestro valor esperado sea 0?
6. (Más difícil) ¿Cuál debería ser la tasa de interés para que la probabilidad de perder dinero sea 1 en 20? En notación matemática, ¿cuál debería ser la tasa de interés para que  $\Pr(S < 0) = 0.05$ ?
7. Si el banco quiere minimizar las probabilidades de perder dinero, ¿cuál de las siguientes opciones **no** hace que suban las tasas de interés?
  - a. Un grupo más pequeño de préstamos.
  - b. Una mayor probabilidad de incumplimiento.
  - c. Una menor probabilidad requerida de perder dinero.
  - d. El número de simulaciones Monte Carlo.



# 15

---

## Inferencia estadística

---

En el Capítulo 16, describiremos, con cierto detalle, cómo los agregadores de encuestas, como FiveThirtyEight, usan los datos para predecir los resultados de las elecciones. Para entender cómo lo hacen, primero debemos aprender los conceptos básicos de la *inferencia estadística*, la parte de la estadística que ayuda a distinguir los patrones reales de esos que surgen del azar. La inferencia estadística es un tema amplio y aquí repasaremos los conceptos básicos utilizando las encuestas como un ejemplo motivador. Para describir los conceptos, complementamos las fórmulas matemáticas con simulaciones Monte Carlo y el código R.

---

### 15.1 Encuestas

Las encuestas de opinión se han llevado a cabo desde el siglo XIX. El objetivo general es describir las opiniones de una población específica sobre un set particular de temas. Recientemente, estas encuestas han sido más notables durante las elecciones presidenciales en EE.UU. Las encuestas son útiles cuando entrevistar a cada miembro de una población particular es lógicamente imposible. La estrategia general es entrevistar a un grupo más pequeño elegido al azar y luego inferir las opiniones de toda la población a partir de las opiniones del grupo más pequeño. La teoría estadística que se usa para justificar el proceso se conoce como *inferencia* y es el tema principal de este capítulo.

Quizás las encuestas de opinión más conocidas son esas realizadas para determinar el candidato favorito de los votantes en una elección determinada. Los estrategas políticos hacen uso extensivo de las encuestas para decidir, entre otras cosas, cómo invertir recursos. Por ejemplo, es posible que quieran saber en qué regiones geográficas enfocar sus esfuerzos de “sacar el voto”.

Las elecciones son un caso particularmente interesante de encuestas de opinión porque la opinión real de toda la población se revela el día de las elecciones. Por supuesto, cuesta millones de dólares realizar una elección real, lo que hace que las encuestas sean una estrategia efectiva para aquellos que quieren pronosticar los resultados.

Aunque típicamente los resultados de estas encuestas se mantienen privados, las organizaciones de noticias realizan encuestas similares porque los resultados tienden a ser de interés público y los datos se hacen públicos. Eventualmente estaremos analizando tales datos.

Real Clear Politics<sup>1</sup> es un ejemplo de un agregador de noticias que organiza y publica resultados de encuestas. Por ejemplo, presentan los siguientes resultados de las encuestas que ofrecen estimados del voto popular para las elecciones presidenciales del 2016<sup>2</sup>:

---

<sup>1</sup><http://www.realclearpolitics.com>

<sup>2</sup>[http://www.realclearpolitics.com/epolls/2016/president/us/general\\_election\\_trump\\_vs\\_clinton-5491.html](http://www.realclearpolitics.com/epolls/2016/president/us/general_election_trump_vs_clinton-5491.html)

| Poll          | Date        | Sample  | MoE | Clinton | Trump | Spread       |
|---------------|-------------|---------|-----|---------|-------|--------------|
| Final Results | –           | –       | –   | 48.2    | 46.1  | Clinton +2.1 |
| RCP Average   | 11/1 - 11/7 | –       | –   | 46.8    | 43.6  | Clinton +3.2 |
| Bloomberg     | 11/4 - 11/6 | 799 LV  | 3.5 | 46.0    | 43.0  | Clinton +3   |
| IBD           | 11/4 - 11/7 | 1107 LV | 3.1 | 43.0    | 42.0  | Clinton +1   |
| Economist     | 11/4 - 11/7 | 3669 LV | –   | 49.0    | 45.0  | Clinton +4   |
| LA Times      | 11/1 - 11/7 | 2935 LV | 4.5 | 44.0    | 47.0  | Trump +3     |
| ABC           | 11/3 - 11/6 | 2220 LV | 2.5 | 49.0    | 46.0  | Clinton +3   |
| FOX News      | 11/3 - 11/6 | 1295 LV | 2.5 | 48.0    | 44.0  | Clinton +4   |
| Monmouth      | 11/3 - 11/6 | 748 LV  | 3.6 | 50.0    | 44.0  | Clinton +6   |
| NBC News      | 11/3 - 11/5 | 1282 LV | 2.7 | 48.0    | 43.0  | Clinton +5   |
| CBS News      | 11/2 - 11/6 | 1426 LV | 3.0 | 47.0    | 43.0  | Clinton +4   |
| Reuters       | 11/2 - 11/6 | 2196 LV | 2.3 | 44.0    | 39.0  | Clinton +5   |

Aunque en Estados Unidos el voto popular no determina el resultado de las elecciones presidenciales, lo utilizaremos como un ejemplo ilustrativo y sencillo de cuán bien funcionan las encuestas. Pronosticar la elección es un proceso más complejo ya que implica combinar resultados de 50 estados y DC y lo describiremos en la Sección 16.8.

Por ahora, hagamos algunas observaciones sobre la tabla anterior. Primero, tengan en cuenta que las diferentes encuestas, todas tomadas días antes de las elecciones, informan una *diferencia* (*spread* en inglés) diferente: la diferencia estimada entre el apoyo a los dos candidatos. Observen también que la diferencia indicada ronda en torno a lo que acabó siendo el resultado real: Clinton ganó el voto popular por 2.1%. También vemos una columna titulada **MoE** que significa *margen de error* (*margin of error* en inglés).

En esta sección, mostraremos cómo se pueden aplicar los conceptos de probabilidad que aprendimos en capítulos anteriores para desarrollar enfoques estadísticos que hacen de las encuestas una herramienta eficaz. Estudiaremos los conceptos estadísticos necesarios para definir *estimadores y márgenes de errores*, y mostraremos cómo podemos usar estos para pronosticar resultados finales relativamente precisos y también proveer un estimador de la precisión de nuestro pronóstico. Una vez que aprendamos esto, podremos entender dos conceptos que son ubicuos en la ciencia de datos: *intervalos de confianza* y *valores-p*. Finalmente, para comprender las declaraciones probabilísticas sobre la probabilidad de que un candidato gane, tendremos que aprender sobre los modelos bayesianos. En las secciones finales, lo reunimos todo para recrear la versión simplificada del modelo de FiveThirtyEight y aplicarlo a las elecciones presidenciales del 2016.

Comenzamos conectando la teoría de probabilidad con la tarea de usar encuestas para aprender sobre una población.

### 15.1.1 El modelo de muestreo para encuestas

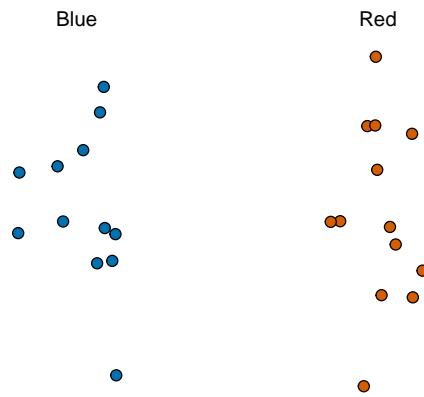
Para ayudarnos a entender la conexión entre las encuestas y lo que hemos aprendido, vamos a construir una situación similar a la que enfrentan los encuestadores. Para imitar el desafío que enfrentan en términos de competir con otros encuestadores para la atención de los medios, utilizaremos una urna llena de cuentas para representar a los votantes y fingiremos que estamos compitiendo por un premio de 25 dólares. El desafío es adivinar la diferencia entre la proporción de cuentas azules y de cuentas rojas en esta urna (en este caso, un frasco de pepinillos):



Antes de hacer una predicción, pueden tomar una muestra (con reemplazo) de la urna. Para imitar el hecho de que realizar encuestas es costoso, les cuesta \$0.10 cada vez que escogen una cuenta. Por lo tanto, si el tamaño de su muestra es 250 y ganan, ni ganarán ni perderán ya que acabarían pagando \$25 por un premio de \$25. Su entrada en la competencia puede ser un intervalo. Si el intervalo que someten contiene la proporción real, obtienen la mitad de lo que pagaron y pasan a la segunda fase de la competencia. En la segunda fase, la entrada con el intervalo más pequeño será la ganadora.

El paquete **dslabs** incluye una función que muestra un sorteo aleatorio de esta urna:

```
library(tidyverse)
library(dslabs)
take_poll(25)
```



Piensen en cómo construirían su intervalo según los datos que se muestran arriba.

Acabamos de describir un modelo de muestreo sencillo para encuestas de opinión. Las cuentas dentro de la urna representan a las personas que votarán el día de las elecciones. Los que votarán por el candidato republicano están representados con cuentas rojas y los demócratas con cuentas azules. Para simplificar, suponemos que no hay otros colores. Es decir, que solo hay dos partidos: republicano y demócrata.

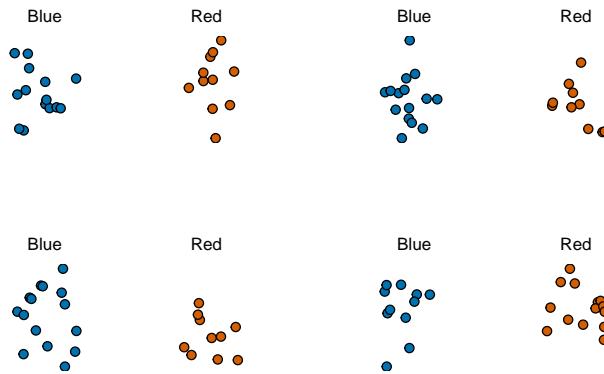
## 15.2 Poblaciones, muestras, parámetros y estimadores

Queremos predecir la proporción de cuentas azules en la urna. Llamemos a esta cantidad  $p$ , que luego nos dice la proporción de cuentas rojas  $1 - p$  y la diferencia  $p - (1 - p)$ , que se simplifica a  $2p - 1$ .

En los libros de texto estadísticos, las cuentas en la urna se llaman la *población*. La proporción de cuentas azules en la población  $p$  se llama un *parámetro*. Las 25 cuentas que vemos en el gráfico anterior se llaman la *muestra*. La tarea de la inferencia estadística es predecir el parámetro  $p$  utilizando los datos observados en la muestra.

¿Podemos hacer esto con las 25 observaciones anteriores? Ciertamente es informativo. Por ejemplo, dado que vemos 13 cuentas rojas y 12 azules, es poco probable que  $p > .9$  o  $p < .1$ . ¿Pero estamos listos para predecir con certeza que hay más cuentas rojas que azules en el frasco?

Queremos construir un estimador de  $p$  utilizando solo la información que observamos. Un estimador se debe considerar un resumen de los datos observados que consideramos informativos sobre el parámetro de interés. Parece intuitivo pensar que la proporción de cuentas azules en la muestra 0.48 debe estar al menos relacionado con la proporción real  $p$ . ¿Pero entonces predecimos que  $p$  es 0.48? Primero, recuerden que la proporción muestral es una variable aleatoria. Si ejecutamos el comando `take_poll(25)` cuatro veces, obtenemos una respuesta diferente cada vez, ya que la proporción muestral es una variable aleatoria.



Tengan en cuenta que en las cuatro muestras aleatorias de arriba, las proporciones de la muestra varían de 0.44 a 0.60. Al describir la distribución de esta variable aleatoria, podremos obtener información sobre cuán buena es este estimador y cómo mejorarla.

### 15.2.1 El promedio de la muestra

La realización de una encuesta de opinión se modela como la toma de una muestra aleatoria de una urna. Estamos proponiendo el uso de la proporción de cuentas azules en nuestra muestra como un estimador del parámetro  $p$ . Una vez que tengamos este estimador, podemos

reportar fácilmente un estimador para la diferencia  $2p - 1$ , pero, para simplificar, ilustraremos los conceptos para estimar  $p$ . Usaremos nuestro conocimiento de probabilidad para defender nuestro uso de la proporción muestral y cuantificaremos cuán cerca creemos que está de la proporción poblacional  $p$ .

Comenzamos definiendo la variable aleatoria  $X$  como:  $X = 1$  si elegimos una cuenta azul al azar y  $X = 0$  si es roja. Esto implica que la población es una lista de 0s y 1s. Si muestramos  $N$  cuentas, entonces el promedio de los sorteos  $X_1, \dots, X_N$  es equivalente a la proporción de cuentas azules en nuestra muestra. Esto es porque sumar las  $X$ s es equivalente a contar las cuentas azules y dividir esta suma por el total  $N$  a calcular una proporción. Usamos el símbolo  $\bar{X}$  para representar este promedio. En general, en los libros de texto de estadísticas, una barra en la parte superior de un símbolo significa el promedio. La teoría que acabamos de aprender sobre la suma de los sorteos es útil porque el promedio es una suma de sorteos multiplicada por la constante  $1/N$ :

$$\bar{X} = 1/N \times \sum_{i=1}^N X_i$$

Para simplificar, supongan que los sorteos son independientes: después de ver cada cuenta muestreada, la devolvemos a la urna. En este caso, ¿qué sabemos sobre la distribución de la suma de los sorteos? Primero, sabemos que el valor esperado de la suma de los sorteos es  $N$  veces el promedio de los valores en la urna. Además, sabemos que el promedio de los 0s y 1s en la urna debe ser  $p$ , la proporción de cuentas azules.

Aquí encontramos una diferencia importante con lo que hicimos en el capítulo de probabilidad: no sabemos qué hay en la urna. Sabemos que hay cuentas azules y rojas, pero no sabemos cuántas de cada una. Esto es lo que queremos descubrir: estamos tratando de **estimar  $p$** .

### 15.2.2 Parámetros

Al igual que usamos variables para definir las cantidades desconocidas en los sistemas de ecuaciones, en la inferencia estadística definimos *parámetros* para definir los componentes desconocidos de nuestros modelos. En el modelo de urna que estamos utilizando para imitar una encuesta de opinión, no sabemos la proporción de cuentas azules en la urna. Definimos los parámetros  $p$  para representar esta cantidad.  $p$  es el promedio de la urna porque si tomamos el promedio de 1s (azul) y 0s (rojo), obtenemos la proporción de cuentas azules. Dado que nuestro objetivo principal es descubrir qué es  $p$ , vamos a estimar este parámetro.

Las ideas presentadas aquí sobre cómo estimar los parámetros y proveer información sobre cuán buenos son estos estimadores, se extrapolan a muchas tareas de la ciencia de datos. Por ejemplo, es posible que queramos saber: ¿cuánto más mejora la salud de los pacientes que reciben un tratamiento comparado a un grupo control? Podemos preguntarnos, ¿cuáles son los efectos de fumar en la salud de una población? ¿Cuáles son las diferencias entre grupos raciales de disparos mortales por parte de la policía? ¿Cuál es la tasa de cambio en esperanza de vida en Estados Unidos durante los últimos 10 años? Todas estas preguntas se pueden considerar como una tarea de estimar un parámetro de una muestra.

### 15.2.3 Encuesta versus pronóstico

Antes de continuar, hagamos una aclaración importante relacionada con el problema práctico de pronosticar las elecciones. Si se realiza una encuesta cuatro meses antes de las elecciones, se estima la  $p$  para ese momento y no para el día de las elecciones. La  $p$  para la noche de las elecciones podría ser diferente ya que las opiniones de las personas fluctúan a través del tiempo. Las encuestas realizadas la noche anterior a las elecciones tienden a ser las más precisas ya que las opiniones no cambian tanto en un día. Sin embargo, los pronosticadores intentan crear herramientas que modelan cómo las opiniones varían a lo largo del tiempo e intentan predecir los resultados de la noche de elecciones tomando en cuenta el hecho de que las opiniones fluctúan. Describiremos algunos enfoques para hacer esto en una sección posterior.

### 15.2.4 Propiedades de nuestro estimador: valor esperado y error estándar

Para comprender cuán bueno es nuestro estimador, describiremos las propiedades estadísticas de la variable aleatoria definida anteriormente: la proporción muestral  $\bar{X}$ . Recuerden que  $\bar{X}$  es la suma de los sorteos independientes, por lo que aplican las reglas que cubrimos en el capítulo de probabilidad.

Usando lo que ya hemos aprendido, el valor esperado de la suma  $N\bar{X}$  es  $N \times$  el promedio de la urna,  $p$ . Entonces, dividir por la constante no aleatoria  $N$  nos da que el valor esperado del promedio  $\bar{X}$  es  $p$ . Podemos escribirlo usando nuestra notación matemática:

$$\text{E}(\bar{X}) = p$$

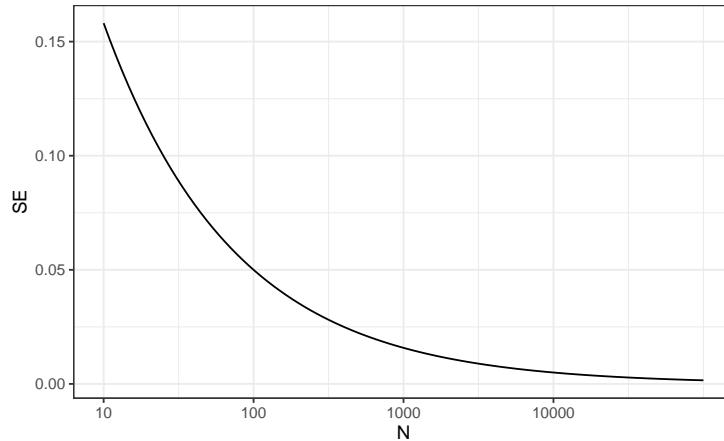
También podemos usar lo que aprendimos para determinar el error estándar: el error estándar de la suma es  $\sqrt{N} \times$  la desviación estándar de la urna. ¿Podemos calcular el error estándar de la urna? Aprendimos una fórmula que nos dice que es  $(1 - 0)\sqrt{p(1 - p)} = \sqrt{p(1 - p)}$ . Como estamos dividiendo la suma por  $N$ , llegamos a la siguiente fórmula para el error estándar del promedio:

$$\text{SE}(\bar{X}) = \sqrt{p(1 - p)/N}$$

Este resultado demuestra el poder de las encuestas. El valor esperado de la proporción muestral  $\bar{X}$  es el parámetro de interés  $p$  y podemos hacer que el error estándar sea tan pequeño como queramos aumentando  $N$ . La ley de los grandes números nos dice que con una encuesta lo suficientemente grande, nuestra estimación converge a  $p$ .

Si realizamos una encuesta lo suficientemente grande como para que nuestro error estándar sea alrededor de 1%, estaremos bastante seguros de quién ganará. Pero, ¿cuán grande debe ser la encuesta para que el error estándar sea tan pequeño?

Un problema es que no sabemos  $p$ , por lo que no podemos calcular el error estándar. Sin embargo, para fines ilustrativos, supongan que  $p = 0.51$  y grafiquemos el error estándar versus el tamaño de la muestra  $N$ :



Del gráfico vemos que necesitaríamos una encuesta de más de 10,000 personas para obtener un error estándar tan bajo. Raras veces vemos encuestas de este tamaño debido en parte a los costos. De la tabla de Real Clear Politics, aprendemos que los tamaños de muestra en las encuestas de opinión oscilan entre 500-3,500 personas. Para un tamaño de muestra de 1,000 y  $p = 0.51$ , el error estándar es:

```
sqrt(p*(1-p))/sqrt(1000)
#> [1] 0.0158
```

o 1.5 puntos porcentuales. Entonces, incluso con grandes encuestas, para elecciones cerradas,  $\bar{X}$  puede llevarnos por mal camino si no nos damos cuenta de que es una variable aleatoria. Sin embargo, podemos decir más sobre cuán cerca nos acercamos con el  $p$  y los hacemos en la Sección 15.4.

### 15.3 Ejercicios

1. Suponga que sondea una población en la que una proporción  $p$  de los votantes son demócratas y  $1 - p$  son republicanos. Su tamaño de muestra es  $N = 25$ . Considere la variable aleatoria  $S$  que es el **total** número de demócratas en su muestra. ¿Cuál es el valor esperado de esta variable aleatoria? Sugerencia: es una función de  $p$ .
2. ¿Cuál es el error estándar de  $S$ ? Sugerencia: es una función de  $p$ .
3. Considere la variable aleatoria  $S/N$ . Esta es equivalente al promedio de la muestra, que hemos estado denotando como  $\bar{X}$ . ¿Cuál es el valor esperado de la  $\bar{X}$ ? Sugerencia: es una función de  $p$ .
4. ¿Cuál es el error estándar de  $\bar{X}$ ? Sugerencia: es una función de  $p$ .
5. Escriba una línea de código que le dé el error estándar  $se$  para el problema anterior para varios valores de  $p$ , específicamente para  $p <- seq(0, 1, length = 100)$ . Haga un gráfico de  $se$  versus  $p$ .

6. Copie el código anterior y póngalo dentro de un bucle-for para hacer el gráfico para  $N = 25$ ,  $N = 100$  y  $N = 1000$ .
  7. Si nos interesa la diferencia en proporciones,  $p - (1 - p)$ , nuestra estimación es  $d = \bar{X} - (1 - \bar{X})$ . Use las reglas que aprendimos sobre sumas de variables aleatorias y variables aleatorias escaladas para derivar el valor esperado de  $d$ .
  8. ¿Cuál es el error estándar de  $d$ ?
  9. Si el valor verdadero de  $p = .45$ , eso significa que los republicanos están ganando por un margen relativamente grande dado que  $d = -.1$ , que es un margen de victoria de 10%. En este caso, ¿cuál es el error estándar de  $2\bar{X} - 1$  si tomamos una muestra de  $N = 25$ ?
  10. Dada la respuesta a 9, ¿cuál de las siguientes opciones describe mejor su estrategia de usar un tamaño de muestra de  $N = 25$ ?
    - a. El valor esperado de nuestra estimación  $2\bar{X} - 1$  es  $d$ , por lo que nuestra predicción será cierta.
    - b. Nuestro error estándar es mayor que la diferencia, por lo que las posibilidades de que  $2\bar{X} - 1$  sea positivo y nos confunda no son tan pequeñas. Deberíamos elegir un tamaño de muestra más grande.
    - c. La diferencia es de 10% y el error estándar es de aproximadamente 0.2, por lo tanto, mucho más pequeño que la diferencia.
    - d. Como no sabemos  $p$ , no tenemos manera de saber si hacer  $N$  más grande mejoraría nuestro error estándar.
- 

#### 15.4 Teorema del límite central en la práctica

El teorema del límite central (CLT) nos dice que la función de distribución para una suma de sorteos es aproximadamente normal. También aprendimos que dividir una variable aleatoria normalmente distribuida por una constante también es una variable normalmente distribuida. Esto implica que la distribución de  $\bar{X}$  es aproximadamente normal.

En resumen, determinamos que  $\bar{X}$  tiene una distribución aproximadamente normal con valor esperado  $p$  y error estándar  $\sqrt{p(1-p)/N}$ .

Ahora, ¿cómo nos ayuda esto? Supongan que queremos saber cuál es la probabilidad de que estamos a 1% de  $p$ . Básicamente estamos preguntando cuánto es:

$$\Pr(|\bar{X} - p| \leq .01)$$

que es lo mismo que:

$$\Pr(\bar{X} \leq p + .01) - \Pr(\bar{X} \leq p - .01)$$

Para contestar a esta pregunta, podemos usar el truco matemático que aprendimos en el capítulo anterior. Resten el valor esperado y dividan por el error estándar para obtener una variable aleatoria que sigue la distribución normal unitaria, llámenla  $Z$ , a la izquierda. Ya que  $p$  es el valor esperado y  $\text{SE}(\bar{X}) = \sqrt{p(1-p)/N}$  es el error estándar, obtenemos:

$$\Pr \left( Z \leq \frac{.01}{\text{SE}(\bar{X})} \right) - \Pr \left( Z \leq -\frac{.01}{\text{SE}(\bar{X})} \right)$$

Un problema que tenemos es que como no sabemos  $p$ , no sabemos  $\text{SE}(\bar{X})$ . Pero resulta que el CLT aún funciona si estimamos el error estándar usando  $\bar{X}$  en lugar de  $p$ . En inglés, decimos que tenemos que *plug-in* el estimador. Por lo tanto, nuestro estimador del error estándar es:

$$\hat{\text{SE}}(\bar{X}) = \sqrt{\bar{X}(1 - \bar{X})/N}$$

En los libros de texto de estadísticas, usamos un sombrerito para denotar estimadores. El estimador se puede construir utilizando los datos observados y  $N$ .

Ahora continuamos con nuestro cálculo, pero dividiendo por  $\hat{\text{SE}}(\bar{X}) = \sqrt{\bar{X}(1 - \bar{X})/N}$ . En nuestra primera muestra teníamos 12 azules y 13 rojos así que  $\bar{X} = 0.48$  y nuestro estimador del error estándar es:

```
x_hat <- 0.48
se <- sqrt(x_hat*(1-x_hat)/25)
se
#> [1] 0.0999
```

Y ahora podemos responder a la pregunta sobre la probabilidad de estar cerca de  $p$ . La respuesta es:

```
pnorm(0.01/se) - pnorm(-0.01/se)
#> [1] 0.0797
```

Por lo tanto, existe una pequeña posibilidad de que estamos cerca. Una encuesta de solo  $N = 25$  personas no es realmente muy útil, al menos no para una elección cerrada.

Anteriormente mencionamos el *margen de error*. Ahora podemos definirlo porque es simplemente dos veces el error estándar, que ahora podemos estimar. En nuestro caso es:

```
1.96*se
#> [1] 0.196
```

¿Por qué multiplicamos por 1.96? Porque si preguntan cuál es la probabilidad de que estemos dentro de 1.96 errores estándar de  $p$ , obtenemos:

$$\Pr \left( Z \leq 1.96 \text{SE}(\bar{X})/\text{SE}(\bar{X}) \right) - \Pr \left( Z \leq -1.96 \text{SE}(\bar{X})/\text{SE}(\bar{X}) \right)$$

que es:

$$\Pr (Z \leq 1.96) - \Pr (Z \leq -1.96)$$

que sabemos es aproximadamente 95%:

```
pnorm(1.96)-pnorm(-1.96)
#> [1] 0.95
```

Por lo tanto, hay un 95% de probabilidad de que  $\bar{X}$  estará dentro  $1.96 \times \hat{SE}(\bar{X})$ , en nuestro caso aproximadamente dentro de 0.2, de  $p$ . Tengan en cuenta que el 95% es una elección arbitraria y, a veces, se utilizan otros porcentajes, pero es el valor más utilizado para definir el margen de error. A menudo redondeamos 1.96 a 2 para simplificar la presentación.

En resumen, el CLT nos dice que nuestra encuesta que se basa en un tamaño de muestra de 25 no es muy útil. Efectivamente no aprendemos mucho cuando el margen de error es tan grande. Lo único que realmente podemos decir es que el voto popular no se ganará por un margen amplio. Esta es la razón por la cual los encuestadores tienden a usar tamaños de muestra más grandes.

En la tabla anterior, vemos que los tamaños de muestra típicos oscilan entre 700 y 3500. Para ver cómo esto nos da un resultado mucho más práctico, noten que si hubiéramos obtenido un  $\bar{X} = 0.48$  con un tamaño de muestra de 2,000, nuestro error estándar  $\hat{SE}(\bar{X})$  habría sido 0.011. Entonces nuestro resultado es un estimador de 48% con un margen de error de 2%. En este caso, el resultado es mucho más informativo y nos haría pensar que hay más cuentas rojas que azules. Recuerden, sin embargo, que esto es hipotético. No hicimos una encuesta de 2,000 ya que no queremos dañar el concurso.

#### 15.4.1 Una simulación Monte Carlo

Supongan que queremos usar una simulación Monte Carlo para corroborar las herramientas que hemos construido utilizando la teoría de la probabilidad. Para crear la simulación, escribiríamos código como este:

```
B <- 10000
N <- 1000
x_hat <- replicate(B, {
 x <- sample(c(0,1), size = N, replace = TRUE, prob = c(1-p, p))
 mean(x)
})
```

El problema es, por supuesto, que no sabemos  $p$ . Podríamos construir una urna como la que se muestra arriba y ejecutar una simulación analógica (sin una computadora). Nos tomaría mucho tiempo, pero podríamos tomar 10,000 muestras, contar las cuentas y registrar las proporciones de azul. Podemos usar la función `take_poll(n=1000)` en lugar de escoger de una urna real, pero todavía tomaría tiempo contar las cuentas y registrar los resultados.

Por lo tanto, algo que hacemos para corroborar los resultados teóricos es elegir uno o varios valores de  $p$  y ejecutar las simulaciones. Vamos a configurar  $p=0.45$ . Entonces podemos simular una encuesta:

```
p <- 0.45
N <- 1000

x <- sample(c(0,1), size = N, replace = TRUE, prob = c(1-p, p))
x_hat <- mean(x)
```

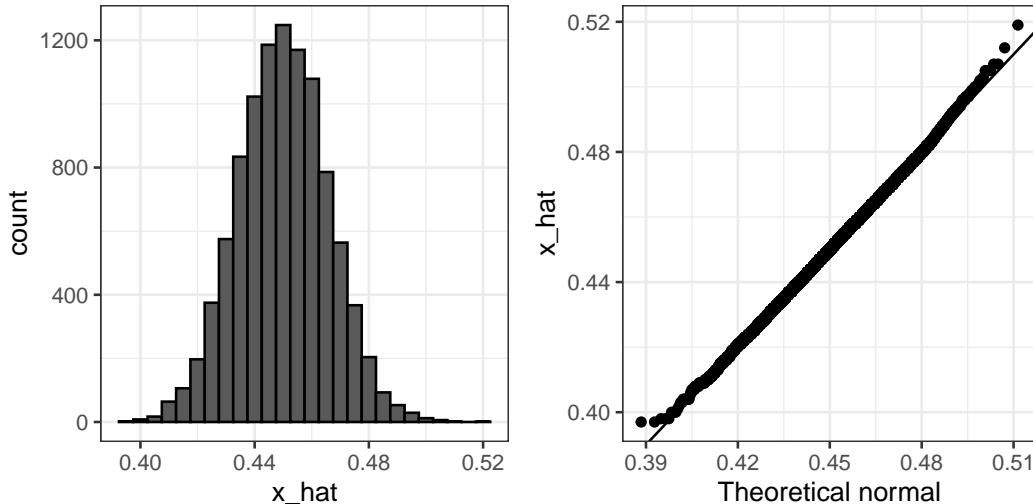
En esta muestra particular, nuestro estimador es `x_hat`. Podemos usar ese código para hacer una simulación Monte Carlo:

```
B <- 10000
x_hat <- replicate(B, {
 x <- sample(c(0,1), size = N, replace = TRUE, prob = c(1-p, p))
 mean(x)
})
```

Para repasar, la teoría nos dice que la distribución de  $\bar{X}$  es aproximadamente normal, tiene valor esperado  $p = 0.45$  y error estándar  $\sqrt{p(1-p)/N} = 0.016$ . La simulación confirma esto:

```
mean(x_hat)
#> [1] 0.45
sd(x_hat)
#> [1] 0.0158
```

Un histograma y un gráfico Q-Q confirman que la aproximación normal también es precisa:



Por supuesto, en la vida real nunca podríamos realizar un experimento así porque no sabemos  $p$ . Pero podríamos ejecutarlo para varios valores de  $p$  y  $N$  y ver que la teoría realmente funciona bien para la mayoría de los valores. Pueden hacerlo fácilmente volviendo a ejecutar el código anterior después de cambiar  $p$  y  $N$ .

### 15.4.2 La diferencia

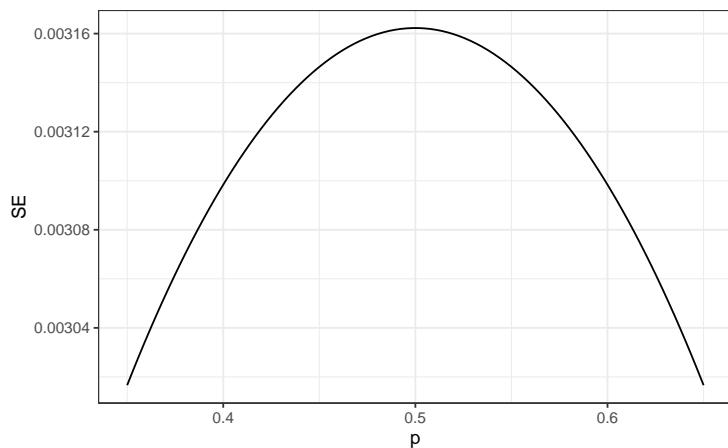
El reto es predecir la diferencia, no la proporción  $p$ . Sin embargo, dado que suponemos que solo hay dos partidos, sabemos que la diferencia es  $p - (1-p) = 2p - 1$ . Como resultado, todo lo que hemos hecho se puede adaptar fácilmente a estimar  $2p - 1$ . Una vez que tengamos

nuestro estimador  $\bar{X}$  y  $\hat{SE}(\bar{X})$ , estimamos la diferencia con  $2\bar{X} - 1$  y, dado que estamos multiplicando por 2, el error estándar es  $2\hat{SE}(\bar{X})$ . Noten que restar 1 no añade variabilidad, por lo que no afecta el error estándar.

Para nuestra muestra anterior de 25 artículos, nuestro  $p$  estimado es .48 con margen de error .20 y nuestro estimador de la diferencia es 0.04 con margen de error .40. Nuevamente, no es un tamaño de muestra muy útil. Sin embargo, el punto es que una vez que tengamos un estimador y un error estándar para  $p$ , lo tenemos para la diferencia  $2p - 1$ .

### 15.4.3 Sesgo: ¿por qué no realizar una encuesta bien grande?

Para valores realistas de  $p$ , digamos de 0.35 a 0.65, si realizamos una encuesta bien grande con 100,000 personas, la teoría nos dice que predeciríamos la elección perfectamente ya que el mayor margen de error posible es de alrededor de 0.3%:



Una razón es que realizar una encuesta de este tipo es muy costosa. Otra razón posiblemente más importante es que la teoría tiene sus limitaciones. El sondeo es mucho más complicado que escoger cuentas de una urna. Algunas personas pueden mentirle a los encuestadores y otras pueden no tener teléfonos. Pero quizás la manera más importante en que una encuesta real difiere de un modelo de urna es que no sabemos con certeza quién está en nuestra población y quién no. ¿Cómo sabemos quién va a votar? ¿Todos los votantes tienen la misma posibilidad de ser encuestado? Aunque nuestro margen de error es bien pequeño, es posible que nuestro valor esperado no sea exactamente  $p$ . A esto lo llamamos sesgo (*bias* en inglés). Históricamente, observamos que las encuestas están sesgadas, aunque no por mucho. El sesgo típico parece ser de aproximadamente 1-2%. Esto hace que el pronóstico de las elecciones sea un poco más interesante y hablaremos sobre cómo modelar esto en un capítulo posterior.

## 15.5 Ejercicios

1. Escriba una función que modele una urna que toma la proporción de demócratas  $p$  y el tamaño de la muestra  $N$  como argumentos y devuelve el promedio de la muestra si los demócratas son 1s y los republicanos son 0s. Llame a la función `take_sample`.
2. Ahora suponga que  $p <- 0.45$  y que su tamaño de muestra es  $N = 100$ . Tome una muestra 10,000 veces y guarde el vector de `mean(X) - p` en un objeto llamado `errors`. Sugerencia: use la función que escribió para el ejercicio 1 para escribir esto en una línea de código.
3. El vector `errors` contiene, para cada muestra simulada, la diferencia entre el valor real  $p$  y nuestro estimador  $\bar{X}$ . Nos referimos a esta diferencia como el *error*. Calcule el promedio y haga un histograma de los errores generados en la simulación Monte Carlo y seleccione cuál de las siguientes opciones describe mejor sus distribuciones:

```
mean(errors)
hist(errors)
```

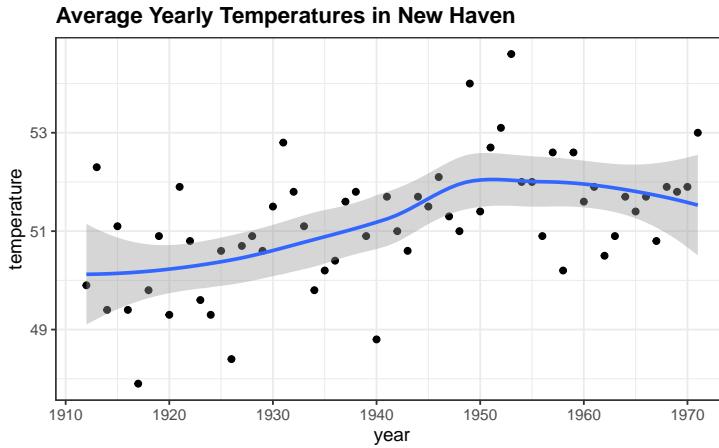
- a. Los errores son alrededor de 0.05.
  - b. Los errores son todos alrededor de -0.05.
  - c. Los errores se distribuyen simétricamente alrededor de 0.
  - d. Los errores varían de -1 a 1.
4. El error  $\bar{X} - p$  es una variable aleatoria. En la práctica, el error no se observa porque no sabemos  $p$ . Aquí lo observamos porque construimos la simulación. ¿Cuál es el tamaño promedio del error si definimos el tamaño tomando el valor absoluto  $| \bar{X} - p |$ ?
  5. El error estándar está relacionado con el **tamaño** típico del error que cometemos al predecir. Decimos **tamaño** porque acabamos de ver que los errores están centrados alrededor de 0, por lo que el valor de error promedio es 0. Por razones matemáticas relacionadas con el teorema del límite central, utilizamos la desviación estándar de `errors` en lugar del promedio de los valores absolutos para cuantificar el tamaño típico. ¿Cuál es esta desviación estándar de los errores?
  6. La teoría que acabamos de aprender nos dice cuál será esta desviación estándar porque es el error estándar de  $\bar{X}$ . Según la teoría, ¿cuánto es el error estándar de  $\bar{X}$  para un tamaño de muestra de 100?
  7. En la práctica, no sabemos  $p$ , por lo que construimos un estimador de la predicción teórica basada en remplazar  $p$  con  $\bar{X}$ . Calcule este estimador. Fije la semilla en 1 con `set.seed(1)`.
  8. Observe cuán cerca están los estimadores de error estándar obtenidos de la simulación Monte Carlo (ejercicio 5), la predicción teórica (ejercicio 6) y el estimador de la predicción teórica (ejercicio 7). La teoría está funcionando y nos da un enfoque práctico para conocer el error típico que cometemos si predecimos  $p$  con  $\bar{X}$ . Otra ventaja que provee el resultado teórico es que da una idea de cuán grande tiene que ser el tamaño de muestra para obtener la precisión que necesitamos. Anteriormente vimos que los errores estándar más grandes ocurren para  $p = 0.5$ . Cree un gráfico del error estándar más grande para  $N$  que va desde 100 hasta 5,000. Según este gráfico, ¿cuán grande debe ser el tamaño de la muestra para tener un error estándar de aproximadamente 1%?

- a. 100
  - b. 500
  - c. 2,500
  - d. 4,000
9. Para el tamaño de la muestra  $N = 100$ , el teorema del límite central nos dice que la distribución de  $\bar{X}$  es:
- a. prácticamente igual a  $p$ .
  - b. aproximadamente normal con el valor esperado  $p$  y error estándar  $\sqrt{p(1-p)/N}$ .
  - c. aproximadamente normal con el valor esperado  $\bar{X}$  y error estándar  $\sqrt{\bar{X}(1-\bar{X})/N}$ .
  - d. no es una variable aleatoria.
10. Según la respuesta del ejercicio 8, el error  $\bar{X} - p$  es:
- a. prácticamente igual a 0.
  - b. aproximadamente normal con el valor esperado 0 y error estándar  $\sqrt{p(1-p)/N}$ .
  - c. aproximadamente normal con el valor esperado  $p$  y error estándar  $\sqrt{p(1-p)/N}$ .
  - d. No es una variable aleatoria.
11. Para corroborar su respuesta al ejercicio 9, haga un gráfico Q-Q de los **errors** que generó en el ejercicio 2 para ver si siguen una distribución normal.
12. Si  $p = 0.45$  y  $N = 100$  como en el ejercicio 2, use el CLT para estimar la probabilidad de que  $\bar{X} > 0.5$ . Puede suponer que sabe que  $p = 0.45$  para este cálculo
13. Suponga que está en una situación práctica y no sabe  $p$ . Tome una muestra de tamaño  $N = 100$  y obtenga una muestra promedio de  $\bar{X} = 0.51$ . ¿Cuál es la aproximación del CLT para la probabilidad de que su error sea igual o mayor que 0.01?

---

## 15.6 Intervalos de confianza

Los *intervalos de confianza* (*confidence intervals* en inglés) son un concepto muy útil ampliamente utilizado por los analistas de datos. Una versión de estos que vemos comúnmente proviene de la geometría `geom_smooth` de `ggplot`. Aquí tenemos un ejemplo usando un set de datos de temperatura disponible en R:



En la parte sobre *machine learning*, aprenderemos cómo se forma la curva, pero por ahora consideren el área sombreada alrededor de la curva. Esto se crea utilizando el concepto de intervalos de confianza.

En nuestro concurso anterior, se les pidió que dieran un intervalo. Si el intervalo que indicaron incluye el  $p$ , obtienen la mitad del dinero que gastaron en su “encuesta” y pasan a la siguiente etapa del concurso. Una forma de pasar a la segunda ronda es informar un intervalo muy grande. Por ejemplo, el intervalo  $[0, 1]$  está garantizado a siempre incluir  $p$ . Sin embargo, con un intervalo tan grande, no tenemos posibilidades de ganar el concurso. Del mismo modo, si ustedes son pronosticadores de elecciones y predicen que la diferencia será entre -100% y 100%, serán ridiculizados por decir lo obvio. Incluso hasta un intervalo más pequeño, como decir que la diferencia será entre -10 y 10%, no se consideraría serio.

Por otro lado, entre más pequeño sea el intervalo que escogemos, más bajas serán nuestras posibilidades de ganar el premio. Del mismo modo, un encuestador audaz que informa intervalos demasiado pequeños y se equivoca la mayor parte del tiempo no se considerará un buen encuestador. Queremos estar en algún punto intermedio.

Podemos usar la teoría estadística que hemos aprendido para calcular la probabilidad de cualquier intervalo dado, incluyendo  $p$ . Si se nos pide crear un intervalo con, digamos, una probabilidad de 95% de incluir  $p$ , podemos hacer eso también. Estos se denominan intervalos de confianza de 95%.

Cuando un encuestador informa un estimador y un margen de error, de alguna manera informa un intervalo de confianza de 95%. Mostremos cómo funciona esto matemáticamente.

Queremos saber la probabilidad de que el intervalo  $[\bar{X} - 2\hat{SE}(\bar{X}), \bar{X} + 2\hat{SE}(\bar{X})]$  contenga la verdadera proporción  $p$ . Primero, consideren que el inicio y el final de estos intervalos son variables aleatorias: cada vez que tomamos una muestra, cambian. Para ilustrar esto, ejecuten la simulación Monte Carlo arriba dos veces. Usamos los mismos parámetros que arriba:

```
p <- 0.45
N <- 1000
```

Y observen que el intervalo aquí:

```
x <- sample(c(0, 1), size = N, replace = TRUE, prob = c(1-p, p))
x_hat <- mean(x)
se_hat <- sqrt(x_hat * (1 - x_hat) / N)
c(x_hat - 1.96 * se_hat, x_hat + 1.96 * se_hat)
#> [1] 0.416 0.478
```

es diferente de este:

```
x <- sample(c(0,1), size=N, replace=TRUE, prob=c(1-p, p))
x_hat <- mean(x)
se_hat <- sqrt(x_hat * (1 - x_hat) / N)
c(x_hat - 1.96 * se_hat, x_hat + 1.96 * se_hat)
#> [1] 0.419 0.481
```

Sigan muestreando y creando intervalos y verán la variación aleatoria.

Para determinar la probabilidad de que el intervalo incluya  $p$ , necesitamos calcular esto:

$$\Pr(\bar{X} - 1.96\hat{SE}(\bar{X}) \leq p \leq \bar{X} + 1.96\hat{SE}(\bar{X}))$$

Al restar y dividir las mismas cantidades en todas las partes de la ecuación, nosotros obtenemos que lo anterior es equivalente a:

$$\Pr\left(-1.96 \leq \frac{\bar{X} - p}{\hat{SE}(\bar{X})} \leq 1.96\right)$$

El término en el medio es una variable aleatoria aproximadamente normal con valor esperado 0 y error estándar 1, que hemos estado denotando con  $Z$ , y por lo tanto tenemos:

$$\Pr(-1.96 \leq Z \leq 1.96)$$

que podemos calcular rápidamente usando:

```
pnorm(1.96) - pnorm(-1.96)
#> [1] 0.95
```

demonstrando que tenemos una probabilidad de 95%.

Si queremos tener una probabilidad más grande, digamos 99%, necesitamos multiplicar por cualquier  $z$  que cumpla lo siguiente:

$$\Pr(-z \leq Z \leq z) = 0.99$$

Utilizando:

```
z <- qnorm(0.995)
z
#> [1] 2.58
```

lograremos esto porque por definición `pnorm(qnorm(0.995))` es 0.995 y por simetría `pnorm(1-qnorm(0.995))` es 1 - 0.995. Como consecuencia, tenemos que:

```
pnorm(z) - pnorm(-z)
#> [1] 0.99
```

es  $0.995 - 0.005 = 0.99$ . Podemos usar este enfoque para cualquier proporción  $p$ : nosotros fijamos  $z = qnorm(1 - (1 - p)/2)$  porque  $1 - (1 - p)/2 - (1 - p)/2 = p$ .

Entonces, por ejemplo, para  $p = 0.95$ ,  $1 - (1 - p)/2 = 0.975$  y obtenemos el 1.96 que hemos estado usando:

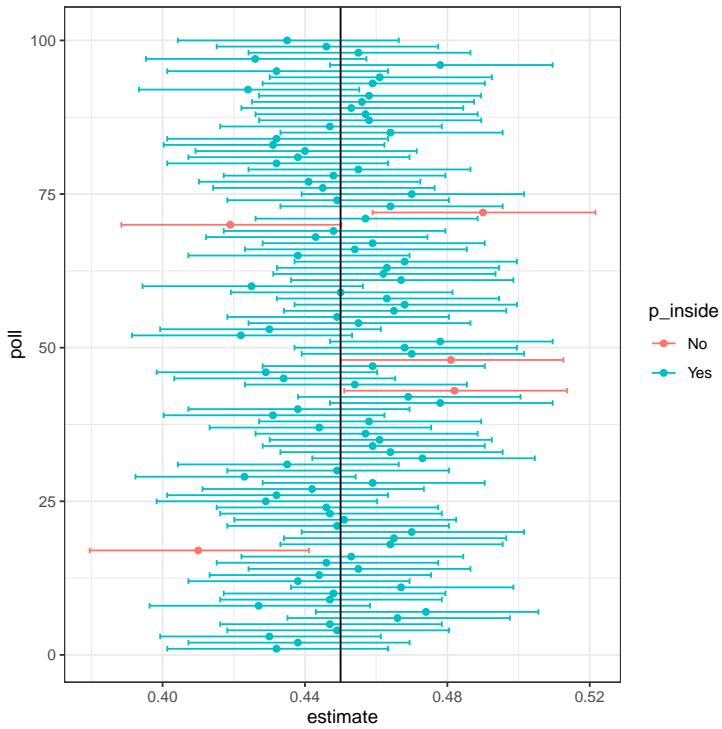
```
qnorm(0.975)
#> [1] 1.96
```

### 15.6.1 Una simulación Monte Carlo

Podemos ejecutar una simulación Monte Carlo para confirmar que, de hecho, un intervalo de confianza de 95% incluye  $p$  95% del tiempo.

```
N <- 1000
B <- 10000
inside <- replicate(B, {
 x <- sample(c(0,1), size = 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 - 1.96 * se_hat, x_hat + 1.96 * se_hat)
})
mean(inside)
#> [1] 0.948
```

El siguiente gráfico muestra los primeros 100 intervalos de confianza. En este caso, creamos la simulación para que la línea negra denote el parámetro que estamos tratando de estimar:



### 15.6.2 El idioma correcto

Al usar la teoría que describimos anteriormente, es importante recordar que los intervalos son aleatorios, no  $p$ . En el gráfico anterior, podemos ver los intervalos aleatorios moviéndose. En cambio, la proporción de cuentas azules en la urna,  $p$ , representada por la línea vertical, no se mueve. Entonces el 95% se refiere a la probabilidad de que este intervalo aleatorio caiga encima de  $p$ . Decir que  $p$  tiene una probabilidad de 95% de estar entre esto y eso es técnicamente una declaración incorrecta porque  $p$  no es aleatorio.

## 15.7 Ejercicios

Para estos ejercicios, utilizaremos encuestas reales de las elecciones del 2016. Puede cargar los datos del paquete **dslabs**.

```
library(dslabs)
data("polls_us_election_2016")
```

Específicamente, utilizaremos todas las encuestas nacionales que acabaron dentro de una semana antes de las elecciones.

```
library(tidyverse)
polls <- polls_us_election_2016 %>%
 filter(enddate >= "2016-10-31" & state == "U.S.")
```

1. Para la primera encuesta, puede obtener el tamaño de las muestras y el porcentaje estimado para Clinton con:

```
N <- polls$samplesize[1]
x_hat <- polls$rawpoll_clinton[1]/100
```

Suponga que solo hay dos candidatos. Construya un intervalo de confianza de 95% para la proporción  $p$  observada la noche de elecciones.

2. Ahora use **dplyr** para añadir dos columnas al objeto **poll1**, llámelas **lower** y **upper**, para representar el intervalo de confianza. Luego use **select** para mostrar los variables **pollster**, **enddate**, **x\_hat**, **lower**, **upper**. Sugerencia: defina columnas temporeras **x\_hat** y **se\_hat**.

3. El conteo final para el voto popular fue Clinton 48.2% y Trump 46.1%. Agregue una columna, llámela **hit**, a la tabla anterior que indica si el intervalo de confianza incluía la proporción verdadera  $p = 0.482$  o no.

4. Para la tabla que acaba de crear, ¿qué proporción de intervalos de confianza incluyeron  $p$ ?

5. Si estos intervalos de confianza se construyen correctamente y la teoría se sostiene, ¿qué proporción debería incluir  $p$ ?

6. De estas encuestas, una proporción menor de lo esperado resulta en intervalos de confianza que contienen  $p$ . Si examina la tabla cuidadosamente, verá que la mayoría de las encuestas que no incluyen  $p$  están subestimando. La razón es que hay votantes indecisos, las personas encuestadas que aún no saben por quién votarán o no quieren decir. Debido a que históricamente los indecisos se dividen igualmente entre los dos candidatos principales el día de las elecciones, es más informativo estimar la variabilidad o la diferencia entre la proporción de dos candidatos  $d$ , que en esta elección fue  $0.482 - 0.461 = 0.021$ . Suponga que solo hay dos partidos y que  $d = 2p - 1$ , redefina **polls** como se hace abajo y repita el ejercicio 1, pero para la diferencia.

```
polls <- polls_us_election_2016 %>%
 filter(enddate >= "2016-10-31" & state == "U.S.") %>%
 mutate(d_hat = rawpoll_clinton/ 100 - rawpoll_trump/ 100)
```

7. Ahora repita el ejercicio 3, pero para la diferencia.

8. Ahora repita el ejercicio 4, pero para la diferencia.

9. Aunque la proporción de intervalos de confianza aumenta sustancialmente, sigue siendo menor que 0.95. En el próximo capítulo, aprendemos la razón de esto. Para motivar esto, haga un gráfico del error, la diferencia entre el estimador de cada encuesta y la diferencia real  $d = 0.021$ . Estratifique por encuestador.

10. Vuelva a hacer el gráfico que hizo para el ejercicio 9, pero solo para los encuestadores que tomaron cinco o más encuestas.

## 15.8 Poder

Los encuestadores no se consideran exitosos al proveer intervalos de confianza correctos, sino al predecir quién ganará. Cuando tomamos un tamaño de muestra de 25 cuentas, el intervalo de confianza para la diferencia:

```
N <- 25
x_hat <- 0.48
(2 * x_hat - 1) + c(-1.96, 1.96) * 2 * sqrt(x_hat * (1 - x_hat)/ N)
#> [1] -0.432 0.352
```

incluye 0. Si esto fuera una encuesta y nos viéramos obligados a hacer una declaración, tendríamos que decir que ambos resultados son probables.

Un problema con los resultados de nuestra encuesta es que, dado el tamaño de la muestra y el valor de  $p$ , tendríamos que sacrificar la probabilidad de una predicción incorrecta para crear un intervalo que no incluya 0.

Esto no significa que la elección está cerrada. Solo significa que tenemos un tamaño de muestra pequeño. En los libros de texto estadísticos esto se llama falta de *poder*. En el contexto de las encuestas, el *poder* es la probabilidad de detectar diferencias que no sean 0.

Al aumentar el tamaño de nuestra muestra, disminuimos nuestro error estándar y, por lo tanto, tenemos muchas más posibilidades de detectar la dirección de la diferencia.

## 15.9 valores-p

Los *valores-p* (*p-values* en inglés) son ubicuos en la literatura científica. Están relacionados con los intervalos de confianza, por lo que presentamos el concepto aquí.

Consideremos las cuentas azules y rojas. Supongan que, en lugar de querer un estimador de la diferencia o de la proporción de azul, solo nos interesa la pregunta: ¿hay más cuentas azules o cuentas rojas? Queremos saber si la diferencia  $2p - 1 > 0$ .

Digamos que tomamos una muestra aleatoria de  $N = 100$  y observamos 52 cuentas azules, lo que nos da  $2\bar{X} - 1 = 0.04$ . Esto parece estar apuntando a la existencia de más cuentas azules que rojas ya que 0.04 es mayor que 0. Sin embargo, como científicos de datos, debemos ser escépticos. Sabemos que el azar afecta este proceso y podríamos obtener un 52 incluso cuando la diferencia real es 0. Llamamos a la suposición de que la diferencia es  $2p - 1 = 0$  una *hipótesis nula*. La hipótesis nula es la hipótesis del escéptico. Hemos observado una variable aleatoria  $2\bar{X} - 1 = 0.04$  y el valor-p es la respuesta a la pregunta: ¿cuán probable es ver un valor tan grande, cuando la hipótesis nula es cierta? Entonces escribimos:

$$\Pr(|\bar{X} - 0.5| > 0.02)$$

suponiendo que  $2p - 1 = 0$  o  $p = 0.5$ . Bajo la hipótesis nula sabemos que:

$$\sqrt{N} \frac{\bar{X} - 0.5}{\sqrt{0.5(1 - 0.5)}}$$

es normal unitaria. Por lo tanto, podemos calcular la probabilidad anterior, que es el valor-p.

$$\Pr \left( \sqrt{N} \frac{|\bar{X} - 0.5|}{\sqrt{0.5(1 - 0.5)}} > \sqrt{N} \frac{0.02}{\sqrt{0.5(1 - 0.5)}} \right)$$

```
N <- 100
z <- sqrt(N)*0.02/0.5
1 - (pnorm(z) - pnorm(-z))
#> [1] 0.689
```

En este caso, existe una gran posibilidad de ver 52 o más bajo la hipótesis nula.

Tengan en cuenta que existe una conexión entre los valores-p y los intervalos de confianza. Si un intervalo de confianza de 95% de la diferencia no incluye 0, sabemos que el valor-p tiene que ser menor que 0.05.

Para aprender más sobre los valores-p, pueden consultar cualquier libro de texto de estadísticas. Sin embargo, en general, preferimos resumir nuestros resultados con intervalos de confianza en vez de valores-p, ya que nos da una idea del tamaño del estimador. Si solo informamos el valor-p, no proveemos información sobre la importancia del hallazgo en el contexto del problema.

## 15.10 Pruebas de asociación

Las pruebas estadísticas que hemos estudiado hasta ahora no incluyen varios tipos de datos. Específicamente, no hemos discutido la inferencia para datos binarios, categóricos y ordinales. Para dar un ejemplo muy específico, consideren el siguiente estudio de caso.

Una publicación del 2014 de PNAS<sup>3</sup> analizó las tasas de éxito de las agencias de financiamiento en los Países Bajos y concluyó que:

los resultados revelan un sesgo de género que favorece a los hombres solicitantes sobre las mujeres solicitantes en la priorización de sus evaluaciones de “calidad de investigador” (pero no de “calidad de propuesta”), así como en el uso del lenguaje en los materiales de instrucción y de evaluación.

La evidencia principal de esta conclusión se reduce a una comparación de los porcentajes. La Tabla S1 en el documento incluye la información que necesitamos. Aquí están las tres columnas que muestran los resultados generales:

<sup>3</sup><http://www.pnas.org/content/112/40/12349.abstract>

```
library(tidyverse)
library(dslabs)
data("research_funding_rates")
research_funding_rates %>% select(discipline, applications_total,
 success_rates_total) %>% head()
#> discipline applications_total success_rates_total
#> 1 Chemical sciences 122 26.2
#> 2 Physical sciences 174 20.1
#> 3 Physics 76 26.3
#> 4 Humanities 396 16.4
#> 5 Technical sciences 251 17.1
#> 6 Interdisciplinary 183 15.8
```

Tenemos estos valores para cada género:

```
names(research_funding_rates)
#> [1] "discipline" "applications_total" "applications_men"
#> [4] "applications_women" "awards_total" "awards_men"
#> [7] "awards_women" "success_rates_total" "success_rates_men"
#> [10] "success_rates_women"
```

Podemos calcular el total de los que tuvieron éxito y el total de los que no lo tuvieron de la siguiente manera:

```
totals <- research_funding_rates %>%
 select(-discipline) %>%
 summarize_all(sum) %>%
 summarize(yes_men = awards_men,
 no_men = applications_men - awards_men,
 yes_women = awards_women,
 no_women = applications_women - awards_women)
```

Entonces vemos que un mayor porcentaje de hombres que mujeres recibieron premios:

```
totals %>% summarize(percent_men = yes_men/(yes_men+no_men),
 percent_women = yes_women/(yes_women+no_women))
#> percent_men percent_women
#> 1 0.177 0.149
```

Pero, ¿esto se debe solo a la variabilidad aleatoria? Aquí aprenderemos a llevar a cabo inferencia estadística para este tipo de datos.

### 15.10.1 Lady Tasting Tea

R.A. Fisher<sup>4</sup> fue uno de los primeros en formalizar las pruebas de hipótesis. El “Lady Tasting Tea” es uno de los ejemplos más famosos.

---

<sup>4</sup>[https://en.wikipedia.org/wiki/Ronald\\_Fisher](https://en.wikipedia.org/wiki/Ronald_Fisher)

La historia es la siguiente: una conocida de Fisher insistió que ella podía detectar si añadían leche antes o después de verter el té. Fisher se mostró escéptico. Diseñó un experimento para probar esta afirmación. Él le dio cuatro pares de tazas de té. Cada par incluía una taza con leche vertida primero y la otra después del té. El orden era aleatorio. La hipótesis nula aquí es que ella está adivinando. Fisher dedujo la distribución del número de selecciones correctas suponiendo que las elecciones eran aleatorias e independientes.

Como ejemplo, supongan que la amiga escogió 3 de 4 correctamente. ¿Creemos que ella tiene una habilidad especial? La pregunta básica que hacemos es: si ella realmente está adivinando, ¿cuáles son las posibilidades de que ella saque 3 o más correctas? Tal como lo hemos hecho antes, podemos calcular una probabilidad bajo la hipótesis nula de que ella está adivinando 4 con leche vertida primero y 4 después. Bajo esta hipótesis nula, podemos pensar en este ejemplo particular como sacar 4 cuentas de una urna con 4 cuentas azules (respuesta correcta) y 4 cuentas rojas (respuesta incorrecta). Recuerden, ella sabe que hay cuatro tazas con leche antes del té y cuatro con leche después.

Bajo la hipótesis nula de que ella simplemente está adivinando, cada cuenta tiene la misma posibilidad de ser elegida. Entonces podemos usar combinaciones para averiguar cada probabilidad. La probabilidad de elegir 3 es  $\binom{4}{3} \binom{4}{1} / \binom{8}{4} = 16/70$ . La probabilidad de elegir bien las 4 veces es  $\binom{4}{4} \binom{4}{0} / \binom{8}{4} = 1/70$ . Por lo tanto, la posibilidad de observar un 3 o algo más extremo, bajo la hipótesis nula, es  $\approx 0.24$ . Este es el valor-p. El procedimiento que produjo este valor-p se llama la *prueba exacta de Fisher* (*Fisher's exact test* en inglés) y utiliza la *distribución hipergeométrica*.

### 15.10.2 Tablas 2x2

Los datos del experimento generalmente se resumen en una tabla como esta:

```
tab <- matrix(c(3,1,1,3), 2, 2)
rownames(tab) <- c("Poured Before", "Poured After")
colnames(tab) <- c("Guessed before", "Guessed after")
tab
#> Guessed before Guessed after
#> Poured Before 3 1
#> Poured After 1 3
```

que se conoce como una tabla 2x2. Para cada una de las cuatro combinaciones que se pueden obtener con un par de variables binarias, la tabla muestra los recuentos observados para cada ocurrencia.

La función `fisher.test` realiza los cálculos de inferencia anteriores:

```
fisher.test(tab, alternative="greater")$p.value
#> [1] 0.243
```

### 15.10.3 Prueba de chi-cuadrado

En cierto sentido, nuestro ejemplo de tasas de financiamiento es parecido al de “Lady Tasting Tea”. Sin embargo, en el ejemplo de “Lady Tasting Tea”, el número de cuentas azules y rojas

se fija experimentalmente y el número de respuestas dadas para cada categoría también. Esto se debe a que Fisher se aseguró de que se vertieran cuatro tazas con leche antes del té y cuatro tazas con leche después del té y la señora lo sabía, por lo que las respuestas también tendrían que incluir cuatro antes y cuatro después. Si este es el caso, la suma de las filas y la suma de las columnas son fijas. Esto define restricciones sobre las posibles formas en que podemos llenar la tabla 2x2 y también nos permite usar la distribución hipergeométrica. En general, este no es el caso. No obstante, hay otro enfoque, la prueba de chi-cuadrado, que se describe a continuación.

Imaginen que tenemos 290, 1,345, 177, 1,011 solicitantes, algunos son hombres y otros son mujeres y algunos reciben financiamiento, mientras que otros no. Vimos que las tasas de éxito para hombres y mujeres eran:

```
totals %>% summarize(percent_men = yes_men/(yes_men+no_men),
 percent_women = yes_women/(yes_women+no_women))
#> percent_men percent_women
#> 1 0.177 0.149
```

respectivamente. ¿Volveríamos a ver esto si asignamos fondos al azar usando como tasa la tasa general?

```
rate <- totals %>%
 summarize(percent_total =
 (yes_men + yes_women) /
 (yes_men + no_men + yes_women + no_women)) %>%
 pull(percent_total)
rate
#> [1] 0.165
```

La prueba de chi-cuadrado responde a esta pregunta. El primer paso es crear la tabla de datos 2x2:

```
two_by_two <- data.frame(awarded = c("no", "yes"),
 men = c(totals$no_men, totals$yes_men),
 women = c(totals$no_women, totals$yes_women))
two_by_two
#> awarded men women
#> 1 no 1345 1011
#> 2 yes 290 177
```

La idea general de la prueba de chi-cuadrado es comparar esta tabla 2x2 con lo que esperamos ver, que sería:

```
data.frame(awarded = c("no", "yes"),
 men = (totals$no_men + totals$yes_men) * c(1 - rate, rate),
 women = (totals$no_women + totals$yes_women) * c(1 - rate, rate))
#> awarded men women
#> 1 no 1365 991
#> 2 yes 270 197
```

Podemos ver que más hombres y menos mujeres de lo esperado recibieron fondos. Sin embargo, bajo la hipótesis nula, estas observaciones son variables aleatorias. La prueba de chi-cuadrado nos dice cuán probable es ver una desviación así de grande o más grande. Esta prueba utiliza un resultado asintótico, similar al CLT, relacionado con las sumas de resultados binarios independientes. La función `chisq.test` de R toma una tabla 2x2 y devuelve los resultados de la prueba:

```
chisq_test <- two_by_two %>% select(-awarded) %>% chisq.test()
```

Vemos que el valor-p es 0.0509:

```
chisq_test$p.value
#> [1] 0.0509
```

#### 15.10.4 Riesgo relativo

Un resumen estadístico informativo para tablas 2x2 es el *riesgo relativo* (*odds ratio* en inglés). Definan las dos variables como  $X = 1$  si eres hombre y 0 de lo contrario, e  $Y = 1$  si recibe financiamiento y 0 de lo contrario. Las probabilidades de obtener fondos si eres hombre se definen así:

$$\Pr(Y = 1 | X = 1)/\Pr(Y = 0 | X = 1)$$

y se pueden calcular así:

```
odds_men <- with(two_by_two, (men[2]/sum(men))/ (men[1]/sum(men)))
odds_men
#> [1] 0.216
```

Y las probabilidades de recibir financiamiento si eres mujer son:

$$\Pr(Y = 1 | X = 0)/\Pr(Y = 0 | X = 0)$$

y se pueden calcular así:

```
odds_women <- with(two_by_two, (women[2]/sum(women))/ (women[1]/sum(women)))
odds_women
#> [1] 0.175
```

El riesgo relativo es la razón de estas dos probabilidades: ¿cuántas veces más grandes son las probabilidades para los hombres que para las mujeres?

```
odds_men/ odds_women
#> [1] 1.23
```

A menudo vemos tablas de 2x2 escritas usando  $a$ ,  $b$ ,  $c$ , y  $d$  como en la siguiente tabla. En este caso, el riesgo relativo es  $\frac{a/c}{b/d}$  que es equivalente a  $(ad)/(bc)$ .

|             | Men | Women |
|-------------|-----|-------|
| Awarded     | a   | b     |
| Not Awarded | c   | d     |

### 15.10.5 Intervalos de confianza para el riesgo relativo

Calcular intervalos de confianza para el riesgo relativo no es matemáticamente sencillo. A diferencia de otras estadísticas, para las cuales podemos derivar aproximaciones útiles de sus distribuciones, el riesgo relativo no es solo una razón, sino una razón de razones. Por lo tanto, no hay una forma sencilla de utilizar, por ejemplo, el CLT.

Sin embargo, la teoría estadística nos dice que cuando las cuatro entradas de la tabla 2x2 son lo suficientemente grandes, entonces el logaritmo del riesgo relativo es aproximadamente normal con error estándar:

$$\sqrt{1/a + 1/b + 1/c + 1/d}$$

Esto implica que un intervalo de confianza de 95% para el logaritmo del riesgo relativo se puede formar por:

$$\log\left(\frac{ad}{bc}\right) \pm 1.96\sqrt{1/a + 1/b + 1/c + 1/d}$$

Exponenciando estos dos números podemos construir un intervalo de confianza del riesgo relativo.

Usando R, podemos calcular este intervalo de confianza de la siguiente manera:

```
log_or <- log(odds_men / odds_women)
se <- two_by_two %>% select(-awarded) %>%
 summarize(se = sqrt(sum(1/men) + sum(1/women))) %>%
 pull(se)
ci <- log_or + c(-1,1) * qnorm(0.975) * se
```

Si queremos convertirlo de nuevo a la escala de riesgo relativo, podemos exponenciar:

```
exp(ci)
#> [1] 1.00 1.51
```

Observen que 1 no está incluido en el intervalo de confianza, lo que significa que el valor-p es menor que 0.05. Podemos confirmar esto usando:

```
2*(1 - pnorm(log_or, 0, se))
#> [1] 0.0454
```

Este es un valor-p un poco diferente al de la prueba de chi-cuadrado. Esto se debe a que estamos utilizando una aproximación asintótica diferente a la distribución nula. Para obtener más información sobre la inferencia y la teoría asintótica del riesgo relativo, consulten el libro *Generalized Linear Models* de McCullagh y Nelder.

### 15.10.6 Corrección de recuento pequeño

Si cualquiera de las celdas de la tabla 2x2 es 0, el logaritmo del riesgo relativo es indefinido. Esto se debe a que si  $a$ ,  $b$ ,  $c$  o  $d$  es 0, el  $\log(\frac{ad}{bc})$  es el logaritmo de 0 o tiene un 0 en el denominador. Para esta situación, es una práctica común evitar los 0 añadiendo 0.5 a cada celda. Esto se conoce como la *corrección de Haldane-Anscombe* y se ha demostrado, tanto en la práctica como en la teoría, que funciona bien.

### 15.10.7 Muestras grandes, valores-p pequeños

Como se mencionó anteriormente, informar solo valores-p no es una forma apropiada de informar los resultados del análisis de datos. En revistas científicas, por ejemplo, algunos estudios parecen enfatizar demasiado los valores-p. Algunos de estos estudios tienen muestras de gran tamaño e indican valores-p impresionantemente pequeños. Sin embargo, cuando uno mira de cerca los resultados, se da cuenta que los riesgos relativos son pequeños: apenas mayores que 1. En este caso, la diferencia puede que no sea *prácticamente significativa* o *científicamente significativa*.

Tengan en cuenta que la relación entre el riesgo relativo y el valor-p no es una correspondencia uno-a-uno. La relación depende del tamaño de la muestra. Por lo tanto, un valor-p muy pequeño no necesariamente significa un riesgo relativo muy grande. Observen lo que sucede con el valor-p si multiplicamos nuestra tabla 2x2 por 10, lo cual no cambia el riesgo relativo:

```
two_by_two %>% select(-awarded) %>%
 mutate(men = men*10, women = women*10) %>%
 chisq.test() %>% .\$p.value
#> [1] 2.63e-10
```

---

## 15.11 Ejercicios

1. Una atleta famosa tiene una carrera impresionante, ganando 70% de los 500 partidos de su carrera. Sin embargo, critican a esta atleta porque en eventos importantes, como los Juegos Olímpicos, tiene un récord perdedor de 8 victorias y 9 derrotas. Realice una prueba de chi-cuadrado para determinar si este récord se debe simplemente al azar en vez de no competir bien bajo presión.
2. ¿Por qué usamos la prueba de chi-cuadrado en lugar de la prueba exacta de Fisher en el ejercicio anterior?
  - a. Realmente no importa ya que dan exactamente el mismo valor-p.
  - b. La prueba exacta de Fisher y la de chi-cuadrado son nombres diferentes para la misma prueba.
  - c. Debido a que la suma de las filas y columnas de la tabla 2x2 no son fijas, la distribución hipergeométrica no es una suposición apropiada para la hipótesis nula. Por esta razón, la prueba exacta de Fisher rara vez es aplicable a datos observacionales.

- d. Porque la prueba de chi-cuadrado se ejecuta más rápido.
3. Calcule el riesgo relativo de “perder bajo presión” junto con un intervalo de confianza.
4. Observe que el valor-p es mayor que 0.05, pero el intervalo de confianza de 95% no incluye 1. ¿Qué explica esto?
  - a. Cometimos un error en nuestro código.
  - b. Estas no son estadísticas t, por lo que no aplica la conexión entre el valor-p y los intervalos de confianza.
  - c. Se utilizan diferentes aproximaciones para el valor-p y el cálculo del intervalo de confianza. Si tuviéramos un tamaño de muestra más grande, la coincidencia sería mejor.
  - d. Deberíamos usar la prueba exacta de Fisher para obtener intervalos de confianza.
5. Multiplique la tabla 2x2 por dos y vea si el valor-p y el intervalo de confianza coinciden mejor.

# 16

## Modelos estadísticos

“Todos los modelos están equivocados, pero algunos son útiles.” –George E. P. Box

El día antes de las elecciones presidenciales del 2008, FiveThirtyEight de Nate Silver declaró que “Barack Obama parece estar listo para una victoria electoral decisiva”. Fueron hasta más lejos y predijeron que Obama ganaría las elecciones con 349 votos electorales a 189 y el voto popular por un margen de 6.1%. FiveThirtyEight también añadió una declaración probabilística a su predicción declarando que Obama tenía una probabilidad de 91% de ganar las elecciones. Las predicciones fueron bastante precisas y, en los resultados finales, Obama ganó el colegio electoral 365 a 173 y el voto popular por una diferencia de 7.2%. El desempeño de FiveThirtyEight en las elecciones del 2008 atrajo la atención de expertos políticos y personalidades de la televisión. Cuatro años después, la semana antes de las elecciones presidenciales del 2012, Nate Silver de FiveThirtyEight le estaba dando a Obama una probabilidad de 90% de ganar a pesar de que muchos de los expertos pensaban que los resultados finales estarían más cerca. El comentarista político Joe Scarborough dijo durante su show<sup>1</sup>:

Cualquiera que piense que esta elección no está cerrada en este momento es un tremendo ideólogo ... son un chiste.

A lo que Nate Silver respondió a través de Twitter:

Si cree que la elección está cerrada, apostemos. Si Obama gana, Ud. dona \$1,000 a la Cruz Roja Americana. Si Romney gana, yo lo hago. ¿De acuerdo?

En 2016, Silver no estaba tan seguro y le dio a Hillary Clinton solo una probabilidad de 71% de ganar. En cambio, la mayoría de los otros pronosticadores estaban casi seguros de que ella ganaría. Ella perdió. Pero 71% sigue siendo más de 50%, ¿se equivocó el Sr. Silver? Además, ¿qué significa la probabilidad en este contexto? ¿Alguien está tirando dados?

En este capítulo demostraremos cómo los *agregadores de encuestas*, como FiveThirtyEight, recopilaron y combinaron datos informados por diferentes expertos para producir mejores predicciones. Presentaremos las ideas detrás de los *modelos estadísticos*, también conocidos como *modelos de probabilidad*, que utilizaron los agregadores de encuestas para mejorar los pronósticos electorales en comparación a las encuestas individuales. En este capítulo, motivamos los modelos, construyendo sobre los conceptos de inferencia estadística que aprendimos en el Capítulo 15. Comenzamos con modelos relativamente sencillos, tomando en cuenta que el ejercicio real de la ciencia de datos de pronosticar elecciones involucra algunos modelos bastante complejos, que presentamos al final del capítulo en la Sección 16.8.

<sup>1</sup><https://www.youtube.com/watch?v=TbKkjm-gheY>

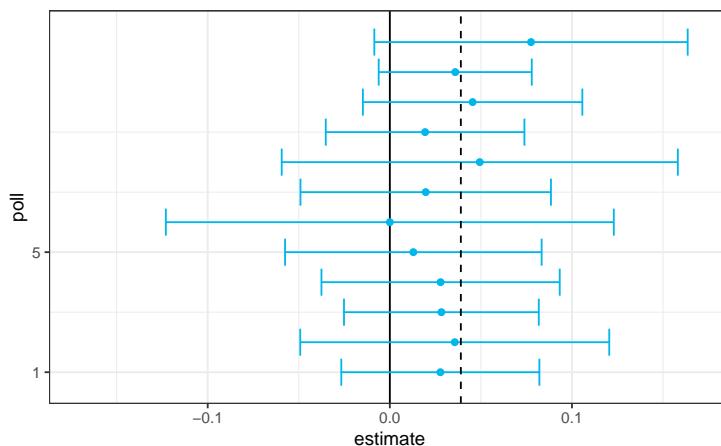
## 16.1 Agregadores de encuestas

Como describimos anteriormente, unas semanas antes de las elecciones del 2012, Nate Silver le estaba dando a Obama una probabilidad de 90% de ganar. ¿Por qué tenía tanta confianza el señor Silver? Utilizaremos una simulación Monte Carlo para ilustrar la idea que tuvo el Sr. Silver y que otros no vieron. Para hacer esto, generamos resultados para 12 encuestas realizadas la semana anterior a las elecciones. Imitaremos tamaños de muestra de encuestas reales y construiremos e informaremos intervalos de confianza de 95% para cada una de las 12 encuestas. Guardaremos los resultados de esta simulación en un set de datos y añadiremos una columna de ID de encuesta.

```
library(tidyverse)
library(dslabs)
d <- 0.039
Ns <- c(1298, 533, 1342, 897, 774, 254, 812, 324, 1291, 1056, 2172, 516)
p <- (d + 1)/ 2

polls <- map_df(Ns, function(N) {
 x <- sample(c(0,1), size=N, replace=TRUE, prob=c(1-p, p))
 x_hat <- mean(x)
 se_hat <- sqrt(x_hat * (1 - x_hat)/ N)
 list(estimate = 2 * x_hat - 1,
 low = 2*(x_hat - 1.96*se_hat) - 1,
 high = 2*(x_hat + 1.96*se_hat) - 1,
 sample_size = N)
}) %>% mutate(poll = seq_along(Ns))
```

Aquí tenemos una visualización que muestra los intervalos que los encuestadores reportaron para la diferencia entre Obama y Romney:



No es sorprendente que las 12 encuestas informen intervalos de confianza que incluyen el resultado de la noche electoral (línea discontinua). Sin embargo, las 12 encuestas también

incluyen 0 (línea negra sólida). Por lo tanto, si se les pide individualmente una predicción, los encuestadores tendrían que decir: las probabilidades están parejas. A continuación describimos una idea clave que no consideraron.

Los agregadores de encuestas, como Nate Silver, se dieron cuenta de que al combinar los resultados de diferentes encuestas, la precisión podría mejorar enormemente. Al hacer esto, estamos llevando a cabo una encuesta con un gran tamaño de muestra. Por lo tanto, podemos informar un intervalo de confianza menor de 95% y una predicción más precisa.

Aunque como agregadores no tenemos acceso a los datos sin procesar de la encuesta, podemos usar las matemáticas para reconstruir lo que habríamos obtenido si hubiéramos hecho una encuesta grande con:

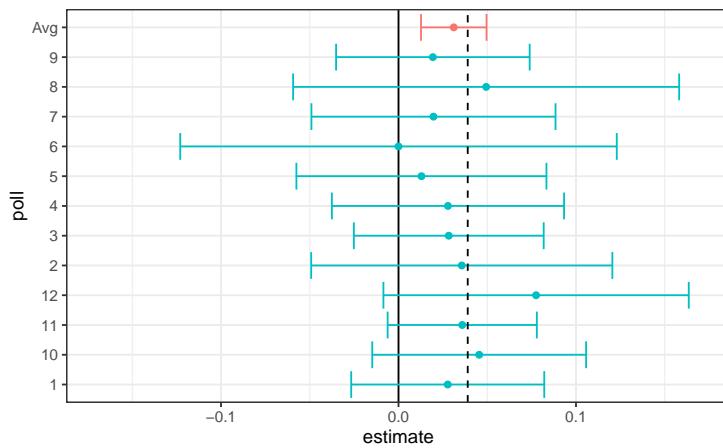
```
sum(polls$sample_size)
#> [1] 11269
```

participantes. Básicamente, construimos un estimador de la diferencia, llamémosla  $d$ , con un promedio ponderado de la siguiente manera:

```
d_hat <- polls %>%
 summarize(avg = sum(estimate*sample_size)/ sum(sample_size)) %>%
 pull(avg)
```

Una vez que tengamos un estimador de  $d$ , podemos construir un estimador de la proporción votando por Obama, que luego podemos usar para estimar el error estándar. Tan pronto hacemos esto, vemos que nuestro margen de error es 0.018.

Por lo tanto, podemos predecir que la diferencia será 3.1 más o menos 1.8, que no solo incluye el resultado real que observamos en la noche de las elecciones, sino que está bastante lejos de incluir 0. Al combinar las 12 encuestas, acabamos seguros de que Obama ganará el voto popular.



Por supuesto, esto fue solo una simulación para ilustrar la idea. El ejercicio real de la ciencia de datos de pronosticar elecciones es mucho más complicado y requiere modelos estadísticos. A continuación explicamos cómo los encuestadores ajustan los modelos multinivel a los datos y los utilizan para pronosticar los resultados electorales. En las elecciones presidenciales

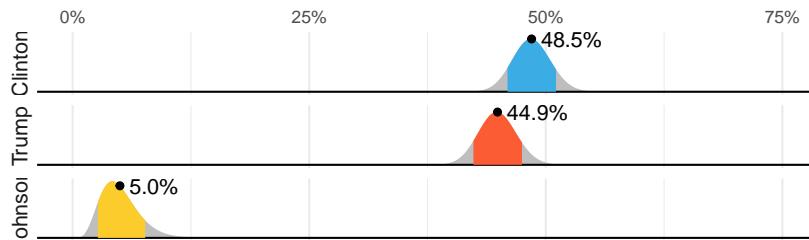
estadounidenses del 2008 y 2012, Nate Silver utilizó este enfoque para hacer una predicción casi perfecta y callar a los expertos.

Desde las elecciones del 2008, otras organizaciones han establecido sus propios grupos de pronóstico de elecciones que, como el de Nate Silver, agregan datos de encuestas y utilizan modelos estadísticos para hacer predicciones. En 2016, los pronosticadores subestimaron por mucho las probabilidades de Trump de ganar. El día antes de las elecciones, el *New York Times* informó<sup>2</sup> las siguientes probabilidades de que Hillary Clinton ganara la presidencia:

|          | NYT | 538 | HuffPost | PW  | PEC  | DK  | Cook     | Roth     |
|----------|-----|-----|----------|-----|------|-----|----------|----------|
| Win Prob | 85% | 71% | 98%      | 89% | >99% | 92% | Lean Dem | Lean Dem |

Por ejemplo, el Consorcio Electoral de Princeton (*Princeton Election Consortium* en inglés) le dio a Trump menos de 1% de probabilidad de ganar, mientras que el *Huffington Post* le dio una probabilidad de 2%. Por el contrario, FiveThirtyEight le daba a Trump una probabilidad de ganar de 29%, más que la probabilidad de lanzar dos monedas y obtener dos caras. De hecho, cuatro días antes de las elecciones, FiveThirtyEight publicó un artículo titulado *Trump is Just A Normal Polling Error Behind Clinton*<sup>3</sup>. Al entender los modelos estadísticos y cómo los pronosticadores los usan, comenzaremos a entender cómo sucedió esto.

Aunque no tan interesante como predecir el colegio electoral, para fines ilustrativos comenzaremos analizando las predicciones para el voto popular. FiveThirtyEight predijo una ventaja de 3.6% para Clinton<sup>4</sup> y su intervalo de confianza incluyó el resultado real de una diferencia de 2.1% (48.2% a 46.1%). Además, FiveThirtyEight estuvo mucho más seguro sobre la posibilidad de que Clinton ganara el voto popular, dándole una probabilidad de 81.4%. Su predicción se resumió con un gráfico como este:



Las áreas coloreadas representan valores con una probabilidad de 80% de incluir el resultado real, según el modelo de FiveThirtyEight.

Presentamos datos reales de las elecciones presidenciales de EE. UU. del 2016 para mostrar cómo se motivan y se construyen los modelos para producir estas predicciones. Para comprender la declaración “81.4% de probabilidad”, necesitamos describir las estadísticas bayesianas, lo que hacemos en las Secciones 16.4 y 16.8.1.

<sup>2</sup><https://www.nytimes.com/interactive/2016/upshot/presidential-polls-forecast.html>

<sup>3</sup><https://fivethirtyeight.com/features/trump-is-just-a-normal-polling-error-behind-Clinton/>

<sup>4</sup><https://projects.fivethirtyeight.com/2016-election-forecast/>

### 16.1.1 Datos de encuesta

Utilizamos datos públicos de encuestas organizados por FiveThirtyEight para las elecciones presidenciales del 2016. Los datos se incluyen como parte del paquete **dslabs**:

```
data(polls_us_election_2016)
```

La tabla incluye los resultados de las encuestas nacionales, así como las encuestas estatales, tomadas durante el año anterior a la elección. Para este primer ejemplo, filtraremos los datos para incluir encuestas nacionales realizadas durante la semana previa a las elecciones. También eliminamos las encuestas que FiveThirtyEight ha determinado que no son confiables y calificaron con una “B” o menos. Algunas encuestas no han sido calificadas e incluimos aquellas:

```
polls <- polls_us_election_2016 %>%
 filter(state == "U.S." & enddate >= "2016-10-31" &
 (grade %in% c("A+","A","A-","B+") | is.na(grade)))
```

Agregamos el estimador de la diferencia:

```
polls <- polls %>%
 mutate(spread = rawpoll_clinton/100 - rawpoll_trump/100)
```

Para este ejemplo, suponemos que solo hay dos partes y llamaremos  $p$  a la proporción de votos para Clinton y  $1 - p$  a la proporción votando por Trump. Estamos interesados en la diferencia  $2p - 1$ . Llamemos a la diferencia  $d$ .

Tenemos 49 estimadores de la diferencia. La teoría que aprendimos nos dice que estos estimadores son una variable aleatoria con una distribución de probabilidad que es aproximadamente normal. El valor esperado es la diferencia de la noche electoral  $d$  y el error estándar es  $2\sqrt{p(1-p)/N}$ . Suponiendo que el modelo de urna que describimos anteriormente es bueno, podemos usar esta información para construir un intervalo de confianza basado en los datos agregados. El estimador de la diferencia es:

```
d_hat <- polls %>%
 summarize(d_hat = sum(spread * samplesize) / sum(samplesize)) %>%
 pull(d_hat)
```

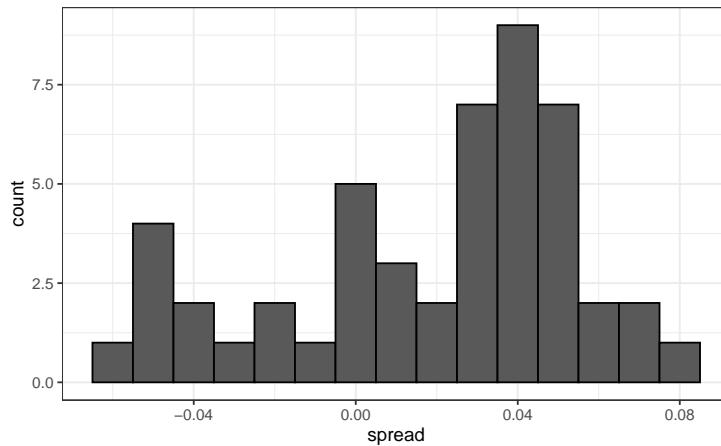
y el error estándar es:

```
p_hat <- (d_hat+1)/2
moe <- 1.96 * 2 * sqrt(p_hat * (1 - p_hat)) / sum(polls$samplesize)
moe
#> [1] 0.00662
```

Entonces informamos una diferencia de 1.43% con un margen de error de 0.66%. En la noche de las elecciones, descubrimos que el porcentaje real era 2.1%, que está fuera de un intervalo de confianza de 95%. ¿Qué pasó?

Un histograma de las variabilidades reportadas muestra un problema:

```
polls %>%
 ggplot(aes(spread)) +
 geom_histogram(color="black", binwidth = .01)
```



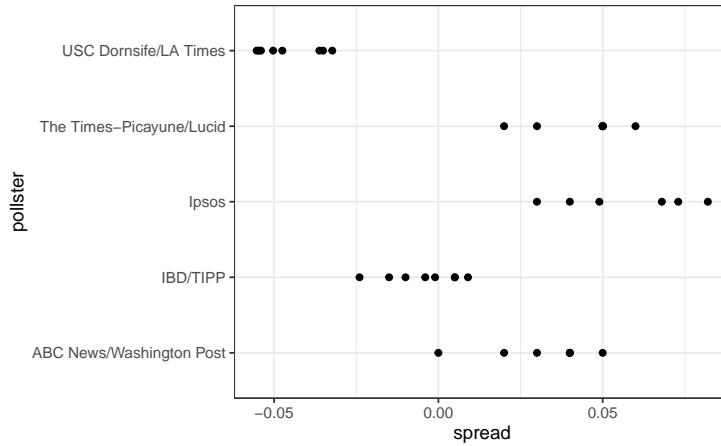
Los datos no parecen estar distribuidos normalmente y el error estándar parece ser mayor que 0.007. La teoría no está funcionando bien aquí.

### 16.1.2 Sesgo de los encuestadores

Observen que varios encuestadores están involucrados y algunos toman varias encuestas por 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
#> # ... with 10 more rows
```

Visualicemos los datos de los encuestadores que sondean regularmente:



Este gráfico revela un resultado inesperado. Primero, consideren que el error estándar predicho por la teoría para cada encuesta:

```
polls %>% group_by(pollster) %>%
 filter(n() >= 6) %>%
 summarize(se = 2 * sqrt(p_hat * (1-p_hat)/ median(samplesize)))
#> # A tibble: 5 x 2
#> pollster se
#> <fct> <dbl>
#> 1 ABC News/Washington Post 0.0265
#> 2 IBD/TIPP 0.0333
#> 3 Ipsos 0.0225
#> 4 The Times-Picayune/Lucid 0.0196
#> 5 USC Dornsife/LA Times 0.0183
```

está entre 0.018 y 0.033, que concuerda con la variación de encuesta a encuesta que vemos para cada encuestador. Sin embargo, parece haber diferencias *entre los encuestadores*. Observen, por ejemplo, cómo el encuestador USC Dornsife/LA Times predice una ventaja de 4% para Trump, mientras que Ipsos predice una ventaja mayor de 5% para Clinton. La teoría que aprendimos no dice nada acerca de diferentes encuestadores que producen encuestas con diferentes valores esperados. Todas las encuestas deben tener el mismo valor esperado. FiveThirtyEight se refiere a estas diferencias como “house effects”. También las llamamos *sesgo de encuestadores*.

En la siguiente sección, en lugar de utilizar la teoría del modelo de urna, desarrollaremos un modelo basado en datos.

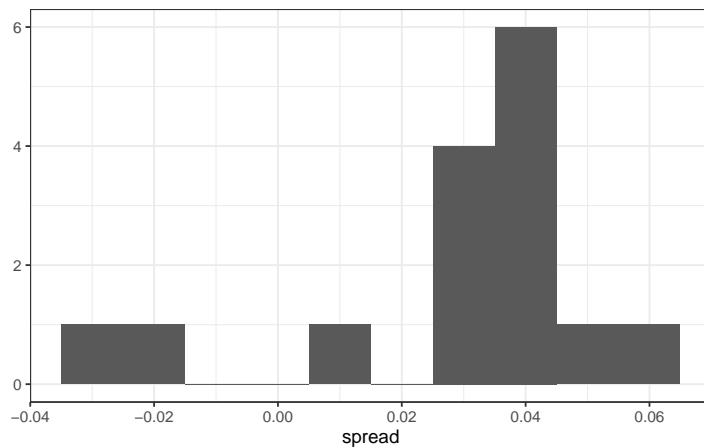
## 16.2 Modelos basados en datos

Para cada encuestador, recopilemos el último resultado que informan antes de las elecciones:

```
one_poll_per_pollster <- polls %>% group_by(pollster) %>%
 filter(enddate == max(enddate)) %>%
 ungroup()
```

Aquí hay un histograma de los datos para estos 15 encuestadores:

```
qplot(spread, data = one_poll_per_pollster, binwidth = 0.01)
```



En la sección anterior, vimos que usar la teoría del modelo de urna para combinar estos resultados a veces no es apropiado debido al efecto de encuestador. En cambio, modelaremos estos datos de las diferencias directamente.

El nuevo modelo también puede considerarse como un modelo de urna, aunque la conexión no es tan directa. En lugar de 0s (republicanos) y 1s (demócratas), nuestra urna ahora contiene los resultados de las encuestas de todos los posibles encuestadores. Suponemos que el valor esperado de nuestra urna es la diferencia real  $d = 2p - 1$ .

Dado que en lugar de 0s y 1s, nuestra urna contiene números continuos entre -1 y 1, la desviación estándar de la urna ya no es  $\sqrt{p(1-p)}$ . En vez de la variabilidad del muestreo de votantes, el error estándar ahora incluye la variabilidad entre encuestadores. Nuestra nueva urna también incluye la variabilidad de muestreo del sondeo. De cualquier manera, esta desviación estándar ahora es un parámetro desconocido. En los libros de texto de estadística, el símbolo griego  $\sigma$  se usa para representar este parámetro.

En resumen, tenemos dos parámetros desconocidos: el valor esperado  $d$  y la desviación estándar  $\sigma$ .

Nuestra tarea es estimar  $d$ . Como modelamos los valores observados  $X_1, \dots, X_N$  como una muestra aleatoria de la urna, el CLT aún podría funcionar en esta situación porque es un promedio de variables aleatorias independientes. Para un tamaño de muestra suficientemente grande  $N$ , la distribución de probabilidad del promedio de la muestra  $\bar{X}$  es aproximadamente normal con valor esperado  $\mu$  y error estándar  $\sigma/\sqrt{N}$ . Si estamos dispuestos a considerar  $N = 15$  como suficientemente grande, podemos usar esto para construir intervalos de confianza.

Un problema es que no sabemos  $\sigma$ . Pero la teoría nos dice que podemos estimar el modelo de urna  $\sigma$  con la desviación estándar de la muestra definida como  $s = \sqrt{\sum_{i=1}^N (X_i - \bar{X})^2 / (N - 1)}$ .

A diferencia de la definición de desviación estándar de la población, ahora dividimos por  $N - 1$ . Esto hace  $s$  un mejor estimador de  $\sigma$ . Hay una explicación matemática para esto, que se enseña en la mayoría de los libros de texto de estadística, pero no la cubrimos aquí.

En R, la función `sd` calcula la desviación estándar de la muestra:

```
sd(one_poll_per_pollster$spread)
#> [1] 0.0242
```

Ahora estamos listos para formar un nuevo intervalo de confianza basado en nuestro nuevo modelo y en datos:

```
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
```

Nuestro intervalo de confianza ahora es más amplio ya que incorpora la variabilidad de encuestador. Incluye el resultado de la noche electoral de 2.1%. Además, observen que era lo suficientemente pequeño como para no incluir 0, lo que significa que estábamos seguros de que Clinton ganaría el voto popular.

¿Estamos listos ahora para declarar una probabilidad de que Clinton gane el voto popular? Aún no. En nuestro modelo,  $d$  es un parámetro fijo, por lo que no podemos hablar de probabilidades. Para ofrecer probabilidades, necesitaremos aprender sobre las estadísticas bayesianas.

### 16.3 Ejercicios

Hemos estado utilizando modelos de urna para motivar el uso de modelos de probabilidad. La mayoría de las aplicaciones de ciencia de datos no están relacionadas con datos obtenidos de urnas. Más comunes son los datos que provienen de individuos. La razón por la que la probabilidad importa aquí es porque los datos provienen de una muestra aleatoria. La muestra aleatoria se toma de una población y la urna sirve como analogía para la población.

Volvamos al set de datos de alturas. Suponga que consideramos a los varones de nuestra clase como la población.

```
library(dslabs)
data(heights)
x <- heights %>% filter(sex == "Male") %>%
 pull(height)
```

1. Matemáticamente hablando,  $\mathbf{x}$  es nuestra población. Usando la analogía de la urna, tenemos una urna con los valores de  $\mathbf{x}$  dentro de ella. ¿Cuáles son el promedio y la desviación estándar de nuestra población?
2. Llame al promedio de población calculado arriba  $\mu$  y la desviación estándar  $\sigma$ . Ahora tome una muestra de tamaño 50, con reemplazo, y construya un estimador para  $\mu$  y  $\sigma$ .
3. ¿Qué nos dice la teoría sobre el promedio de la muestra  $\bar{X}$  y como se relaciona con  $\mu$ ?
  - a. Es prácticamente idéntico a  $\mu$ .
  - b. Es una variable aleatoria con valor esperado  $\mu$  y error estándar  $\sigma/\sqrt{N}$ .
  - c. Es una variable aleatoria con valor esperado  $\mu$  y error estándar  $\sigma$ .
  - d. No contiene información.
4. Entonces, ¿cómo es esto útil? Vamos a utilizar un ejemplo simplificado pero ilustrativo. Suponga que queremos saber la altura promedio de nuestros estudiantes varones, pero solo llegamos a medir 50 de los 708. Usaremos  $\bar{X}$  como nuestro estimador. Sabemos por la respuesta al ejercicio 3 que el estimador estándar de nuestro error  $\bar{X} - \mu$  es  $\sigma/\sqrt{N}$ . Queremos calcular esto, pero no sabemos  $\sigma$ . Según lo que se describe en esta sección, indique su estimador de  $\sigma$ .
5. Ahora que tenemos un estimador de  $\sigma$ , llamemos a nuestro estimador  $s$ . Construya un intervalo de confianza de 95% para  $\mu$ .
6. Ahora ejecute una simulación Monte Carlo en la que calcula 10,000 intervalos de confianza como acaba de hacer. ¿Qué proporción de estos intervalos incluye  $\mu$ ?
7. En esta sección, discutimos el sesgo de encuestador. Utilizamos la visualización para motivar la presencia de tal sesgo. Aquí le daremos un tratamiento más riguroso. Consideraremos dos encuestadores que realizaron encuestas diarias. Examinaremos las encuestas nacionales del mes anterior a las elecciones.

```
data(polls_us_election_2016)
polls <- polls_us_election_2016 %>%
 filter(pollster %in% c("Rasmussen Reports/Pulse Opinion Research",
 "The Times-Picayune/Lucid") &
 enddate >= "2016-10-15" &
 state == "U.S.") %>%
 mutate(spread = rawpoll_clinton/100 - rawpoll_trump/100)
```

Queremos contestar la pregunta: ¿hay un sesgo en la encuesta? Haga un gráfico que muestre la diferencia para cada encuesta.

8. Los datos parecen sugerir que hay una diferencia. Sin embargo, estos datos están sujetos a variabilidad. Quizás las diferencias que observamos se deben al azar.

La teoría del modelo de urna no dice nada sobre el efecto del encuestador. Bajo el modelo de urna, ambos encuestadores tienen el mismo valor esperado: la diferencia del día de las elecciones, que llamamos  $d$ .

Para responder a la pregunta “¿hay un modelo de urna?”, modelaremos los datos observados  $Y_{i,j}$  de la siguiente manera:

$$Y_{i,j} = d + b_i + \varepsilon_{i,j}$$

con  $i = 1, 2$  indexando los dos encuestadores,  $b_i$  el sesgo para el encuestador  $i$  y  $\varepsilon_{ij}$  representando la variabilidad aleatoria de las encuestas. Suponemos que los  $\varepsilon$  son independientes entre sí, tienen valor esperado 0 y desviación estándar  $\sigma_i$  independientemente de  $j$ .

¿Cuál de las siguientes mejor representa nuestra pregunta?

- ¿Es  $\varepsilon_{i,j} = 0$ ?
- ¿Cuán cerca están los  $Y_{i,j}$  a  $d$ ?
- ¿Es  $b_1 \neq b_2$ ?
- ¿Son  $b_1 = 0$  y  $b_2 = 0$ ?

9. En el lado derecho de este modelo solo  $\varepsilon_{i,j}$  es una variable aleatoria. Los otros dos son constantes. ¿Cuál es el valor esperado de  $Y_{1,j}$ ?

10. Suponga que definimos  $\bar{Y}_1$  como el promedio de los resultados de la encuesta del primer encuestador,  $Y_{1,1}, \dots, Y_{1,N_1}$  con  $N_1$  el número de encuestas realizadas por el primer encuestador:

```
polls %>%
 filter(pollster=="Rasmussen Reports/Pulse Opinion Research") %>%
 summarize(N_1 = n())
```

¿Cuál es el valor esperado de  $\bar{Y}_1$ ?

11. ¿Cuál es el error estándar de  $\bar{Y}_1$ ?

12. Suponga que definimos  $\bar{Y}_2$  como el promedio de los resultados de la encuesta de la primera encuesta,  $Y_{2,1}, \dots, Y_{2,N_2}$  con  $N_2$  el número de encuestas realizadas por el primer encuestador. ¿Cuál es el valor esperado  $\bar{Y}_2$ ?

13. ¿Cuál es el error estándar de  $\bar{Y}_2$ ?

14. Usando lo que aprendimos al responder a las preguntas anteriores, ¿cuál es el valor esperado de  $\bar{Y}_2 - \bar{Y}_1$ ?

15. Usando lo que aprendimos al responder a las preguntas anteriores, ¿cuál es el error estándar de  $\bar{Y}_2 - \bar{Y}_1$ ?

16. La respuesta a la pregunta anterior depende de  $\sigma_1$  y  $\sigma_2$ , que no sabemos. Aprendimos que podemos estimarlos con la desviación estándar de la muestra. Escriba un código que calcule estos dos estimadores.

17. ¿Qué nos dice el CLT sobre la distribución de  $\bar{Y}_2 - \bar{Y}_1$ ?

- Nada porque este no es el promedio de una muestra.
- Como el  $Y_{ij}$  son aproximadamente normales, también lo son los promedios.
- Como  $\bar{Y}_2$  y  $\bar{Y}_1$  son promedios de muestras, si suponemos que  $N_2$  y  $N_1$  son lo suficientemente grandes, cada uno es aproximadamente normal. La diferencia de normales también es normal.
- Los datos no son 0 o 1, por lo que el CLT no se aplica.

18. Hemos construido una variable aleatoria que tiene un valor esperado  $b_2 - b_1$ , la diferencia de sesgo del encuestador. Si nuestro modelo funciona, entonces esta variable aleatoria tiene una distribución aproximadamente normal y sabemos su error estándar. El error estándar

depende de  $\sigma_1$  y  $\sigma_2$ , pero podemos usar las desviaciones estándar de muestra que calculamos anteriormente. Comenzamos preguntando: ¿ $b_2 - b_1$  es diferente de 0? Use toda la información que hemos aprendido anteriormente para construir un intervalo de confianza de 95% para la diferencia  $b_2 - b_1$ .

19. El intervalo de confianza nos dice que hay un efecto encuestador relativamente fuerte que resulta en una diferencia de aproximadamente 5%. La variabilidad aleatoria no parece explicarlo. Podemos calcular un valor-p para explicar el hecho de que el azar no lo explica. ¿Cuál es el valor-p?
20. La estadística formada al dividir nuestro estimador de  $b_2 - b_1$  por su error estándar estimado:

$$\frac{\bar{Y}_2 - \bar{Y}_1}{\sqrt{s_2^2/N_2 + s_1^2/N_1}}$$

se llama la estadística t. Ahora observe que tenemos más de dos encuestadores. También podemos probar para el efecto de encuestador utilizando todos los encuestadores, no solo dos. La idea es comparar la variabilidad entre encuestas con la variabilidad dentro de las encuestas. De hecho, podemos construir estadísticas para probar los efectos y aproximar su distribución. El área de estadísticas que hace esto se llama *análisis de la varianza* (ANOVA por sus siglas en inglés). No lo cubrimos aquí, pero ANOVA provee un set muy útil de herramientas para responder a preguntas como: ¿hay un efecto encuestador?

Para este ejercicio, cree una nueva tabla:

```
polls <- polls_us_election_2016 %>%
 filter(enddate >= "2016-10-15" &
 state == "U.S.") %>%
 group_by(pollster) %>%
 filter(n() >= 5) %>%
 mutate(spread = rawpoll_clinton/100 - rawpoll_trump/100) %>%
 ungroup()
```

Calcule el promedio y la desviación estándar para cada encuestador y examine la variabilidad entre los promedios y cómo se compara con la variabilidad dentro de los encuestadores, resumida por la desviación estándar.

## 16.4 Estadísticas bayesianas

¿Qué significa que un pronosticador electoral nos diga que un candidato tiene un 90% probabilidad de ganar? En el contexto del modelo de urna, esto sería equivalente a afirmar que la probabilidad  $p > 0.5$  es 90%. Sin embargo, como discutimos anteriormente, en el modelo de urna  $p$  es un parámetro fijo y no tiene sentido hablar de probabilidad. Con estadísticas bayesianas, modelamos  $p$  como variable aleatoria y, por lo tanto, una declaración como “90% probabilidad de ganar” es coherente.

Los pronosticadores también usan modelos para describir la variabilidad en diferentes niveles. Por ejemplo, la variabilidad de muestreo, la variabilidad de encuestador a encuestador,

la variabilidad diaria y la variabilidad de elección a elección. Uno de los enfoques más exitosos utilizados para esto son los modelos jerárquicos, que pueden explicarse en el contexto de las estadísticas bayesianas.

En este capítulo describimos brevemente las estadísticas bayesianas. Para una exploración más profunda de este tema, recomendamos uno de los siguientes libros de texto:

- Berger JO (1985). *Statistical Decision Theory and Bayesian Analysis*, 2nd edition. Springer-Verlag.
- Lee PM (1989). *Bayesian Statistics: An Introduction*. Oxford.

#### 16.4.1 Teorema de Bayes

Comenzamos describiendo el teorema de Bayes. Hacemos esto usando una prueba hipotética de fibrosis quística como ejemplo. Supongan que una prueba de fibrosis quística tiene una exactitud de 99%. Vamos a utilizar la siguiente notación:

$$\text{Prob}(+ \mid D = 1) = 0.99, \text{Prob}(- \mid D = 0) = 0.99$$

con + significando una prueba positiva y  $D$  representando si realmente tiene la enfermedad (1) o no (0).

Supongan que seleccionamos una persona al azar y dan positivo. ¿Cuál es la probabilidad de que tengan la enfermedad? Escribimos esto como  $\text{Prob}(D = 1 \mid +)$ . La tasa de fibrosis quística es de 1 en 3,900, lo que implica que  $\text{Prob}(D = 1) = 0.00025$ . Para responder a esta pregunta, utilizaremos el teorema de Bayes, que por lo general nos dice que:

$$\text{Pr}(A \mid B) = \frac{\text{Pr}(B \mid A)\text{Pr}(A)}{\text{Pr}(B)}$$

Esta ecuación aplicada a nuestro problema se convierte en:

$$\begin{aligned} \text{Pr}(D = 1 \mid +) &= \frac{P(+ \mid D = 1) \cdot P(D = 1)}{\text{Pr}(+)} \\ &= \frac{\text{Pr}(+ \mid D = 1) \cdot P(D = 1)}{\text{Pr}(+ \mid D = 1) \cdot P(D = 1) + \text{Pr}(+ \mid D = 0) \text{Pr}(D = 0)} \end{aligned}$$

usando estos números obtenemos:

$$\frac{0.99 \cdot 0.00025}{0.99 \cdot 0.00025 + 0.01 \cdot (0.99975)} = 0.02$$

Esto dice que a pesar de que la prueba tiene una precisión de 0.99, la probabilidad de tener la enfermedad dado una prueba positiva es solo 0.02. Aunque parezca contrario al sentido común, la razón de esto es porque tenemos que considerar la muy rara probabilidad de que una persona, elegida al azar, tenga la enfermedad. Para ilustrar esto, ejecutamos una simulación Monte Carlo.

## 16.5 Simulación del teorema de Bayes

La siguiente simulación está destinada a ayudarles visualizar el teorema de Bayes. Comenzamos seleccionando aleatoriamente 100,000 personas de una población en la cual la enfermedad en cuestión tiene una prevalencia de 1 en 4,000.

```
prev <- 0.00025
N <- 100000
outcome <- sample(c("Disease", "Healthy"), N, replace = TRUE,
 prob = c(prev, 1 - prev))
```

Recuerden que hay muy pocas personas con la enfermedad:

```
N_D <- sum(outcome == "Disease")
N_D
#> [1] 23
N_H <- sum(outcome == "Healthy")
N_H
#> [1] 99977
```

Además, hay muchas sin la enfermedad, lo que hace más probable que veamos algunos falsos positivos dado que la prueba no es perfecta. Ahora cada persona se hace la prueba, que acierta 99% del tiempo:

```
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))
```

Debido a que hay muchos más controles que casos, incluso con una tasa baja de falsos positivos obtenemos más controles que los casos en el grupo que dio positivo:

```
table(outcome, test)
#> test
#> outcome - +
#> Disease 0 23
#> Healthy 99012 965
```

De esta tabla, vemos que la proporción de pruebas positivas que tienen la enfermedad es 23 de 988. Podemos ejecutar esto una y otra vez para ver que, de hecho, la probabilidad converge a aproximadamente 0.022.

### 16.5.1 Bayes en la práctica

José Iglesias es un jugador de béisbol profesional. En abril de 2013, cuando comenzaba su carrera, se desempeñaba bastante bien:

| Mes   | At Bats | H | AVG  |
|-------|---------|---|------|
| abril | 20      | 9 | .450 |

La estadística del promedio de bateo (AVG) es una forma de medir éxito. En términos generales, nos dice la tasa de éxito al batear. Un AVG de .450 significa que José ha tenido éxito el 45% de las veces que ha bateado (At Bats) que es bastante alto, históricamente hablando. Tengan en cuenta que nadie ha terminado una temporada con un AVG de .400 o más desde que Ted Williams lo hizo en 1941. Para ilustrar la forma en que los modelos jerárquicos son eficaces, intentaremos predecir el promedio de bateo de José al final de la temporada. Recuerden que en una temporada típica, los jugadores tienen alrededor de 500 turnos al bate.

Con las técnicas que hemos aprendido hasta ahora, denominadas *técnicas frecuentistas*, lo mejor que podemos hacer es ofrecer un intervalo de confianza. Podemos pensar en los resultados de batear como un binomio con una tasa de éxito de  $p$ . Entonces, si la tasa de éxito es .450, el error estándar de solo 20 turnos al bate es:

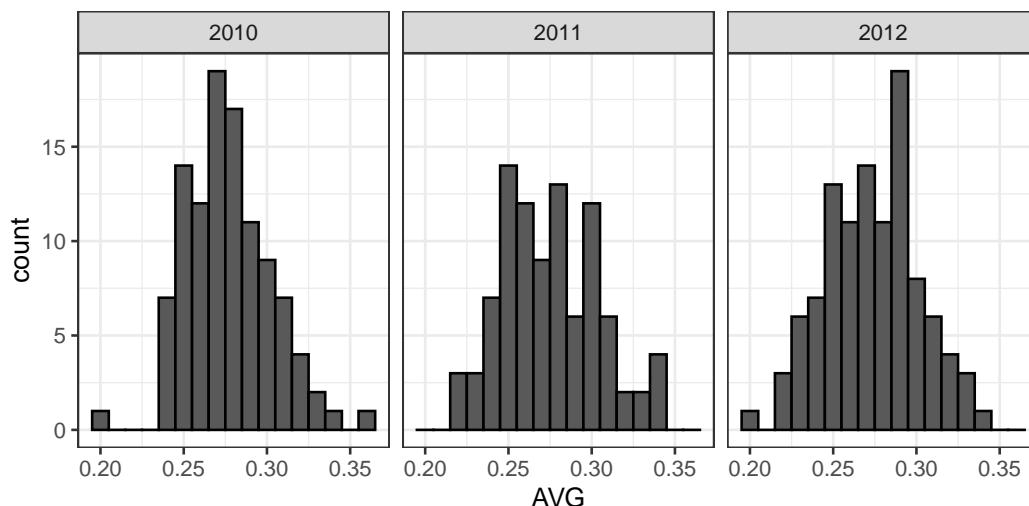
$$\sqrt{\frac{.450(1 - .450)}{20}} = .111$$

Esto significa que nuestro intervalo de confianza es  $.450 - .222$  a  $.450 + .222$  o  $.228$  a  $.672$ .

Esta predicción tiene dos problemas. Primero, es muy grande, por lo que no es muy útil. Segundo, está centrada en .450, lo que implica que nuestra mejor conjetura es que este nuevo jugador romperá el récord de Ted Williams.

Sin embargo, para los fanáticos del béisbol, esta última afirmación no tiene sentido. Los fanáticos implícitamente emplean un modelo jerárquico que toma en cuenta la información de años de seguir el béisbol. Aquí mostramos cómo podemos cuantificar esta intuición.

Primero, exploremos la distribución de los promedios de bateo para todos los jugadores con más de 500 turnos al bate durante las tres temporadas anteriores:



El jugador promedio tuvo un AVG de .275 y la desviación estándar de la población de jugadores fue 0.027. Entonces podemos ver que .450 sería una anomalía, ya que está a más de seis desviaciones estándar de la media.

Entonces, ¿tiene suerte José o es el mejor bateador de los últimos 50 años? Quizás sea una combinación de suerte y talento. ¿Pero cuánto de cada uno? Si nos convencemos de que tiene suerte, deberíamos cambiarlo a otro equipo que confíe en la observación de .450 y tal vez sobreestime su potencial.

## 16.6 Modelos jerárquicos

El modelo jerárquico ofrece una descripción matemática de cómo llegamos a ver la observación de .450. Primero, elegimos un jugador al azar con una habilidad intrínseca resumida por, por ejemplo,  $p$ . Luego vemos 20 resultados aleatorios con probabilidad de éxito  $p$ .

Utilizamos un modelo para representar dos niveles de variabilidad en nuestros datos. Primero, a cada jugador se le asigna una habilidad natural para batear. Usaremos el símbolo  $p$  para representar esta habilidad. Pueden pensar en  $p$  como el promedio de bateo al que convergería si este jugador en particular bateara repetidas veces.

De acuerdo con los gráficos que mostramos anteriormente, suponemos que  $p$  tiene una distribución normal, con valor esperado .270 y error estándar 0.027.

Ahora el segundo nivel de variabilidad tiene que ver con la suerte al batear. Independientemente de lo bueno que sea el jugador, a veces tiene mala suerte y a veces tiene buena suerte. En cada turno al bate, este jugador tiene una probabilidad de éxito  $p$ . Si sumamos estos éxitos y fracasos, entonces el CLT nos dice que el promedio observado, llámelo  $Y$ , tiene una distribución normal con el valor esperado  $p$  y error estándar  $\sqrt{p(1-p)/N}$  con  $N$  el número de turnos al bate.

Los libros de texto estadísticos escribirán el modelo así:

$$\begin{aligned} p &\sim N(\mu, \tau^2) \\ Y \mid p &\sim N(p, \sigma^2) \end{aligned}$$

Aquí el símbolo  $\sim$  nos dice que la variable aleatoria a la izquierda del símbolo sigue la distribución a la derecha y  $N(a, b^2)$  representa la distribución normal con media  $a$  y desviación estándar  $b$ . El  $|$  significa que estamos *condicionando en* la variable aleatoria a la derecha del símbolo como si se conociera su valor. Nos referimos al modelo como jerárquico porque necesitamos saber  $p$ , el primer nivel, para modelar  $Y$ , el segundo nivel. En nuestro ejemplo, el primer nivel describe la aleatoriedad en la asignación de talento a un jugador y en el segundo se describe la aleatoriedad en el desempeño de este jugador una vez fijemos el parámetro de talento. En un marco bayesiano, el primer nivel se llama *distribución a priori* y el segundo la *distribución muestral*. El análisis de datos que hemos realizado aquí sugiere que establezcamos  $\mu = .270$ ,  $\tau = 0.027$  y  $\sigma^2 = p(1-p)/N$ .

Ahora, usemos este modelo para los datos de José. Supongan que queremos predecir su habilidad innata en la forma de su verdadero promedio de bateo  $p$ . Este sería el modelo jerárquico para nuestros datos:

$$\begin{aligned} p &\sim N(.275, .027^2) \\ Y \mid p &\sim N(p, .111^2) \end{aligned}$$

Ahora estamos listos para calcular una distribución a posteriori para resumir nuestra predicción de  $p$ . La versión continua de la regla de Bayes se puede usar aquí para derivar la *función de probabilidad a posteriori*, que es la distribución de  $p$  suponiendo que observemos  $Y = y$ . En nuestro caso, podemos demostrar que cuando fijamos  $Y = y$ ,  $p$  sigue una distribución normal con el valor esperado:

$$\begin{aligned} E(p \mid Y = y) &= B\mu + (1 - B)y \\ &= \mu + (1 - B)(y - \mu) \\ \text{with } B &= \frac{\sigma^2}{\sigma^2 + \tau^2} \end{aligned}$$

Este es un promedio ponderado del promedio de la población  $\mu$  y los datos observados  $y$ . El peso depende de la SD de la población  $\tau$  y de la SD de nuestros datos observados  $\sigma$ . Este promedio ponderado a veces se denomina *contracción* (*shrinking* en inglés) porque *contrae* los estimadores hacia la media de la distribución a priori. En el caso de José Iglesias tenemos:

$$\begin{aligned} E(p \mid Y = .450) &= B \times .275 + (1 - B) \times .450 \\ &= .275 + (1 - B)(.450 - .275) \\ B &= \frac{.111^2}{.111^2 + .027^2} = 0.944 \\ E(p \mid Y = 450) &\approx .285 \end{aligned}$$

No mostramos la derivación aquí, pero el error estándar se puede demostrar que es:

$$SE(p \mid y)^2 = \frac{1}{1/\sigma^2 + 1/\tau^2} = \frac{1}{1/.111^2 + 1/.027^2} = 0.00069$$

y, por lo tanto, la desviación estándar es 0.026. Entonces comenzamos con un intervalo de confianza frecuentista de 95% que ignoraba los datos de otros jugadores y resumía solo los datos de José:  $.450 \pm 0.220$ . Luego usamos un enfoque bayesiano que incorporaba datos de otros jugadores y otros años para obtener una probabilidad a posteriori. De hecho, esto se conoce como un enfoque empírico bayesiano porque utilizamos datos para construir la distribución a priori. Desde la distribución a posteriori, podemos calcular lo que se llama un *intervalo de confianza de Bayes* o *intervalo de Bayes* (*credible interval* en inglés) de 95%. Para hacer esto, construimos una región, centrada en la media, con una probabilidad de 95% de ocurrir. En nuestro caso, esto resulta ser:  $.285 \pm 0.052$ .

El intervalo de Bayes sugiere que si otro equipo está impresionado por el promedio observado de .450, deberíamos considerar cambiar a José, ya que pronosticamos que estará ligeramente por encima del promedio. Curiosamente, los Red Sox cambiaron a José a los Detroit Tigers en julio. Estos son los promedios de bateo de José Iglesias para los próximos cinco meses:

| Mes   | At Bat | Hits | AVG  |
|-------|--------|------|------|
| abril | 20     | 9    | .450 |
| mayo  | 26     | 11   | .423 |
| junio | 86     | 34   | .395 |

| Mes             | At Bat | Hits | AVG  |
|-----------------|--------|------|------|
| julio           | 83     | 17   | .205 |
| agosto          | 85     | 25   | .294 |
| septiembre      | 50     | 10   | .200 |
| Total sin abril | 330    | 97   | .293 |

Aunque ambos intervalos incluyeron el promedio final de bateo, el intervalo de Bayes ofreció una predicción mucho más precisa. En particular, predijo que no sería tan bueno durante el resto de la temporada.

---

## 16.7 Ejercicios

1. En 1999, en Inglaterra, Sally Clark<sup>5</sup> fue declarada culpable del asesinato de dos de sus hijos. Ambos bebés fueron encontrados muertos por la mañana, uno en 1996 y otro en 1998. En ambos casos, Clark afirmó que la causa de la muerte fue el síndrome de muerte súbita del lactante (*Sudden Infant Death Syndrome* o SIDS por sus siglas en inglés). A ninguno de los niños le encontraron lesiones físicas, por lo que la principal evidencia en su contra fue el testimonio del profesor Sir Roy Meadow, quien testificó que las probabilidades de que dos niños de la misma madre murieran de SIDS eran de 1 en 73 millones. Llegó a esta cifra al encontrar que la tasa de SIDS era de 1 en 8,500 y luego calcular que la posibilidad de dos casos de SIDS era  $8,500 \times 8,500 \approx 73$  millones. ¿Con cuál de las siguientes declaraciones está de acuerdo?
  - a. Sir Meadow supuso que la probabilidad de que el segundo hijo fuera afectado por el SIDS era independiente de la del primer hijo afectado, ignorando así posibles causas genéticas. Si la genética juega un papel, entonces:  $\Pr(\text{second case of SIDS} | \text{first case of SIDS}) < \Pr(\text{first case of SIDS})$ .
  - b. Nada. La regla de multiplicación siempre se aplica de esta manera:  $\Pr(A \text{ and } B) = \Pr(A)\Pr(B)$
  - c. Sir Meadow es un experto y debemos confiar en sus cálculos.
  - d. Los números no mienten.
2. Suponga que definitivamente hay un componente genético para el SIDS y la probabilidad de  $\Pr(\text{second case of SIDS} | \text{first case of SIDS}) = 1/100$ , es mucho mayor que 1 en 8,500. ¿Cuál es la probabilidad de que sus dos hijos mueran de SIDS?
3. Muchos informes de prensa declararon que el experto afirmó que la probabilidad de que Sally Clark fuera inocente era 1 en 73 millones. Quizás el jurado y el juez también interpretaron el testimonio de esta manera. Esta probabilidad se puede escribir como la probabilidad de que *una madre sea una psicópata asesina de hijos, dado que encuentran a dos de sus hijos muertos sin lesiones físicas*. Según la regla de Bayes, ¿cuánta es esta probabilidad?
4. Suponga que la probabilidad de que una psicópata asesina de hijos encuentre la manera de matar a sus hijos, sin dejar evidencia física, es:

<sup>5</sup>[https://en.wikipedia.org/wiki/Sally\\_Clark](https://en.wikipedia.org/wiki/Sally_Clark)

$$\Pr(A | B) = 0.50$$

con  $A$  = dos de sus hijos los encuentran muertos sin lesiones físicas y  $B$  = una madre es una psicópata asesina de hijos = 0.50. Suponga que la tasa de madres psicópatas que asesinan hijos es 1 en 1,000,000. Según el teorema de Bayes, ¿cuál es la probabilidad de  $\Pr(B | A)$ ?

5. Después de que Sally Clark fue declarada culpable, la Royal Statistical Society emitió un comunicado diciendo que “no había base estadística” para el reclamo del experto. Expresaron preocupación por el “mal uso de las estadísticas en los tribunales”. Sally Clark fue absuelta en junio de 2003. ¿Qué no consideró el experto Sir Roy Meadow?

- a. Cometió un error aritmético.
- b. Cometió dos errores. Primero, hizo un mal uso de la regla de multiplicación y, segundo, no tomó en cuenta lo raro que es que una madre asesine a sus hijos. Después de usar la regla de Bayes, encontramos una probabilidad más cercana a 0.5 que a 1 en 73 millones.
- c. Confundió el numerador y el denominador de la regla de Bayes.
- d. No usó R.

6. Florida es uno de los estados más vigilados en las elecciones de EE. UU. porque tiene muchos votos electorales y las elecciones generalmente son cerradas. Además, Florida tiende a ser un estado decisivo que puede votar por cualquiera de los dos partidos. Cree la siguiente tabla con las encuestas realizadas durante las últimas dos semanas:

```
library(tidyverse)
library(dslabs)
data(polls_us_election_2016)
polls <- polls_us_election_2016 %>%
 filter(state == "Florida" & enddate >= "2016-11-04") %>%
 mutate(spread = rawpoll_clinton/100 - rawpoll_trump/100)
```

Tome la diferencia promedio de estas encuestas. El CLT nos dice que este promedio es aproximadamente normal. Calcule un promedio y provea un estimador del error estándar. Guarde sus resultados en un objeto llamado **results**.

7. Ahora suponga un modelo bayesiano con distribución a priori normal para la diferencia de la noche electoral de Florida  $d$  con valor esperado  $\mu$  y desviación estandar  $\tau$ . ¿Cuáles son las interpretaciones de  $\mu$  y  $\tau$ ?

- a.  $\mu$  y  $\tau$  son números arbitrarios que nos permiten hacer declaraciones de probabilidad sobre  $d$ .
- b.  $\mu$  y  $\tau$  resumen lo que predeciríamos para Florida antes de ver las encuestas. Basado en elecciones pasadas, fijaríamos  $\mu$  cerca de 0 porque tanto republicanos como demócratas han ganado y  $\tau$  en aproximadamente 0.02 porque estas elecciones tienden a ser cerradas.
- c.  $\mu$  y  $\tau$  resumen lo que queremos que sea verdad. Por lo tanto, fijamos  $\mu$  en 0.10 y  $\tau$  en 0.01.
- d. La decisión de que distribución a priori usar no tiene ningún efecto en el análisis bayesiano.

8. El CLT nos dice que nuestro estimador de la diferencia  $\hat{d}$  tiene distribución normal con valor esperado  $d$  y desviación estándar  $\sigma$  calculada en el problema 6. Use las fórmulas que mostramos para la distribución a posteriori para calcular el valor esperado de la distribución a posteriori si fijamos  $\mu = 0$  y  $\tau = 0.01$ .
9. Ahora calcule la desviación estándar de la distribución a posteriori.
10. Usando el hecho de que la distribución a posteriori es normal, cree un intervalo que tenga un 95% de probabilidad de ocurrir centrado en el valor esperado a posteriori. Recuerden que estos los llamamos intervalos de Bayes.
11. Según este análisis, ¿cuál fue la probabilidad de que Trump ganara Florida?
12. Ahora use la función `sapply` para cambiar la varianza de la probabilidad a priori de `seq(0.05, 0.05, len = 100)` y observe cómo cambia la probabilidad haciendo un gráfico.

## 16.8 Estudio de caso: pronóstico de elecciones

En una sección anterior, generamos las siguientes tablas de datos:

```
library(tidyverse)
library(dslabs)
polls <- polls_us_election_2016 %>%
 filter(state == "U.S." & enddate >= "2016-10-31" &
 (grade %in% c("A+","A","A-","B+") | is.na(grade))) %>%
 mutate(spread = rawpoll_clinton/100 - rawpoll_trump/100)

one_POLL_per_POLLSTER <- polls %>% group_by(pollster) %>%
 filter(enddate == max(enddate)) %>%
 ungroup()

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)
```

A continuación, las utilizaremos para nuestro pronóstico.

### 16.8.1 Enfoque bayesiano

Los encuestadores tienden a hacer declaraciones probabilísticas sobre los resultados de las elecciones. Por ejemplo, “La probabilidad de que Obama gane el colegio electoral es 91%” es una declaración probabilística sobre un parámetro que en secciones anteriores hemos denotado con  $d$ . Mostramos que para las elecciones del 2016, FiveThirtyEight le dio a Clinton una probabilidad de 81.4% de ganar el voto popular. Para hacer esto, utilizaron el enfoque bayesiano que describimos anteriormente.

Suponemos un modelo jerárquico similar al que hicimos para predecir el desempeño de un jugador de béisbol. Los libros de texto estadísticos escribirán el modelo así:

$d \sim N(\mu, \tau^2)$  describes our best guess had we not seen any polling data  
 $\bar{X} | d \sim N(d, \sigma^2)$  describes randomness due to sampling and the pollster effect

Para hacer nuestro mejor pronóstico, notamos que antes de que haya datos de encuestas disponibles, podemos usar fuentes de datos que no son datos de encuestas. Un enfoque popular es utilizar la situación económica y demográfica que históricamente parecen tener un efecto a favor o en contra del partido en poder. No usaremos estos aquí. En cambio, usaremos  $\mu = 0$ , que se interpreta como un modelo que no ofrece información sobre quién ganará. Para la desviación estándar, usaremos datos históricos recientes que muestran que el ganador del voto popular tiene una variabilidad promedio de aproximadamente 3.5%. Por lo tanto, fijamos  $\tau = 0.035$ .

Ahora podemos usar las fórmulas para la distribución a posteriori del parámetro  $d$ : la probabilidad de que  $d > 0$  dado los datos de la encuesta observada:

```
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 hacer una declaración de probabilidad, usamos el hecho de que la distribución a posteriori también es normal. Y tenemos un intervalo de confianza de Bayes de:

```
posterior_mean + c(-1.96, 1.96)*posterior_se
#> [1] 0.0160 0.0401
```

La probabilidad a posteriori  $\Pr(d > 0 | \bar{X})$  se puede calcular así:

```
1 - pnorm(0, posterior_mean, posterior_se)
#> [1] 1
```

Esto dice que estamos 100% seguros de que Clinton ganará el voto popular, lo que parece demasiado confiado. Además, no está de acuerdo con el 81.4% de FiveThirtyEight. ¿Qué explica esta diferencia?

### 16.8.2 El sesgo general

Una vez finalizadas las elecciones, se puede observar la diferencia entre las predicciones de los encuestadores y el resultado real. Una observación importante que nuestro modelo no

considera es que es común ver un sesgo general que afecta a muchos encuestadores de la misma manera, que entonces conduce a que los datos observados estén correlacionados. No hay una buena explicación para esto, pero se observa en datos históricos: en una elección, el promedio de las encuestas favorece a los demócratas por 2%, luego en las siguientes elecciones favorece a los republicanos por 1%, entonces en las próximas elecciones no hay sesgo, luego en la siguiente los republicanos son los favoritos por 3%, y así sucesivamente. En 2016, las encuestas favorecieron a los demócratas por 1-2%.

Aunque sabemos que este sesgo afecta a nuestras encuestas, no tenemos forma de saber cuán grande es este sesgo hasta la noche de las elecciones. Como consecuencia, no podemos corregir nuestras encuestas para tomar este sesgo en cuenta. Lo que podemos hacer es incluir un término en nuestro modelo que explique esta variabilidad.

### 16.8.3 Representaciones matemáticas de modelos

Imagínense que estamos recopilando datos de un encuestador y suponemos que no hay sesgo general. El encuestador recoge varias encuestas con un tamaño de muestra de  $N$ , por lo que observamos varias mediciones de la variabilidad  $X_1, \dots, X_J$ . La teoría nos dice que estas variables aleatorias tienen un valor esperado  $d$  y un error estándar  $2\sqrt{p(1-p)/N}$ . Comencemos usando el siguiente modelo para describir la variabilidad observada:

$$X_j = d + \varepsilon_j.$$

Usamos el índice  $j$  para representar las diferentes encuestas y definimos  $\varepsilon_j$  para ser una variable aleatoria que explica la variabilidad entre encuestas individuales introducida por el error de muestreo. Para hacer esto, suponemos que su promedio es 0 y su error estándar es  $2\sqrt{p(1-p)/N}$ . Si  $d$  es 2.1 y el tamaño de la muestra para estas encuestas es de 2,000, podemos simular  $J = 6$  puntos de datos de este modelo así:

```
set.seed(3)
J <- 6
N <- 2000
d <- .021
p <- (d + 1)/2
X <- d + rnorm(J, 0, 2 * sqrt(p * (1 - p) / N))
```

Ahora supongan que tenemos  $J = 6$  puntos de datos de  $I = 5$  diferentes encuestadores. Para representar esto, necesitamos dos índices, uno para el encuestador y otro para las encuestas que cada encuestador toma. Usamos  $X_{ij}$  con  $i$  representando al encuestador y  $j$  representando la encuesta número  $j$  de ese encuestador. Si aplicamos el mismo modelo, escribimos:

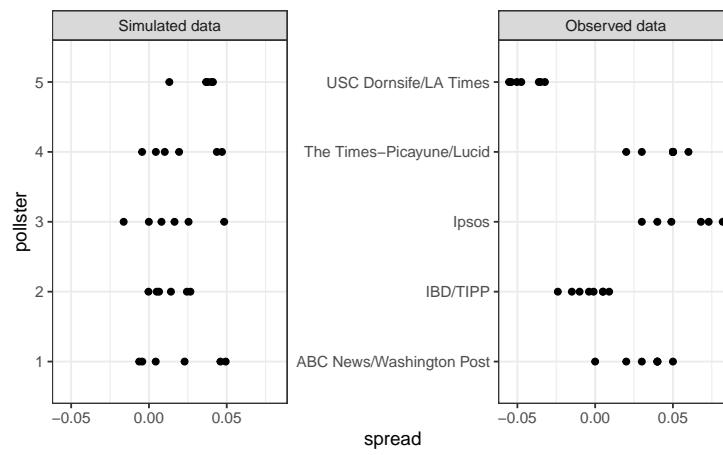
$$X_{i,j} = d + \varepsilon_{i,j}$$

Para simular datos, ahora tenemos que usar un bucle para simular los datos de cada encuestador:

```
I <- 5
J <- 6
```

```
N <- 2000
X <- sapply(1:I, function(i){
 d + rnorm(J, 0, 2 * sqrt(p * (1 - p) / N))
})
```

Los datos simulados realmente no parecen capturar las características de los datos reales:



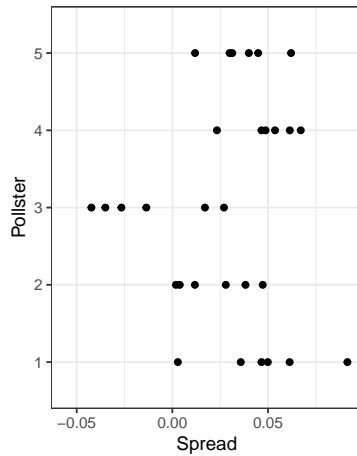
El modelo anterior no toma en cuenta la variabilidad entre encuestadores. Para arreglar esto, añadimos un nuevo término para el efecto de los encuestadores. Usaremos  $h_i$  para representar el sesgo del encuestador número  $i$ . Le añadimos este nuevo término al modelo:

$$X_{i,j} = d + h_i + \varepsilon_{i,j}$$

Para simular datos de un encuestador específico, ahora necesitamos escoger un  $h_i$  y luego añadir los  $\varepsilon$ s. Entonces, para un encuestador específico, suponemos que  $\sigma_h$  es 0.025:

```
I <- 5
J <- 6
N <- 2000
d <- .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))
})
```

Los datos simulados ahora se parecen más a los datos reales:



Noten que  $h_i$  es común a todas las variabilidades observadas de un encuestador específico. Diferentes encuestadores tienen una  $h_i$  diferente, lo que explica por qué cuando vemos los datos de los distintos encuestadores, podemos ver los diferentes grupos de puntos desplazarse hacia arriba y hacia abajo.

Ahora, en el modelo anterior, suponemos que el promedio de los sesgos de los encuestadores es 0. Creemos que para cada encuestador sesgado a favor de nuestro partido, hay otro a favor del otro partido y suponemos que la desviación estándar es  $\sigma_h$ . Pero históricamente vemos que cada elección tiene un sesgo general que afecta a todas las encuestas. Podemos observar esto con los datos del 2016, pero si recopilamos datos históricos, vemos que el promedio de las encuestas falla por más de lo que predicen modelos como el anterior. Para ver esto, tomariamos el promedio de las encuestas para cada año electoral y lo compararíamos con el valor real. Si hiciéramos esto, veríamos una diferencia con una desviación estándar de entre 2-3%. Para incorporar esto en el modelo, podemos añadir otro término para explicar esta variabilidad:

$$X_{i,j} = d + b + h_i + \varepsilon_{i,j}.$$

Aquí  $b$  es una variable aleatoria que explica la variabilidad de elección a elección. Esta variable aleatoria cambia de elección a elección, pero para cualquier elección dada, es la misma para todos los encuestadores y las encuestas dentro de la elección. Por eso no tiene índices. Esto implica que todas las variables aleatorias  $X_{i,j}$  para un año electoral están correlacionadas ya que todas tienen  $b$  en común.

Una forma de interpretar  $b$  es como la diferencia entre el promedio de todas las encuestas de todos los encuestadores y el resultado real de la elección. Como no conocemos el resultado real hasta después de las elecciones, no podemos estimar  $b$  hasta entonces. Sin embargo, podemos estimar  $b$  de las elecciones anteriores y estudiar la distribución de estos valores. Conforme a este enfoque, suponemos que, a lo largo de los años electorales,  $b$  tiene el valor esperado 0 y el error estándar es aproximadamente  $\sigma_b = 0.025$ .

Una implicación de añadir este término al modelo es que la desviación estándar de  $X_{i,j}$  es mayor que lo que llamamos anteriormente  $\sigma$ , que combina la variabilidad del encuestador y la variabilidad de la muestra, y que se estimó con:

```
sd(one_poll_per_pollster$spread)
#> [1] 0.0242
```

Este estimador no incluye la variabilidad introducida por  $b$ . Tengan en cuenta que como:

$$\bar{X} = d + b + \frac{1}{N} \sum_{i=1}^N X_i,$$

la desviación estándar de  $\bar{X}$  es:

$$\sqrt{\sigma^2/N + \sigma_b^2}.$$

Ya que la misma  $b$  está en cada medición, el promedio no reduce la variabilidad introducida por este término. Este es un punto importante: no importa cuántas encuestas realicen, este sesgo no se reduce.

Si rehacemos el cálculo bayesiano tomando en cuenta esta variabilidad, obtenemos un resultado mucho más cercano al de FiveThirtyEight:

```
mu <- 0
tau <- 0.035
sigma <- sqrt(results$se^2 + .025^2)
Y <- results$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
```

#### 16.8.4 Prediciendo el colegio electoral

Hasta ahora nos hemos enfocado en el voto popular. Pero en Estados Unidos, las elecciones no se deciden por el voto popular, sino por lo que se conoce como el colegio electoral. Cada estado obtiene una cantidad de votos electorales que dependen, de una manera algo compleja, del tamaño de la población del estado. Aquí están los 5 principales estados clasificados por votos electorales en 2016.

```
results_us_election_2016 %>% top_n(5, electoral_votes)
#> #> 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
```

Con algunas excepciones que no discutimos, los votos electorales se ganan todo o nada. Por ejemplo, si un candidato gana California con solo 1 voto, aún obtiene los 55 votos electorales. Esto significa que al ganar algunos estados grandes por un amplio margen, pero al perder muchos estados pequeños por pequeños márgenes, se puede ganar el voto popular,

pero perder el colegio electoral que es lo que decide el ganador. Esto sucedió en 1876, 1888, 2000 y 2016. La idea detrás de esto es evitar que algunos estados grandes tengan el poder de dominar las elecciones presidenciales. Sin embargo, muchas personas en Estados Unidos consideran que el colegio electoral es injusto y les gustaría abolirlo.

Ahora estamos listos para predecir el resultado del colegio electoral para 2016. Comenzamos agregando los resultados de una encuesta realizada durante la última semana antes de las elecciones. Utilizamos `str_detect`, una función que discutiremos más adelante en la Sección 24.1, para eliminar encuestas que cubren solo parte de un estado.

```
results <- polls_us_election_2016 %>%
 filter(state!="U.S." &
 !str_detect(state, "CD") &
 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))
```

Aquí están los cinco estados con los resultados más cerrados según las encuestas:

```
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
#> # ... with 42 more rows
```

Ahora utilizaremos el comando `left_join` que nos permitirá añadir fácilmente el número de votos electorales para cada estado del set de datos `us_electoral_votes_2016`. Describiremos esta función en detalle en el capítulo “Wrangling de datos”. Aquí, simplemente observaremos que la función combina los dos sets de datos para que la información del segundo argumento se agregue a la información del primero:

```
results <- left_join(results, results_us_election_2016, by = "state")
```

Observen que algunos estados no tienen encuestas porque prácticamente se conoce el ganador:

```
results_us_election_2016 %>% filter(!state %in% results$state) %>%
 pull(state)
#> [1] "Rhode Island" "Alaska" "Wyoming"
#> [4] "District of Columbia"
```

No se realizaron encuestas en DC, Rhode Island, Alaska y Wyoming porque los demócratas seguramente ganarán en los primeros dos y los republicanos en los últimos dos.

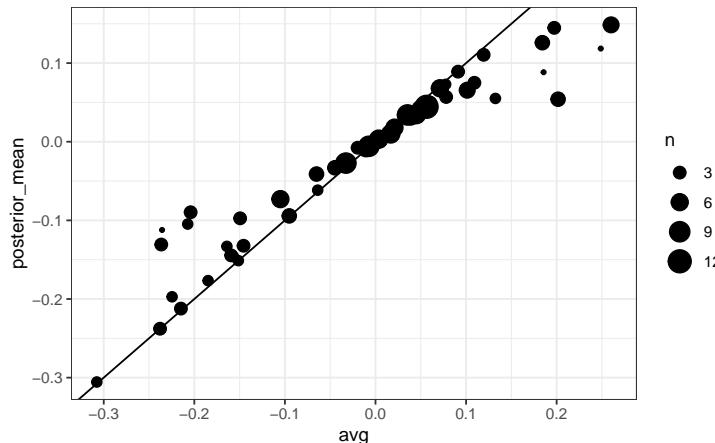
Debido a que no podemos estimar la desviación estándar para los estados con una sola encuesta, la calcularemos como la mediana de las desviaciones estándar estimadas para los estados con más de una encuesta:

```
results <- results %>%
 mutate(sd = ifelse(is.na(sd), median(results$sd, na.rm = TRUE), sd))
```

Para hacer argumentos probabilísticos, utilizaremos una simulación Monte Carlo. Para cada estado, aplicamos el enfoque bayesiano para generar una  $d$  para el día de elecciones. Podríamos construir las probabilidades a priori de cada estado basado en la historia reciente. Sin embargo, para simplificar, asignamos una probabilidad a priori a cada estado que supone que no sabemos nada sobre lo que sucederá. Dado que de un año electoral a otro, los resultados de un estado específico no cambian tanto, asignaremos una desviación estándar de 2% o  $\tau = 0.02$ . Por ahora, vamos a suponer incorrectamente que los resultados de la encuesta de cada estado son independientes. El código para el cálculo bayesiano bajo estos supuestos se ve así:

```
#> # A tibble: 47 x 12
#> state avg sd n electoral_votes clinton trump others
#> <chr> <dbl> <dbl> <int> <dbl> <dbl> <dbl> <dbl>
#> 1 Alabama -0.149 0.0253 3 9 34.4 62.1 3.6
#> 2 Arizona -0.0326 0.0270 9 11 45.1 48.7 6.2
#> 3 Arkansas -0.151 0.000990 2 6 33.7 60.6 5.8
#> 4 California 0.260 0.0387 5 55 61.7 31.6 6.7
#> 5 Colorado 0.0452 0.0295 7 9 48.2 43.3 8.6
#> # ... with 42 more rows, and 4 more variables: sigma <dbl>, B <dbl>,
#> # posterior_mean <dbl>, posterior_se <dbl>
```

Los estimadores basadas en las probabilidades a posteriori mueven los estimadores hacia 0, aunque los estados con muchas encuestas están menos influenciados. Esto se espera ya que mientras más datos de encuestas recolectamos, más confiamos en esos resultados:



Ahora repetimos esto 10,000 veces y generamos un resultado de la probabilidad a posteriori. En cada iteración, hacemos un seguimiento del número total de votos electorales para

Clinton. Recuerden que Trump obtiene 270 votos electorales menos los votos para Clinton. También noten que la razón por la que añadimos 7 en el código es para tomar en cuenta Rhode Island y DC:

```
B <- 10000
mu <- 0
tau <- 0.02
clinton_EV <- replicate(B, {
 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)),
 result = rnorm(length(posterior_mean),
 posterior_mean, posterior_se),
 clinton = ifelse(result > 0, electoral_votes, 0)) %>%
 summarize(clinton = sum(clinton)) %>%
 pull(clinton) + 7
})

mean(clinton_EV > 269)
#> [1] 0.998
```

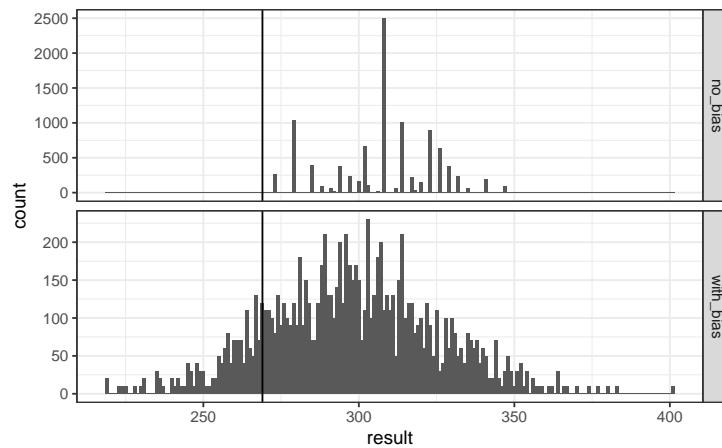
Este modelo le da a Clinton una probabilidad de ganar mayor que 99%. El Consorcio Electoral de Princeton hizo una predicción similar. Ahora sabemos que fallaron por mucho. ¿Qué pasó?

El modelo anterior ignora el sesgo general y supone que los resultados de diferentes estados son independientes. Después de las elecciones, nos dimos cuenta de que el sesgo general en 2016 no era tan grande: estaba entre 1 y 2%. Pero debido a que la elección estuvo cerrada en varios estados grandes y estos estados tenían una gran cantidad de encuestas, los encuestadores que ignoraron el sesgo general subestimaron considerablemente el error estándar. Utilizando la notación que presentamos, supusieron que el error estándar era  $\sqrt{\sigma^2/N}$  que con N grande es bastante más pequeño que el estimador más preciso  $\sqrt{\sigma^2/N + \sigma_b^2}$ . FiveThirtyEight, que modela el sesgo general de una manera bastante sofisticada, informó un resultado más cercano. Podemos simular los resultados ahora con un término de sesgo. Para el nivel de estado, el sesgo general puede ser mayor, por lo que lo establecemos en  $\sigma_b = 0.03$ :

```
tau <- 0.02
bias_sd <- 0.03
clinton_EV_2 <- replicate(1000, {
 results %>% mutate(sigma = sqrt(sd^2/n + bias_sd^2),
 B = sigma^2/ (sigma^2 + tau^2),
 posterior_mean = B*mu + (1-B)*avg,
 posterior_se = sqrt(1/ (1/sigma^2 + 1/tau^2)),
 result = rnorm(length(posterior_mean),
 posterior_mean, posterior_se),
 clinton = ifelse(result>0, electoral_votes, 0)) %>%
 summarize(clinton = sum(clinton) + 7) %>%
 pull(clinton)
})
```

```
mean(clinton_EV_2 > 269)
#> [1] 0.848
```

Esto nos da un estimador mucho más sensato. Al observar los resultados de la simulación, vemos cómo el término de sesgo agrega variabilidad a los resultados finales.



El modelo de FiveThirtyEight incluye muchas otras características que no describimos aquí. Una es que modelan la variabilidad con distribuciones que tienen altas probabilidades para eventos extremos en comparación con la distribución normal. Una forma que nosotros podemos hacerlo es cambiando la distribución utilizada en la simulación de una distribución normal a una distribución t. FiveThirtyEight predijo una probabilidad de 71%.

### 16.8.5 Pronósticos

A los pronosticadores les gusta hacer predicciones mucho antes de las elecciones. Las predicciones se adaptan a medida que salen nuevas encuestas. Sin embargo, una pregunta importante que deben hacer los pronosticadores es: ¿cuán informativas son las encuestas que se hacen varias semanas antes de las elecciones sobre la elección real? Aquí estudiamos la variabilidad de los resultados de las encuestas a lo largo del tiempo.

Para asegurarnos de que la variabilidad que observamos no se debe a efectos del encuestador, estudiemos los datos de un encuestador:

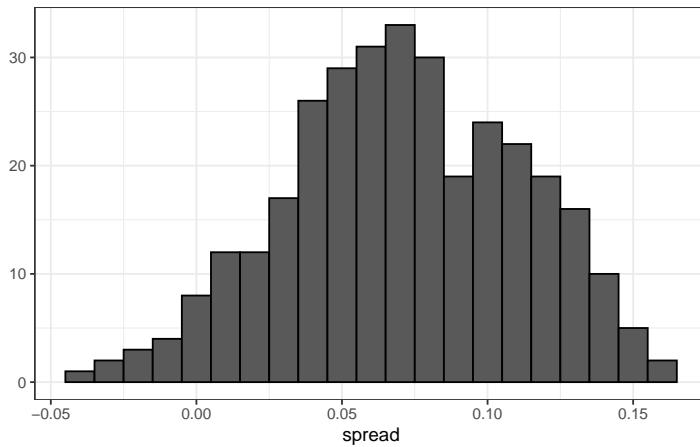
```
one_pollster <- polls_us_election_2016 %>%
 filter(pollster == "Ipsos" & state == "U.S.") %>%
 mutate(spread = rawpoll_clinton/100 - rawpoll_trump/100)
```

Como no hay efecto de encuestador, quizás el error estándar teórico coincide con la desviación estándar derivada de los datos. Calculamos ambos aquí:

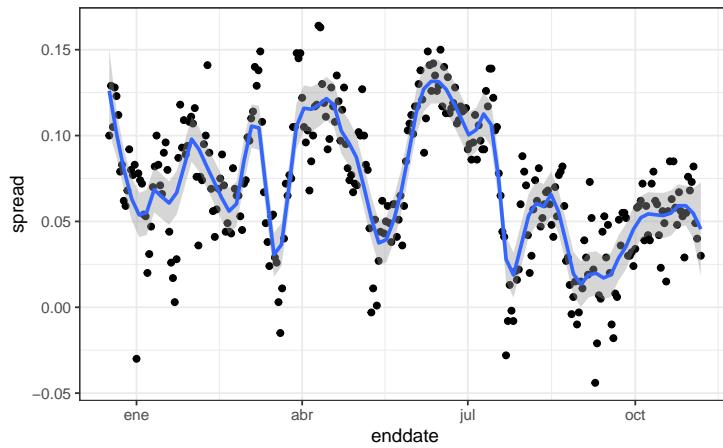
```
se <- one_pollster %>%
 summarize(empirical = sd(spread),
 theoretical = 2 * sqrt(mean(spread) * (1 - mean(spread)))/
```

```
min(samplesize))
se
#> empirical theoretical
#> 1 0.0403 0.0326
```

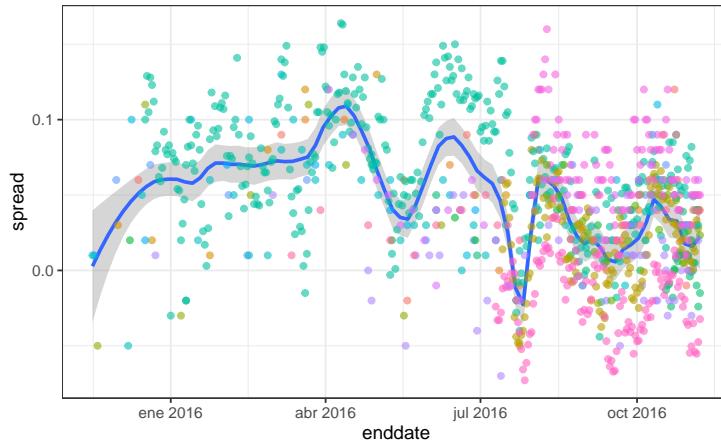
Pero la desviación estándar empírica es más alta que el estimador teórico más alto posible. Además, los datos de la variabilidad no se ven normales como la teoría predeciría:



Los modelos que hemos descrito incluyen la variabilidad entre encuestadores y el error de muestreo. Pero este gráfico es para un encuestador y la variabilidad que vemos ciertamente no la explica el error de muestreo. ¿De dónde viene la variabilidad extra? Los siguientes gráficos muestran un fuerte argumento de que esta variabilidad proviene de fluctuaciones de tiempo no explicadas por la teoría que supone que  $p$  es fija:



Algunos de los picos y valles que vemos coinciden con eventos como las convenciones de los partidos, que tienden a dar un impulso a los candidatos. Vemos consistencia entre los distintos encuestadores en cuanto a la localización de los picos y valles.



Esto implica que, si vamos a pronosticar, nuestro modelo debe incluir un término que toma en cuenta el efecto temporero. Necesitamos escribir un modelo que incluya un término de sesgo para el tiempo:

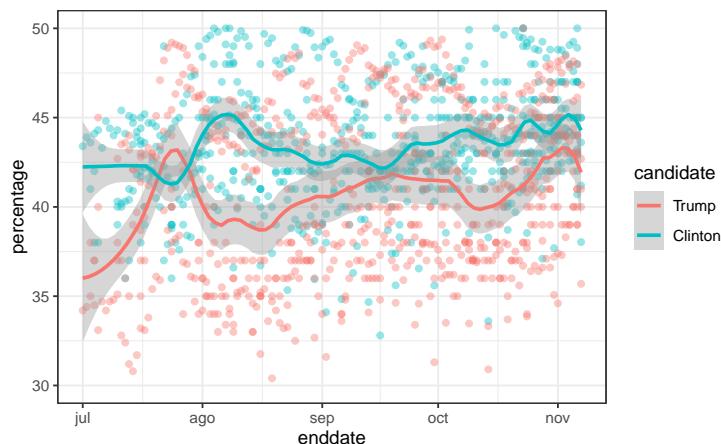
$$Y_{i,j,t} = d + b + h_j + b_t + \varepsilon_{i,j,t}$$

La desviación estándar de  $b_t$  va a depender de  $t$  ya que en cuanto más nos acercamos al día de las elecciones, más cerca de 0 debería estar este término de sesgo.

Los encuestadores también intentan estimar las tendencias de estos datos e incorporarlos en sus predicciones. Podemos modelar la tendencia temporera con una función  $f(t)$  y reescribir el modelo así:

$$Y_{i,j,t} = d + b + h_j + b_t + f(t) + \varepsilon_{i,j,t},$$

Usualmente vemos el estimador de  $f(t)$  no para la diferencia, sino para los porcentajes reales para cada candidato así:



Una vez que se seleccione un modelo como el anterior, podemos usar datos históricos y

actuales para estimar todos los parámetros necesarios para hacer predicciones. Existe una variedad de métodos para estimar tendencias  $f(t)$  que discutimos en la parte de *machine learning*.

## 16.9 Ejercicios

1. Cree esta tabla:

```
library(tidyverse)
library(dslabs)
data("polls_us_election_2016")
polls <- polls_us_election_2016 %>%
 filter(state != "U.S." & enddate >= "2016-10-31") %>%
 mutate(spread = rawpoll_clinton/100 - rawpoll_trump/100)
```

Ahora, para cada encuesta, use el CLT para crear un intervalo de confianza de 95% para la diferencia informada por cada encuesta. Llame al objeto resultante `cis` con columnas inferior y superior para los límites de los intervalos de confianza. Utilice la función `select` para mantener las columnas `state`, `startdate`, `end date`, `pollster`, `grade`, `spread`, `lower`, `upper`.

2. Puede añadir el resultado final a la tabla `cis` que acaba de crear utilizando la función `right_join` así:

```
add <- results_us_election_2016 %>%
 mutate(actual_spread = clinton/100 - trump/100) %>%
 select(state, actual_spread)
cis <- cis %>%
 mutate(state = as.character(state)) %>%
 left_join(add, by = "state")
```

Ahora determine con qué frecuencia el intervalo de confianza de 95% incluye el resultado real.

3. Repita esto, pero muestre la proporción de veces que cada encuestador acierta. Muestre solo encuestadores con más de 5 encuestas y póngalos en orden de mejor a peor. Muestre el número de encuestas realizadas por cada encuestador y la calificación de FiveThirtyEight para cada encuestador. Sugerencia: use `n=n()`, `grade = grade[1]` en la llamada a `summarize`.

4. Repita el ejercicio 3, pero en lugar de estratificar por encuestador, estratifique por estado. Recuerden que aquí no podemos mostrar calificaciones.

5. Haga un diagrama de barras basado en el resultado del ejercicio 4. Use `coord_flip`.

6. Para cada encuesta, calcule la diferencia entre la diferencia que predijimos y la diferencia observada. Añada una columna a la tabla `cis`. Entonces, añada otra columna llamada `hit` que es TRUE cuando los signos son los mismos. Sugerencia: use la función `sign`. Llame al objeto `resids`.

7. Cree un gráfico como en el ejercicio 5, pero para la proporción de veces que los signos de la diferencia fueron iguales.
8. En el ejercicio 7, vemos que para la mayoría de los estados las encuestas acertaron el 100% de las veces. En solo 9 estados las encuestas fallaron más de 25% de las veces. En particular, observe que en Wisconsin todas las encuestas se equivocaron. En Pennsylvania y Michigan, más de 90% de las encuestas predijeron incorrectamente el ganador. Haga un histograma de los errores. ¿Cuál es la mediana de estos errores?
9. Vemos que a nivel estatal, el error medio fue 3% a favor de Clinton. La distribución no está centrada en 0, sino en 0.03. Este es el sesgo general que describimos en la sección anterior. Cree un diagrama de caja para ver si el sesgo fue general para todos los estados o si afectó a algunos estados de manera diferente. Utilice `filter(grade %in% c("A+", "A", "A-", "B+") | is.na(grade))` para incluir solo encuestadores con altas calificaciones.
10. Algunos de estos estados solo tienen unas pocas encuestas. Repita el ejercicio 9, pero solo incluya estados con 5 o más encuestas buenas. Sugerencia: use `group_by`, `filter` y luego `ungroup`. Verá que el Oeste (Washington, Nuevo México, California) subestimó el desempeño de Hillary, mientras que el Medio Oeste (Michigan, Pennsylvania, Wisconsin, Ohio, Missouri) lo sobreestimó. En nuestra simulación, no modelamos este comportamiento ya que añadimos un sesgo general, en lugar de un sesgo regional. Tenga en cuenta que algunos encuestadores ahora pueden modelar la correlación entre estados similares y estimar esta correlación a partir de datos históricos. Para obtener más información sobre esto, puede aprender sobre efectos aleatorios y modelos mixtos.

## 16.10 La distribución t

Arriba utilizamos el CLT con un tamaño de muestra de 15. Como estamos estimando un segundo parámetro  $\sigma$ , se introduce más variabilidad a nuestro intervalo de confianza, lo que da como resultado intervalos muy pequeños. Para tamaños de muestra muy grandes, esta variabilidad adicional es insignificante, pero, en general, para valores menores de 30 debemos ser cautelosos al usar el CLT.

Sin embargo, si se sabe que los datos en la urna siguen una distribución normal, entonces tenemos una teoría matemática que nos dice cuánto más grande necesitamos hacer los intervalos para tomar en cuenta el estimador de  $\sigma$ . Usando esta teoría, podemos construir intervalos de confianza para cualquier  $N$ . Pero, de nuevo, esto funciona solo si **sabemos que los datos en la urna siguen una distribución normal**. Entonces, para los datos 0, 1 de nuestro modelo de urna anterior, esta teoría definitivamente no aplica.

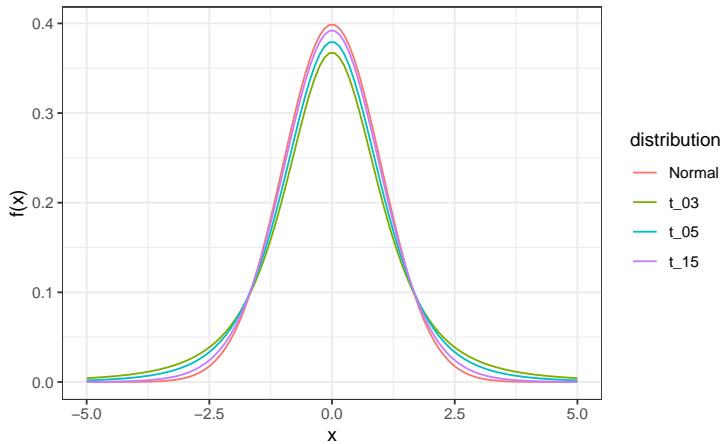
La estadística en la que se basan los intervalos de confianza para  $d$  es:

$$Z = \frac{\bar{X} - d}{\sigma/\sqrt{N}}$$

El CLT nos dice que la distribución de  $Z$  es aproximadamente normal con valor esperado 0 y error estándar 1. Pero en la práctica no sabemos  $\sigma$ , entonces usamos:

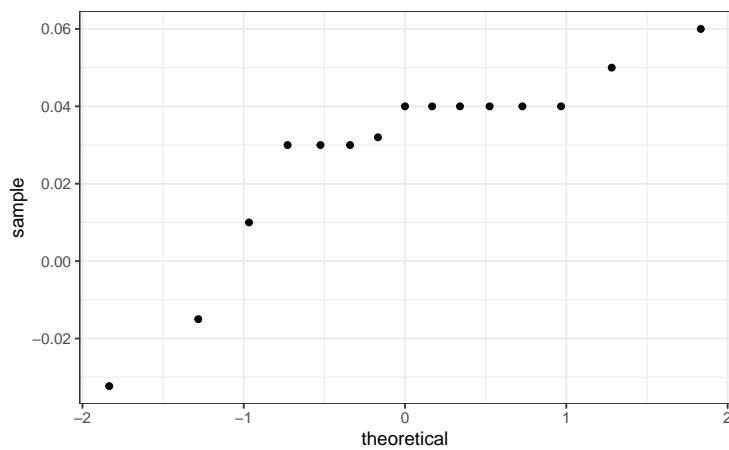
$$t = \frac{\bar{X} - d}{s/\sqrt{N}}$$

A esto se le conoce la estadística t. Al sustituir  $\sigma$  con  $s$ , introducimos cierta variabilidad. La teoría nos dice que  $t$  sigue una distribución t con  $N - 1$  *grados de libertad*. Los grados de libertad son un parámetro que controla la variabilidad a través de colas más pesadas:



Si estamos dispuestos a suponer que los datos del efecto del encuestador siguen una distribución normal, según la muestra de datos  $X_1, \dots, X_N$ ,

```
one_poll_per_pollster %>%
 ggplot(aes(sample=spread)) + stat_qq()
```



entonces  $t$  sigue una distribución t con  $N - 1$  grados de libertad. Por eso, quizás un mejor intervalo de confianza para  $d$  es:

```
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
```

que es un poco más grande que cuando usamos la distribución normal. Esto es porque:

```
qt(0.975, 14)
#> [1] 2.14
```

es más grande que:

```
qnorm(0.975)
#> [1] 1.96
```

La distribución t y la estadística t son la base para llevar acabo pruebas t, un acercamiento comúnmente usado para calcular valores p. Para aprender más sobre las prueba t, puede consultar un libro de texto de estadística.

La distribución t también se puede usar para modelar errores cuando esperamos que la probabilidad de grandes desviaciones de la media sea mayor de lo que dicta la distribución normal. FiveThirtyEight utiliza la distribución t para generar errores que modelan mejor las desviaciones que vemos en los datos electorales. Por ejemplo, en Wisconsin, el promedio de seis encuestas fue 7% a favor de Clinton con una desviación estándar de 1%, pero Trump ganó por 0.7%. Incluso después de tomar en cuenta el sesgo general, este residuo de 7.7% está más en línea con datos que siguen la distribución t, que con datos que siguen la distribución normal.

```
data("polls_us_election_2016")
polls_us_election_2016 %>%
 filter(state == "Wisconsin" &
 enddate >= "2016-10-31" &
 grade %in% c("A+","A","A-","B+") | is.na(grade))) %>%
 mutate(spread = rawpoll_clinton/100 - rawpoll_trump/100) %>%
 mutate(state = as.character(state)) %>%
 left_join(results_us_election_2016, by = "state") %>%
 mutate(actual = clinton/100 - trump/100) %>%
 summarize(actual = first(actual), avg = mean(spread),
 sd = sd(spread), n = n()) %>%
 select(actual, avg, sd, n)
#> actual avg sd n
#> 1 -0.007 0.0711 0.0104 6
```



# 17

---

## Regresión

---

Hasta ahora, este libro se ha enfocado principalmente en variables individuales. Sin embargo, en aplicaciones de ciencia de datos, es muy común estar interesado en la relación entre dos o más variables. Por ejemplo, en el Capítulo 18, utilizaremos un enfoque basado en datos que examina la relación entre las estadísticas de los jugadores y el éxito para guiar la construcción de un equipo de béisbol con un presupuesto limitado. Antes de profundizar en este ejemplo más complejo, presentamos los conceptos necesarios para entender la regresión utilizando una ilustración más sencilla. De hecho, utilizamos el set de datos con el cual se inventó la regresión.

El ejemplo es de la genética. Francis Galton<sup>1</sup> estudió la variación y la herencia de los rasgos humanos. Entre muchos otros rasgos, Galton recopiló y estudió datos de estatura de familias para tratar de entender la herencia. Mientras hacía eso, desarrolló los conceptos de correlación y regresión, así como una conexión a pares de datos que siguen una distribución normal. Por supuesto, en el momento en que se recopilaron estos datos, nuestro conocimiento de la genética era bastante limitado en comparación con lo que sabemos hoy. Una pregunta muy específica que Galton intentó contestar fue: ¿cuán bien podemos predecir la altura de un niño basándose en la altura de los padres? La técnica que desarrolló para responder a esta pregunta, la regresión, también se puede aplicar a nuestra pregunta de béisbol y a muchas otras circunstancias.

**Nota histórica:** Galton hizo importantes contribuciones a las estadísticas y la genética, pero también fue uno de los primeros defensores de la eugenésica, un movimiento filosófico científicamente defectuoso favorecido por muchos biólogos de la época de Galton, pero con terribles consecuencias históricas<sup>2</sup>.

---

### 17.1 Estudio de caso: ¿la altura es hereditaria?

Tenemos acceso a los datos de altura familiar de Galton a través del paquete **HistData**. Estos datos contienen alturas de varias docenas de familias: madres, padres, hijas e hijos. Para imitar el análisis de Galton, crearemos un set de datos con las alturas de los padres y un hijo de cada familia seleccionado al azar:

```
library(tidyverse)
library(HistData)
data("GaltonFamilies")
```

---

<sup>1</sup>[https://en.wikipedia.org/wiki/Francis\\_Galton](https://en.wikipedia.org/wiki/Francis_Galton)

<sup>2</sup><https://pged.org/history-eugenics-and-genetics/>

```
set.seed(1983)
galton_heights <- GaltonFamilies %>%
 filter(gender == "male") %>%
 group_by(family) %>%
 sample_n(1) %>%
 ungroup() %>%
 select(father, childHeight) %>%
 rename.son = childHeight)
```

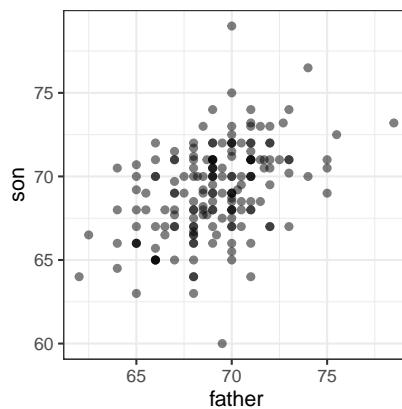
En los ejercicios, veremos otras relaciones, incluyendo la de madres e hijas.

Supongan que nos piden que resumamos los datos de padre e hijo. Dado que ambas distribuciones están bien aproximadas por la distribución normal, podríamos usar los dos promedios y las dos desviaciones estándar como resúmenes:

```
galton_heights %>%
 summarize(mean(father), sd(father), mean.son, sd.son))
#> # A tibble: 1 x 4
#> `mean(father)` `sd(father)` `mean.son` `sd.son`
#> <dbl> <dbl> <dbl> <dbl>
#> 1 69.1 2.55 69.2 2.71
```

Sin embargo, este resumen no describe una característica importante de los datos: la tendencia de que entre más alto es el padre, más alto es el hijo.

```
galton_heights %>% ggplot(aes(father, son)) +
 geom_point(alpha = 0.5)
```



Aprenderemos que el coeficiente de correlación es un resumen informativo de cómo dos variables se mueven juntas y luego veremos cómo esto se puede usar para predecir una variable usando la otra.

## 17.2 El coeficiente de correlación

El coeficiente de correlación se define para una lista de pares  $(x_1, y_1), \dots, (x_n, y_n)$  como el promedio del producto de los valores estandarizados:

$$\rho = \frac{1}{n} \sum_{i=1}^n \left( \frac{x_i - \mu_x}{\sigma_x} \right) \left( \frac{y_i - \mu_y}{\sigma_y} \right)$$

con  $\mu_x, \mu_y$  los promedios de  $x_1, \dots, x_n$  y  $y_1, \dots, y_n$ , respectivamente, y  $\sigma_x, \sigma_y$  las desviaciones estándar. La letra griega para  $r$ ,  $\rho$ , se usa comúnmente en libros de estadística para denotar la correlación porque es la primera letra de la palabra regresión. Pronto aprenderemos sobre la conexión entre correlación y regresión. Podemos representar la fórmula anterior con código R usando:

```
rho <- mean(scale(x) * scale(y))
```

Para entender por qué esta ecuación resume cómo se mueven juntas dos variables, consideren que la entrada número  $i$  de  $x$  está  $\left(\frac{x_i - \mu_x}{\sigma_x}\right)$  desviaciones estándar del promedio  $\mu_x$ . Del mismo modo, el  $y_i$  que se combina con  $x_i$ , está  $\left(\frac{y_i - \mu_y}{\sigma_y}\right)$  desviaciones estándar del promedio  $\mu_y$ . Si  $x$  e  $y$  no están relacionadas, el producto  $\left(\frac{x_i - \mu_x}{\sigma_x}\right) \left(\frac{y_i - \mu_y}{\sigma_y}\right)$  será positivo ( $+ \times +$  y  $- \times -$ ) tan frecuentemente como negativo ( $+ \times -$  y  $- \times +$ ) y tendrá un promedio de alrededor de 0. La correlación es el promedio de estos productos y, por lo tanto, las variables no relacionadas tendrán correlación 0. En cambio, si las cantidades varían juntas, entonces estamos promediando productos mayormente positivos ( $+ \times +$  y  $- \times -$ ) y obtenemos una correlación positiva. Si varían en direcciones opuestas, obtenemos una correlación negativa.

El coeficiente de correlación siempre está entre -1 y 1. Podemos mostrar esto matemáticamente: consideren que no podemos tener una correlación más alta que cuando comparamos una lista consigo misma (correlación perfecta) y, en este caso, la correlación es:

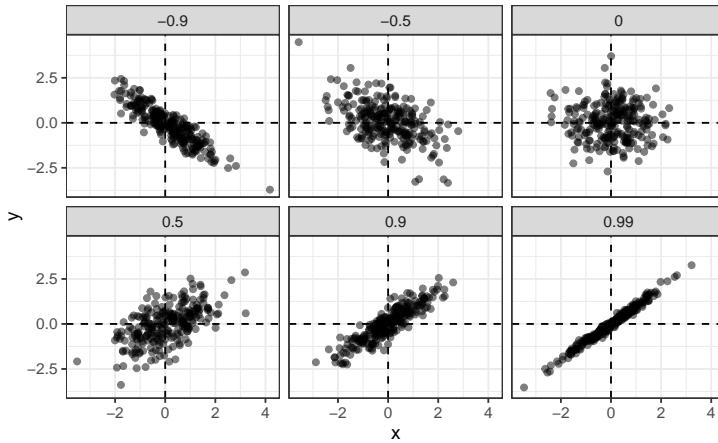
$$\rho = \frac{1}{n} \sum_{i=1}^n \left( \frac{x_i - \mu_x}{\sigma_x} \right)^2 = \frac{1}{\sigma_x^2} \frac{1}{n} \sum_{i=1}^n (x_i - \mu_x)^2 = \frac{1}{\sigma_x^2} \sigma_x^2 = 1$$

Una derivación similar, pero con  $x$  y su opuesto exacto, prueba que la correlación tiene que ser mayor o igual a -1.

Para otros pares, la correlación está entre -1 y 1. La correlación entre las alturas de padre e hijo es de aproximadamente 0.5:

```
galton_heights %>% summarize(r = cor(father, son)) %>% pull(r)
#> [1] 0.433
```

Para ver cómo se ven los datos para diferentes valores de  $\rho$ , aquí tenemos seis ejemplos de pares con correlaciones que van desde -0.9 a 0.99:



### 17.2.1 La correlación de muestra es una variable aleatoria

Antes de continuar conectando la correlación con la regresión, recordemos la variabilidad aleatoria.

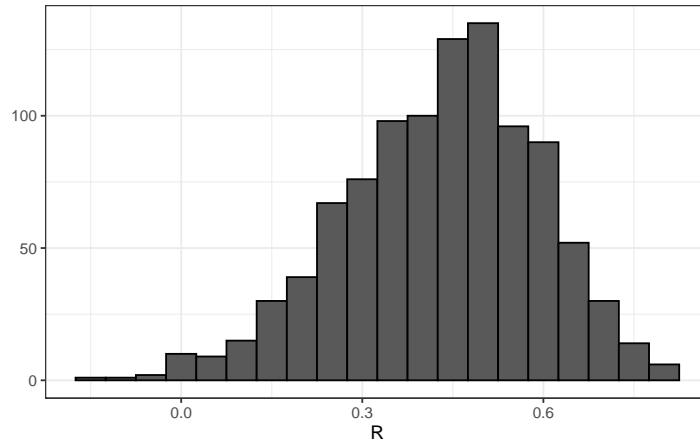
En la mayoría de las aplicaciones de ciencia de datos, observamos datos que incluyen variación aleatoria. Por ejemplo, en muchos casos, no observamos datos para toda la población de interés, sino para una muestra aleatoria. Al igual que con la desviación promedio y estándar, la *correlación de muestra* (*sample correlation* en inglés) es el estimador más comúnmente utilizado de la correlación de la población. Esto implica que la correlación que calculamos y usamos como resumen es una variable aleatoria.

A modo de ilustración, supongan que los 179 pares de padres e hijos es toda nuestra población. Un genetista menos afortunado solo puede permitirse mediciones de una muestra aleatoria de 25 pares. La correlación de la muestra se puede calcular con:

```
R <- sample_n(galton_heights, 25, replace = TRUE) %>%
 summarize(r = cor(father, son)) %>% pull(r)
```

R es una variable aleatoria. Podemos ejecutar una simulación Monte Carlo para ver su distribución:

```
B <- 1000
N <- 25
R <- replicate(B, {
 sample_n(galton_heights, N, replace = TRUE) %>%
 summarize(r=cor(father, son)) %>%
 pull(r)
})
qplot(R, geom = "histogram", binwidth = 0.05, color = I("black"))
```



Vemos que el valor esperado de R es la correlación de la población:

```
mean(R)
#> [1] 0.431
```

y que tiene un error estándar relativamente alto en relación con el rango de valores que R puede tomar:

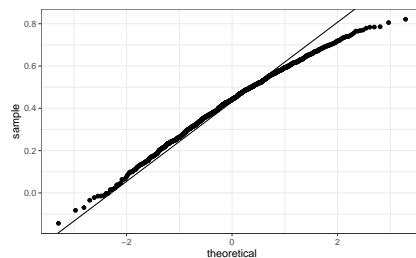
```
sd(R)
#> [1] 0.161
```

Entonces, al interpretar las correlaciones, recuerden que las correlaciones derivadas de las muestras son estimadores que contienen incertidumbre.

Además, tengan en cuenta que debido a que la correlación de la muestra es un promedio de sorteos independientes, el límite central aplica. Por lo tanto, para  $N$  suficientemente grande, la distribución de R es aproximadamente normal con valor esperado  $\rho$ . La desviación estándar, que es algo compleja para derivar, es  $\sqrt{\frac{1-\rho^2}{N-2}}$ .

En nuestro ejemplo,  $N = 25$  no parece ser lo suficientemente grande como para que la aproximación sea buena:

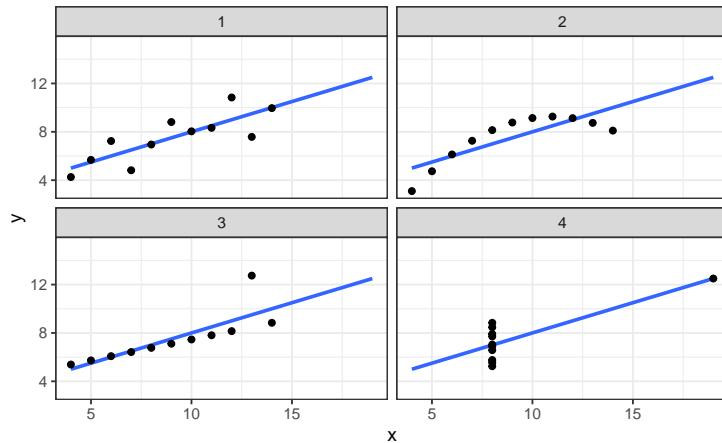
```
ggplot(aes(sample=R), data = data.frame(R)) +
 stat_qq() +
 geom_abline(intercept = mean(R), slope = sqrt((1-mean(R)^2)/(N-2)))
```



Si aumentan  $N$ , verán que la distribución converge a la normalidad.

### 17.2.2 La correlación no siempre es un resumen útil

La correlación no siempre es un buen resumen de la relación entre dos variables. Los siguientes cuatro sets de datos artificiales, conocidos como el cuarteto de Anscombe, ilustran este punto. Todos estos pares tienen una correlación de 0.82:



La correlación solo tiene sentido en un contexto particular. Para ayudarnos a entender cuándo la correlación tiene sentido como resumen estadístico, volveremos al ejemplo de predecir la altura de un hijo usando la altura de su padre. Esto ayudará a motivar y definir la regresión lineal. Comenzamos demostrando cómo la correlación puede ser útil para la predicción.

---

### 17.3 Valor esperado condicional

Supongan que se nos pide que adivinemos la altura de un hijo, seleccionado al azar, y no sabemos la altura de su padre. Debido a que la distribución de las alturas de los hijos es aproximadamente normal, conocemos que la altura promedio, 69.2, es el valor con la mayor proporción y sería la predicción con la mayor probabilidad de minimizar el error. Pero, ¿qué pasa si nos dicen que el padre es más alto que el promedio, digamos que mide 72 pulgadas, todavía adivinaríamos 69.2 para el hijo?

Resulta que si pudiéramos recopilar datos de un gran número de padres que miden 72 pulgadas, la distribución de las alturas de sus hijos sigue una distribución normal. Esto implica que el promedio de la distribución calculada en este subconjunto sería nuestra mejor predicción.

En general, llamamos a este enfoque *condicionar* (*conditioning* en inglés). La idea general es que estratificamos una población en grupos y calculamos resúmenes para cada grupo. Por lo tanto, condicionar está relacionado con el concepto de estratificación descrito en la Sección 8.13. Para proveer una descripción matemática del condicionamiento, consideren que tenemos una población de pares de valores  $(x_1, y_1), \dots, (x_n, y_n)$ , por ejemplo, todas las alturas de padres e hijos en Inglaterra. En el capítulo anterior, aprendimos que si tomamos un

par aleatorio  $(X, Y)$ , el valor esperado y el mejor predictor de  $Y$  es  $E(Y) = \mu_y$ , el promedio de la población  $1/n \sum_{i=1}^n y_i$ . Sin embargo, ya no estamos interesados en la población general, sino en el subconjunto de la población con un valor específico de  $x_i$ , 72 pulgadas en nuestro ejemplo. Este subconjunto de la población también es una población y, por lo tanto, los mismos principios y propiedades que hemos aprendido aplican. Los  $y_i$  en la subpoblación tienen una distribución, denominada *distribución condicional* (*conditional distribution* en inglés), y esta distribución tiene un valor esperado denominado *valor esperado condicional* (*conditional expectation* en inglés). En nuestro ejemplo, el valor esperado condicional es la altura promedio de todos los hijos en Inglaterra con padres que miden 72 pulgadas. La notación estadística para el valor esperado condicional es:

$$E(Y | X = x)$$

con  $x$  representando el valor fijo que define ese subconjunto, por ejemplo, en nuestro caso 72 pulgadas. Del mismo modo, denotamos la desviación estándar de los estratos con:

$$\text{SD}(Y | X = x) = \sqrt{\text{Var}(Y | X = x)}$$

Como el valor esperado condicional  $E(Y | X = x)$  es el mejor predictor para la variable aleatoria  $Y$  para un individuo en los estratos definidos por  $X = x$ , muchos retos de la ciencia de datos se reducen a los estimadores de esta cantidad. La desviación estándar condicional cuantifica la precisión de la predicción.

En el ejemplo que hemos estado considerando, queremos calcular la altura promedio del hijo *condicionado en* que el padre mida 72 pulgadas. Queremos estimar  $E(Y|X = 72)$  utilizando la muestra recopilada por Galton. Anteriormente aprendimos que el promedio de la muestra es el enfoque preferido para estimar el promedio de la población. Sin embargo, un reto al usar este enfoque para estimar los valores esperados condicionales es que para los datos continuos no tenemos muchos puntos de datos que coincidan exactamente con un valor en nuestra muestra. Por ejemplo, solo tenemos:

```
sum(galton_heights$father == 72)
#> [1] 8
```

padres que miden exactamente 72 pulgadas. Si cambiamos el número a 72.5, obtenemos aún menos puntos de datos:

```
sum(galton_heights$father == 72.5)
#> [1] 1
```

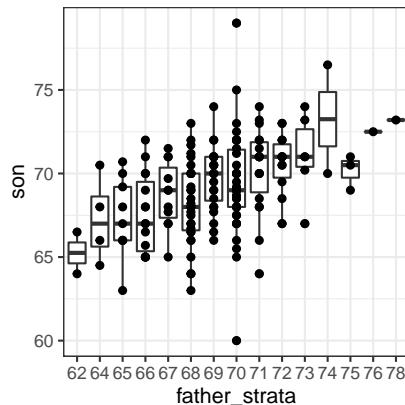
Una forma práctica de mejorar estos estimadores de los valores esperados condicionales es definir estratos con valores similares de  $x$ . En nuestro ejemplo, podemos redondear las alturas del padre a la pulgada más cercana y suponer que todas son 72 pulgadas. Si hacemos esto, terminamos con la siguiente predicción para el hijo de un padre que mide 72 pulgadas:

```
conditional_avg <- galton_heights %>%
 filter(round(father) == 72) %>%
 summarize(avg = mean(son)) %>%
 pull(avg)
conditional_avg
#> [1] 70.5
```

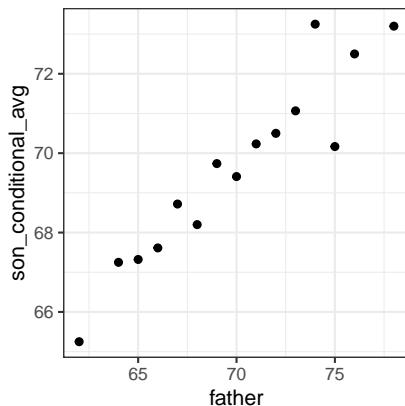
Noten que un padre que mide 72 pulgadas es más alto que el promedio, específicamente  $(72.0 - 69.1)/2.5 = 1.1$  desviaciones estándar más alto que el padre promedio. Nuestra predicción 70.5 también es más alta que el promedio, pero solo 0.49 desviaciones estándar más que el hijo promedio. Los hijos de padres que miden 72 pulgadas han *retrocedido* (*regressed* en inglés) un poco hacia la altura promedio. Observamos que la reducción en la cantidad de desviaciones estandár es de aproximadamente 0.5, que es la correlación. Como veremos en una sección posterior, esto no es una coincidencia.

Si queremos hacer una predicción de cualquier altura, no solo 72 pulgadas, podríamos aplicar el mismo enfoque a cada estrato. La estratificación seguida por diagramas de caja nos permite ver la distribución de cada grupo:

```
galton_heights %>% mutate(father_strata = factor(round(father))) %>%
 ggplot(aes(father_strata, son)) +
 geom_boxplot() +
 geom_point()
```



No es sorprendente que los centros de los grupos estén aumentando con la altura. Además, estos centros parecen seguir una relación lineal. A continuación graficamos los promedios de cada grupo. Si tomamos en cuenta que estos promedios son variables aleatorias con errores estándar, los datos son consistentes con estos puntos siguiendo una línea recta:



El hecho de que estos promedios condicionales sigan una línea no es una coincidencia. En la siguiente sección, explicamos que la línea que siguen estos promedios es lo que llamamos la *línea de regresión* (*regression line* en inglés), que mejora la precisión de nuestros estimadores. Sin embargo, no siempre es apropiado estimar los valores esperados condicionales con la línea de regresión, por lo que también describimos la justificación teórica de Galton para usar la línea de regresión.

## 17.4 La línea de regresión

Si estamos prediciendo una variable aleatoria  $Y$  sabiendo el valor de otra variable  $X = x$  y usando una línea de regresión, entonces predecimos que por cada desviación estándar  $\sigma_X$  que  $x$  aumenta por encima del promedio  $\mu_X$ ,  $Y$  aumenta  $\rho$  desviaciones estándar  $\sigma_Y$  por encima del promedio  $\mu_Y$  con  $\rho$  la correlación entre  $X$  y  $Y$ . Por lo tanto, la fórmula para la regresión es:

$$\left( \frac{Y - \mu_Y}{\sigma_Y} \right) = \rho \left( \frac{x - \mu_X}{\sigma_X} \right)$$

Podemos reescribirla así:

$$Y = \mu_Y + \rho \left( \frac{x - \mu_X}{\sigma_X} \right) \sigma_Y$$

Si hay una correlación perfecta, la línea de regresión predice un aumento que es el mismo número de desviaciones estandár. Si hay 0 correlación, entonces no usamos  $x$  para la predicción y simplemente predecimos el promedio  $\mu_Y$ . Para valores entre 0 y 1, la predicción está en algún punto intermedio. Si la correlación es negativa, predecimos una reducción en vez de un aumento.

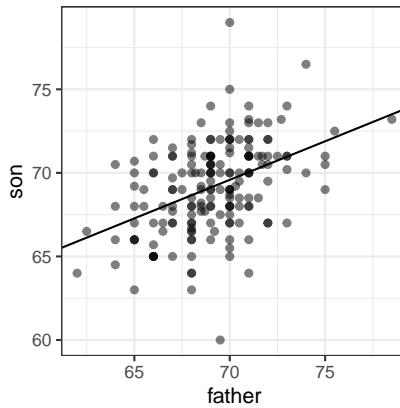
Tengan en cuenta que si la correlación es positiva e inferior a 1, nuestra predicción está más cerca, en unidades estándar, de la altura promedio que el valor utilizado para predecir,  $x$ , está del promedio de las  $xs$ . Es por eso que lo llamamos *regresión*: el hijo retrocede a la altura promedio. De hecho, el título del artículo de Galton era: *Regression towards mediocrity in hereditary stature*. Para añadir líneas de regresión a los gráficos, necesitaremos la siguiente versión de la fórmula anterior:

$$y = b + mx \text{ with slope } m = \rho \frac{\sigma_y}{\sigma_x} \text{ and intercept } b = \mu_y - m\mu_x$$

Aquí añadimos la línea de regresión a los datos originales:

```
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)
```

```
galton_heights %>%
 ggplot(aes(father, son)) +
 geom_point(alpha = 0.5) +
 geom_abline(slope = r * s_y/s_x, intercept = mu_y - r * s_y/s_x * mu_x)
```



La fórmula de regresión implica que si primero estandarizamos las variables, es decir, restamos el promedio y dividimos por la desviación estándar, entonces la línea de regresión tiene un intercepto 0 y una pendiente igual a la correlación  $r$ . Pueden hacer el mismo gráfico, pero usando unidades estándar así:

```
galton_heights %>%
 ggplot(aes(scale(father), scale(son))) +
 geom_point(alpha = 0.5) +
 geom_abline(intercept = 0, slope = r)
```

#### 17.4.1 La regresión mejora la precisión

Comparemos los dos enfoques de predicción que hemos presentado:

1. Redondear las alturas de los padres a la pulgada más cercana, estratificar y luego tomar el promedio.
2. Calcular la línea de regresión y usarla para predecir.

Utilizamos una simulación Monte Carlo que muestrea  $N = 50$  familias:

```
B <- 1000
N <- 50

set.seed(1983)
conditional_avg <- replicate(B, {
 dat <- sample_n(galton_heights, N)
 dat %>% filter(round(father) == 72) %>%
 summarize(avg = mean(son)) %>%
 pull(avg)
```

```

})
regression_prediction <- replicate(B, {
 dat <- sample_n(galton_heights, N)
 mu_x <- mean(dat$father)
 mu_y <- mean(dat$son)
 s_x <- sd(dat$father)
 s_y <- sd(dat$son)
 r <- cor(dat$father, dat$son)
 mu_y + r*(72 - mu_x)/s_x*s_y
})

```

Aunque el valor esperado de estas dos variables aleatorias es casi el mismo:

```

mean(conditional_avg, na.rm = TRUE)
#> [1] 70.5
mean(regression_prediction)
#> [1] 70.5

```

El error estándar para la predicción usando la regresión es sustancialmente más pequeño:

```

sd(conditional_avg, na.rm = TRUE)
#> [1] 0.964
sd(regression_prediction)
#> [1] 0.452

```

La línea de regresión es, por lo tanto, mucho más estable que la media condicional. Hay una razón intuitiva para esto. El promedio condicional se basa en un subconjunto relativamente pequeño: los padres que miden aproximadamente 72 pulgadas. De hecho, en algunas de las permutaciones no tenemos datos, por eso utilizamos `na.rm=TRUE`. La regresión siempre usa todos los datos.

Entonces, ¿por qué no siempre usar la regresión para predecir? Porque no siempre es apropiado. Por ejemplo, Anscombe ofreció casos para los cuales los datos no tienen una relación lineal. Entonces, ¿tiene sentido usar la línea de regresión para predecir en nuestro ejemplo? Galton encontró que sí, en el caso de los datos de altura. La justificación, que incluimos en la siguiente sección, es algo más avanzada que el resto del capítulo.

#### 17.4.2 Distribución normal de dos variables (avanzada)

La correlación y la pendiente de regresión son resúmenes estadísticos ampliamente utilizados, pero que a menudo se malinterpretan o se usan mal. Los ejemplos de Anscombe ofrecen casos simplificados de sets de datos en los que resumir con correlación sería un error. Sin embargo, hay muchos más ejemplos de la vida real.

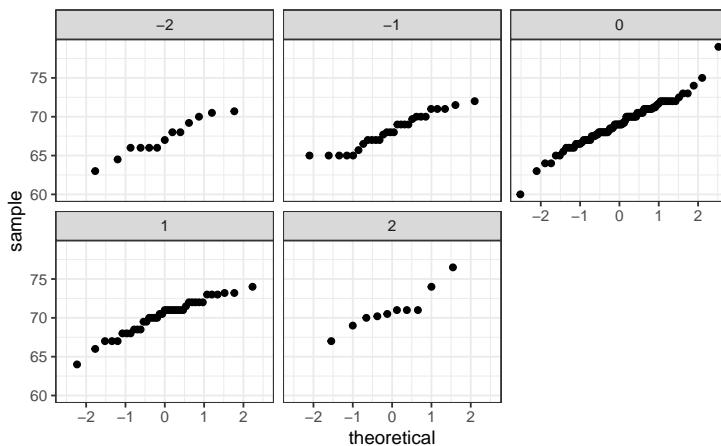
La manera principal en que motivamos el uso de la correlación involucra lo que se llama la *distribución normal de dos variables* (*bivariate normal distribution* en inglés).

Cuando un par de variables aleatorias se aproxima por la distribución normal de dos variables, los diagramas de dispersión parecen óvalos. Como vimos en la Sección 17.2, pueden ser delgados (alta correlación) o en forma de círculo (sin correlación).

Una forma más técnica de definir la distribución normal de dos variables es la siguiente: si  $X$  es una variable aleatoria normalmente distribuida,  $Y$  también es una variable aleatoria normalmente distribuida, y la distribución condicional de  $Y$  para cualquier  $X = x$  es aproximadamente normal, entonces el par sigue una distribución normal de dos variables.

Si creemos que los datos de altura están bien aproximados por la distribución normal de dos variables, entonces deberíamos ver que la aproximación normal aplica a cada estrato. Aquí estratificamos las alturas de los hijos por las alturas estandarizadas de los padres y vemos que el supuesto parece ser válido:

```
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)
```



Ahora volvemos a tratar de definir correlación. Galton utilizó estadísticas matemáticas para demostrar que, cuando dos variables siguen una distribución normal de dos variables, calcular la línea de regresión es equivalente a calcular los valores esperados condicionales. No mostramos la derivación aquí, pero podemos mostrar que bajo este supuesto, para cualquier valor dado de  $x$ , el valor esperado de  $Y$  en pares para los cuales  $X = x$  es:

$$E(Y|X = x) = \mu_Y + \rho \frac{X - \mu_X}{\sigma_X} \sigma_Y$$

Esta es la línea de regresión, con pendiente:

$$\rho \frac{\sigma_Y}{\sigma_X}$$

e intercepto  $\mu_Y - \rho \mu_X$ . Es equivalente a la ecuación de regresión que mostramos anteriormente que se puede escribir así:

$$\frac{E(Y | X = x) - \mu_Y}{\sigma_Y} = \rho \frac{x - \mu_X}{\sigma_X}$$

Esto implica que, si la distribución de nuestros datos se puede aproximar con una distribución normal de dos variables, la línea de regresión da el valor esperado condicional. Por lo tanto, podemos obtener un estimador mucho más estable del valor esperado condicional al encontrar la línea de regresión y usarla para predecir.

En resumen, si nuestros datos se pueden aproximar con una distribución normal de dos variables, entonces el valor esperado condicional, la mejor predicción de  $Y$  cuando sabemos el valor de  $X$ , lo da la línea de regresión.

### 17.4.3 Varianza explicada

La teoría de la distribución normal de dos variables también nos dice que la desviación estándar de la distribución *condicional* descrita anteriormente es:

$$\text{SD}(Y | X = x) = \sigma_Y \sqrt{1 - \rho^2}$$

Para ver por qué esto es intuitivo, observen que sin condicionar,  $\text{SD}(Y) = \sigma_Y$ , estamos viendo la variabilidad de todos los hijos. Pero tan pronto condicionamos, solo observamos la variabilidad de los hijos con un padre alto, de 72 pulgadas. Este grupo tenderá a ser algo alto, por lo que se reduce la desviación estándar.

Específicamente, se reduce a  $\sqrt{1 - \rho^2} = \sqrt{1 - 0.25} = 0.87$  de lo que era originalmente. Podríamos decir que las alturas de los padres “explican” el 13% de la variabilidad observada en las alturas de los hijos.

La declaración “ $X$  explica tal y cual porcentaje de la variabilidad” se usa comúnmente en los trabajos académicos. En este caso, este porcentaje realmente se refiere a la varianza (la desviación estándar al cuadrado). Entonces si los datos siguen una distribución normal de dos variables, la varianza se reduce por  $1 - \rho^2$  y entonces decimos que  $X$  explica  $1 - (1 - \rho^2) = \rho^2$  (la correlación al cuadrado) de la varianza.

Pero es importante recordar que la declaración de “varianza explicada” solo tiene sentido cuando los datos se aproximan mediante una distribución normal de dos variables.

### 17.4.4 Advertencia: hay dos líneas de regresión

Calculamos una línea de regresión para predecir la altura del hijo basada en la altura del padre. Utilizamos estos cálculos:

```
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_1 <- r * s_y / s_x
b_1 <- mu_y - m_1 * mu_x
```

que nos da la función  $E(Y | X = x) = 37.3 + 0.46 x$ .

¿Qué pasa si queremos predecir la altura del padre basada en la del hijo? Es importante

saber que esto no se determina calculando la función inversa:  $x = \{\text{E}(Y | X = x) - 37.3\}/0.5$ .

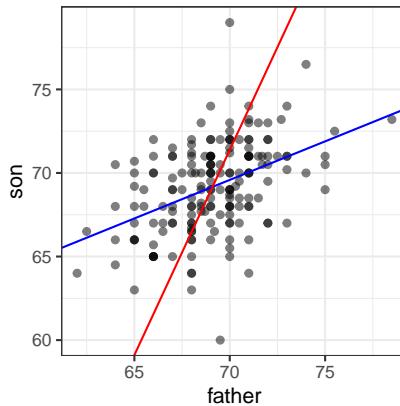
Necesitamos calcular  $\text{E}(X | Y = y)$ . Dado que los datos se aproximan mediante una distribución normal de dos variables, la teoría descrita anteriormente nos dice que este valor esperado condicional seguirá una línea con pendiente e intercepto:

```
m_2 <- r * s_x / s_y
b_2 <- mu_x - m_2 * mu_y
```

Entonces obtenemos  $\text{E}(X | Y = y) = 40.9 + 0.41y$ . Nuevamente vemos una regresión al promedio: la predicción para la altura del padre está más cerca del promedio del padre que la altura del hijo  $y$  está a la altura del hijo promedio.

Aquí tenemos un gráfico que muestra las dos líneas de regresión. La azul predice las alturas de los hijos según las alturas de los padres y la roja predice las alturas de los padres según las alturas de los hijos:

```
galton_heights %>%
 ggplot(aes(father, son)) +
 geom_point(alpha = 0.5) +
 geom_abline(intercept = b_1, slope = m_1, col = "blue") +
 geom_abline(intercept = -b_2/m_2, slope = 1/m_2, col = "red")
```



## 17.5 Ejercicios

1. Cargue los datos `GaltonFamilies` de **HistData**. Los niños de cada familia se enumeran por género y luego por estatura. Cree un set de datos llamado `galton_heights` eligiendo un varón y una hembra al azar.
2. Haga un diagrama de dispersión para las alturas entre madres e hijas, madres e hijos, padres e hijas, y padres e hijos.
3. Calcule la correlación en las alturas entre madres e hijas, madres e hijos, padres e hijas, y padres e hijos.

# 18

---

## Modelos lineales

---

Desde el desarrollo original de Galton, la regresión se ha convertido en una de las herramientas más utilizadas en la ciencia de datos. Una razón por esto es que la regresión nos permite encontrar relaciones entre dos variables tomando en cuenta los efectos de otras variables que afectan a ambas. Esto ha sido particularmente popular en campos donde los experimentos aleatorios son difíciles de ejecutar, como la economía y la epidemiología.

Cuando no podemos asignar aleatoriamente a cada individuo a un grupo de tratamiento o control, la confusión (*confounding* en inglés) es particularmente frecuente. Por ejemplo, consideren estimar el efecto de comer comidas rápidas en la esperanza de vida utilizando datos recopilados de una muestra aleatoria de personas en una jurisdicción. Es más probable que los consumidores de comida rápida sean fumadores, bebedores y tengan ingresos más bajos. Por lo tanto, un modelo sencillo de regresión puede sobreestimar el efecto negativo de la comida rápida en la salud. Entonces, ¿cómo explicamos la confusión en la práctica? En este capítulo aprendemos cómo los modelos lineales pueden ayudar con estas situaciones y cómo pueden usarse para describir cómo una o más variables afectan el resultado.

---

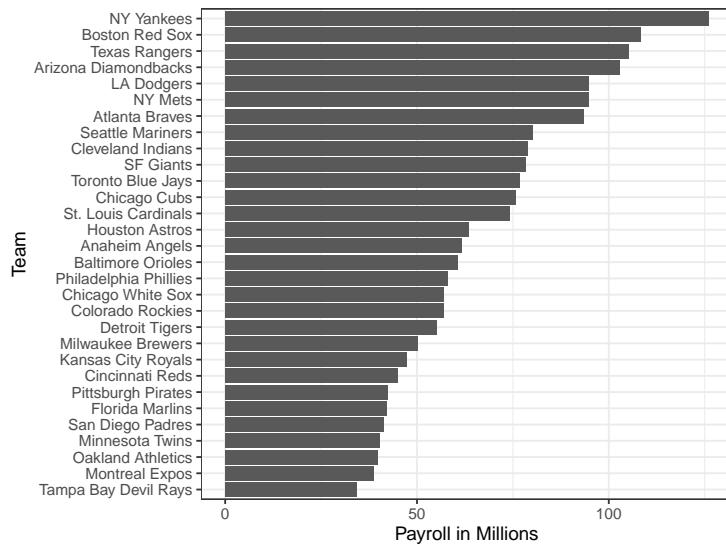
### 18.1 Estudio de caso: *Moneyball*

El libro *Moneyball: El arte de ganar un juego injusto* de Michael Lewis se trata del equipo de béisbol los Atléticos de Oakland, también conocidos como los A's, y su gerente general, la persona encargada de construir el equipo, Billy Beane.

Tradicionalmente, los equipos de béisbol usan *scouts*, o buscadores de talento, para ayudarlos a decidir qué jugadores contratar. Estos *scouts* evalúan a los jugadores viéndolos jugar. Por esta razón, tienden a favorecer a los jugadores atléticos con habilidades físicas observables y, en general, hay consenso entre los *scouts* sobre quiénes son los mejores jugadores. Como consecuencia, hay mucha demanda para estos jugadores, lo cual aumenta sus salarios.

De 1989 a 1991, los A's tuvieron una de las nóminas más altas del béisbol. Pudieron comprar los mejores jugadores y, durante estos años, fueron uno de los mejores equipos. Sin embargo, en 1995, el dueño del equipo cambió y la nueva administración recortó drásticamente el presupuesto, dejando al entonces gerente general, Sandy Alderson, con una de las nóminas más bajas en el béisbol. Éste ya no podía permitirse el lujo de los jugadores más codiciados y, por eso, comenzó a utilizar un enfoque estadístico para encontrar inefficiencias en el mercado. Alderson fue mentor de Billy Beane, quien lo sucedió en 1998 y aceptó por completo la ciencia de los datos, en vez de los *scouts*, como un método para encontrar jugadores de bajo costo que, según los datos, ayudarían al equipo a ganar. Hoy, esta estrategia ha sido adaptada por la mayoría de los equipos de béisbol. Como veremos, la regresión juega un papel importante en este enfoque.

Como motivación para este capítulo, fingiremos que es 2002 y trataremos de construir un equipo de béisbol con un presupuesto limitado, tal como lo hicieron los Atléticos. Para apreciar la dificultad del reto, tengan en cuenta que en 2002 la nómina de los Yankees de \$125,928,583 era más del triple de la de los Atléticos de Oakland de \$39,679,746.



### 18.1.1 Sabermetrics

Las estadísticas se han utilizado en el béisbol desde sus inicios. El set de datos que usaremos, que se incluye en el paquete **Lahman**, se remonta al siglo XIX. Por ejemplo, un resumen estadístico que describiremos pronto, el *promedio de bateo* (*batting average* en inglés), se ha utilizado durante décadas para resumir el éxito de un bateador. Otras estadísticas<sup>1</sup> como cuadrangulares (HR o *homeruns* en inglés), carreras impulsadas (RBI o *runs batted in* en inglés) y bases robadas (SB o *stolen bases* en inglés) se indican para cada jugador en los resúmenes del juego que se incluyen en la sección de deportes de periódicos, con jugadores recompensados por números altos. Aunque resúmenes estadísticos como estos se utilizaron ampliamente en el béisbol, el análisis de datos en sí se ignoraba. Estas estadísticas se escogieron arbitrariamente sin pensar mucho en si realmente predecían algo o si estaban relacionadas con ayudar a un equipo a ganar.

Esto cambió con Bill James<sup>2</sup>. A fines de la década de 1970, este fanático del béisbol y aspirante a escritor comenzó a publicar artículos que describían un análisis más profundo de los datos del béisbol. Denominó *sabermetrics*<sup>3</sup> al enfoque de usar datos para pronosticar qué resultados mejor predicen si un equipo ganará. Hasta que Billy Beane convirtió a *sabermetrics* en el centro de su operación de béisbol, el mundo del béisbol por lo general ignoró el trabajo de Bill James. Actualmente, la popularidad de *sabermetrics* ya no se limita solo al béisbol; varios otros deportes también han comenzado a usar este enfoque.

En este capítulo, para simplificar el ejercicio, nos enfocaremos en carreras (R o *runs* en inglés) anotadas e ignoraremos los otros dos aspectos importantes del juego: lanzar y fildear.

<sup>1</sup>[http://mlb.mlb.com/stats/league\\_leaders.jsp](http://mlb.mlb.com/stats/league_leaders.jsp)

<sup>2</sup>[https://en.wikipedia.org/wiki/Bill\\_James](https://en.wikipedia.org/wiki/Bill_James)

<sup>3</sup><https://en.wikipedia.org/wiki/Sabermetrics>

Veremos cómo el análisis de regresión puede ayudar a desarrollar estrategias para construir un equipo de béisbol competitivo con un presupuesto limitado. El acercamiento se puede dividir en dos análisis de datos separados. En el primero, determinamos qué estadísticas específicas del jugador predicen carreras. En el segundo, examinamos si los jugadores estaban infravalorados según lo que predice nuestro primer análisis.

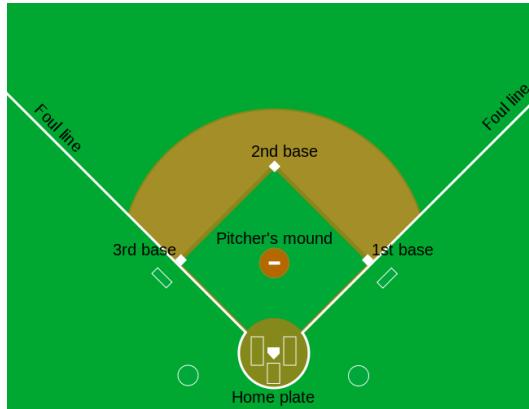
### 18.1.2 Conceptos básicos de béisbol

Para ver cómo la regresión nos ayudará a encontrar jugadores infravalorados, no necesitamos entender todos los detalles sobre el juego de béisbol, que tiene más de 100 reglas. Aquí, desfilamos el deporte al conocimiento básico que uno necesita saber para atacar efectivamente el reto de analizar los datos.

El objetivo de un juego de béisbol es anotar más carreras (puntos) que el otro equipo. Cada equipo tiene 9 bateadores que tienen la oportunidad de darle a una pelota con un bate en un orden predeterminado. Después de que el noveno bateador haya tenido su turno, el primer bateador vuelve a batear, luego el segundo y así sucesivamente. Cada vez que un bateador tiene la oportunidad de batear, lo llamamos una *turno al bate* (PA o *plate appearance* en inglés). En cada PA, el lanzador (*pitcher* en inglés) del otro equipo lanza la pelota y el bateador intenta darle. El PA termina con un resultado binario: el bateador hace un *out* (falla) y regresa al banco o el bateador le da a la bola (éxito) y puede correr alrededor de las bases, y potencialmente anotar una carrera (llegar a las 4 bases). Cada equipo tiene nueve intentos, denominados *entradas* (*innings* en inglés), para anotar carreras y cada entrada termina después de tres *outs*.

Aquí pueden ver un video que muestra un éxito: <https://www.youtube.com/watch?v=HL-XjMCPfio>. Y aquí hay uno que muestra una falla: <https://www.youtube.com/watch?v=NeloljCx-1g>. En estos videos, vemos cómo la suerte está involucrada en el proceso. Cuando está al bate, el bateador quiere darle a la pelota con fuerza. Si le da lo suficientemente fuerte, es un HR o cuadrangular, el mejor resultado posible ya que el bateador obtiene al menos una carrera automática. Pero a veces, debido al azar, el bateador le da a la pelota muy fuerte y un defensor la atrapa, lo que resulta en un *out*. Por el contrario, a veces el bateador le da a la pelota suavemente, pero cae justo en el lugar correcto. El hecho de que el azar afecta sugiere por qué los modelos de probabilidad son útiles.

Ahora hay varias formas de tener éxito. Entender esta distinción será importante para nuestro análisis. Cuando el bateador le da a la pelota, él quiere pisar cuantas más bases sea posible. Hay cuatro bases y la cuarta se llama *home* o *home plate*. Ahí es donde los bateadores comienzan bateando, por lo que las bases forman un ciclo.



(Cortesía de Cburnett<sup>4</sup>. Licencia CC BY-SA 3.0<sup>5</sup>.)

Un bateador que *llega a todas las bases* y a *home*, anota una carrera.

Estamos simplificando un poco, pero hay cinco formas en que un bateador puede tener éxito, es decir, no hacer un *out*:

- Bases por bolas (BB): el lanzador no logra lanzar la pelota dentro de un área predefinida donde el bateador le puede dar (la zona de *strike*), por lo que el bateador puede ir a primera base.
- Sencillo: el bateador le da a la bola y llega a primera base.
- Doble (2B): el bateador le da a la bola y llega a segunda base.
- Triple (3B): el bateador le da a la bola y llega a tercera base.
- Cuadrangular (HR)<sup>6</sup>: el bateador le da a la bola, llega a *home* y anota una carrera.

Si un bateador llega a una base, ese bateador aún tiene la posibilidad de llegar a *home* y anotar una carrera si el siguiente bateador batea con éxito. Mientras el bateador está *en base*, él también puede intentar robarse una base (SB o *stolen bases* en inglés). Si un bateador corre lo suficientemente rápido, el bateador puede intentar ir de una base a la siguiente sin que el otro equipo lo toque (*tag* en inglés)<sup>7</sup>.

Todos estos eventos se registran durante la temporada y están disponibles para nosotros a través del paquete **Lahman**. Ahora comenzaremos a discutir cómo el análisis de datos puede ayudarnos a decidir cómo usar estas estadísticas para evaluar a los jugadores.

### 18.1.3 No hay premios para BB

Históricamente, el *promedio de bateo* se ha considerado la estadística ofensiva más importante. Para definir este promedio, definimos un *hit* (H) y un *al bate* (AB o *at bat* en inglés). Sencillos, dobles, triples y cuadrangulares son éxitos. La quinta forma de tener éxito, BB, no es un éxito. Un AB es la cantidad de veces que un bateador logra un *hit* o un *out*; los BB se excluyen. El promedio de bateo es simplemente H/AB y se considera la medida principal de una tasa de éxito. Hoy esta tasa de éxito oscila entre el 20% y el 38%. Nos referimos al

<sup>4</sup><https://en.wikipedia.org/wiki/User:Cburnett>

<sup>5</sup><https://creativecommons.org/licenses/by-sa/3.0/deed.en>

<sup>6</sup><https://www.youtube.com/watch?v=xYxSZJ9GZ-w>

<sup>7</sup><https://www.youtube.com/watch?v=JSE5kfxkzfk>

promedio de bateo en miles, por lo que, por ejemplo, si su índice de éxito es 28%, decimos que *está bateando 280*.



(Imagen cortesía de Keith Allison<sup>8</sup>. Licencia CC BY-SA 2.0<sup>9</sup>. )

Una de las primeras ideas importantes de Bill James es que el promedio de bateo ignora BB, pero un BB es un éxito. James propuso que se usara el *on-base percentage* (OBP), el porcentaje de veces que un bateador llega a una base, en lugar del promedio de bateo. Definió OBP como  $(H + BB)/(AB + BB)$ , que es simplemente la proporción de turnos al bate que no resultan en un *out*, una medida muy intuitiva. Señaló que un jugador que obtiene muchos más BB que el jugador promedio podría no ser reconocido si su promedio de bateo no es alto. ¿Pero este jugador no está ayudando a producir carreras? Aún así, no se le otorga premio al jugador con más BB. Además, el béisbol no adoptó de inmediato el OBP como una estadística importante. En cambio, el total de bases robadas se considera importante y le otorgan un premio al jugador con la mayor cantidad<sup>10</sup>. Pero los jugadores con altos totales de SB también hacen más *outs* ya que no siempre tienen éxito. ¿Un jugador con un alto total de SB ayuda a producir carreras? ¿Podemos usar la ciencia de datos para determinar si es mejor pagar por jugadores con totales altos de BB o de SB?

#### 18.1.4 ¿Base por bolas o bases robadas?

Uno de los desafíos en este análisis es que no es obvio cómo determinar si un jugador produce carreras porque mucho depende de sus compañeros de equipo. Llevamos un registro del número de carreras anotadas por un jugador. Sin embargo, recuerden que si un jugador X batea justo antes de alguien que logra muchos cuadrangulares, el bateador X marcará muchas carreras. Pero estas carreras no necesariamente suceden si contratamos al jugador X pero no a su compañero de equipo que batea cuadrangulares. No obstante, podemos examinar las estadísticas a nivel de equipo. ¿Cómo se comparan los equipos con muchos SB con los equipos con pocos? ¿Qué tal BB? ¡Tenemos datos! Vamos a examinar algunos.

Comencemos con una obvia: HR. ¿Los equipos que tienen más cuadrangulares anotan más carreras? Examinamos los datos de 1961 a 2001. La visualización de las opciones al explorar la relación entre dos variables, como HR y triunfos, es un diagrama de dispersión:

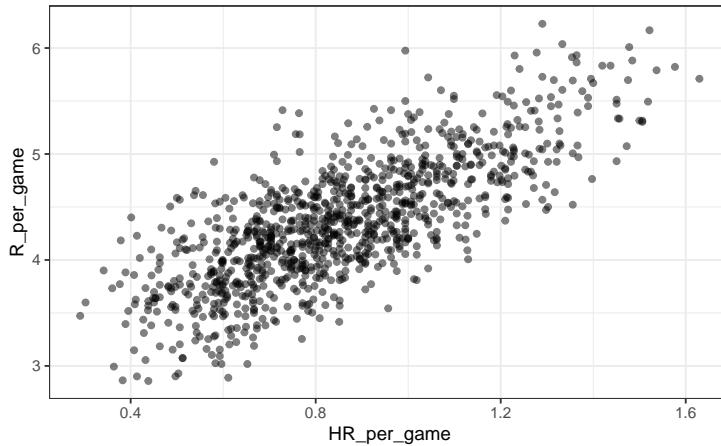
<sup>8</sup><https://www.flickr.com/people/27003603@N00>

<sup>9</sup><https://creativecommons.org/licenses/by-sa/2.0>

<sup>10</sup>[http://www.baseball-almanac.com/awards/lou\\_brock\\_award.shtml](http://www.baseball-almanac.com/awards/lou_brock_award.shtml)

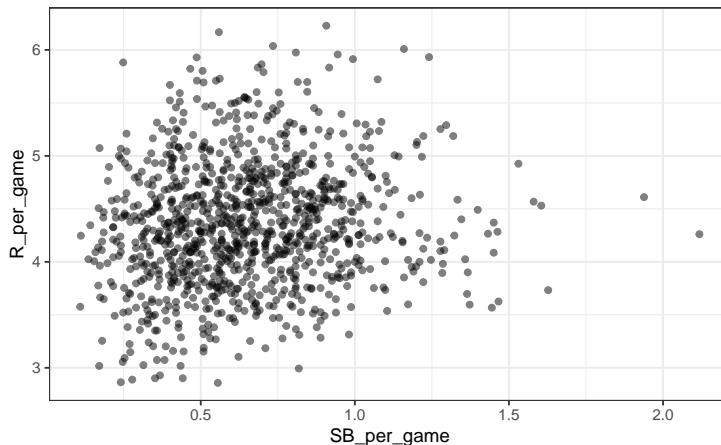
```
library(Lahman)
```

```
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)
```



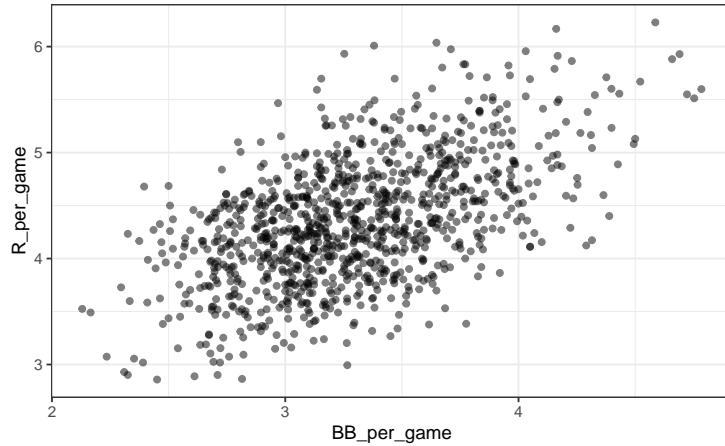
El gráfico muestra una fuerte asociación: los equipos con más HR tienden a anotar más carreras. Ahora examinemos la relación entre bases robadas y carreras:

```
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)
```



Aquí la relación no es tan clara. Finalmente, examinemos la relación entre BB y carreras:

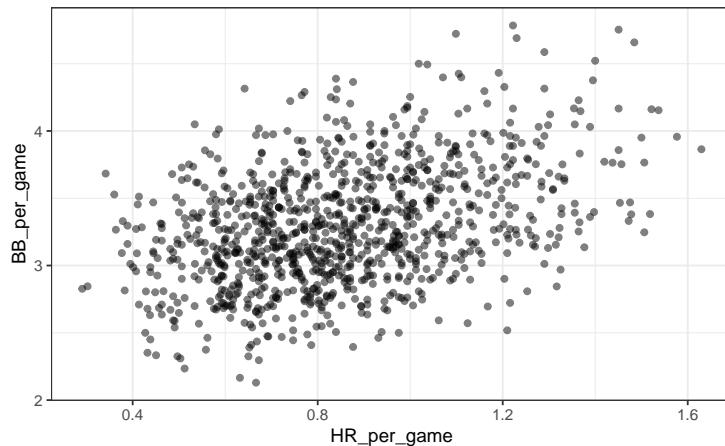
```
Teams %>% filter(yearID %in% 1961:2001) %>%
 mutate(BB_per_game = BB/G, R_per_game = R/G) %>%
 ggplot(aes(BB_per_game, R_per_game)) +
 geom_point(alpha = 0.5)
```



Aquí nuevamente vemos una asociación clara. Pero, ¿esto significa que aumentar las BB de un equipo **causa** un aumento en las carreras? Una de las lecciones más importantes que aprenderemos en este libro es que **la asociación no implica causalidad**.

De hecho, parece que los BB y HR también están asociados:

```
Teams %>% filter(yearID %in% 1961:2001) %>%
 mutate(HR_per_game = HR/G, BB_per_game = BB/G) %>%
 ggplot(aes(HR_per_game, BB_per_game)) +
 geom_point(alpha = 0.5)
```



Sabemos que los HR causan carreras porque, como su nombre sugiere, cuando un jugador logra un “home run”, se le garantiza al menos una carrera. ¿Podría ser que los HR también causen BB y esto hace que parezca que los BB también causen carreras? Cuando esto sucede,

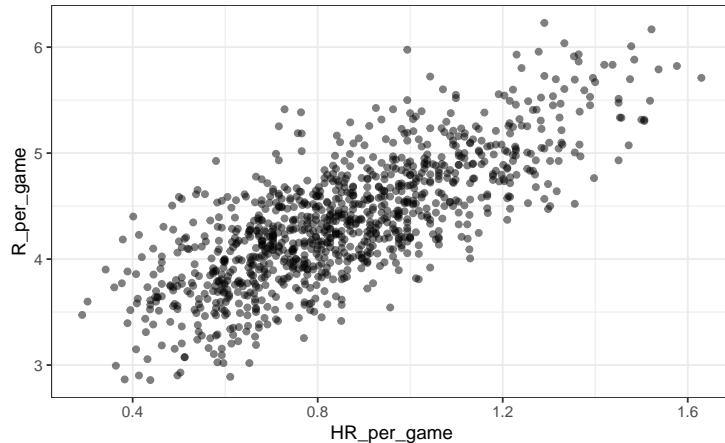
decimos que hay *confusión*, un concepto importante sobre el que aprenderemos más a lo largo de este capítulo.

La regresión lineal nos ayudará a analizar todo esto y cuantificar las asociaciones para determinar qué jugadores reclutar. Específicamente, trataremos de predecir cosas como cuántas carreras más anotará un equipo si aumentamos el número de BB, pero mantenemos los HR fijos. La regresión nos ayudará a responder preguntas como esta.

### 18.1.5 Regresión aplicada a las estadísticas de béisbol

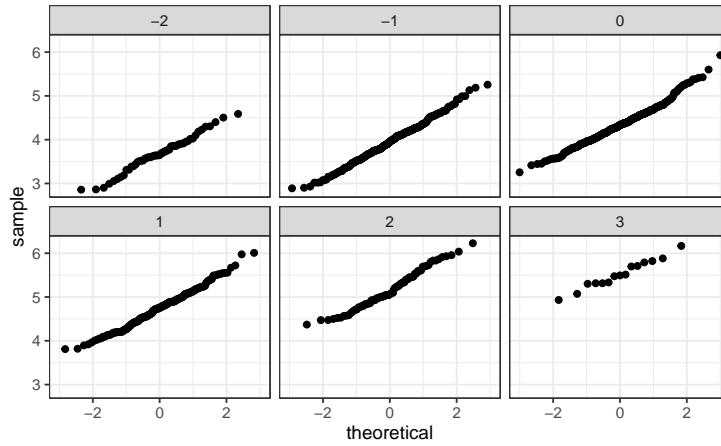
¿Podemos usar la regresión con estos datos? Primero, observen que los datos de HR y carreras parecen seguir una distribución normal de dos variables. Guardamos el gráfico en el objeto `p` ya que lo usaremos más tarde.

```
library(Lahman)
p <- 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)
```



Los gráficos Q-Q confirman que la aproximación normal es útil aquí:

```
Teams %>% filter(yearID %in% 1961:2001) %>%
 mutate(z_HR = round((HR - mean(HR))/sd(HR)),
 R_per_game = R/G) %>%
 filter(z_HR %in% -2:3) %>%
 ggplot() +
 stat_qq(aes(sample=R_per_game)) +
 facet_wrap(~z_HR)
```



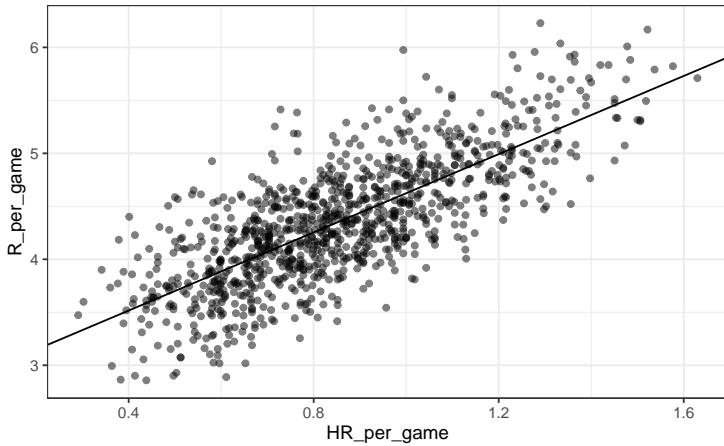
Ahora estamos listos para usar la regresión lineal para predecir el número de carreras que anotará un equipo si sabemos cuántos cuadrangulares logrará el equipo. Lo único que necesitamos hacer es calcular los cinco resúmenes estadísticos:

```
summary_stats <- Teams %>%
 filter(yearID %in% 1961:2001) %>%
 mutate(HR_per_game = HR/G, R_per_game = R/G) %>%
 summarize(avg_HR = mean(HR_per_game),
 s_HR = sd(HR_per_game),
 avg_R = mean(R_per_game),
 s_R = sd(R_per_game),
 r = cor(HR_per_game, R_per_game))
summary_stats
#> avg_HR s_HR avg_R s_R r
#> 1 0.855 0.243 4.36 0.589 0.762
```

y usar las fórmulas dadas arriba para crear las líneas de regresión:

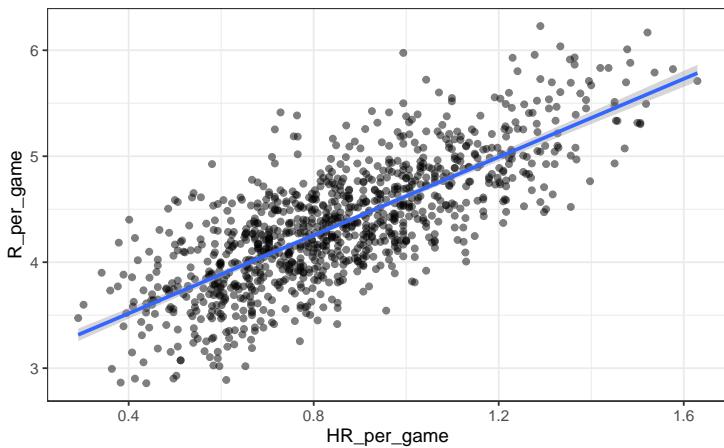
```
reg_line <- summary_stats %>% summarize(slope = r*s_R/s_HR,
 intercept = avg_R - slope*avg_HR)

p + geom_abline(intercept = reg_line$intercept, slope = reg_line$slope)
```



Pronto aprenderemos más sobre las funciones de R, como `lm`, que facilitan el ajuste de las líneas de regresión. Otro ejemplo que estudiaremos es la función `geom_smooth` de `ggplot2` que calcula y agrega una línea de regresión junto con intervalos de confianza al gráfico. Usamos el argumento `method = "lm"` que significa *modelo lineal* (*linear model* en inglés), el título de una próxima sección. Entonces podemos simplificar el código anterior así:

```
p + geom_smooth(method = "lm")
```



En el ejemplo anterior, la pendiente es 1.845. Esto nos dice que los equipos que logran 1 HR más por juego que el equipo promedio, anotan más carreras por juego que el equipo promedio. Dado que la puntuación final más común es la diferencia de una carrera, esto ciertamente puede conducir a un gran aumento en victorias. No es sorprendente que los jugadores con muchos HR sean muy caros. Debido a que estamos trabajando con un presupuesto limitado, necesitaremos encontrar otra forma de aumentar las victorias. Entonces, en la siguiente sección, trasladamos nuestra atención a BB.

## 18.2 Confusión

Anteriormente, notamos una fuerte relación entre carreras y BB. Si encontramos la línea de regresión para predecir carreras desde bases por bolas, obtendremos una pendiente de:

```
library(tidyverse)
library(Lahman)
get_slope <- function(x, y) cor(x, y) * sd(y)/ sd(x)

bb_slope <- Teams %>%
 filter(yearID %in% 1961:2001) %>%
 mutate(BB_per_game = BB/G, R_per_game = R/G) %>%
 summarize(slope = get_slope(BB_per_game, R_per_game))

bb_slope
#> slope
#> 1 0.735
```

Entonces, ¿esto significa que si contratamos jugadores de bajo salario con muchos BB y así aumentamos por 2 el número de BB por juego, nuestro equipo marcará 1.5 más carreras por juego?

Nuevamente debemos recordar que la asociación no implica la causalidad. Los datos ofrecen evidencia sólida de que un equipo con dos BB más por juego que el equipo promedio, anota 1.5 carreras por juego. Pero esto no significa que los BB sean la causa.

Noten que si calculamos la pendiente de la línea de regresión para sencillos obtenemos:

```
singles_slope <- Teams %>%
 filter(yearID %in% 1961:2001) %>%
 mutate(Singles_per_game = (H-HR-X2B-X3B)/G, R_per_game = R/G) %>%
 summarize(slope = get_slope(Singles_per_game, R_per_game))

singles_slope
#> slope
#> 1 0.449
```

que es un valor más bajo que el que obtenemos para BB.

Además, observen que un sencillo lleva a un jugador a primera base igual que un BB. Aquellos que saben de béisbol señalarán que con un sencillo, los corredores en base tienen una mejor oportunidad de anotar que con un BB. Entonces, ¿cómo puede un BB ser más predictivo de las carreras? La razón por la que esto sucede es por *confusión*. Aquí mostramos la correlación entre HR, BB y sencillos:

```
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
```

Resulta que los lanzadores, temerosos de los HR, a veces evitarán lanzar *strikes* a los bateadores de HR. Como resultado, los bateadores de HR tienden a tener más BB y un equipo con muchos bateadores de HR también tendrá más BB. Aunque puede parecer que BB causan carreras, realmente son HR los que causan la mayoría de estas carreras. Decimos que BB están *confundidos* (*confounded* en inglés) con HR. Sin embargo, ¿es posible que las BB todavía ayuden? Para averiguar, tenemos que ajustar para el efecto de HR. La regresión también puede ayudar con esto.

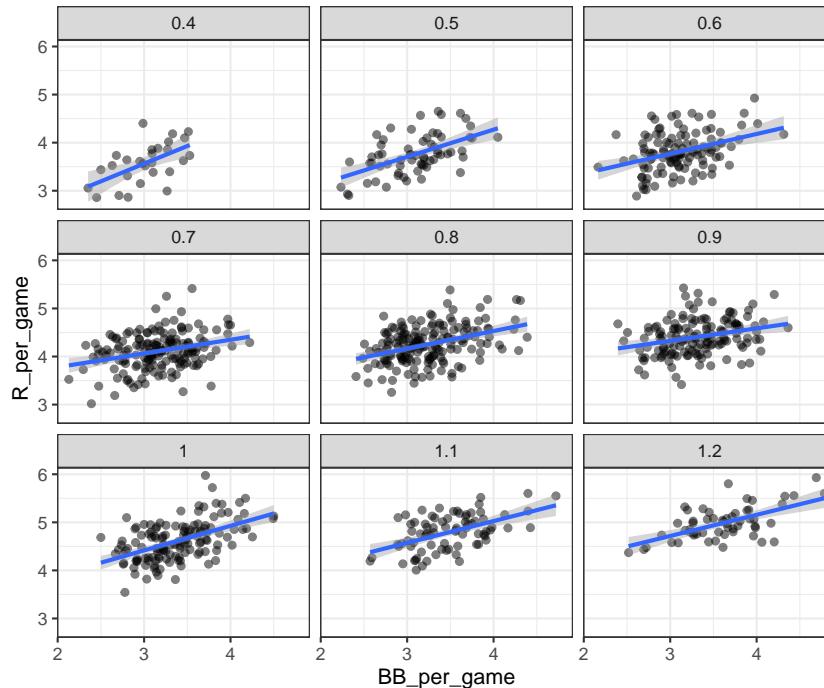
### 18.2.1 Cómo entender la confusión a través de la estratificación

Un primer acercamiento es mantener los HR fijos a un valor determinado y luego examinar la relación entre BB y las carreras. Como lo hicimos cuando estratificamos a los padres redondeando a la pulgada más cercana, aquí podemos estratificar HR por juego a los diez más cercanos. Filtramos los estratos con pocos puntos para evitar estimadores muy variables:

```
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)
```

y luego hacemos un diagrama de dispersión para cada estrato:

```
dat %>%
 ggplot(aes(BB_per_game, R_per_game)) +
 geom_point(alpha = 0.5) +
 geom_smooth(method = "lm") +
 facet_wrap(~ HR_strata)
```

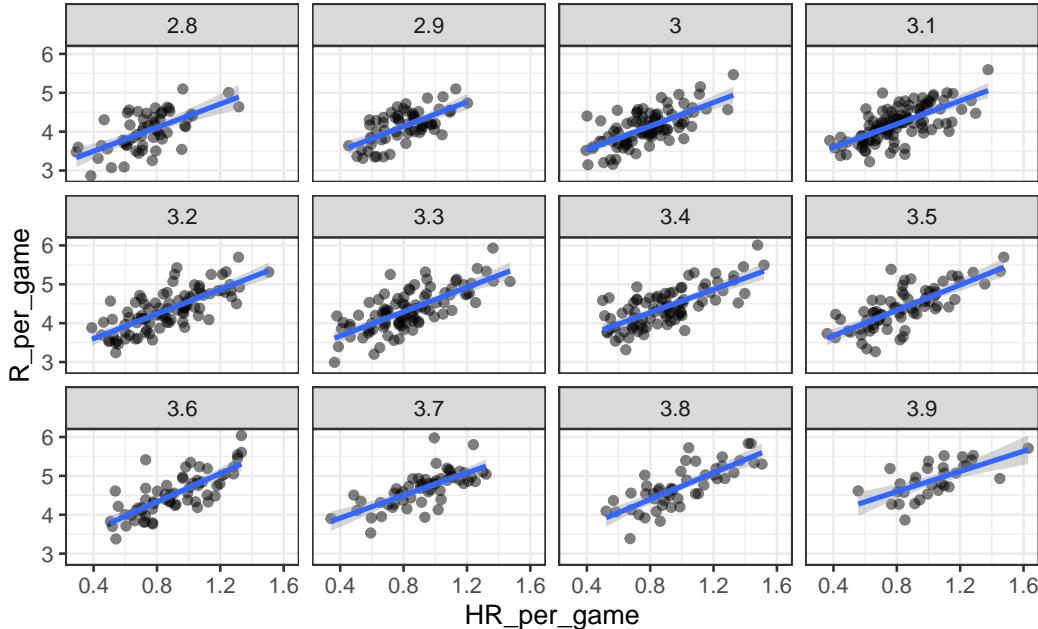


Recuerden que la pendiente de regresión para predecir carreras con BB era 0.7. Una vez que estratificamos por HR, estas pendientes se reducen sustancialmente:

```
dat %>%
 group_by(HR_strata) %>%
 summarize(slope = get_slope(BB_per_game, R_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
#> # ... with 4 more rows
```

Las pendientes se reducen, pero no son 0, lo que indica que las BB son útiles para producir carreras, pero no tanto como se pensaba anteriormente. De hecho, los valores anteriores están más cerca de la pendiente que obtuvimos de sencillos, 0.45, que es más consistente con nuestra intuición. Dado que tanto los sencillos como los BB nos llevan a primera base, deberían tener aproximadamente el mismo poder predictivo.

Aunque nuestra comprensión de la aplicación nos dice que los HR causan BB pero no al revés, aún podemos verificar si la estratificación por BB hace que el efecto de HR disminuya. Para hacer esto, usamos el mismo código, excepto que intercambiamos HR y BB para obtener este gráfico:



En este caso, las pendientes no cambian mucho del valor original:

```
dat %>% group_by(BB_strata) %>%
 summarize(slope = get_slope(HR_per_game, R_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
#> # ... with 7 more rows
```

Se reducen un poco, lo que es consistente con el hecho de que BB sí conducen a algunas carreras.

```
hr_slope <- Teams %>%
 filter(yearID %in% 1961:2001) %>%
 mutate(HR_per_game = HR/G, R_per_game = R/G) %>%
 summarize(slope = get_slope(HR_per_game, R_per_game))

hr_slope
#> slope
#> 1 1.84
```

De todos modos, parece que si estratificamos por HR, tenemos una distribución normal de dos variables para carreras versus BB. Del mismo modo, si estratificamos por BB, tenemos una distribución normal de dos variables aproximada para HR versus carreras.

### 18.2.2 Regresión lineal múltiple

Es un poco complejo calcular líneas de regresión para cada estrato. Básicamente, estamos ajustando modelos como este:

$$E[R | BB = x_1, HR = x_2] = \beta_0 + \beta_1(x_2)x_1 + \beta_2(x_1)x_2$$

con las pendientes para  $x_1$  cambiando para diferentes valores de  $x_2$  y viceversa. ¿Pero hay un acercamiento más fácil?

Si tomamos en cuenta la variabilidad aleatoria, las pendientes en los estratos no parecen cambiar mucho. Si estas pendientes son iguales, esto implica que  $\beta_1(x_2)$  y  $\beta_2(x_1)$  son constantes. Esto a su vez implica que la expectativa de carreras condicionadas en HR y BB se puede escribir así:

$$E[R | BB = x_1, HR = x_2] = \beta_0 + \beta_1x_1 + \beta_2x_2$$

Este modelo sugiere que si el número de HR se fija en  $x_2$ , observamos una relación lineal entre carreras y BB con un intercepto de  $\beta_0 + \beta_2x_2$ . Nuestro análisis exploratorio de datos sugirió esto. El modelo también sugiere que a medida que aumenta el número de HR, el crecimiento del intercepto también es lineal y está determinado por  $\beta_1x_1$ .

En este análisis, denominado *regresión lineal múltiple* (*multivariable linear regression* en inglés), a menudo escucharán a la gente decir que la pendiente BB  $\beta_1$  está *ajustada* (*adjusted* en inglés) para el efecto HR. Si el modelo es correcto, entonces se ha tomado en cuenta la confusión. ¿Pero cómo estimamos  $\beta_1$  y  $\beta_2$  de los datos? Para esto, aprendemos sobre modelos lineales y estimaciones de mínimos cuadrados.

## 18.3 Estimaciones de mínimos cuadrados

Hemos explicado cómo cuando los datos tienen una distribución normal de dos variables, entonces los valores esperados condicionales siguen la línea de regresión. El hecho de que el valor esperado condicional es una línea no es una suposición adicional, sino más bien un resultado derivado. Sin embargo, en la práctica es común escribir un modelo que describa la relación entre dos o más variables utilizando un *modelo lineal* (*linear model* en inglés).

Notamos que “lineal” aquí no se refiere exclusivamente a líneas, sino al hecho de que el valor esperado condicional es una combinación lineal de cantidades conocidas. En matemáticas, cuando multiplicamos cada variable por una constante y luego las sumamos, decimos que formamos una combinación lineal de las variables. Por ejemplo,  $3x - 4y + 5z$  es una combinación lineal de  $x$ ,  $y$  y  $z$ . Además, podemos añadir una constante y por eso  $2 + 3x - 4y + 5z$  también es una combinación lineal de  $x$ ,  $y$  y  $z$ .

Entonces  $\beta_0 + \beta_1x_1 + \beta_2x_2$  es una combinación lineal de  $x_1$  y  $x_2$ . El modelo lineal más sencillo es una constante  $\beta_0$ ; el segundo más sencillo es una línea  $\beta_0 + \beta_1x$ . Si tuviéramos que especificar un modelo lineal para los datos de Galton, denotaríamos las  $N$  alturas de padres observadas con  $x_1, \dots, x_n$ , entonces modelamos las  $N$  alturas de hijos que estamos tratando de predecir con:

$$Y_i = \beta_0 + \beta_1 x_i + \varepsilon_i, \quad i = 1, \dots, N.$$

Aquí  $x_i$  es la altura del padre, que es fija (no aleatoria) debido al condicionamiento, y  $Y_i$  es la altura aleatoria del hijo que queremos predecir. Suponemos además que  $\varepsilon_i$  son independientes entre sí, tienen valor esperado 0 y la desviación estándar, llámenla  $\sigma$ , no depende de  $i$ .

En el modelo anterior, sabemos el  $x_i$ , pero para tener un modelo útil para la predicción, necesitamos  $\beta_0$  y  $\beta_1$ . Los estimamos a partir de los datos. Una vez que hagamos esto, podemos predecir alturas de hijos para cualquier altura de padre  $x$ . Mostramos cómo hacer esto en la siguiente sección.

Noten que si además suponemos que el  $\varepsilon$  tiene una distribución normal, entonces este modelo es exactamente el mismo que obtuvimos anteriormente suponiendo que los datos siguen una distribución normal de dos variables. Una diferencia algo matizada es que en el primer acercamiento suponemos que los datos siguen una distribución normal de dos variables y que no suponemos un modelo lineal, sino que lo derivamos. En la práctica, los modelos lineales son simplemente supuestos y no necesariamente suponemos normalidad: la distribución de  $\varepsilon$  no se especifica. Sin embargo, si sus datos siguen una distribución normal de dos variables, se cumple el modelo lineal anterior. Si sus datos no siguen una distribución normal de dos variables, necesitarán justificar el modelo de otra forma.

### 18.3.1 Interpretando modelos lineales

Una razón por la que los modelos lineales son populares es porque son interpretables. En el caso de los datos de Galton, podemos interpretar los datos de esta manera: debido a los genes heredados, la predicción de la altura del hijo crece por  $\beta_1$  para cada pulgada que aumentamos la altura del padre  $x$ . Porque no todos los hijos con padres de estatura  $x$  son de la misma altura, necesitamos el término  $\varepsilon$ , lo que explica la variabilidad restante. Esta variabilidad restante incluye el efecto genético de la madre, los factores ambientales y otros factores biológicos aleatorios.

Dada la forma en que escribimos el modelo anterior, el intercepto  $\beta_0$  no es muy interpretable, ya que es la altura pronosticada de un hijo con un padre que mide 0 pulgadas. Debido a la regresión a la media, la predicción generalmente será un poco más grande que 0. Para hacer que el parámetro de pendiente sea más interpretable, podemos reescribir el modelo como:

$$Y_i = \beta_0 + \beta_1(x_i - \bar{x}) + \varepsilon_i, \quad i = 1, \dots, N$$

con  $\bar{x} = 1/N \sum_{i=1}^N x_i$  el promedio de  $x$ . En este caso,  $\beta_0$  representa la altura cuando  $x_i = \bar{x}$ , que es la altura del hijo de un padre promedio.

### 18.3.2 Estimadores de mínimos cuadrados (LSE)

Para que los modelos lineales sean útiles, tenemos que estimar los  $\beta$ s desconocidos. El enfoque estándar en la ciencia es encontrar los valores que minimizan la distancia del modelo ajustado a los datos. La siguiente ecuación se llama la ecuación de mínimos cuadrados (LS o *least squares* en inglés) y la veremos a menudo en este capítulo. Para los datos de Galton, escribiríamos:

$$RSS = \sum_{i=1}^n \{y_i - (\beta_0 + \beta_1 x_i)\}^2$$

Esta cantidad se denomina suma de errores cuadrados (*residual sum of squares* o RSS por sus siglas en inglés). Una vez que encontremos los valores que minimizan el RSS, llamaremos a los valores los estimadores de mínimos cuadrados (*least squares estimates* o LSE por sus siglas en inglés) y los denotaremos con  $\hat{\beta}_0$  y  $\hat{\beta}_1$ . Demostremos esto con el set de datos previamente definido:

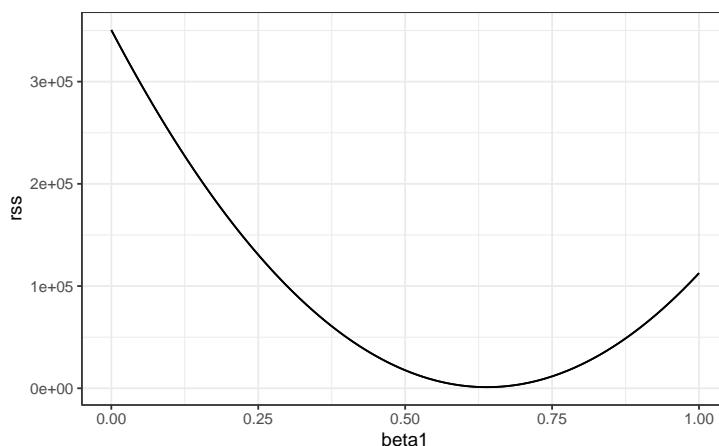
```
library(HistData)
data("GaltonFamilies")
set.seed(1983)
galton_heights <- GaltonFamilies %>%
 filter(gender == "male") %>%
 group_by(family) %>%
 sample_n(1) %>%
 ungroup() %>%
 select(father, childHeight) %>%
 rename(son = childHeight)
```

Escribamos una función que calcule el RSS para cualquier par de valores  $\beta_0$  y  $\beta_1$ .

```
rss <- function(beta0, beta1, data){
 resid <- galton_heights$son - (beta0+beta1*galton_heights$father)
 return(sum(resid^2))
}
```

Entonces, para cualquier par de valores, obtenemos un RSS. Aquí hay un gráfico del RSS como función de  $\beta_1$  cuando mantenemos el  $\beta_0$  fijo en 25.

```
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))
```



Podemos ver un mínimo claro para  $\beta_1$  alrededor de 0.65. Sin embargo, este mínimo para  $\beta_1$  es para cuando  $\beta_0 = 25$ , un valor que elegimos arbitrariamente. No sabemos si (25, 0.65) es el par que minimiza la ecuación en todos los pares posibles.

El método de prueba y error no funcionarán en este caso. Podríamos buscar un mínimo dentro de una cuadrícula fina de valores de  $\beta_0$  y  $\beta_1$ , pero esto requiere mucho tiempo innecesariamente ya que podemos usar cálculo: tomen las derivadas parciales, fíjenlas en 0 y resuelvan para  $\beta_1$  y  $\beta_2$ . Por supuesto, si tenemos muchos parámetros, estas ecuaciones pueden volverse bastante complejas. Pero hay funciones en R que hacen estos cálculos por nosotros. Aprenderemos esto a continuación. Para aprender las matemáticas detrás de esto, pueden consultar un libro sobre modelos lineales.

### 18.3.3 La función lm

En R, podemos obtener los estimadores de mínimos cuadrados usando la función `lm`. Para ajustar el modelo:

$$Y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$$

con  $Y_i$  la altura del hijo y  $x_i$  la altura del padre, podemos usar este código para obtener los estimadores de mínimos cuadrados.

```
fit <- lm(son ~ father, data = galton_heights)
fit$coef
#> (Intercept) father
#> 37.288 0.461
```

La forma más común que usamos `lm` es mediante el uso del carácter `~` para dejar `lm` saber cuál es la variable que estamos prediciendo (a la izquierda de `~`) y que estamos utilizando para predecir (a la derecha de `~`). El intercepto se agrega automáticamente al modelo que se ajustará.

El objeto `fit` incluye más información sobre el ajuste. Podemos usar la función `summary` para extraer más de esta información (que no mostramos):

```
summary(fit)
#>
#> Call:
#> lm(formula = son ~ father, data = galton_heights)
#>
#> Residuals:
#> Min 1Q Median 3Q Max
#> -9.354 -1.566 -0.008 1.726 9.415
#>
#> Coefficients:
#> Estimate Std. Error t value Pr(>|t|)
#> (Intercept) 37.2876 4.9862 7.48 3.4e-12 ***
#> father 0.4614 0.0721 6.40 1.4e-09 ***
#> ---
#> Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
#>
#> Residual standard error: 2.45 on 177 degrees of freedom
#> Multiple R-squared: 0.188, Adjusted R-squared: 0.183
#> F-statistic: 40.9 on 1 and 177 DF, p-value: 1.36e-09
```

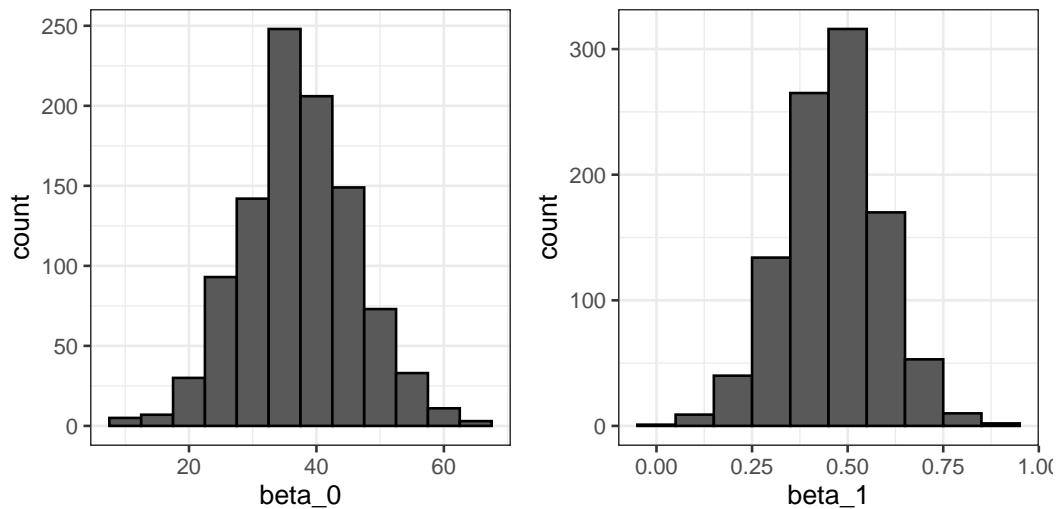
Para entender parte de la información incluida en este resumen, debemos recordar que el LSE consiste de variables aleatorias. La estadística matemática nos da algunas ideas sobre la distribución de estas variables aleatorias.

#### 18.3.4 El LSE consiste de variables aleatorias

El LSE se deriva de los datos  $y_1, \dots, y_N$ , que son una realización de variables aleatorias  $Y_1, \dots, Y_N$ . Esto implica que nuestros estimadores son variables aleatorias. Para ver esto, podemos ejecutar una simulación Monte Carlo en la que suponemos que los datos de altura del hijo y del padre definen una población, tomar una muestra aleatoria de tamaño  $N = 50$ , y calcular el coeficiente de pendiente de regresión para cada uno:

```
B <- 1000
N <- 50
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])
```

Podemos ver la variabilidad de los estimadores graficando sus distribuciones:



La razón por la que se ven normales es porque el teorema del límite central también aplica

aquí: para  $N$  suficientemente grande, los estimadores de mínimos cuadrados serán aproximadamente normales con el valor esperado  $\beta_0$  y  $\beta_1$ , respectivamente. Los errores estándar son un poco complicados para calcular, pero la teoría matemática nos permite calcularlos y están incluidos en el resumen proporcionado por la función `lm`. Aquí lo vemos para uno de nuestros sets de datos simulados:

```
sample_n(galton_heights, N, replace = TRUE) %>%
 lm(son ~ father, data = .) %>%
 summary %>% .$coef
#> Estimate Std. Error t value Pr(>|t|)
#> (Intercept) 19.28 11.656 1.65 1.05e-01
#> father 0.72 0.169 4.25 9.79e-05
```

Pueden ver que los estimadores de errores estándar informados por la función `summary` están cerca de los errores estándar de la simulación:

```
lse %>% summarize(se_0 = sd(beta_0), se_1 = sd(beta_1))
#> se_0 se_1
#> 1 8.84 0.128
```

La función `summary` también informa estadísticas t (`t value`) y valores-p (`Pr(>|t|)`). La estadística t no se basa realmente en el teorema del límite central, sino más bien en la suposición de que los  $\varepsilon$ s siguen una distribución normal. Bajo este supuesto, la teoría matemática nos dice que el LSE dividido por su error estándar,  $\hat{\beta}_0/\hat{S.E}(\hat{\beta}_0)$  y  $\hat{\beta}_1/\hat{S.E}(\hat{\beta}_1)$ , sigue una distribución t con  $N - p$  grados de libertad, con  $p$  el número de parámetros en nuestro modelo. En el caso de la altura  $p = 2$ , los dos valores-p prueban la hipótesis nula de que  $\beta_0 = 0$  y  $\beta_1 = 0$ , respectivamente.

Recuerden que, como describimos en la Sección 16.10, para  $N$  suficientemente grande, el CLT funciona y la distribución t se vuelve casi igual a la distribución normal. Además, tengan en cuenta que podemos construir intervalos de confianza, pero pronto aprenderemos sobre `broom`, un paquete adicional que lo hace fácil.

Aunque no ofrecemos ejemplos en este libro, las pruebas de hipótesis con modelos de regresión se usan comúnmente en epidemiología y economía para hacer afirmaciones como “el efecto de A en B fue estadísticamente significativo después de ajustar por X, Y y Z”. Sin embargo, varios supuestos tienen que ser válidos para que estas afirmaciones sean verdaderas.

### 18.3.5 Valores pronosticados son variables aleatorias

Una vez que ajustemos nuestro modelo, podemos obtener predicciones de  $Y$  usando los estimadores del modelo de regresión. Por ejemplo, si la altura del padre es  $x$ , entonces nuestra predicción  $\hat{Y}$  para la altura del hijo será:

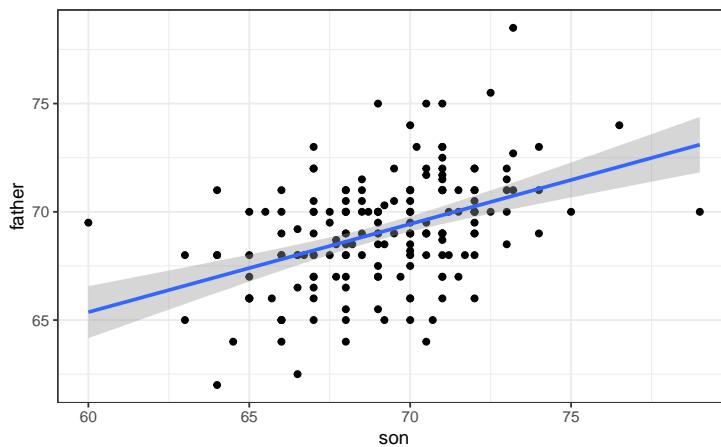
$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 x$$

Cuando graficamos  $\hat{Y}$  versus  $x$ , vemos la línea de regresión.

Tengan en cuenta que la predicción  $\hat{Y}$  también es una variable aleatoria y la teoría

matemática nos dice cuáles son los errores estándar. Si suponemos que los errores son normales o tienen un tamaño de muestra lo suficientemente grande, además podemos usar la teoría para construir intervalos de confianza. De hecho, la capa `geom_smooth(method = "lm")` de `ggplot2` que anteriormente usamos grafica  $\hat{Y}$  y lo rodea por intervalos de confianza:

```
galton_heights %>% ggplot(aes(son, father)) +
 geom_point() +
 geom_smooth(method = "lm")
```



La función `predict` de R toma un objeto `lm` como entrada y devuelve la predicción. Si se lo pedimos, también provee los errores estándar y la información necesaria para construir intervalos de confianza:

```
fit <- galton_heights %>% lm(son ~ father, data = .)

y_hat <- predict(fit, se.fit = TRUE)

names(y_hat)
#> [1] "fit" "se.fit" "df" "residual.scale"
```

## 18.4 Ejercicios

Hemos demostrado cómo BB y sencillos tienen un poder predictivo similar para anotar carreras. Otra forma de comparar la utilidad de estas métricas de béisbol es evaluando cuán estables son a lo largo de los años. Dado que tenemos que elegir jugadores a base de sus desempeños anteriores, preferiremos métricas que sean más estables. En estos ejercicios, compararemos la estabilidad de sencillos y BBs.

1. Antes de comenzar, queremos generar dos tablas. Una para 2002 y otra para el promedio de las temporadas 1999-2001. Queremos definir estadísticas por turnos al bate. Aquí vemos

como creamos la tabla para el 2017, quedándonos solo con jugadores con más de 100 turnos al bate.

```
library(Lahman)
dat <- Batting %>% filter(yearID == 2002) %>%
 mutate(pa = AB + BB,
 singles = (H - X2B - X3B - HR)/ pa, bb = BB/ pa) %>%
 filter(pa >= 100) %>%
 select(playerID, singles, bb)
```

Ahora calcule una tabla similar pero con tasas calculadas durante 1999-2001.

2. En la Sección 22.1, aprenderemos sobre `inner_join`, que se puede usar para poner los datos y promedios de 2001 en la misma tabla:

```
dat <- inner_join(dat, avg, by = "playerID")
```

Calcule la correlación entre 2002 y las temporadas anteriores para sencillos y BB.

3. Note que la correlación es mayor para BB. Para rápidamente tener una idea de la incertidumbre asociada con este estimador de correlación, ajustaremos un modelo lineal y calcularemos los intervalos de confianza para el coeficiente de pendiente. Sin embargo, primero haga diagramas de dispersión para confirmar que es apropiado ajustar un modelo lineal.

4. Ahora ajuste un modelo lineal para cada métrica y use la función `confint` para comparar los estimadores.

## 18.5 Regresión lineal en el tidyverse

Para ver cómo usamos la función `lm` en un análisis más complejo, volvamos al ejemplo del béisbol. En ese caso, estimamos líneas de regresión para predecir carreras por BB en diferentes estratos de HR. Primero construimos un *data frame* similar a este:

```
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)
```

Como en ese momento no sabíamos de la función `lm` para calcular la línea de regresión en cada estrato, utilizamos la fórmula así:

```
get_slope <- function(x, y) cor(x, y) * sd(y)/ sd(x)
dat %>%
 group_by(HR) %>%
 summarize(slope = get_slope(BB, R))
```

Argumentamos que las pendientes son similares y que las diferencias quizás se debieron a una variación aleatoria. Para ofrecer una defensa más rigurosa de que las pendientes eran las mismas, lo que condujo a nuestro modelo de múltiples variables, pudimos calcular los intervalos de confianza para cada pendiente. No hemos aprendido la fórmula para esto, pero la función `lm` provee suficiente información para construirlos.

Primero, noten que si intentamos usar la función `lm` para obtener la pendiente estimada de esta manera:

```
dat %>%
 group_by(HR) %>%
 lm(R ~ BB, data = .) %>% .$coef
#> (Intercept) BB
#> 2.198 0.638
```

no obtenemos el resultado que queremos. La función `lm` ignora el `group_by` ya que `lm` no es parte del `tidyverse` y no sabe cómo manejar el resultado de un tibble agrupado.

Las funciones de `tidyverse` saben cómo interpretar los tibbles agrupados. Además, para facilitar la creación de una secuencia de comandos con el *pipe* `%>%`, las funciones de `tidyverse` consistentemente devuelven *data frames*, ya que esto asegura que el resultado de una función sea aceptado como la entrada de otra. Pero la mayoría de las funciones de R no reconocen los tibbles agrupados ni devuelven *data frames*. La función `lm` es un ejemplo. Sin embargo, podemos escribir una función que usa `lm` para calcular y devolver los resúmenes relevantes en un *data frame* y entonces usar `summarize`:

```
get_slope <- function(x, y){
 fit <- lm(y ~ x)
 tibble(slope = fit$coefficients[2],
 se = summary(fit)$coefficient[2,2])
}
dat %>%
 group_by(HR) %>%
 summarize(get_slope(BB, R))
#> # A tibble: 9 x 3
#> 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
#> # ... with 4 more rows
```

Aquí un ejemplo que devuelve estimadores para ambos parametros:

```
get_lse <- function(x, y){
 fit <- lm(y ~ x)
 data.frame(term = names(fit$coefficients),
 slope = fit$coefficients,
 se = summary(fit)$coefficient[,2])
```

```

}

dat %>%
 group_by(HR) %>%
 summarize(get_lse(BB, R))
#> # 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 x 0.734 0.208
#> 3 0.5 (Intercept) 2.01 0.344
#> 4 0.5 x 0.566 0.110
#> 5 0.6 (Intercept) 2.53 0.305
#> # ... with 13 more rows

```

Si creen que todo esto es demasiado complicado, no son los únicos. Para simplificar las cosas, presentamos el paquete **broom** que fue diseñado para facilitar el uso de funciones que ajustan modelos, como `lm`, con el **tidyverse**.

### 18.5.1 El paquete broom

Nuestra tarea original era proveer un estimador y un intervalo de confianza para los estimadores de la pendiente de cada estrato. El paquete **broom** hará esto bastante fácil.

El paquete **broom** tiene tres funciones principales, todas de las cuales extraen información del objeto devuelto por `lm` y lo devuelve en un *data frame* que **tidyverse** entiende. Estas funciones son `tidy`, `glance` y `augment`. La función `tidy` devuelve estimadores e información relacionada como 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

```

Podemos agregar otros resúmenes importantes, como los intervalos de confianza:

```

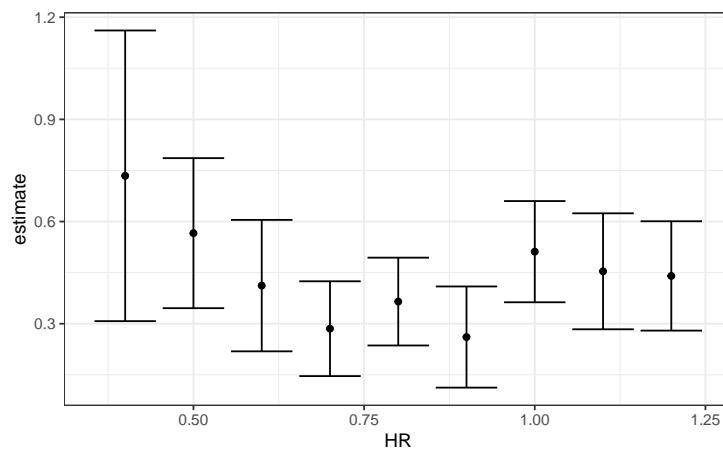
tidy(fit, conf.int = TRUE)
#> # A tibble: 2 x 7
#> term estimate std.error statistic p.value conf.low conf.high
#> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1 (Intercept) 2.20 0.113 19.4 1.12e-70 1.98 2.42
#> 2 BB 0.638 0.0344 18.5 1.35e-65 0.570 0.705

```

Debido a que el resultado es un *data frame*, podemos usarlo inmediatamente con `summarize`

para unir los comandos que producen la tabla que queremos. Como se devuelve un *data frame*, podemos filtrar y seleccionar las filas y columnas que queramos, que facilita trabajar con **ggplot2**:

```
dat %>%
 group_by(HR) %>%
 summarize(tidy(lm(R ~ BB), 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()
```



Ahora volvemos a discutir nuestra tarea original de determinar si las pendientes cambiaron. El gráfico que acabamos de hacer, usando **summarize** y **tidy**, muestra que los intervalos de confianza se superponen, que provee una buena confirmación visual de que nuestra suposición de que la pendiente no cambia es cierta.

Las otras funciones ofrecidas por **broom**, **glance** y **augment**, se relacionan con resultados específicos del modelo y de la observación, respectivamente. Aquí podemos ver los resúmenes que resultan de ajustar modelos que **glance** devuelve:

```
glance(fit)
#> # A tibble: 1 x 12
#> r.squared adj.r.squared sigma statistic p.value df logLik AIC
#> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1 0.266 0.265 0.454 343. 1.35e-65 1 -596. 1199.
#> # ... with 4 more variables: BIC <dbl>, deviance <dbl>,
#> # df.residual <int>, nobs <int>
```

Pueden obtener más información sobre estos resúmenes en cualquier libro de texto de regresión.

Veremos un ejemplo de **augment** en la siguiente sección.

## 18.6 Ejercicios

1. En una sección anterior, calculamos la correlación entre madres e hijas, madres e hijos, padres e hijas, y padres e hijos, y notamos que la correlación más alta es entre padres e hijos y la más baja es entre madres e hijos. Podemos calcular estas correlaciones usando:

```
library(HistData)
data("GaltonFamilies")
set.seed(1)
galton_heights <- GaltonFamilies %>%
 group_by(family, gender) %>%
 sample_n(1) %>%
 ungroup()

cors <- galton_heights %>%
 pivot_longer(father:mother, names_to = "parent", values_to = "parentHeight") %>%
 mutate(child = ifelse(gender == "female", "daughter", "son")) %>%
 unite(pair, c("parent", "child")) %>%
 group_by(pair) %>%
 summarize(cor = cor(parentHeight, childHeight))
```

¿Son estas diferencias estadísticamente significativas? Para responder, calcularemos las pendientes de la línea de regresión junto con sus errores estándar. Comience usando `lm` y el paquete `broom` para calcular el LSE de las pendientes y los errores estándar.

2. Repita el ejercicio anterior, pero calcule también un intervalo de confianza.
3. Grafique los intervalos de confianza y observe que se superponen, que implica que los datos son consistentes con que la herencia de altura y sexo son independientes.
4. Debido a que estamos seleccionando niños al azar, podemos hacer algo como una prueba de permutación aquí. Repita el cálculo de correlaciones 100 veces tomando una muestra diferente cada vez. Sugerencia: use un código similar al que usamos con las simulaciones.
5. Ajuste un modelo de regresión lineal para obtener los efectos de BB y HR en las carreras (a nivel de equipo) para el año 1971. Utilice la función `tidy` del paquete `broom` para obtener los resultados en un *data frame*.
6. Ahora repita lo anterior para cada año desde 1961 y haga un gráfico. Utilice `summarize` y el paquete `broom` para ajustar este modelo para cada año desde 1961.
7. Use los resultados del ejercicio anterior para graficar los efectos estimados de BB en las carreras.
8. **Avanzado:** Escriba una función que tome R, HR y BB como argumentos y ajuste dos modelos lineales:  $R \sim BB$  y  $R \sim BB + HR$ . Luego use la función `summarize` para obtener el BB para ambos modelos para cada año desde 1961. Finalmente, grafiquelos como función de tiempo y compárelos.

## 18.7 Estudio de caso: *Moneyball* (continuación)

Al tratar de responder de cuán bien las BB predicen las carreras, la exploración de datos nos llevó a un modelo:

$$E[R | BB = x_1, HR = x_2] = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

Aquí, los datos son aproximadamente normales y las distribuciones condicionales también fueron normales. Por lo tanto, tiene sentido usar un modelo lineal:

$$Y_i = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \varepsilon_i$$

con  $Y_i$  representando carreras por juego para el equipo  $i$ ,  $x_{i,1}$  representando BB por juego, y  $x_{i,2}$  representando HR por juego. Para usar `lm` aquí, necesitamos que la función sepa que tenemos dos variables predictivas. Entonces usamos el símbolo `+` de la siguiente manera:

```
fit <- Teams %>%
 filter(yearID %in% 1961:2001) %>%
 mutate(BB = BB/G, HR = HR/G, R = R/G) %>%
 lm(R ~ BB + HR, data = .)
```

Nosotros podemos usar `tidy` para ver un buen resumen:

```
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
```

Cuando ajustamos el modelo con una sola variable, las pendientes estimadas fueron 0.735328761775897 y 1.84482406982372 para BB y HR, respectivamente. Tengan en cuenta que cuando se ajusta el modelo de múltiples variables, ambos disminuyen, con el efecto BB disminuyendo mucho más.

Ahora queremos construir una métrica para elegir jugadores. Tenemos que considerar sencillos, dobles y triples. ¿Podemos construir un modelo que prediga carreras basado en todos estos resultados?

Ahora vamos a dar un “salto de fe” y suponer que estas cinco variables son conjuntamente normales. Esto significa que si elegimos cualquiera de ellas y mantenemos las otras cuatro fijas, la relación con el resultado es lineal y la pendiente no depende de los cuatro valores que se mantienen constantes. Si esto es cierto, entonces un modelo lineal para nuestros datos es:

$$Y_i = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \beta_3 x_{i,3} + \beta_4 x_{i,4} + \beta_5 x_{i,5} + \varepsilon_i$$

con  $x_{i,1}, x_{i,2}, x_{i,3}, x_{i,4}, x_{i,5}$  representando BB, sencillos, dobles, triples y HR respectivamente.

Utilizando `lm`, podemos encontrar rápidamente el LSE para los parámetros usando:

```
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 = .)
```

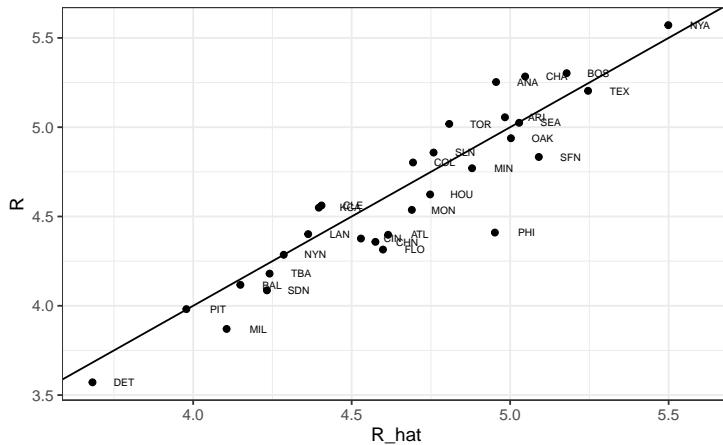
Podemos ver los coeficientes usando `tidy`:

```
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
#> # ... with 1 more row
```

Para ver cuán bien nuestra métrica predice carreras, podemos predecir el número de carreras para cada equipo en 2002 usando la función `predict` y entonces hacer un gráfico:

```
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)) +
 geom_point() +
 geom_text(nudge_x=0.1, cex = 2) +
 geom_abline()
```



Nuestro modelo hace un buen trabajo, como lo demuestra el hecho de que los puntos del gráfico observado versus los del gráfico previsto caen cerca de la línea de identidad.

Entonces, en lugar de usar el promedio de bateo o solo el número de HR como una medida de selección de jugadores, podemos usar nuestro modelo ajustado para formar una métrica que se relacione más directamente con la producción de carreras. Específicamente, para definir una métrica para un jugador A, imaginamos un equipo compuesto por jugadores como el jugador A y usamos nuestro modelo de regresión ajustado para predecir cuántas carreras produciría este equipo. La fórmula se vería así:  $-2.769 + 0.371 \times BB + 0.519 \times \text{singles} + 0.771 \times \text{dobles} + 1.24 \times \text{triples} + 1.443 \times HR$ .

Para definir una métrica específica al jugador, tenemos un poco más de trabajo por hacer. Un reto aquí es que derivamos la métrica para equipos, basada en estadísticas de resumen a nivel de equipo. Por ejemplo, el valor de HR que se ingresa en la ecuación es HR por juego para todo el equipo. Si en cambio calculamos el HR por juego para un jugador, el valor será mucho más bajo dado que ya no son 9 bateadores contribuyendo al total sino un solo jugador. Además, si un jugador solo juega parte del juego y obtiene menos oportunidades que el promedio, todavía se considera un juego jugado. Para los jugadores, una tasa que toma en cuenta oportunidades es la tasa por turnos al bate.

Para hacer que la tasa de equipo por juego sea comparable a la tasa de jugador por turno al bate, calculamos el número promedio de turnos al bate por juego:

```
pa_per_game <- Batting %>% filter(yearID == 2002) %>%
 group_by(teamID) %>%
 summarize(pa_per_game = sum(AB+BB) / max(G)) %>%
 pull(pa_per_game) %>%
 mean
```

Calculamos las tasas de turnos al bate para jugadores disponibles en 2002 con datos de 1997-2001. Para evitar pequeños artefactos de muestra, filtramos jugadores con menos de 200 turnos al bate por año. Aquí está el cálculo completo en una línea:

```
players <- Batting %>% filter(yearID %in% 1997:2001) %>%
 group_by(playerID) %>%
 mutate(PA = BB + AB) %>%
```

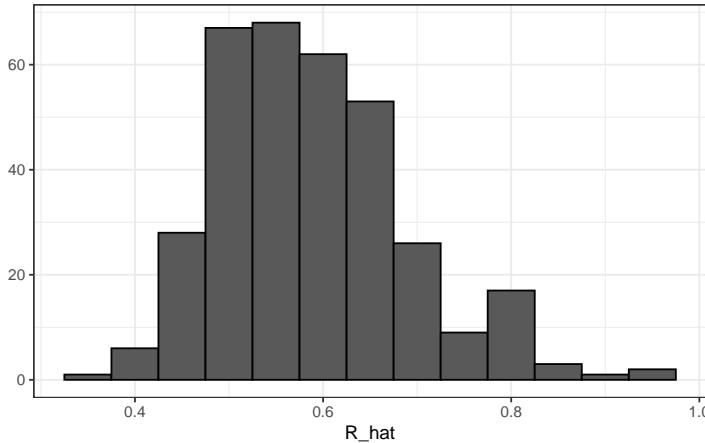
```

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 >= 1000) %>%
select(-G) %>%
mutate(R_hat = predict(fit, newdata = .) / 9)

```

Como el modelo fue ajustado a data de equipos, dividimos la predicción entre 9 (el número de jugadores que batean) para que represente las carreras que predecimos cada jugador producirá por juego. La distribución demuestra que existe una gran variabilidad entre los jugadores:

```
qplot(R_hat, data = players, binwidth = 0.05, color = I("black"))
```



### 18.7.1 Añadiendo información sobre salario y posición

Para realmente construir el equipo, necesitaremos conocer sus salarios y su posición defensiva. Para hacer esto, unimos el *data frame* `players` que acabamos de crear con el *data frame* de información del jugador incluido en algunas de las otras tablas de datos de **Lahman**. Aprenderemos más sobre la función `join` en la Sección 22.1.

Comiencen añadiendo los salarios del 2002 para cada jugador:

```

players <- Salaries %>%
 filter(yearID == 2002) %>%
 select(playerID, salary) %>%
 right_join(players, by="playerID")

```

A continuación, agregamos su posición defensiva. Esta es una tarea algo complicada porque

los jugadores juegan más de una posición cada año. La tabla `Appearances` del paquete **Lahman** indica cuántos juegos jugó cada jugador en cada posición y podemos elegir la posición que más se jugó usando `which.max` en cada fila. Usamos `apply` para hacer esto. Sin embargo, debido a que algunos jugadores son intercambiados, aparecen más de una vez en la tabla, por lo que primero sumamos sus turnos al bate en los equipos. Aquí, escogemos la posición en la que más jugó el jugador usando la función `top_n`. Para asegurarnos de que solo elegimos una posición, en el caso de empates, elegimos la primera fila del *data frame* resultante. También eliminamos la posición `OF` que significa *outfielder*, una generalización de tres posiciones: jardín izquierdo (LF o *left field* en inglés), jardín central (CF o *center field* en inglés) y campo derecho (RF o *right field* en inglés). Además, eliminamos los lanzadores, ya que no batean en la liga en la que juegan los Atléticos.

```
position_names <-
 paste0("G_", c("P", "C", "1b", "2b", "3b", "ss", "lf", "cf", "rf", "dh"))

tmp <- Appearances %>%
 filter(yearID == 2002) %>%
 group_by(playerID) %>%
 summarize_at(position_names, sum) %>%
 ungroup()

pos <- tmp %>%
 select(position_names) %>%
 apply(., 1, which.max)
#> Note: Using an external vector in selections is ambiguous.
#> i Use `all_of(position_names)` instead of `position_names` to silence this message.
#> i See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
#> This message is displayed once per session.

players <- tibble(playerID = tmp$playerID, POS = position_names[pos]) %>%
 mutate(POS = str_to_upper(str_remove(POS, "G_"))) %>%
 filter(POS != "P") %>%
 right_join(players, by="playerID") %>%
 filter(!is.na(POS) & !is.na(salary))
```

Finalmente, agregamos su nombre y apellido:

```
players <- Master %>%
 select(playerID, nameFirst, nameLast, debut) %>%
 mutate(debut = as.Date(debut)) %>%
 right_join(players, by="playerID")
```

Si son fanáticos del béisbol, reconocerán a los 10 mejores jugadores:

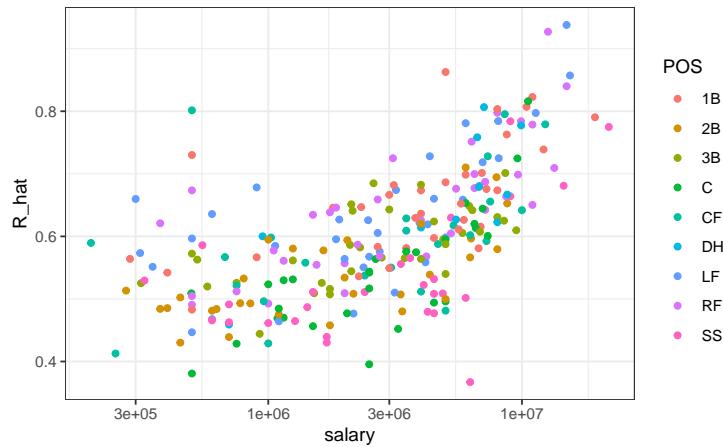
```
players %>% select(nameFirst, nameLast, POS, salary, R_hat) %>%
 arrange(desc(R_hat)) %>% top_n(10)
#> Selecting by R_hat
#> nameFirst nameLast POS salary R_hat
#> 1 Barry Bonds LF 15000000 0.938
#> 2 Larry Walker RF 12666667 0.927
```

```
#> 3 Todd Helton 1B 5000000 0.863
#> 4 Manny Ramirez LF 15462727 0.857
#> 5 Sammy Sosa RF 15000000 0.840
#> 6 Jeff Bagwell 1B 11000000 0.823
#> 7 Mike Piazza C 10571429 0.816
#> 8 Jason Giambi 1B 10428571 0.807
#> 9 Edgar Martinez DH 7086668 0.807
#> 10 Jim Thome 1B 8000000 0.804
```

### 18.7.2 Escoger nueve jugadores

En promedio, los jugadores con una métrica más alta tienen salarios más altos:

```
players %>% ggplot(aes(salary, R_hat, color = POS)) +
 geom_point() +
 scale_x_log10()
```



Podemos buscar buenas ofertas mirando a los jugadores que producen muchas más carreras que otros con salarios similares. Podemos usar esta tabla para decidir qué jugadores escoger y mantener nuestro salario total por debajo de los 40 millones de dólares con los que tuvo que trabajar Billy Beane. Esto se puede hacer usando lo que los científicos de la computación llaman programación lineal. Esto no es algo que enseñamos, pero en la siguiente tabla mostramos los jugadores seleccionados con este acercamiento, la posición que juegan, su salario y las carreras que predecimos producirían. Vemos que, en promedio, nuestro equipo produciría 6.6 carreras por juego! En la tabla también mostramos estadísticas estandarizadas usando todos los jugadores, de modo que, por ejemplo, los bateadores de HR por encima del promedio tienen valores superiores a 0. Vemos que todos estos jugadores tienen BB por encima del promedio y la mayoría tienen tasas de HR por encima del promedio, mientras que lo mismo no es cierto para sencillos:

| nameFirst | nameLast  | POS | salary     | R_hat | BB  | singles | doubles | triples | HR  | AVG  |
|-----------|-----------|-----|------------|-------|-----|---------|---------|---------|-----|------|
| Todd      | Helton    | 1B  | 5,000,000  | 0.9   | 0.9 | -0.2    | 2.6     | -0.3    | 1.5 | 2.7  |
| Mike      | Piazza    | C   | 10,571,429 | 0.8   | 0.3 | 0.4     | 0.2     | -1.4    | 1.8 | 2.2  |
| Edgar     | Martinez  | DH  | 7,086,668  | 0.8   | 2.1 | 0.0     | 1.3     | -1.2    | 0.8 | 2.2  |
| Jim       | Edmonds   | CF  | 7,333,333  | 0.7   | 1.1 | -0.6    | 0.8     | -1.2    | 1.0 | 0.9  |
| Jeff      | Kent      | 2B  | 6,000,000  | 0.7   | 0.2 | -0.7    | 2.0     | 0.4     | 0.8 | 0.8  |
| Phil      | Nevin     | 3B  | 2,600,000  | 0.7   | 0.3 | -0.9    | 0.5     | -1.2    | 1.2 | 0.1  |
| Matt      | Stairs    | RF  | 500,000    | 0.7   | 1.1 | -1.5    | 0.0     | -1.1    | 1.1 | -0.6 |
| Henry     | Rodriguez | LF  | 300,000    | 0.7   | 0.2 | -1.6    | 0.3     | -0.8    | 1.3 | -0.7 |
| John      | Valentin  | SS  | 550,000    | 0.6   | 0.2 | -0.9    | 1.8     | -0.4    | 0.0 | -0.5 |

## 18.8 La falacia de la regresión

Wikipedia define la *maldición de segundo año* (*sophomore slump* en inglés) como:

Una caída de segundo año o maldición de segundo año se refiere a una instancia en la que un segundo esfuerzo, o segundo año, no cumple con los estándares del primer esfuerzo. Se usa comúnmente para referirse a la apatía de los estudiantes (segundo año de secundaria, colegio o universidad), el rendimiento de los atletas (segunda temporada de juego), cantantes/bandas (segundo álbum), programas de televisión (segunda temporada) y películas (secuelas/precuelas).

En las Grandes Ligas de Béisbol, el premio al novato del año (*Rookie of the Year* o ROY por sus siglas en inglés) se otorga al jugador de primer año que se considera que ha tenido el mejor desempeño. La frase *maldición de segundo año* se usa para describir la observación de que a los ganadores del premio ROY no les va tan bien durante su segundo año. Por ejemplo, este artículo de Fox Sports<sup>11</sup> que pregunta “¿La impresionante clase de novatos del MLB de 2015 sufrirá una maldición de segundo año?”

¿Los datos confirman la existencia de una maldición de segundo año? Vamos a ver. Al examinar los datos para el promedio de bateo, vemos que esta observación es válida para los ROY de mayor rendimiento:

| nameFirst | nameLast | rookie_year | rookie | sophomore |
|-----------|----------|-------------|--------|-----------|
| Willie    | McCovey  | 1959        | 0.354  | 0.238     |
| Ichiro    | Suzuki   | 2001        | 0.350  | 0.321     |
| Al        | Bumbry   | 1973        | 0.337  | 0.233     |
| Fred      | Lynn     | 1975        | 0.331  | 0.314     |
| Albert    | Pujols   | 2001        | 0.329  | 0.314     |

De hecho, la proporción de jugadores que tienen un promedio de bateo más bajo en su segundo año es 0.686.

Entonces, ¿es “nerviosismo” o “maldición”? Para responder a esta pregunta, volvamos nuestra atención a todos los jugadores que jugaron las temporadas 2013 y 2014 y batearon más de 130 veces (mínimo para ganar el ROY).

<sup>11</sup><https://www.foxsports.com/stories/mlb/will-mlbs-tremendous-rookie-class-of-2015-suffer-a-sophomore-slump>

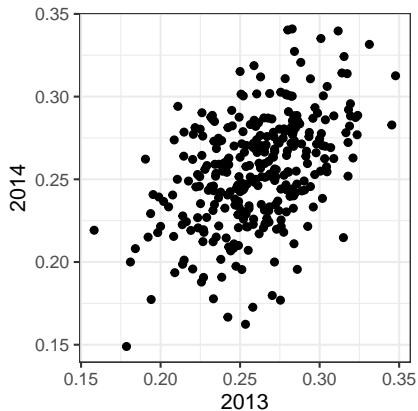
El mismo patrón surge cuando miramos a los jugadores con el mejor desempeño: los promedios de bateo disminuyen para la mayoría de los mejores jugadores.

| nameFirst | nameLast | 2013  | 2014  |
|-----------|----------|-------|-------|
| Miguel    | Cabrera  | 0.348 | 0.313 |
| Hanley    | Ramirez  | 0.345 | 0.283 |
| Michael   | Cuddyer  | 0.331 | 0.332 |
| Scooter   | Gennett  | 0.324 | 0.289 |
| Joe       | Mauer    | 0.324 | 0.277 |

¡Pero estos no son novatos! Además, miren lo que les sucede a los peores jugadores del 2013:

| nameFirst | nameLast | 2013  | 2014  |
|-----------|----------|-------|-------|
| Danny     | Espinosa | 0.158 | 0.219 |
| Dan       | Uggla    | 0.179 | 0.149 |
| Jeff      | Mathis   | 0.181 | 0.200 |
| B. J.     | Upton    | 0.184 | 0.208 |
| Adam      | Rosales  | 0.190 | 0.262 |

¡Sus promedios de bateo en su mayoría suben! ¿Es esto una especie de “bendición” de segundo año? No lo es. No hay tal cosa como una maldición de segundo año. Todo esto se explica con un simple hecho estadístico: la correlación para el rendimiento en dos años separados es alta, pero no perfecta:



La correlación es 0.46 y los datos se parecen mucho a una distribución normal de dos variables, que significa que predecimos un promedio de bateo  $Y$  del 2014 para cualquier jugador que tuviera un promedio de bateo  $X$  en el 2013 con:

$$\frac{Y - .255}{.032} = 0.46 \left( \frac{X - .261}{.023} \right)$$

Como la correlación no es perfecta, la regresión nos dice que, en promedio, esperamos que los jugadores de alto desempeño del 2013 tengan un peor desempeño en 2014. No es una maldición; es solo por casualidad. El ROY se selecciona de los valores superiores de  $X$ , por lo cual se espera que  $Y$  muestre regresión a la media.

## 18.9 Modelos de error de medición

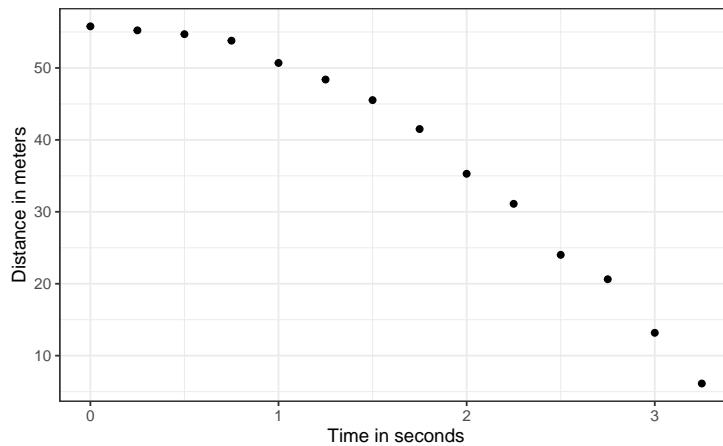
Hasta ahora, todos nuestros ejemplos de regresión lineal se han aplicado a dos o más variables aleatorias. Suponemos que los pares siguen una distribución normal de dos variables y lo usamos para motivar un modelo lineal. Este enfoque cubre la mayoría de los ejemplos reales de regresión lineal. La otra aplicación importante proviene de los modelos de errores de medición. En estas aplicaciones, es común tener una covariante no aleatoria, como el tiempo, y la aleatoriedad se introduce por error de medición en lugar de muestreo o variabilidad natural.

Para entender estos modelos, imaginen que son Galileo en el siglo XVI tratando de describir la velocidad de un objeto que cae. Un asistente sube a la torre de Pisa y deja caer una pelota, mientras que otros asistentes registran la posición en diferentes momentos. Simulemos algunos datos usando las ecuaciones que conocemos hoy y agregando algunos errores de medición. La función `rfalling_object` de `dslabs` genera estas simulaciones:

```
library(dslabs)
falling_object <- rfalling_object()
```

Los asistentes le entregan los datos a Galileo y esto es lo que él ve:

```
falling_object %>%
 ggplot(aes(time, observed_distance)) +
 geom_point() +
 ylab("Distance in meters") +
 xlab("Time in seconds")
```



Galileo no conoce la ecuación exacta, pero al observar el gráfico anterior, deduce que la posición debe seguir una parábola, que podemos escribir así:

$$f(x) = \beta_0 + \beta_1 x + \beta_2 x^2$$

Los datos no caen exactamente en una parábola. Galileo sabe que esto se debe a un error de

medición. Sus ayudantes cometen errores al medir la distancia. Para tomar esto en cuenta, modela los datos con:

$$Y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \varepsilon_i, i = 1, \dots, n$$

con  $Y_i$  representando la distancia en metros,  $x_i$  representando el tiempo en segundos y  $\varepsilon$  tomando en cuenta el error de medición. Se supone que el error de medición sea aleatorio, independiente el uno del otro y que con la misma distribución para cada  $i$ . También suponemos que no hay sesgo, que significa que el valor esperado  $E[\varepsilon] = 0$ .

Noten que este es un modelo lineal porque es una combinación lineal de cantidades conocidas ( $x$  y  $x^2$  son conocidas) y parámetros desconocidos (los  $\beta$ s son parámetros desconocidos para Galileo). A diferencia de nuestros ejemplos anteriores, aquí  $x$  es una cantidad fija; no estamos condicionando.

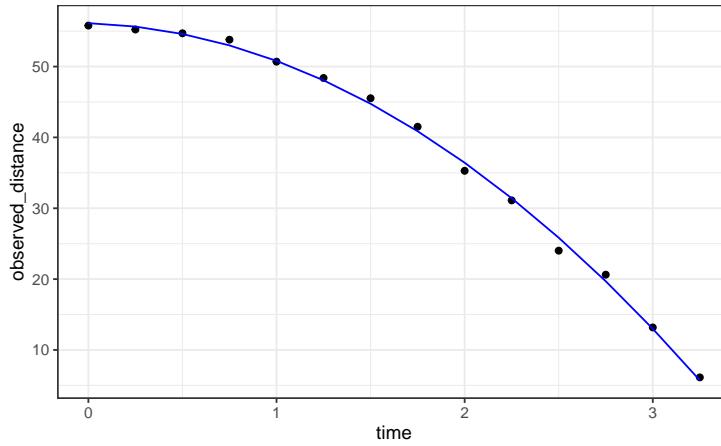
Para plantear una nueva teoría física y comenzar a hacer predicciones sobre la caída de otros objetos, Galileo necesita números reales, en lugar de parámetros desconocidos. Usar el LSE parece un enfoque razonable. ¿Cómo encontramos el LSE?

Los cálculos del LSE no requieren que los errores sean aproximadamente normales. La función `lm` encontrará los  $\beta$ s que minimizarán la suma residual de cuadrados:

```
fit <- falling_object %>%
 mutate(time_sq = time^2) %>%
 lm(observed_distance ~ time + time_sq, data = .)
tidy(fit)
#> # A tibble: 3 x 5
#> term estimate std.error statistic p.value
#> <chr> <dbl> <dbl> <dbl> <dbl>
#> 1 (Intercept) 56.1 0.592 94.9 2.23e-17
#> 2 time -0.786 0.845 -0.930 3.72e- 1
#> 3 time_sq -4.53 0.251 -18.1 1.58e- 9
```

Verifiquemos si la parábola estimada se ajusta a los datos. La función `augment` de **broom** nos permite hacer esto fácilmente:

```
augment(fit) %>%
 ggplot() +
 geom_point(aes(time, observed_distance)) +
 geom_line(aes(time, .fitted), col = "blue")
```



Gracias a nuestros maestros de física de escuela secundaria, sabemos que la ecuación para la trayectoria de un objeto que cae es:

$$d = h_0 + v_0 t - 0.5 \times 9.8 t^2$$

con  $h_0$  y  $v_0$  la altura inicial y la velocidad, respectivamente. Los datos que simulamos anteriormente siguieron esta ecuación y agregaron un error de medición a fin de simular n observaciones para dejar caer una pelota ( $v_0 = 0$ ) desde la torre de Pisa ( $h_0 = 55.86$ ).

Estos son consistentes con los estimadores de los parámetros:

```
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) 56.1 0.592 94.9 2.23e-17 54.8 57.4
#> 2 time -0.786 0.845 -0.930 3.72e- 1 -2.65 1.07
#> 3 time_sq -4.53 0.251 -18.1 1.58e- 9 -5.08 -3.98
```

La altura de la torre de Pisa está dentro del intervalo de confianza para  $\beta_0$ , la velocidad inicial 0 está en el intervalo de confianza para  $\beta_1$  (recuerden que el valor-p es mayor que 0.05) y la constante de aceleración está en un intervalo de confianza para  $-2 \times \beta_2$ .

## 18.10 Ejercicios

Desde la década de 1980, los *sabermetricians* han utilizado una estadística de resumen diferente del promedio de bateo para evaluar a los jugadores. Se dieron cuenta de que las BB eran importantes y que a los dobles, triples y HR se les debe dar más peso que los sencillos. Como resultado, propusieron la siguiente métrica:

$$\frac{\text{BB}}{\text{PA}} + \frac{\text{Singles} + 2\text{Doubles} + 3\text{Triples} + 4\text{HR}}{\text{AB}}$$

Denominaron a la métrica: *on-base-percentage plus slugging percentage* o OPS. Aunque los *sabermetricians* probablemente no usaron la regresión, aquí mostramos cómo OPS está cerca de lo que se obtiene con la regresión.

1. Calcule el OPS para cada equipo en la temporada 2001. Luego grafique carreras por juego versus OPS.
2. Para cada año desde 1961, calcule la correlación entre carreras por juego y OPS. Entonces grafique estas correlaciones como función del año.
3. Tenga en cuenta que podemos reescribir OPS como un promedio ponderado de BB, sencillos, dobles, triples y HR. Sabemos que los coeficientes para dobles, triples y HR son 2, 3 y 4 veces mayores que para los sencillos. ¿Pero y los BB? ¿Cuál es el peso para BB en relación con sencillos? Sugerencia: el peso de BB en relación con sencillos será una función de AB y PA.
4. Tenga en cuenta que el peso para BB,  $\frac{AB}{PA}$ , cambiará de un equipo a otro. Para ver cuán variable es, calcule y grafique esta cantidad para cada equipo para cada año desde 1961. Luego vuelva a graficarla, pero en lugar de calcularla para cada equipo, calcule y grafique la razón para todo el año. Entonces, una vez que esté claro de que no hay mucha tendencia de tiempo o equipo, indique el promedio general.
5. Ahora sabemos que la fórmula para OPS es proporcional a  $0.91 \times BB + singles + 2 \times doubles + 3 \times triples + 4 \times HR$ . Veamos cómo se comparan estos coeficientes con esos obtenidos con la regresión. Ajuste un modelo de regresión a los datos después de 1961, como se hizo anteriormente: usando estadísticas por juego para cada año para cada equipo. Después de ajustar este modelo, indique los coeficientes como pesos relativos al coeficiente para sencillos.
6. Vemos que nuestros coeficientes del modelo de regresión lineal siguen la misma tendencia general que esos utilizados por OPS, pero con un peso ligeramente menor para las métricas que no son sencillos. Para cada equipo en los años posteriores a 1961, calcule el OPS, las carreras predichas con el modelo de regresión y calcule la correlación entre los dos, así como la correlación con carreras por juego.
7. Vemos que el uso del enfoque de regresión predice carreras un poco mejor que OPS, pero no tanto. Sin embargo, tenga en cuenta que hemos estado calculando OPS y prediciendo carreras para los equipos cuando estas medidas se utilizan para evaluar a los jugadores. Demostremos que OPS es bastante similar a lo que se obtiene con la regresión a nivel de jugador. Para la temporada de 1961 y las posteriores, calcule el OPS y las carreras previstas de nuestro modelo para cada jugador y grafíquelas. Use la corrección de PA por juego que usamos en el capítulo anterior.
8. ¿Qué jugadores han mostrado la mayor diferencia entre su rango por carreras predichas y OPS?

# 19

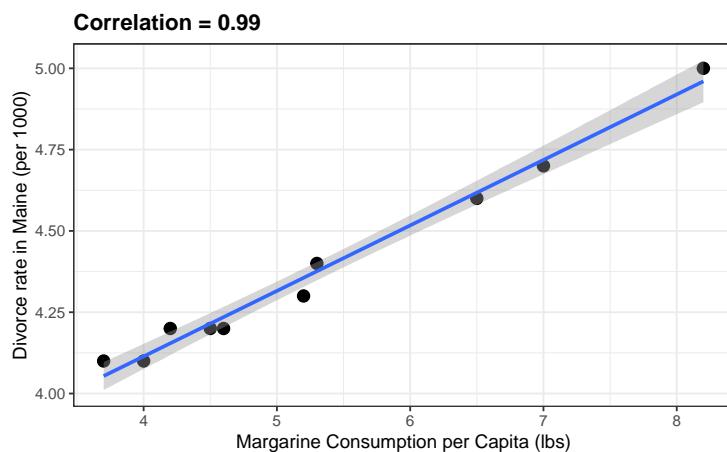
## *La correlación no implica causalidad*

La correlación no implica causalidad es quizás la lección más importante que uno aprende en una clase de estadística. A lo largo de la parte de estadísticas del libro, hemos descrito herramientas útiles para cuantificar asociaciones entre variables. Sin embargo, debemos tener cuidado de no malinterpretar estas asociaciones.

Hay muchas razones por las que una variable  $X$  se puede correlacionar con una variable  $Y$  sin tener ningún efecto directo sobre  $Y$ . A continuación examinaremos cuatro formas comunes que pueden conducir a una malinterpretación de los datos.

### 19.1 Correlación espuria

El siguiente ejemplo cómico subraya que la correlación no es causalidad. Muestra una correlación muy fuerte entre las tasas de divorcio y el consumo de margarina.



¿Significa esto que la margarina causa divorcios? ¿O los divorcios hacen que las personas coman más margarina? Por supuesto, la respuesta a ambas preguntas es “no”. Esto es solo un ejemplo de lo que llamamos una correlación espuria.

Pueden ver muchos más ejemplos absurdos en el sitio web *Spurious Correlations*<sup>1</sup>.

Los casos que se presentan en el sitio de web de correlaciones espurias son todas instancias de

<sup>1</sup><http://tylervigen.com/spurious-correlations>

lo que generalmente se llama *dragado de datos* o *pesca de datos* (*data dredging*, *data fishing*, o *data snooping* en inglés). Básicamente es cuando se escogen los datos selectivamente para confirmar cierta hipótesis. Un ejemplo de dragado de datos sería si observamos muchos resultados producidos por un proceso aleatorio y elegimos solo los que muestran una relación que respalda una teoría que queremos defender.

Se puede usar una simulación Monte Carlo para mostrar cómo el dragado de datos puede resultar en altas correlaciones entre variables no correlacionadas. Guardaremos los resultados de nuestra simulación en un tibble:

```
N <- 25
g <- 1000000
sim_data <- tibble(group = rep(1:g, each=N),
 x = rnorm(N * g),
 y = rnorm(N * g))
```

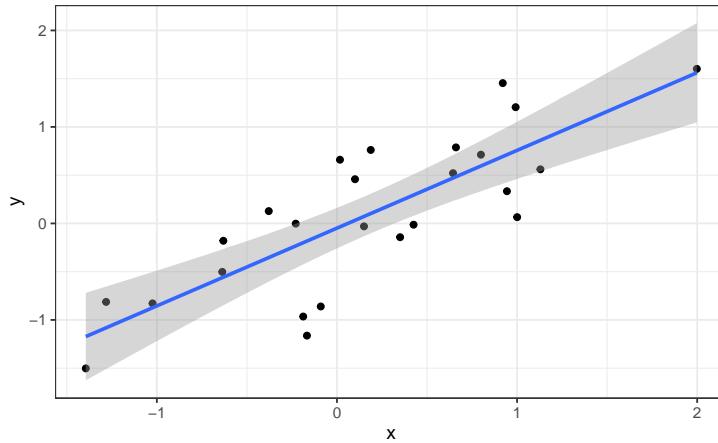
La primera columna denota grupo. Creamos grupos y para cada uno generamos un par de vectores independientes,  $X$  e  $Y$ , cada una con 25 observaciones, almacenadas en la segunda y tercera columnas. Debido a que construimos la simulación, sabemos que  $X$  e  $Y$  no están correlacionadas.

A continuación, calculamos la correlación entre  $X$  e  $Y$  para cada grupo y miramos el máximo:

```
res <- sim_data %>%
 group_by(group) %>%
 summarize(r = cor(x, y)) %>%
 arrange(desc(r))
res
#> # A tibble: 1,000,000 x 2
#> group r
#> <int> <dbl>
#> 1 566605 0.803
#> 2 387075 0.784
#> 3 230548 0.782
#> 4 944766 0.782
#> 5 47755 0.778
#> # ... with 999,995 more rows
```

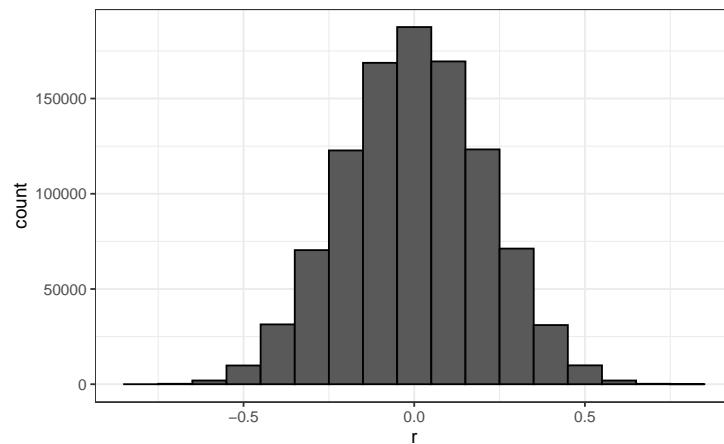
Vemos una correlación máxima de 0.803 y, si solo graficamos los datos del grupo con esta correlación, vemos un gráfico convincente que  $X$  e  $Y$  sí están correlacionados:

```
sim_data %>% filter(group == res$group[which.max(res$r)]) %>%
 ggplot(aes(x, y)) +
 geom_point() +
 geom_smooth(method = "lm")
```



Recuerden que el resumen de correlación es una variable aleatoria. Aquí tenemos la distribución generada por la simulación Monte Carlo:

```
res %>% ggplot(aes(x=r)) + geom_histogram(binwidth = 0.1, color = "black")
```



Es un hecho matemático que si observamos correlaciones aleatorias que se esperan que sean 0, pero tienen un error estándar de 0.204, la más grande estará cerca de 1.

Si realizamos una regresión en este grupo e interpretamos el valor-p, afirmaríamos incorrectamente que esta es una relación estadísticamente significativa:

```
library(broom)
sim_data %>%
 filter(group == res$group[which.max(res$r)]) %>%
 summarize(tidy(lm(y ~ x))) %>%
 filter(term == "x")
#> # A tibble: 1 x 5
#> term estimate std.error statistic p.value
#> <chr> <dbl> <dbl> <dbl> <dbl>
#> 1 x 0.806 0.125 6.46 0.00000136
```

Esta forma particular de dragado de datos se conoce como *p-hacking*. El *p-hacking* es un tema de mucha discusión porque es un problema en publicaciones científicas. Debido a que los editores tienden a premiar resultados estadísticamente significativos sobre resultados negativos, existe un incentivo para informar resultados significativos. En la epidemiología y las ciencias sociales, por ejemplo, los investigadores pueden buscar asociaciones entre un resultado adverso y varias exposiciones a distintos tipos de riesgo e informar solo la exposición que resultó en un valor-p pequeño. Además, podrían intentar ajustar varios modelos diferentes para tomar en cuenta la confusión y elegir el que da el valor-p más pequeño. En disciplinas experimentales, un experimento puede repetirse más de una vez, pero solo informar los resultados del experimento con un valor-p pequeño. Esto no sucede necesariamente debido a comportamientos antiéticos, sino más bien como resultado de la ignorancia estadística o de meras ilusiones. En los cursos de estadística avanzada, pueden aprender métodos para tomar en cuenta estas múltiples comparaciones.

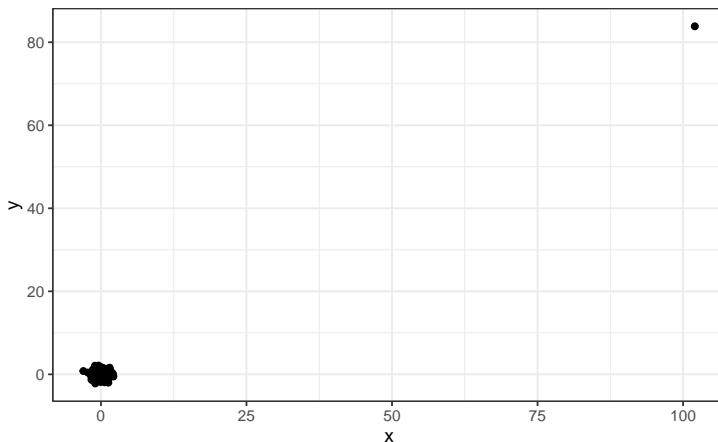
## 19.2 Valores atípicos

Supongan que tomamos medidas de dos resultados independientes,  $X$  e  $Y$ , y estandarizamos las medidas. Sin embargo, cometemos un error y olvidamos estandarizar la entrada 23. Podemos simular dichos datos usando:

```
set.seed(1985)
x <- rnorm(100,100,1)
y <- rnorm(100,84,1)
x[-23] <- scale(x[-23])
y[-23] <- scale(y[-23])
```

Los datos se ven así:

```
qplot(x, y)
```



No es sorprendente que la correlación sea bien alta:

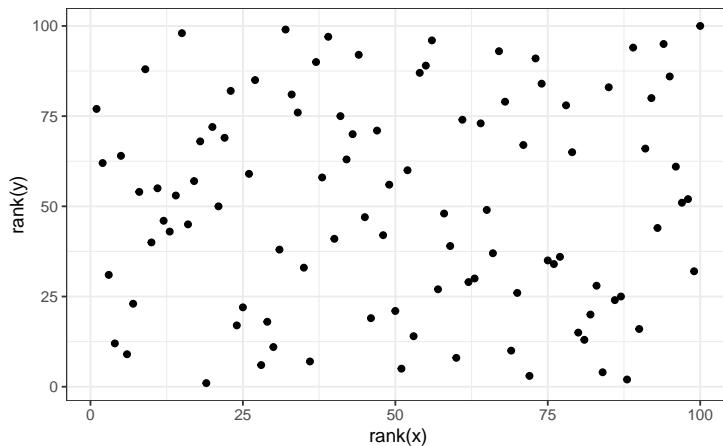
```
cor(x,y)
#> [1] 0.988
```

Pero lo impulsa un valor atípico. Si eliminamos este valor atípico, la correlación se reduce considerablemente a casi 0, que es lo que debería ser:

```
cor(x[-23], y[-23])
#> [1] -0.0442
```

En la Sección 11, describimos alternativas al promedio y la desviación estándar que son robustas a valores atípicos. También hay una alternativa a la correlación muestral para estimar la correlación de población que es robusta a valores atípicos. Se llama *la correlación de Spearman*. La idea es sencilla: calcular la correlación basada en los rangos de los valores. Aquí tenemos un gráfico de los rangos graficados uno contra el otro:

```
qplot(rank(x), rank(y))
```



El valor atípico ya no está asociado con un valor muy grande y la correlación se reduce:

```
cor(rank(x), rank(y))
#> [1] 0.00251
```

La correlación de Spearman también se puede calcular así:

```
cor(x, y, method = "spearman")
#> [1] 0.00251
```

Además, hay métodos robustos para ajustar modelos lineales que pueden aprender, por ejemplo, en el libro *Robust Statistics: Edition 2* de Peter J. Huber y Elvezio M. Ronchetti.

### 19.3 Inversión de causa y efecto

Otra forma en que la asociación se confunde con la causalidad es cuando la causa y el efecto se invierten. Un ejemplo de esto es afirmar que la tutoría afecta negativamente a los estudiantes porque éstos evalúan peor que sus compañeros que no reciben tutoría. En este caso, la tutoría no está causando las bajas puntuaciones en las pruebas, sino al revés.

Una versión de este reclamo se convirtió en un artículo de opinión en el New York Times titulado *Parental Involvement Is Overrated*<sup>2</sup>. Consideren esta cita del artículo:

Cuando examinamos si la ayuda frecuente con la tarea tuvo un impacto positivo en el desempeño académico de los niños, nos sorprendió lo que encontramos. Independientemente de la clase social de la familia, del origen racial o étnico, o del grado de un niño, ayuda consistente con la tarea casi nunca mejoró la puntuación de las pruebas o las notas ... Incluso más sorprendente para nosotros fue que cuando los padres ayudaban frecuentemente con la tarea, los niños generalmente salían peor.

Una posibilidad muy probable es que los niños que necesitan ayuda frecuente de sus padres reciban esta ayuda porque no se desempeñan bien en la escuela.

Fácilmente podemos construir un ejemplo de inversión de causa y efecto utilizando los datos de altura de padre e hijo. Si nos ajustamos al modelo:

$$X_i = \beta_0 + \beta_1 y_i + \varepsilon_i, i = 1, \dots, N$$

a los datos de altura de padre e hijo, con  $X_i$  la altura del padre e  $y_i$  la altura del hijo, obtenemos un resultado estadísticamente significativo:

```
library(HistData)
data("GaltonFamilies")
GaltonFamilies %>%
 filter(childNum == 1 & gender == "male") %>%
 select(father, childHeight) %>%
 rename(son = childHeight) %>%
 summarize(tidy(lm(father ~ son)))
#> term estimate std.error statistic p.value
#> 1 (Intercept) 33.965 4.5682 7.44 4.31e-12
#> 2 son 0.499 0.0648 7.70 9.47e-13
```

El modelo se ajusta muy bien a los datos. Si observamos la formulación matemática del modelo anterior, podría interpretarse fácilmente de manera incorrecta para sugerir que el hijo siendo alto hace que el padre sea alto. Pero dado lo que sabemos sobre genética y biología, sabemos que es al revés. El modelo es técnicamente correcto. Los estimadores y los valores-p también se obtuvieron correctamente. Lo que está mal aquí es la interpretación.

<sup>2</sup><https://opinionator.blogs.nytimes.com/2014/04/12/parental-involvement-is-overrated>

## 19.4 Factores de confusión

Los factores de confusión son quizás la razón más común que conduce a que las asociaciones se malinterpreten.

Si  $X$  e  $Y$  están correlacionados, llamamos  $Z$  un *factor de confusión* (*confounder* en inglés) si cambios en  $Z$  provocan cambios en ambos  $X$  e  $Y$ . Anteriormente, al estudiar los datos del béisbol, vimos cómo los cuadrangulares eran un factor de confusión que resultaban en una correlación más alta de lo esperado al estudiar la relación entre BB y HR. En algunos casos, podemos usar modelos lineales para tomar en cuenta los factores de confusión. Sin embargo, este no siempre es el caso.

La interpretación incorrecta debido a factores de confusión es omnipresente en la prensa laica y, a menudo, son difíciles de detectar. Aquí, presentamos un ejemplo ampliamente utilizado relacionado con las admisiones a la universidad.

### 19.4.1 Ejemplo: admisiones a la Universidad de California, Berkeley

Los datos de admisión de seis concentraciones de U.C. Berkeley, de 1973, mostraron que se admitían a más hombres que mujeres: el 44% de los hombres fueron aceptados en comparación con el 30% de las mujeres. PJ Bickel, EA Hammel & JW O'Connell. Science (1975). Podemos cargar los datos y ejecutar una prueba estadística, que rechaza claramente la hipótesis de que el género y la admisión son independientes:

```
data(admissions)
admissions %>% group_by(gender) %>%
 summarize(total_admitted = round(sum(admitted/ 100 * applicants)),
 not_admitted = sum(applicants) - sum(total_admitted)) %>%
 select(-gender) %>%
 summarize(tidy(chisq.test(.))) %>% .$p.value
#> [1] 1.06e-21
```

Pero una inspección más cuidadosa muestra un resultado paradójico. Aquí están los porcentajes de admisión por concentración :

```
admissions %>% select(major, gender, admitted) %>%
 pivot_wider(names_from = "gender", values_from = "admitted") %>%
 mutate(women_minus_men = women - men)
#> # A tibble: 6 x 4
#> major men women women_minus_men
#> <chr> <dbl> <dbl> <dbl>
#> 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
#> # ... with 1 more row
```

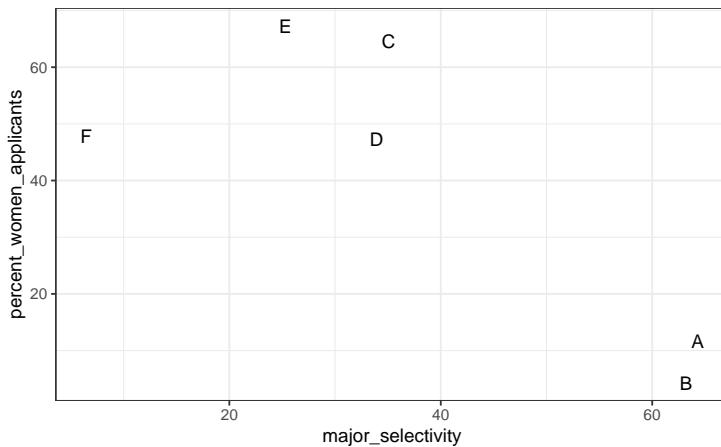
Cuatro de las seis concentraciones favorecen a las mujeres. Más importante aún, todas las diferencias son mucho más pequeñas que la diferencia de 14.2 que vemos al examinar los totales.

La paradoja es que analizar los totales sugiere una dependencia entre admisión y género, pero cuando los datos se agrupan por concentración, esta dependencia parece desaparecer. ¿Qué está pasando? Esto puede suceder cuando un factor de confusión no detectado está impulsando la mayor parte de la variabilidad.

Así que definamos tres variables:  $X$  es 1 para hombres y 0 para mujeres,  $Y$  es 1 para los admitidos y 0 en caso contrario, y  $Z$  cuantifica la selectividad de la concentración. Una afirmación de sesgo de género se basaría en el hecho de que  $\Pr(Y = 1|X = x)$  es mayor para  $x = 1$  que  $x = 0$ . Sin embargo,  $Z$  es un factor de confusión importante para tomar en cuenta. Claramente  $Z$  está asociado con  $Y$ , ya que entre más selectiva sea la concentración,  $\Pr(Y = 1|Z = z)$  será menor. Pero, ¿está asociada la selección de concentración  $Z$  con el género  $X$ ?

Una forma de ver esto es graficar el porcentaje total admitido a una concentración versus el porcentaje de mujeres que componen los solicitantes:

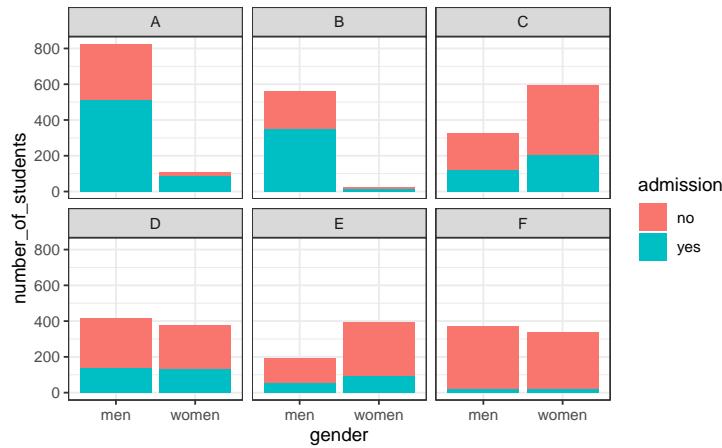
```
admissions %>%
 group_by(major) %>%
 summarize(major_selectivity = sum(admitted * applicants)/sum(applicants),
 percent_women_applicants = sum(applicants * (gender=="women"))/
 sum(applicants) * 100) %>%
 ggplot(aes(major_selectivity, percent_women_applicants, label = major)) +
 geom_text()
```



Parece haber asociación. El gráfico sugiere que las mujeres eran mucho más propensas a solicitar a las dos concentraciones “difíciles”: el género y la selectividad de la concentración están confundidos. Compare, por ejemplo, la concentración B y la E. La concentración B es mucho más difícil de ingresar que la B y más del 60% de los solicitantes a la concentración E eran mujeres, mientras que menos del 30% de los solicitantes a la concentración B eran mujeres.

### 19.4.2 Confusión explicada gráficamente

El siguiente gráfico muestra el número de solicitantes que fueron admitidos y los que no fueron según sexo:

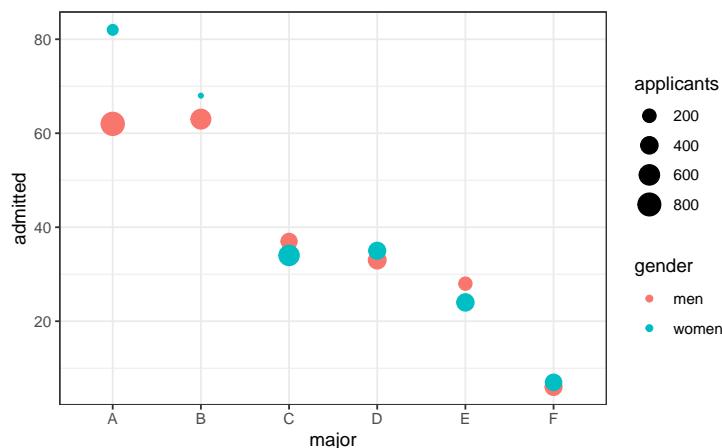


También desglosamos las aceptaciones por concentración. Este gráfico nos permite ver que la mayoría de los hombres aceptados provenían de dos concentraciones: A y B, y que pocas mujeres solicitaron estas concentraciones.

### 19.4.3 Calcular promedio luego de estratificar

En este gráfico, podemos ver que si condicionamos o estratificamos por concentración, controlamos el factor de confusión y este efecto desaparece:

```
admissions %>%
 ggplot(aes(major, admitted, col = gender, size = applicants)) +
 geom_point()
```



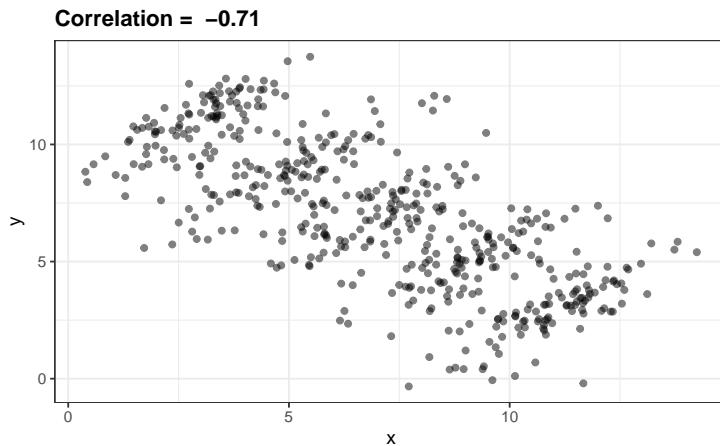
Ahora vemos que concentración por concentración, no hay mucha diferencia. El tamaño del punto representa el número de solicitantes y explica la paradoja: vemos grandes puntos rojos y pequeños puntos azules para las concentraciones más fáciles/menos retantes, A y B.

Si promediamos la diferencia por concentración, encontramos que el porcentaje es 3.5% más alto para las mujeres.

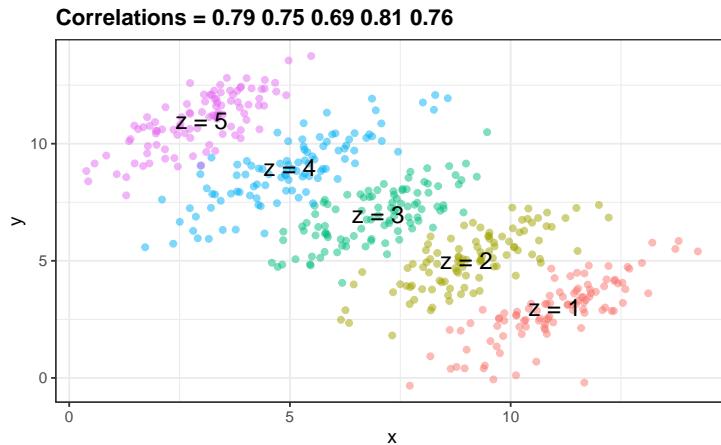
```
admissions %>% group_by(gender) %>% summarize(average = mean(admitted))
#> # A tibble: 2 x 2
#> gender average
#> <chr> <dbl>
#> 1 men 38.2
#> 2 women 41.7
```

## 19.5 La paradoja de Simpson

El caso que acabamos de discutir es un ejemplo de la paradoja de Simpson. Se le llama paradoja porque vemos que el signo de la correlación cambia al comparar la población entera y estratos específicos. Como ejemplo ilustrativo, supongan que observamos realizaciones de las tres variables aleatorias  $X$ ,  $Y$  y  $Z$ . Aquí hay un gráfico de observaciones simuladas para  $X$  e  $Y$  junto con la correlación de muestra:



Pueden ver que  $X$  e  $Y$  están correlacionados negativamente. Sin embargo, una vez que estratificamos por  $Z$  (se muestra en diferentes colores a continuación), surge otro patrón:



Realmente es  $Z$  que está negativamente correlacionada con  $X$ . Si estratificamos por  $Z$ , las  $X$  e  $Y$  están positivamente correlacionadas como se observa en el gráfico anterior.

## 19.6 Ejercicios

Para el próximo set de ejercicios, examinamos los datos de un artículo de PNAS del 2014<sup>3</sup> que analizó las tasas de éxito de agencias de financiación en los Países Bajos y concluyó que:

Nuestros resultados revelan un sesgo de género que favorece a los solicitantes sobre las solicitantes en la priorización de sus evaluaciones y tasas de éxito con respecto a su “calidad de investigador” (pero no “calidad de propuesta”), así como en el lenguaje utilizado en los materiales de instrucción y evaluación.

Unos meses después, se publicó una respuesta<sup>4</sup> titulada *No evidence that gender contributes to personal research funding success in The Netherlands: A reaction to Van der Lee and Ellemers* que concluyó:

Sin embargo, el efecto general del género apenas alcanza significancia estadística, a pesar del tamaño grande de la muestra. Además, su conclusión podría ser un excelente ejemplo de la paradoja de Simpson; si un mayor porcentaje de mujeres solicita subvenciones en disciplinas científicas más competitivas (es decir, con bajas tasas de éxito de solicitudes tanto para hombres como para mujeres), entonces un análisis de todas las disciplinas podría mostrar incorrectamente “evidencia” de desigualdad de género.

¿Quién tiene la razón aquí? ¿El artículo original o la respuesta? Aquí, examinarán los datos y llegarán a su propia conclusión.

1. La evidencia principal para la conclusión del artículo original se reduce a una comparación de los porcentajes. La Tabla S1 en el artículo incluye la información que necesitamos:

<sup>3</sup><http://www.pnas.org/content/112/40/12349.abstract>

<sup>4</sup><http://www.pnas.org/content/112/51/E7036.extract>

```
library(dslabs)
data("research_funding_rates")
research_funding_rates
```

Construya la tabla 2 X 2 utilizada para la conclusión sobre las diferencias en los premios por género.

2. Calcule la diferencia en porcentaje de la tabla 2 X 2.
  3. En el ejercicio anterior, notamos que la tasa de éxito es menor para las mujeres. ¿Pero es significativo? Calcule un valor-p usando una prueba de Chi-cuadrado.
  4. Vemos que el valor-p es aproximadamente 0.05. Entonces parece haber algo de evidencia de una asociación. ¿Pero podemos inferir causalidad aquí? ¿El sesgo de género está causando esta diferencia observada? La respuesta al artículo original afirma que lo que vemos aquí es similar al ejemplo de las admisiones a U.C. Berkeley. Para resolver esta disputa, cree un set de datos con el número de solicitudes, premios y tasas de éxito para cada género. Reordene las disciplinas por su tasa de éxito general. Sugerencia: use la función `reorder` para reordenar las disciplinas como primer paso, luego use `pivot_longer`, `separate` y `pivot_wider` para crear la tabla deseada.
  5. Para verificar si este es un caso de la paradoja de Simpson, grafique las tasas de éxito versus las disciplinas, que han sido ordenadas según éxito general, con colores para denotar los géneros y tamaño para denotar el número de solicitudes.
  6. Definitivamente no vemos el mismo nivel de confusión que en el ejemplo de U.C. Berkeley. Es difícil decir que hay un factor de confusión aquí. Sin embargo, vemos que, según las tasas observadas, algunos campos favorecen a los hombres y otros favorecen a las mujeres. Además, vemos que los dos campos con la mayor diferencia que favorecen a los hombres también son los campos con más solicitudes. Pero, a diferencia del ejemplo de U.C. Berkeley, no es más probable que las mujeres soliciten las concentraciones más difíciles. Entonces, quizás algunos de los comités de selección son parciales y otros no.
- Pero, antes de concluir esto, debemos verificar si estas diferencias son diferentes de lo que obtenemos por casualidad. ¿Alguna de las diferencias vistas anteriormente es estadísticamente significativa? Tengan en cuenta que incluso cuando no hay sesgo, veremos diferencias debido a la variabilidad aleatoria en el proceso de revisión, así como entre los candidatos. Realice una prueba de Chi-cuadrado para cada disciplina. Sugerencia: defina una función que reciba el total de una tabla 2 X 2 y devuelva un *data frame* con el valor-p. Use la corrección 0.5. Luego use la función `summarize`.
7. Para las ciencias médicas, parece haber una diferencia estadísticamente significativa. ¿Pero es esto una correlación espuria? Realice 9 pruebas. Informar solo el caso con un valor-p inferior a 0.05 podría considerarse un ejemplo de dragado de datos. Repita el ejercicio anterior, pero en lugar de un valor-p, calcule un logaritmo de riesgo relativo (*log odds ratio* en inglés) dividido por su error estándar. Entonces use un gráfico Q-Q para ver cuánto se desvían estos logaritmos de riesgo relativo de la distribución normal que esperaríamos: una distribución normal estándar.

---

---

## Part IV

# Wrangling de datos



# 20

---

## *Introducción al wrangling de datos*

---

Los sets de datos utilizados en este libro se han puesto a su disposición como objetos R, específicamente como *data frames*. Los datos de asesinatos de EE. UU., los datos reportados de alturas y los datos de Gapminder son todos *data frames*. Estos sets de datos vienen incluidos en el paquete **dslabs** y los cargamos usando la función `data`. Además, hemos puesto a disposición los datos en formato *tidy*. Los paquetes y las funciones de tidyverse suponen que los datos son *tidy* y esta suposición es una gran parte de la razón por la que estos paquetes funcionan tan bien juntos.

Sin embargo, es bien raro que en un proyecto de ciencia de datos haya datos fácilmente disponibles como parte de un paquete. Hicimos un buen trabajo “tras bastidores” para convertir los datos originales en las tablas *tidy* con que trabajamos. Mucho más común es que los datos estén en un archivo, una base de datos o extraídos de un documento, incluyendo páginas web, tuits o PDF. En estos casos, el primer paso es importar los datos a R y, cuando estemos usando **tidyverse**, ordenar los datos. Este paso inicial en el proceso de análisis de datos generalmente implica varios pasos, a menudo complicados, para convertir datos al formato *tidy* que facilita enormemente el resto del análisis. Nos referimos a este proceso como *wrangling de datos*.

Aquí cubrimos varios pasos comunes del proceso del *wrangling* de datos, incluyendo cómo convertir los datos en formato *tidy*, procesar cadenas, leer y procesar (*parse* en inglés) HTML, trabajar con fechas y horas y, finalmente, *minería de textos* (*text mining* en inglés). Raras veces se necesita hacer todos estos pasos de *wrangling* en un solo análisis, pero los científicos de datos probablemente enfrentarán a todos en algún momento. Algunos de los ejemplos que utilizamos para demostrar las técnicas del *wrangling* de datos se basan en el trabajo que hicimos para convertir datos sin procesar en los sets de datos *tidy* ofrecidos por el paquete **dslabs** y utilizados en el libro como ejemplos.



# 21

---

## Cómo cambiar el formato de datos

---

Como hemos visto a través del libro, tener datos en formato *tidy* es lo que hace que el tidyverse fluya. Después del primer paso en el proceso de análisis de datos, la importación de datos, un siguiente paso común es cambiar la forma de los datos a una que facilite el resto del análisis. El paquete **tidyverse** incluye varias funciones útiles para poner los datos en formato *tidy*.

Utilizaremos el set de datos en formato ancho *fertility*, descrito en la Sección 4.1, como ejemplo en esta sección.

```
library(tidyverse)
library(dslabs)
path <- system.file("extdata", package="dslabs")
filename <- file.path(path, "fertility-two-countries-example.csv")
wide_data <- read_csv(filename)
```

---

### 21.1 pivot\_longer

Una de las funciones más usadas del paquete **tidyverse** es **pivot\_longer**, que nos permite convertir datos anchos (*wide data* en inglés) en datos *tidy*.

Igual que con la mayoría de las funciones de tidyverse, el primer argumento de la función **pivot\_longer** es el *data frame* que será procesado. Aquí queremos cambiar la forma del set de datos `wide_data` para que cada fila represente una observación de fertilidad, que implica que necesitamos tres columnas para almacenar el año, el país y el valor observado. En su forma actual, los datos de diferentes años están en diferentes columnas con los valores de año almacenados en los nombres de las columnas. A través de los argumentos `names_to` y `values_to`, le daremos a **pivot\_longer** los nombres de columna que le queremos asignar a las columnas que contienen los nombres de columna y las observaciones actuales, respectivamente. Por defecto, estos nombres son `name` (nombre) y `value` (valor), los cuales son buenas opciones en general. En este caso, una mejor opción para estos dos argumentos serían `year` y `fertility`. Noten que ninguna parte del archivo nos dice que se trata de datos de fertilidad. En cambio, desciframos esto del nombre del archivo. A través `cols`, el segundo argumento, especificamos las columnas que contienen los valores observados; estas son las columnas que serán *pivotadas*. La acción por defecto es recopilar todas las columnas, por lo que, en la mayoría de los casos, tenemos que especificar las columnas. En nuestro ejemplo queremos las columnas 1960, 1961 hasta 2015.

El código para recopilar los datos de fertilidad se ve así:

```
new_tidy_data <- pivot_longer(wide_data, `1960`:`2015`,
 names_to = "year", values_to = "fertility")
```

También podemos usar el *pipe* de esta manera:

```
new_tidy_data <- wide_data %>%
 pivot_longer(`1960`:`2015`, names_to = "year", values_to = "fertility")
```

Podemos ver que los datos se han convertido al formato *tidy* con columnas **year** y **fertility**:

```
head(new_tidy_data)
#> # A tibble: 6 x 3
#> country year fertility
#> <chr> <chr> <dbl>
#> 1 Germany 1960 2.41
#> 2 Germany 1961 2.44
#> 3 Germany 1962 2.47
#> 4 Germany 1963 2.49
#> 5 Germany 1964 2.49
#> # ... with 1 more row
```

y que cada año resultó en dos filas ya que tenemos dos países y la columna de los países no se recopiló. Una forma un poco más rápida de escribir este código es especificar qué columna **no** se recopilará, en lugar de todas las columnas que se recopilarán:

```
new_tidy_data <- wide_data %>%
 pivot_longer(-country, names_to = "year", values_to = "fertility")
```

El objeto **new\_tidy\_data** se parece al original **tidy\_data** que definimos de esta manera:

```
data("gapminder")
tidy_data <- gapminder %>%
 filter(country %in% c("South Korea", "Germany") & !is.na(fertility)) %>%
 select(country, year, fertility)
```

con solo una pequeña diferencia. ¿La pueden ver? Miren el tipo de datos de la columna del año:

```
class(tidy_data$year)
#> [1] "integer"
class(new_tidy_data$year)
#> [1] "character"
```

La función **pivot\_longer** supone que los nombres de columna son caracteres. Así que necesitamos un poco más de *wrangling* antes de poder graficar. Necesitamos convertir la columna con los años en números. La función **pivot\_longer** incluye el argumento **convert** para este propósito:

```
new_tidy_data <- wide_data %>%
 pivot_longer(-country, names_to = "year", values_to = "fertility") %>%
 mutate(year = as.integer(year))
```

Tengan en cuenta que también podríamos haber utilizado `mutate` y `as.numeric`.

Ahora que los datos están *tidy*, podemos usar este código relativamente sencillo de `ggplot2`:

```
new_tidy_data %>% ggplot(aes(year, fertility, color = country)) +
 geom_point()
```

## 21.2 pivot\_wider

Como veremos en ejemplos posteriores, a veces es útil convertir datos *tidy* en datos anchos para fines de *wrangling* de datos. A menudo usamos esto como un paso intermedio para convertir los datos en formato *tidy*. La función `pivot_wider` es básicamente la inversa de `pivot_longer`. El primer argumento es para los datos, pero como estamos usando el *pipe*, no lo mostramos. El argumento `names_from` le dice a `pivot_longer` qué variable usar como nombre de columna. El argumento `names_to` especifica qué variable usar para completar las celdas:

```
new_wide_data <- new_tidy_data %>%
 pivot_wider(names_from = year, values_from = 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
```

Similar a `pivot_wider`, `names_from` y `values_from` son `name` and `value` por defecto.

## 21.3 separate

El *wrangling* de datos que mostramos arriba es sencillo en comparación con lo que generalmente se requiere. En nuestros archivos de hoja de cálculo que usamos como ejemplo, incluimos una ilustración que es un poco más complicada. Contiene dos variables: esperanza de vida y fertilidad. Sin embargo, la forma en que se almacena no es *tidy* y, como explicaremos, no es óptima.

```
path <- system.file("extdata", package = "dslabs")

filename <- "life-expectancy-and-fertility-two-countries-example.csv"
filename <- file.path(path, filename)

raw_dat <- read_csv(filename)
select(raw_dat, 1:5)
#> # A tibble: 2 x 5
#> country `1960_fertility` `1960_life_expectancy` `1961_fertility`
#> <chr> <dbl> <dbl> <dbl>
#> 1 Germany 2.41 69.3 2.44
#> 2 South Korea 6.16 53.0 5.99
#> # ... with 1 more variable: 1961_life_expectancy <dbl>
```

Primero, tengan en cuenta que los datos están en formato ancho. Además, observen que esta tabla incluye valores para dos variables, fertilidad y esperanza de vida, con el nombre (en inglés) de la columna codificando qué columna representa qué variable. No recomendamos codificar la información en los nombres de las columnas, pero, desafortunadamente, es algo bastante común. Usaremos nuestras habilidades de *wrangling* para extraer esta información y almacenarla de manera *tidy*.

Podemos comenzar el *wrangling* de datos con la función `pivot_longer`, pero ya no deberíamos usar el nombre de la columna `year` para la nueva columna, dado que también contiene el tipo de variable. La nombraremos `name`, el valor predeterminado, por ahora:

```
dat <- raw_dat %>% pivot_longer(-country)
head(dat)
#> # A tibble: 6 x 3
#> country name value
#> <chr> <chr> <dbl>
#> 1 Germany 1960_fertility 2.41
#> 2 Germany 1960_life_expectancy 69.3
#> 3 Germany 1961_fertility 2.44
#> 4 Germany 1961_life_expectancy 69.8
#> 5 Germany 1962_fertility 2.47
#> # ... with 1 more row
```

El resultado no es exactamente lo que llamamos *tidy* ya que cada observación está asociada con dos filas en vez de una. Queremos tener los valores de las dos variables, `fertility` y `life_expectancy`, en dos columnas separadas. El primer reto para lograr esto es separar la columna `name` en año y tipo de variable. Observen que las entradas en esta columna separan el año del nombre de la variable con una barra baja:

```
dat$name[1:5]
#> [1] "1960_fertility" "1960_life_expectancy" "1961_fertility"
#> [4] "1961_life_expectancy" "1962_fertility"
```

Codificar múltiples variables en el nombre de una columna es un problema tan común que el paquete `readr` incluye una función para separar estas columnas en dos o más. Aparte de los datos, la función `separate` toma tres argumentos: el nombre de la columna que se

separará, los nombres que se utilizarán para las nuevas columnas y el carácter que separa las variables. Entonces, un primer intento de hacer esto es:

```
dat %>% separate(name, c("year", "name"), "_")
```

`separate` supone por defecto que `_` es el separador y, por eso, no tenemos que incluirlo en el código:

```
dat %>% separate(name, c("year", "name"))
#> Warning: Expected 2 pieces. Additional pieces discarded in 112 rows [2,
#> 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38,
#> 40, ...].
#> # A tibble: 224 x 4
#> country year name value
#> <chr> <chr> <chr> <dbl>
#> 1 Germany 1960 fertility 2.41
#> 2 Germany 1960 life 69.3
#> 3 Germany 1961 fertility 2.44
#> 4 Germany 1961 life 69.8
#> 5 Germany 1962 fertility 2.47
#> # ... with 219 more rows
```

La función separa los valores, pero nos encontramos con un nuevo problema. Recibimos la advertencia `Too many values at 112 locations`: y la variable `life_expectancy` se corta a `life`. Esto es porque el `_` se usa para separar `life` y `expectancy`, no solo el año y el nombre de la variable. Podríamos añadir una tercera columna para guardar esto y dejar que la función `separate` sepa cual columna *llenar* con los valores faltantes, `NA`, cuando no hay un tercer valor. Aquí le decimos que llene la columna de la derecha:

```
var_names <- c("year", "first_variable_name", "second_variable_name")
dat %>% separate(name, var_names, 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 Germany 1960 life expectancy 69.3
#> 3 Germany 1961 fertility <NA> 2.44
#> 4 Germany 1961 life expectancy 69.8
#> 5 Germany 1962 fertility <NA> 2.47
#> # ... with 219 more rows
```

Sin embargo, si leemos el archivo de ayuda de `separate`, encontramos que un mejor enfoque es fusionar las dos últimas variables cuando hay una separación adicional:

```
dat %>% separate(name, c("year", "name"), extra = "merge")
#> # A tibble: 224 x 4
#> country year name value
#> <chr> <chr> <chr> <dbl>
#> 1 Germany 1960 fertility 2.41
#> 2 Germany 1960 life_expectancy 69.3
```

```
#> 3 Germany 1961 fertility 2.44
#> 4 Germany 1961 life_expectancy 69.8
#> 5 Germany 1962 fertility 2.47
#> # ... with 219 more rows
```

Esto logra la separación que queríamos. Sin embargo, aún no hemos terminado. Necesitamos crear una columna para cada variable. Como aprendimos, la función `pivot_wider` hace eso:

```
dat %>%
 separate(name, c("year", "name"), extra = "merge") %>%
 pivot_wider()
#> # 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
#> # ... with 107 more rows
```

Los datos ahora están en formato *tidy* con una fila para cada observación con tres variables: año, fertilidad y esperanza de vida.

## 21.4 unite

A veces es útil hacer el inverso de `separate`, es decir, unir dos columnas en una. Para demostrar cómo usar `unite`, mostramos un código que, aunque *no* es el acercamiento óptimo, sirve como ilustración. Supongan que no supiéramos sobre `extra` y usáramos este comando para separar:

```
var_names <- c("year", "first_variable_name", "second_variable_name")
dat %>%
 separate(name, var_names, 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 Germany 1960 life expectancy 69.3
#> 3 Germany 1961 fertility <NA> 2.44
#> 4 Germany 1961 life expectancy 69.8
#> 5 Germany 1962 fertility <NA> 2.47
#> # ... with 219 more rows
```

Podemos lograr el mismo resultado final uniendo las segunda y tercera columnas, luego esparciendo las columnas usando `pivot_wider` y renombrando `fertility_NA` a `fertility`:

```
dat %>%
 separate(name, var_names, fill = "right") %>%
 unite(name, first_variable_name, second_variable_name) %>%
 pivot_wider() %>%
 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
#> # ... with 107 more rows
```

## 21.5 Ejercicios

1. Ejecute el siguiente comando para definir el objeto `co2_wide`:

```
co2_wide <- data.frame(matrix(co2, ncol = 12, byrow = TRUE)) %>%
 setNames(1:12) %>%
 mutate(year = as.character(1959:1997))
```

Utilice la función `pivot_longer` para *wrangle* esto en un set de datos *tidy*. Nombre a la columna con las mediciones de CO2 `co2` y nombre a la columna de mes `month`. Nombre al objeto resultante `co2_tidy`.

2. Grafique CO2 versus mes con una curva diferente para cada año usando este código:

```
co2_tidy %>% ggplot(aes(month, co2, color = year)) + geom_line()
```

Si no se realiza el gráfico esperado, probablemente es porque `co2_tidy$month` no es numérico:

```
class(co2_tidy$month)
```

Reescriba el código y que asegúrese que la columna de mes será numérica. Luego haga el gráfico

3. ¿Qué aprendemos de este gráfico?

- Las medidas de CO2 aumentan monotónicamente de 1959 a 1997.
- Las medidas de CO2 son más altas en el verano y el promedio anual aumentó de 1959 a 1997.
- Las medidas de CO2 parecen constantes y la variabilidad aleatoria explica las diferencias.

- d. Las medidas de CO<sub>2</sub> no tienen una tendencia estacional.
4. Ahora cargue el set de datos `admissions`, que contiene información de admisión para hombres y mujeres en seis concentraciones y mantenga solo la columna de porcentaje admitido:

```
load(admissions)
dat <- admissions %>% select(-applicants)
```

Si pensamos en una observación como una concentración, y que cada observación tiene dos variables (porcentaje de hombres admitidos y porcentaje de mujeres admitidas), entonces esto no es *tidy*. Utilice la función `pivot_wider` para *wrangle* en la forma *tidy* que queremos: una fila para cada concentración.

5. Ahora intentaremos un reto más avanzado de *wrangling*. Queremos *wrangle* los datos de admisión para cada concentración para tener 4 observaciones: `admitted_men`, `admitted_women`, `applicants_men` y `applicants_women`. El “truco” que hacemos aquí es realmente bastante común: primero usamos `pivot_longer` para generar un *data frame* intermedio y luego usamos `pivot_wider` para obtener los datos *tidy* que queremos. Iremos paso a paso en este y en los próximos dos ejercicios.

Utilice la función `pivot_longer` para crear un *data frame* `tmp` con una columna que contiene el tipo de observación `admitted` o `applicants`. Nombre a las nuevas columnas `name` y `value`.

6. Ahora tiene un objeto `tmp` con columnas `major`, `gender`, `name` y `value`. Tenga en cuenta que si combina `name` y `gender`, se obtienen los nombres de columna que queremos: `admitted_men`, `admitted_women`, `applicants_men` y `applicants_women`. Use la función `unite` para crear una nueva columna llamada `column_name`.

7. Ahora use la función `pivot_wider` para generar los datos *tidy* con cuatro variables para cada concentración.
8. Ahora use el `pipe` para escribir una línea de código que convierta `admissions` en la tabla producida en el ejercicio anterior.

## 22

---

### Unir tablas

---

Es posible que la información que necesitamos para un análisis no esté en solo en una tabla. Por ejemplo, cuando pronosticamos elecciones usamos la función `left_join` para combinar la información de dos tablas. Aquí usamos un ejemplo más sencillo para ilustrar el desafío general de combinar tablas.

Supongan que queremos explorar la relación entre el tamaño de la población de los estados de EE. UU. y los votos electorales. Tenemos el tamaño de la población en esta tabla:

```
library(tidyverse)
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
```

y los votos electorales en esta:

```
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
```

Simplemente concatenar estas dos tablas no funcionará ya que el orden de los estados no es el mismo.

```
identical(results_us_election_2016$state, murders$state)
#> [1] FALSE
```

Las funciones que usamos para unir (*join* en inglés), descritas a continuación, están diseñadas para manejar este desafío.

## 22.1 Funciones para unir

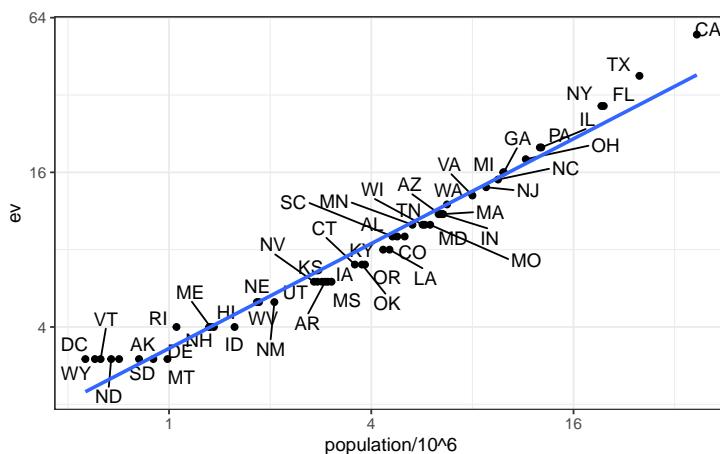
Las funciones para unir del paquete **dplyr** aseguran que las tablas se combinen de tal forma que las filas equivalentes estén juntas. Si conocen SQL, verán que el acercamiento y la sintaxis son muy similares. La idea general es que uno necesita identificar una o más columnas que servirán para emparejar las dos tablas. Entonces se devuelve una nueva tabla con la información combinada. Observen lo que sucede si unimos las dos tablas anteriores por estado usando `left_join` (eliminaremos la columna `others` y renombraremos `electoral_votes` para que las tablas quepen en la página):

```
tab <- left_join(murders, results_us_election_2016, by = "state") %>%
 select(-others) %>% rename(ev = electoral_votes)
head(tab)
```

|      | state      | abb | region | population | total | ev | 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  |

Los datos se han unido exitosamente y ahora podemos, por ejemplo, hacer un diagrama para explorar la relación:

```
library(ggrepel)
tab %>% ggplot(aes(population/10^6, ev, label = abb)) +
 geom_point() +
 geom_text_repel() +
 scale_x_continuous(trans = "log2") +
 scale_y_continuous(trans = "log2") +
 geom_smooth(method = "lm", se = FALSE)
```



Vemos que la relación es casi lineal con aproximadamente dos votos electorales para cada millón de personas, pero con estados muy pequeños obteniendo proporciones más altas.

En la práctica, no siempre ocurre que cada fila de una tabla tiene una fila correspondiente en la otra. Por esta razón, tenemos varias versiones de *join*. Para ilustrar este reto, tomaremos subconjuntos de las tablas anteriores. Creamos las tablas `tab1` y `tab2` para que tengan algunos estados en común pero no todos:

```
tab_1 <- slice(murders, 1:6) %>% select(state, population)
tab_1
#> state population
#> 1 Alabama 4779736
#> 2 Alaska 710231
#> 3 Arizona 6392017
#> 4 Arkansas 2915918
#> 5 California 37253956
#> 6 Colorado 5029196
tab_2 <- results_us_election_2016 %>%
 filter(state%in%c("Alabama", "Alaska", "Arizona",
 "California", "Connecticut", "Delaware")) %>%
 select(state, electoral_votes) %>% rename(ev = electoral_votes)
tab_2
#> state ev
#> 1 California 55
#> 2 Arizona 11
#> 3 Alabama 9
#> 4 Connecticut 7
#> 5 Alaska 3
#> 6 Delaware 3
```

Utilizaremos estas dos tablas como ejemplos en las siguientes secciones.

### 22.1.1 Left join

Supongan que queremos una tabla como `tab_1`, pero agregando votos electorales a cualquier estado que tengamos disponible. Para esto, usamos `left_join` con `tab_1` como el primer argumento. Especifiquemos qué columna usar para que coincida con el argumento `by`.

```
left_join(tab_1, tab_2, by = "state")
#> state population ev
#> 1 Alabama 4779736 9
#> 2 Alaska 710231 3
#> 3 Arizona 6392017 11
#> 4 Arkansas 2915918 NA
#> 5 California 37253956 55
#> 6 Colorado 5029196 NA
```

Tengan en cuenta que `NAs` se agregan a los dos estados que no aparecen en `tab_2`. Además, observen que esta función, así como todas las otras *joins*, pueden recibir los primeros argumentos a través del *pipe*:

```
tab_1 %>% left_join(tab_2, by = "state")
```

### 22.1.2 Right join

Si en lugar de una tabla con las mismas filas que la primera tabla, queremos una con las mismas filas que la segunda tabla, podemos usar `right_join`:

```
tab_1 %>% right_join(tab_2, by = "state")
#> state population ev
#> 1 Alabama 4779736 9
#> 2 Alaska 710231 3
#> 3 Arizona 6392017 11
#> 4 California 37253956 55
#> 5 Connecticut NA 7
#> 6 Delaware NA 3
```

Ahora los `NAs` están en la columna de `tab_1`.

### 22.1.3 Inner join

Si queremos mantener solo las filas que tienen información en ambas tablas, usamos `inner_join`. Pueden pensar en esto como una intersección:

```
inner_join(tab_1, tab_2, by = "state")
#> state population ev
#> 1 Alabama 4779736 9
#> 2 Alaska 710231 3
#> 3 Arizona 6392017 11
#> 4 California 37253956 55
```

### 22.1.4 Full join

Si queremos mantener todas las filas y llenar las partes faltantes con `NAs`, podemos usar `full_join`. Pueden pensar en esto como una unión:

```
full_join(tab_1, tab_2, by = "state")
#> state population ev
#> 1 Alabama 4779736 9
#> 2 Alaska 710231 3
#> 3 Arizona 6392017 11
#> 4 Arkansas 2915918 NA
#> 5 California 37253956 55
#> 6 Colorado 5029196 NA
#> 7 Connecticut NA 7
#> 8 Delaware NA 3
```

### 22.1.5 Semi join

La función `semi_join` nos permite mantener la parte de la primera tabla para la cual tenemos información en la segunda. No agrega las columnas de la segunda:

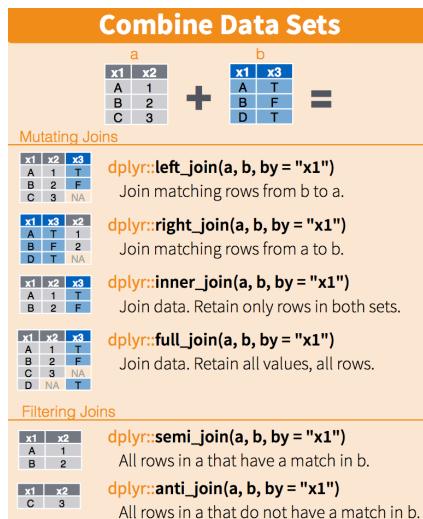
```
semi_join(tab_1, tab_2, by = "state")
#> state population
#> 1 Alabama 4779736
#> 2 Alaska 710231
#> 3 Arizona 6392017
#> 4 California 37253956
```

### 22.1.6 Anti join

La función `anti_join` es la opuesta de `semi_join`. Mantiene los elementos de la primera tabla para los que no hay información en la segunda:

```
anti_join(tab_1, tab_2, by = "state")
#> state population
#> 1 Arkansas 2915918
#> 2 Colorado 5029196
```

El siguiente diagrama resume las funciones `join`:



(Imagen cortesía de RStudio<sup>1</sup>. Licencia CC-BY-4.0<sup>2</sup>. Recortada del original.)

<sup>1</sup><https://github.com/rstudio/cheatsheets>

<sup>2</sup><https://github.com/rstudio/cheatsheets/blob/master/LICENSE>

## 22.2 Binding

Aunque todavía no lo hemos usado en este libro, otra forma común en la que se combinan los sets de datos es *pegándolos* (*binding* en inglés). A diferencia de las funciones *join*, las funciones *binding* no intentan coincidir con una variable, sino que simplemente combinan sets de datos. Si los sets de datos no coinciden con las dimensiones apropiadas, se obtiene un error.

### 22.2.1 Pegando columnas

La función `bind_cols` de `dplyr` pega dos objetos convirtiéndolos en columnas en un *tibble*. Por ejemplo, queremos crear rápidamente un *data frame* que consiste de números que podemos usar.

```
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
```

Esta función requiere que asignemos nombres a las columnas. Aquí elegimos `a` y `b`.

Noten que hay una función de R, `cbind`, con exactamente la misma funcionalidad. Una diferencia importante es que `cbind` puede crear diferentes tipos de objetos, mientras `bind_cols` siempre produce un *data frame*.

`bind_cols` también puede pegar dos *data frames* diferentes. Por ejemplo, aquí sepáramos el *data frame* `tab` en tres *data frames* y luego volvemos a pegarlos:

```
tab_1 <- tab[, 1:3]
tab_2 <- tab[, 4:6]
tab_3 <- tab[, 7:8]
new_tab <- bind_cols(tab_1, tab_2, tab_3)
head(new_tab)
#> state abb region population total ev 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
```

### 22.2.2 Pegando filas

La función `bind_rows` es similar a `bind_cols`, pero pega filas en lugar de columnas:

```
tab_1 <- tab[1:2,]
tab_2 <- tab[3:4,]
bind_rows(tab_1, tab_2)
#> state abb region population total ev 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
```

Esto se basa en la función `rbind` de R.

## 22.3 Operadores de sets

Otro conjunto de comandos útiles para combinar sets de datos son los operadores de sets. Cuando se aplican a los vectores, estos se comportan como lo sugieren sus nombres. Ejemplos son `intersect`, `union`, `setdiff` y `setequal`. Sin embargo, si se carga el `tidyverse`, o más específicamente `dplyr`, estas funciones se pueden usar en *data frames* en lugar de solo en vectores.

### 22.3.1 Intersecar

Pueden tomar intersecciones de vectores de cualquier tipo, como numéricos:

```
intersect(1:10, 6:15)
#> [1] 6 7 8 9 10
```

o caracteres:

```
intersect(c("a", "b", "c"), c("b", "c", "d"))
#> [1] "b" "c"
```

El paquete `dplyr` incluye una función `intersect` que se puede aplicar a tablas con los mismos nombres de columna. Esta función devuelve las filas en común entre dos tablas. Para asegurarnos de que usamos la versión de `dplyr` de `intersect` en lugar de la versión del paquete base, podemos usar `dplyr::intersect` así:

```
tab_1 <- tab[1:5,]
tab_2 <- tab[3:7,]
dplyr::intersect(tab_1, tab_2)
#> state abb region population total ev clinton trump
#> 1 Arizona AZ West 6392017 232 11 45.1 48.7
#> 2 Arkansas AR South 2915918 93 6 33.7 60.6
#> 3 California CA West 37253956 1257 55 61.7 31.6
```

### 22.3.2 Unión

Del mismo modo, `union` toma la unión de vectores. Por ejemplo:

```
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"
```

El paquete `dplyr` incluye una versión de `union` que combina todas las filas de dos tablas con los mismos nombres de columna.

```
tab_1 <- tab[1:5,]
tab_2 <- tab[3:7,]
dplyr::union(tab_1, tab_2)
#> state abb region population total ev 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
```

### 22.3.3 setdiff

La diferencia establecida entre un primer y un segundo argumento se puede obtener con `setdiff`. A diferencia de `intersect` y `union`, esta función no es simétrica:

```
setdiff(1:10, 6:15)
#> [1] 1 2 3 4 5
setdiff(6:15, 1:10)
#> [1] 11 12 13 14 15
```

Al igual que con las funciones que se muestran arriba, `dplyr` tiene una versión para *data frames*:

```
tab_1 <- tab[1:5,]
tab_2 <- tab[3:7,]
dplyr::setdiff(tab_1, tab_2)
#> state abb region population total ev clinton trump
#> 1 Alabama AL South 4779736 135 9 34.4 62.1
#> 2 Alaska AK West 710231 19 3 36.6 51.3
```

### 22.3.4 setequal

Finalmente, la función `setequal` nos dice si dos sets son iguales, independientemente del orden. Noten que:

```
setequal(1:5, 1:6)
#> [1] FALSE
```

pero:

```
setequal(1:5, 5:1)
#> [1] TRUE
```

Cuando se aplica a *data frames* que no son iguales, independientemente del orden, la versión **dplyr** ofrece un mensaje útil que nos permite saber cómo los sets son diferentes:

```
dplyr::setequal(tab_1, tab_2)
#> [1] FALSE
```

---

## 22.4 Ejercicios

1. Instale y cargue la biblioteca **Lahman**. Esta base de datos incluye datos relacionados a equipos de béisbol. Incluya estadísticas sobre cómo se desempeñaron los jugadores ofensiva y defensivamente durante varios años. También incluye información personal sobre los jugadores.

El *data frame* **Batting** contiene las estadísticas ofensivas de todos los jugadores durante muchos años. Puede ver, por ejemplo, los 10 mejores bateadores ejecutando este código:

```
library(Lahman)

top <- Batting %>%
 filter(yearID == 2016) %>%
 arrange(desc(HR)) %>%
 slice(1:10)

top %>% as_tibble()
```

¿Pero quiénes son estos jugadores? Vemos una identificación, pero no los nombres. Los nombres de los jugadores están en esta tabla:

```
Master %>% as_tibble()
```

Podemos ver los nombres de las columnas **nameFirst** y **nameLast**. Utilice la función **left\_join** para crear una tabla de los mejores bateadores de cuadrangulares. La tabla debe tener **playerID**, nombre, apellido y número de cuadrangulares (HR). Reescriba el objeto **top** con esta nueva tabla.

2. Ahora use el *data frame* **Salaries** para añadir el salario de cada jugador a la tabla que creó en el ejercicio 1. Note que los salarios son diferentes cada año, así que asegúrese de filtrar

para el año 2016, luego use `right_join`. Esta vez muestre el nombre, apellido, equipo, HR y salario.

3. En un ejercicio anterior, creamos una versión ordenada del set de datos `co2`:

```
co2_wide <- data.frame(matrix(co2, ncol = 12, byrow = TRUE)) %>%
 setNames(1:12) %>%
 mutate(year = 1959:1997) %>%
 pivot_longer(-year, names_to = "month", values_to = "co2") %>%
 mutate(month = as.numeric(month))
```

Queremos ver si la tendencia mensual está cambiando, por lo que eliminaremos los efectos del año y luego graficaremos los resultados. Primero, calcularemos los promedios del año. Utilice `group_by` y `summarize` para calcular el CO2 promedio de cada año. Guárdelo en un objeto llamado `yearly_avg`.

4. Ahora use la función `left_join` para agregar el promedio anual al set de datos `co2_wide`. Entonces calcule los residuos: medida de CO2 observada - promedio anual.

5. Haga un diagrama de las tendencias estacionales por año, pero solo después de eliminar el efecto del año.

# 23

## Extracción de la web

Los datos que necesitamos para responder a una pregunta no siempre están en una hoja de cálculo lista para leer. Por ejemplo, el set de datos sobre asesinatos de EE.UU. que utilizamos en el Capítulo “Lo básico de R” proviene originalmente de esta página de Wikipedia:

```
url <- paste0("https://en.wikipedia.org/w/index.php?title=",
 "Gun_violence_in_the_United_States_by_state",
 "&direction=prev&oldid=810166167")
```

Pueden ver la tabla de datos en la página web:



The screenshot shows a Wikipedia article titled 'Gun violence in the United States by state'. The page has a standard header with links for 'Article', 'Talk', 'Read', 'Edit', 'View history', and a search bar. Below the header is a note about the revision date and URL. The main content is a table titled 'States' with data for five states: Alabama, Alaska, Arizona, Arkansas, and California. The table includes columns for State, Population (2010) [1], Murders and Nonnegligent Manslaughter (2010) [2], and Murder and Nonnegligent Manslaughter Rate (per 100,000 inhabitants) (2010).

| State      | Population<br>(2010)<br>[1] | Murders and Nonnegligent<br>Manslaughter<br>(2010)<br>[2] | Murder and Nonnegligent<br>Manslaughter Rate<br>(per 100,000 inhabitants)<br>(2010) |
|------------|-----------------------------|-----------------------------------------------------------|-------------------------------------------------------------------------------------|
| Alabama    | 4,853,875                   | 348                                                       | 7.2                                                                                 |
| Alaska     | 737,709                     | 59                                                        | 8.0                                                                                 |
| Arizona    | 6,817,865                   | 309                                                       | 4.5                                                                                 |
| Arkansas   | 2,977,853                   | 181                                                       | 6.1                                                                                 |
| California | 38,993,940                  | 1,861                                                     | 4.8                                                                                 |

(Página web cortesía de Wikipedia<sup>1</sup>. Licencia CC-BY-SA-3.0)<sup>2</sup>.

Desafortunadamente, no hay un enlace a un archivo de datos. Para crear el *data frame* que se carga cuando escribimos `data(murders)`, tuvimos que hacer un poco de *extracción de la web* (*web scraping* o *web harvesting* en inglés).

*Extracción de la web* es el término que se usa para describir el proceso de extracción de datos de un sitio web. La razón por la que podemos hacer esto es porque la información utilizada por un navegador para representar páginas web se recibe como un archivo de texto de un servidor. El texto es un código escrito en lenguaje de marcado de hipertexto (*hyper text markup language* o HTML por sus siglas en inglés). Todos los navegadores tienen una manera de mostrar el código HTML de una página, cada uno diferente. En Chrome, pueden usar Control-U en una PC y comando + alt + U en una Mac. Verán algo como esto:

<sup>1</sup>[https://en.wikipedia.org/w/index.php?title=Gun\\_violence\\_in\\_the\\_United\\_States\\_by\\_state&direction=prev&oldid=810166167](https://en.wikipedia.org/w/index.php?title=Gun_violence_in_the_United_States_by_state&direction=prev&oldid=810166167)

<sup>2</sup>[https://en.wikipedia.org/wiki/Wikipedia:Text\\_of\\_Creative\\_Commons\\_Attribution-ShareAlike\\_3.0\\_Unported\\_License](https://en.wikipedia.org/wiki/Wikipedia:Text_of_Creative_Commons_Attribution-ShareAlike_3.0_Unported_License)

## 23.1 HTML

Debido a que este código es accesible, podemos descargar el archivo HTML, importarlo a R y luego escribir programas para extraer la información que necesitamos de la página. Sin embargo, al ver el código HTML, esto puede parecer una tarea desalentadora. Pero le mostraremos algunas herramientas convenientes para facilitar el proceso. Para tener una idea de cómo funciona, aquí hay unas líneas de código de la página de Wikipedia que proveen los datos de asesinatos en Estados Unidos:

| State   | Population<br>(total inhabitants)<br>(2015) | Murders and Nonnegligent<br>Manslaughter<br>(total deaths)<br>(2015) | Murder and Nonnegligent<br>Manslaughter Rate<br>(per 100,000 inhabitants)<br>(2015) |
|---------|---------------------------------------------|----------------------------------------------------------------------|-------------------------------------------------------------------------------------|
| Alabama | 4,853,875                                   | 348                                                                  |                                                                                     |

```
<td>7.2</td>
</tr>
<tr>
<td>Alaska</td>
<td>737,709</td>
<td>59</td>
<td>8.0</td>
</tr>
<tr>
```

Pueden ver los datos, excepto que los valores de datos están rodeados por un código HTML como `<td>`. También podemos ver un patrón de cómo se almacenan. Si conocen HTML, pueden escribir programas que aprovechan el conocimiento de estos patrones para extraer lo que queremos. Además, nos aprovechamos de *Cascading Style Sheets* (CSS), un lenguaje ampliamente utilizado para hacer que las páginas web se vean “bonitas”. Discutimos más sobre esto en la Sección 23.3.

Aunque ofrecemos herramientas que permiten extraer datos sin conocer HTML, como científicos de datos es bastante útil aprender algo de HTML y CSS. Esto no solo mejora sus habilidades de extracción, sino que puede ser útil si están creando una página web para exhibir su trabajo. Hay muchos cursos y tutoriales en línea para aprenderlos, como Codecademy<sup>3</sup> y W3schools<sup>4</sup>.

## 23.2 El paquete **rvest**

El **tidyverse** provee un paquete de extracción de la web llamado **rvest**. El primer paso para usar este paquete es importar la página web a R. El paquete lo hace bastante fácil:

```
library(tidyverse)
library(rvest)
h <- read_html(url)
```

Tengan en cuenta que la página entera de Wikipedia *Gun violence in the United States* ahora está contenida en `h`. La clase de este objeto es:

```
class(h)
#> [1] "xml_document" "xml_node"
```

El paquete **rvest** es más general; maneja documentos XML. XML es un lenguaje de marcado general (ML siendo las iniciales de *markup language*) que se puede usar para representar cualquier tipo de datos. HTML es un tipo específico de XML desarrollado específicamente para representar páginas web. Aquí nos concentraremos en documentos HTML.

Ahora, ¿cómo extraemos la tabla del objeto `h`? Si imprimimos `h`, realmente no vemos mucho:

<sup>3</sup><https://www.codecademy.com/learn/learn-html>

<sup>4</sup><https://www.w3schools.com/>

```

h
#> {html_document}
#> <html class="client-nojs" lang="en" dir="ltr">
#> [1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset ...
#> [2] <body class="mediawiki ltr sitedir-ltr mw-hide-empty-elt ns-0 ns- ...

```

Podemos ver todo el código que define la página web descargada usando la función `html_text` así:

```
html_text(h)
```

No mostramos el *output* aquí porque incluye miles de caracteres, pero si lo miramos, podemos ver que los datos que buscamos se almacenan en una tabla HTML. Pueden ver esto en esta línea del código HTML anterior: `<table class="wikitable sortable">`. Las diferentes partes de un documento HTML, a menudo definidas con un mensaje entre `< y >`, se conocen como *nodos* (*nodes* en inglés). El paquete `rvest` incluye funciones para extraer nodos de un documento HTML: `html_nodes` extrae todos los nodos de diferentes tipos y `html_node` extrae el primero. Para extraer las tablas del código HTML usamos:

```
tab <- h %>% html_nodes("table")
```

Ahora, en lugar de toda la página web, solo tenemos el código HTML para las tablas de la página:

```

tab
#> {xml_nodeset (2)}
#> [1] <table class="wikitable sortable"><tbody>\n<tr>\n<th>State</th> ...
#> [2] <table class="nowraplinks hlist mw-collapsible mw-collapsed navbo ...

```

La tabla que nos interesa es la primera:

```

tab[[1]]
#> {html_node}
#> <table class="wikitable sortable">
#> [1] <tbody>\n<tr>\n<th>State</th>\n<th>\n<a href="/wiki/List_of_U.S ...

```

Esto claramente no es un set de datos *tidy*, ni siquiera un *data frame*. En el código anterior, podemos ver un patrón y es muy factible escribir código para extraer solo los datos. De hecho, `rvest` incluye una función solo para convertir tablas HTML en *data frames*:

```

tab <- tab[[1]] %>% html_table
class(tab)
#> [1] "tbl_df" "tbl" "data.frame"

```

Ahora estamos mucho más cerca de tener una tabla de datos utilizables:

```
tab <- tab %>% setNames(c("state", "population", "total", "murder_rate"))
head(tab)
#> # A tibble: 6 x 4
#> state population total murder_rate
#> <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
#> # ... with 1 more row
```

Todavía tenemos que hacer un poco de *wrangling*. Por ejemplo, necesitamos eliminar las comas y convertir los caracteres en números. Antes de continuar con esto, aprenderemos un acercamiento más general para extraer información de sitios web.

### 23.3 Selectores CSS

El aspecto por defecto de una página web hecha con el HTML más básico es poco atractivo. Las páginas estéticamente agradables que vemos hoy usan CSS para definir su aspecto y estilo. El hecho de que todas las páginas de una empresa tienen el mismo estilo generalmente resulta del uso del mismo archivo CSS para definir el estilo. La forma general en que funcionan estos archivos CSS es determinando cómo se verá cada uno de los elementos de una página web. El título, los encabezados, las listas detalladas, las tablas y los enlaces, por ejemplo, reciben cada uno su propio estilo, que incluye la fuente, el color, el tamaño y la distancia del margen. CSS hace esto aprovechando los patrones utilizados para definir estos elementos, denominados *selectores*. Un ejemplo de dicho patrón, que utilizamos anteriormente, es `table`, pero hay muchos más.

Si queremos obtener datos de una página web y conocemos un selector que es único para la parte de la página que contiene estos datos, podemos usar la función `html_nodes`. Sin embargo, saber qué selector puede ser bastante complicado. De hecho, la complejidad de las páginas web ha aumentado a medida que se vuelven más sofisticadas. Para algunas de las más avanzadas, parece casi imposible encontrar los nodos que definen un dato en particular. Sin embargo, SelectorGadget lo hace posible.

SelectorGadget<sup>5</sup> es un software que les permite determinar de manera interactiva qué selector CSS necesita para extraer componentes específicos de la página web. Si van a extraer datos que no son tablas de páginas HTML, les recomendamos que lo instalen. Chrome tiene una extensión que les permite encender el *gadget* y luego, al hacer clic en la página, resalta partes y les muestra el selector que necesitan para extraer estas partes. Hay varias demostraciones de cómo hacer esto, incluyendo este artículo de `rvest`<sup>6</sup> y otros tutoriales basados en esa vignette<sup>7 8</sup>.

<sup>5</sup><http://selectorgadget.com/>

<sup>6</sup><https://rvest.tidyverse.org/articles/selectorgadget.html>

<sup>7</sup>[https://stat4701.github.io/edav/2015/04/02/rvest\\_tutorial/](https://stat4701.github.io/edav/2015/04/02/rvest_tutorial/)

<sup>8</sup><https://www.analyticsvidhya.com/blog/2017/03/beginners-guide-on-web-scraping-in-r-using-rvest-with-hands-on-knowledge/>

## 23.4 JSON

Compartir datos en Internet se ha vuelto cada vez más común. Desafortunadamente, los proveedores usan diferentes formatos, lo que añade dificultad para los científicos de datos reorganizar los datos en R. Sin embargo, hay algunos estándares que también se están volviendo más comunes. Actualmente, un formato que se está adoptando ampliamente es la Notación de Objetos JavaScript (*JavaScript Object Notation* o JSON por sus siglas en inglés). Debido a que este formato es muy general, no se parece en nada a una hoja de cálculo. Este archivo JSON se parece más al código que usamos para definir una lista. Aquí un ejemplo de información almacenada en formato JSON:

```
#>
#> Attaching package: 'jsonlite'
#> The following object is masked from 'package:purrr':
#>
#> flatten
#> [
#> {
#> "name": "Miguel",
#> "student_id": 1,
#> "exam_1": 85,
#> "exam_2": 86
#> },
#> {
#> "name": "Sofia",
#> "student_id": 2,
#> "exam_1": 94,
#> "exam_2": 93
#> },
#> {
#> "name": "Aya",
#> "student_id": 3,
#> "exam_1": 87,
#> "exam_2": 88
#> },
#> {
#> "name": "Cheng",
#> "student_id": 4,
#> "exam_1": 90,
#> "exam_2": 91
#> }
#>]
```

El archivo anterior representa un *data frame*. Para leerlo, podemos usar la función `fromJSON` del paquete **jsonlite**. Noten que los archivos JSON a menudo están disponibles a través de Internet. Varias organizaciones proveen una API JSON o un servicio web al que pueden conectarse directamente y obtener datos. Aquí un ejemplo:

```
library(jsonlite)
citi_bike <- fromJSON("http://citibikenyc.com/stations/json")
```

Esto descarga una lista. El primer argumento les dice cuando lo descargaron:

```
citi_bike$executionTime
```

y el segundo es una tabla de datos:

```
citi_bike$stationBeanList %>% as_tibble()
```

Pueden aprender mucho más examinando tutoriales y archivos de ayuda del paquete **jsonlite**. Este paquete está destinado a tareas relativamente sencillas, como convertir datos en tablas. Para mayor flexibilidad, recomendamos **rjson**.

---

## 23.5 Ejercicios

1. Visite la siguiente página web:

<https://web.archive.org/web/20181024132313/http://www.stevetheump.com/Payrolls.htm>

Observe que hay varias tablas. Digamos que estamos interesados en comparar las nóminas de los equipos a lo largo de los años. Los siguientes ejercicios nos lleva por los pasos necesarios para hacer esto.

Comience aplicando lo que aprendió e importe el sitio web a un objeto llamado **h**.

2. Tenga en cuenta que, aunque no es muy útil, podemos ver el contenido de la página escribiendo:

```
html_text(h)
```

El siguiente paso es extraer las tablas. Para esto, podemos usar la función **html\_nodes**. Aprendimos que las tablas en HTML están asociadas con el nodo **table**. Utilice la función **html\_nodes** y el nodo **table** para extraer la primera tabla. Almacénela en un objeto **nodes**.

3. La función **html\_nodes** devuelve una lista de objetos de clase **xml\_node**. Podemos ver el contenido de cada uno usando, por ejemplo, la función **html\_text**. Puede ver el contenido de un componente elegido arbitrariamente así:

```
html_text(nodes[[8]])
```

Si el contenido de este objeto es una tabla HTML, podemos usar la función **html\_table** para convertirlo en un *data frame*. Utilice la función **html\_table** para convertir la octava entrada de **nodes** en una tabla.

4. Repita lo anterior para los primeros 4 componentes de **nodes**. ¿Cuáles de las siguientes son tablas de cálculo de nómina?

- a. Todas.
  - b. 1
  - c. 2
  - d. 2-4
5. Repita lo anterior para los 3 **últimos** componentes de `nodes`. ¿Cuál de los siguientes es cierto?
- a. La última entrada en `nodes` muestra el promedio de todos los equipos a lo largo del tiempo, no la nómina por equipo.
  - b. Las tres son tablas de cálculo de nómina por equipo.
  - c. Las tres son como la primera entrada, no una tabla de cálculo de nómina.
  - d. Todas las anteriores.
6. Hemos aprendido que la primera y la última entrada de `nodes` no son tablas de cálculo de nómina. Redefina `nodes` para que estas dos se eliminen.
7. Vimos en el análisis anterior que el primer nodo de la tabla realmente no es una tabla. Esto sucede a veces en HTML porque las tablas se usan para hacer que el texto se vea de cierta manera, en lugar de almacenar valores numéricos. Elimine el primer componente y luego use `sapply` y `html_table` para convertir cada nodo en `nodes` en una tabla. Tenga en cuenta que en este caso `sapply` devolverá una lista de tablas. También puede usar `lapply` para asegurar que se aplique una lista.
8. Mire las tablas resultantes. ¿Son todas iguales? ¿Podríamos unirlas con `bind_rows`?
9. Cree dos tablas utilizando las entradas 10 y 19. Llámelas `tab_1` y `tab_2`.
10. Utilice una función `full_join` para combinar estas dos tablas. Antes de hacer esto, corrija el problema del encabezado que falta y haga que los nombres coincidan.
11. Después de unir las tablas, verá varias NAs. Esto se debe a que algunos equipos están en una tabla y no en la otra. Utilice la función `anti_join` para tener una mejor idea de por qué sucede esto.
12. Vemos que uno de los problemas es que los Yankees figuran como *N.Y. Yankees* y *NY Yankees*. En la siguiente sección, aprenderemos enfoques eficientes para solucionar problemas como este. Aquí podemos hacerlo “a mano” de la siguiente manera:

```
tab_1 <- tab_1 %>%
 mutate(Team = ifelse(Team == "N.Y. Yankees", "NY Yankees", Team))
```

Ahora una las tablas y muestre solo Oakland y los Yankees y las columnas de cálculo de nómina.

13. **Avanzado:** Extraiga los títulos de las películas que ganaron el premio de *Best Picture* de este sitio web: <https://m.imdb.com/chart/bestpicture/>

# 24

---

## Procesamiento de cadenas

---

Uno de los desafíos más comunes del *wrangling* de datos consiste en extraer datos numéricos contenidos en cadenas de caracteres y convertirlos en las representaciones numéricas requeridas para hacer gráficos, calcular resúmenes o ajustar modelos en R. También, es común tener que procesar texto no organizado y convertirlo en nombres de variables entendibles o variables categóricas. Muchos de los desafíos de procesamiento de cadenas que enfrentan los científicos de datos son únicos y a menudo inesperados. Por lo tanto, es bastante ambicioso escribir una sección completa sobre este tema. Aquí usamos una serie de estudios de casos que nos ayudan a demostrar cómo el procesamiento de cadenas es un paso necesario para muchos desafíos de disputas de datos. Específicamente, para los ejemplos `murders`, `heights` y `research_funding_rates`, describimos el proceso de convertir los datos sin procesar originales, que aún no hemos mostrados, en los *data frames* que hemos estudiado en este libro.

Al repasar estos estudios de caso, cubriremos algunas de las tareas más comunes en el procesamiento de cadenas, incluyendo cómo extraer números de cadenas, eliminar caracteres no deseados del texto, encontrar y reemplazar caracteres, extraer partes específicas de cadenas, convertir texto de forma libre a formatos más uniformes y dividir cadenas en múltiples valores.

Base R incluye funciones para realizar todas estas tareas. Sin embargo, no siguen una convención unificadora, lo que las hace un poco difíciles de memorizar y usar. El paquete `stringr` reempaquetá esta funcionalidad, pero utiliza un enfoque más consistente de nombrar funciones y ordenar sus argumentos. Por ejemplo, en `stringr`, todas las funciones de procesamiento de cadenas comienzan con `str_`. Esto significa que si escriben `str_` y presionan *tab*, R se completará automáticamente y mostrará todas las funciones disponibles. Como resultado, no tenemos que memorizar todos los nombres de las funciones. Otra ventaja es que en las funciones de este paquete, la cadena que se procesa siempre es el primer argumento, que significa que podemos usar el *pipe* más fácilmente. Por lo tanto, comenzaremos describiendo cómo usar las funciones en el paquete `stringr`.

La mayoría de los ejemplos provendrán del segundo estudio de caso que trata sobre las alturas autoreportadas por los estudiantes y la mayor parte del capítulo se dedica al aprendizaje de expresiones regulares (*regular expressions* o *regex* en inglés) y funciones en el paquete `stringr`.

---

### 24.1 El paquete `stringr`

```
library(tidyverse)
library(stringr)
```

En general, las tareas de procesamiento de cadenas se pueden dividir en **detectar**, **localizar**, **extraer** o **reemplazar** patrones en cadenas. Veremos varios ejemplos. La siguiente tabla incluye las funciones disponibles en el paquete **stringr**. Las dividimos por tarea. También incluimos el equivalente de base R cuando está disponible.

Todas estas funciones toman un vector de caracteres como primer argumento. Además, para cada función, las operaciones se vectorizan: la operación se aplica a cada cadena en el vector.

Finalmente, tengan en cuenta que en esta tabla mencionamos *groups*. Estos se explicarán en la Sección 24.5.9.

stringr	Tarea	Descripción	Base R
<code>str_detect</code>	Detectar	¿El patrón está en la cadena?	<code>grep1</code>
<code>str_which</code>	Detectar	Devuelve el índice de entradas que contienen el patrón.	<code>grep</code>
<code>str_subset</code>	Detectar	Devuelve el subconjunto de cadenas que contienen el patrón.	<code>grep</code> con <code>value = TRUE</code>
<code>str_locate</code>	Localizar	Devuelve las posiciones de la primera aparición del patrón en una cadena.	<code>regexpr</code>
<code>str_locate_all</code>	Localizar	Devuelve la posición de todas las apariciones del patrón en una cadena.	<code>gregexpr</code>
<code>str_view</code>	Localizar	Muestra la primera parte de la cadena que corresponde con el patrón.	
<code>str_view_all</code>	Localizar	Muestra todas las partes de la cadena que corresponden con el patrón.	
<code>str_extract</code>	Extraer	Extrae la primera parte de la cadena que corresponde con el patrón.	
<code>str_extract_all</code>	Extraer	Extrae todas las partes de la cadena que corresponden con el patrón.	
<code>str_match</code>	Extraer	Extrae la primera parte de la cadena que corresponde con los grupos y los patrones definidos por los grupos.	
<code>str_match_all</code>	Extraer	Extrae todas las partes de la cadena que corresponden con los grupos y los patrones definidos por los grupos.	
<code>str_sub</code>	Extraer	Extrae una subcadena.	<code>substring</code>
<code>str_split</code>	Extraer	Divide una cadena en una lista con partes separadas por patrón.	<code>strsplit</code>
<code>str_split_fixed</code>	Extraer	Divide una cadena en una matriz con partes separadas por patrón.	<code>strsplit</code> con <code>fixed = TRUE</code>
<code>str_count</code>	Describir	Cuenta el número de veces que aparece un patrón en una cadena.	
<code>str_length</code>	Describir	Número de caracteres en la cadena.	<code>nchar</code>
<code>str_replace</code>	Reemplazar	Reemplaza la primera parte de una cadena que corresponde con un patrón con otro patrón.	

stringr	Tarea	Descripción	Base R
<code>str_replace_all</code>	Reemplazar	Reemplaza todas las partes de una cadena que corresponden con un patrón con otro patrón.	<code>gsub</code>
<code>str_to_upper</code>	Reemplazar	Cambia todos los caracteres a mayúsculas.	<code>toupper</code>
<code>str_to_lower</code>	Reemplazar	Cambia todos los caracteres a minúsculas.	<code>tolower</code>
<code>str_to_title</code>	Reemplazar	Cambia el primer carácter a mayúscula y el resto a minúscula.	
<code>str_replace_na</code>	Reemplazar	Reemplaza todo los NAs con un nuevo valor.	
<code>str_trim</code>	Reemplazar	Elimina el espacio en blanco del inicio y del final de la cadena.	
<code>str_c</code>	Manipular	Une múltiples cadenas.	<code>paste0</code>
<code>str_conv</code>	Manipular	Cambia la codificación de la cadena.	
<code>str_sort</code>	Manipular	Pone el vector en orden alfabético.	<code>sort</code>
<code>str_order</code>	Manipular	Índice necesario para ordenar el vector alfabéticamente.	<code>order</code>
<code>str_trunc</code>	Manipular	Trunca una cadena a un tamaño fijo.	
<code>str_pad</code>	Manipular	Agrega espacio en blanco a la cadena para que tenga un tamaño fijo.	
<code>str_dup</code>	Manipular	Repite una cadena.	<code>rep</code> luego <code>paste</code>
<code>str_wrap</code>	Manipular	Pone la cadena en párrafos formateados.	
<code>str_interp</code>	Manipular	Interpolación de cadenas.	<code>sprintf</code>

## 24.2 Estudio de caso 1: datos de asesinatos en EE. UU.

En esta sección presentamos algunos de los desafíos más sencillos de procesamiento de cadenas, con los siguientes sets de datos como ejemplo:

```
library(rvest)
url <- paste0("https://en.wikipedia.org/w/index.php?title=",
 "Gun_violence_in_the_United_States_by_state",
 "&direction=prev&oldid=810166167")
murders_raw <- read_html(url) %>%
 html_node("table") %>%
 html_table() %>%
 setNames(c("state", "population", "total", "murder_rate"))
```

El código anterior muestra el primer paso para construir el set de datos,

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

de los datos sin procesar, que se trajeron de una página de Wikipedia.

En general, el procesamiento de cadenas implica una cadena y un patrón. En R, generalmente almacenamos cadenas en un vector de caracteres como `murders$population`. Las primeras tres cadenas en este vector definidas por la variable de población son:

```
murders_raw$population[1:3]
#> [1] "4,853,875" "737,709" "6,817,565"
```

Forzar una conversión usando `as.numeric` no funciona aquí:

```
as.numeric(murders_raw$population[1:3])
#> Warning: NAs introducidos por coerción
#> [1] NA NA NA
```

Esto se debe a las comas ,. El procesamiento de cadenas que queremos hacer aquí es eliminar el **patrón** „,” de las **cadenas** en `murders_raw$population` y luego forzar una conversión de los números.

Podemos usar la función `str_detect` para ver que dos de las tres columnas tienen comas en las entradas:

```
commas <- function(x) any(str_detect(x, ","))
murders_raw %>% summarize_all(commas)
#> # A tibble: 1 x 4
#> state population total murder_rate
#> <lgl> <lgl> <lgl> <lgl>
#> 1 FALSE TRUE TRUE FALSE
```

Entonces podemos usar la función `str_replace_all` para eliminarlas:

```
test_1 <- str_replace_all(murders_raw$population, ",", "")
test_1 <- as.numeric(test_1)
```

De ahí podemos usar `mutate_all` para aplicar esta operación a cada columna, ya que no afectará a las columnas sin comas.

Resulta que esta operación es tan común que `readr` incluye la función `parse_number` específicamente para eliminar caracteres no numéricos antes de forzar una conversión:

```
test_2 <- parse_number(murders_raw$population)
identical(test_1, test_2)
#> [1] TRUE
```

Entonces podemos obtener nuestra tabla deseada usando:

```
murders_new <- murders_raw %>% mutate_at(2:3, parse_number)
head(murders_new)
#> # A tibble: 6 x 4
#> state population total murder_rate
#> <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
#> # ... with 1 more row
```

Este caso es relativamente sencillo en comparación con los desafíos de procesamiento de cadenas que normalmente enfrentamos en la ciencia de datos. El siguiente ejemplo es bastante complejo y ofrece varios retos que nos permitirán aprender muchas técnicas de procesamiento de cadenas.

### 24.3 Estudio de caso 2: alturas autoreportadas

El paquete **dslabs** incluye el set de datos sin procesar del cual se obtuvo el set de datos de alturas. Pueden cargarlo así:

```
data(reported_heights)
```

Estas alturas se obtuvieron mediante un formulario web en el que se les pidió a estudiantes que ingresaran sus alturas. Podían ingresar cualquier cosa, pero las instrucciones pedían *altura en pulgadas*, un número. Recopilamos 1,095 respuestas, pero desafortunadamente el vector de columna con las alturas reportadas tenía varias entradas no numéricas y como resultado se convirtió en un vector de caracteres:

```
class(reported_heights$height)
#> [1] "character"
```

Si intentamos analizarlo en números, recibimos una advertencia:

```
x <- as.numeric(reported_heights$height)
#> Warning: NAs introducidos por coerción
```

Aunque la mayoría de los valores parecen ser la altura en pulgadas según lo solicitado:

```
head(x)
#> [1] 75 70 68 74 61 65
```

terminamos con muchos NAs:

```
sum(is.na(x))
#> [1] 81
```

Podemos ver algunas de las entradas que no se convierten correctamente utilizando `filter` para mantener solo las entradas que resultan en NAs:

```
reported_heights %>%
 mutate(new_height = as.numeric(height)) %>%
 filter(is.na(new_height)) %>%
 head(n=10)

#> 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
```

Inmediatamente vemos lo que está sucediendo. Algunos de los estudiantes no reportaron sus alturas en pulgadas según lo solicitado. Podríamos descartar estos datos y continuar. Sin embargo, muchas de las entradas siguen patrones que, en principio, podemos convertir fácilmente a pulgadas. Por ejemplo, en el resultado anterior, vemos varios casos que usan el formato `x'y''` con `x` e `y` representando pies y pulgadas, respectivamente. Cada uno de estos casos puede ser leído y convertido a pulgadas por un humano, por ejemplo `5'4''` es  $5 \times 12 + 4 = 64$ . Entonces podríamos arreglar todas las entradas problemáticas *a mano*. Sin embargo, los humanos son propensos a cometer errores, por lo que es preferible un enfoque automatizado. Además, debido a que planeamos continuar recolectando datos, será conveniente escribir código que lo haga automáticamente.

Un primer paso en este tipo de tarea es examinar las entradas problemáticas e intentar definir patrones específicos seguidos por un gran grupo de entradas. Cuanto más grandes sean estos grupos, más entradas podremos arreglar con un solo enfoque programático. Queremos encontrar patrones que puedan describirse con precisión con una regla, como “un dígito, seguido por un símbolo de pie, seguido por uno o dos dígitos, seguido por un símbolo de pulgadas”.

Para buscar dichos patrones, es útil eliminar las entradas que son consistentes con estar en pulgadas y ver solo las entradas problemáticas. Por lo tanto, escribimos una función para hacer esto automáticamente. Mantenemos entradas que resultan en NAs al aplicar `as.numeric` o que están fuera de un rango de alturas plausibles. Permitimos un rango que cubre aproximadamente el 99.9999% de la población adulta. También usamos `suppressWarnings` para evitar el mensaje de advertencia que sabemos que `as.numeric` nos da.

```
not_inches <- function(x, smallest = 50, tallest = 84){
 inches <- suppressWarnings(as.numeric(x))
 ind <- is.na(inches) | inches < smallest | inches > tallest
 ind
}
```

Aplicamos esta función y encontramos el número de entradas problemáticas:

```
problems <- reported_heights %>%
 filter(not_inches(height)) %>%
 pull(height)
length(problems)
#> [1] 292
```

Ahora podemos ver todos los casos simplemente imprimiéndolos. No hacemos eso aquí porque hay `length(problems)`, pero después de examinarlos cuidadosamente, vemos que podemos usar tres patrones para definir tres grandes grupos dentro de estas excepciones.

1. Un patrón de la forma `x'y` o `x' y'` o `x'y"` con `x` e `y` representando pies y pulgadas, respectivamente. Aquí hay diez ejemplos:

```
#> 5' 4" 5'7 5'7" 5'3" 5'11 5'9" 5'10" 5' 10 5'5" 5'2"
```

2. Un patrón de la forma `x.y` o `x,y` con `x` pies y `y` pulgadas. Aquí hay diez ejemplos:

```
#> 5.3 5.5 6.5 5.8 5.6 5,3 5.9 6,8 5.5 6.2
```

3. Entradas en centímetros en vez de pulgadas. Aquí hay diez ejemplos:

```
#> 150 175 177 178 163 175 178 165 165 180
```

Una vez que veamos que estos grupos grandes siguen patrones específicos, podemos desarrollar un plan de ataque. Recuerden que raras veces hay una sola forma de realizar estas tareas. Aquí elegimos una que nos ayuda a enseñar varias técnicas útiles, aunque seguramente hay una forma más eficiente de realizar la tarea.

**Plan de ataque:** convertiremos las entradas que siguen a los dos primeros patrones en una estandarizada. Luego, aprovecharemos la estandarización para extraer los pies y pulgadas y convertirlos a pulgadas. Entonces, definiremos un procedimiento para identificar entradas que están en centímetros y convertirlas a pulgadas. Después de aplicar estos pasos, verificaremos nuevamente para ver qué entradas no se corrigieron y ver si podemos ajustar nuestro enfoque para que sea más completo.

Al final, esperamos tener un *script* que haga que los métodos de recolección de datos basados en la web sean robustos para los errores más comunes de los usuarios.

Para lograr nuestro objetivo, utilizaremos una técnica que nos permite detectar con precisión patrones y extraer las partes que queremos: expresiones regulares. Pero primero, describimos rápidamente cómo *escaparse* (“escapar” viene de la tecla *esc* o *escape*) de la función de ciertos caracteres para que se puedan incluir en cadenas.

## 24.4 Cómo escapar al definir cadenas

Para definir cadenas en R, podemos usar comillas dobles:

```
s <- "Hello!"
```

o comillas simples:

```
s <- 'Hello!'
```

Asegúrense de elegir la comilla simple correcta, ya que usar la comilla inversa le dará un error:

```
s <- `Hello`
```

```
Error: object 'Hello' not found
```

Ahora, ¿qué sucede si la cadena que queremos definir incluye comillas dobles? Por ejemplo, si queremos escribir 10 pulgadas así: 10"? En este caso no pueden usar:

```
s <- "10""
```

porque esto es solo la cadena 10 seguido por una comilla doble. Si escriben esto en R, obtendrán un error porque tiene comillas doble sin cerrar. Para evitar esto, podemos usar las comillas simples:

```
s <- '10"'
```

Si imprimimos s, vemos que escapamos las comillas dobles con la barra diagonal inversa \.

```
s
#> [1] "10\""
```

De hecho, escapar con la barra diagonal inversa ofrece una forma de definir la cadena y a la vez seguir utilizando las comillas dobles para definir cadenas:

```
s <- "10\\\""
```

En R, la función cat nos permite ver como se ve la cadena:

```
cat(s)
#> 10"
```

Ahora, ¿qué pasa si queremos que nuestra cadena sea de 5 pies escrita así: 5'? En ese caso, podemos usar las comillas dobles:

```
s <- "5"
cat(s)
#> 5'
```

Así que hemos aprendido a escribir 5 pies y 10 pulgadas por separado, pero ¿qué pasa si queremos escribirlos juntos para representar 5 pies y 10 pulgadas así: 5'10"? En este caso, ni las comillas simples ni las dobles funcionarán. Esto:

```
s <- '5'10'"
```

cierra la cadena después de 5 y esto:

```
s <- "5'10""
```

cierra la cadena después de 10. Tengan en cuenta que si escribimos uno de los fragmentos de código anteriores en R, se atascará esperando que cierren la comilla abierta y tendrán que salir de la ejecución con la tecla *esc*.

En esta situación, necesitamos escapar de la función de las comillas con la barra diagonal inversa \. Pueden escapar cualquiera de los dos de la siguiente manera:

```
s <- '5\'10"'
cat(s)
#> 5'10"
```

o así:

```
s <- "5'10\\\""
cat(s)
#> 5'10"
```

Los caracteres de escape son algo que a menudo tenemos que usar al procesar cadenas.

---

## 24.5 Expresiones regulares

Las expresiones regulares (*regex*) son una forma de describir patrones específicos de caracteres de texto. Se pueden usar para determinar si una cadena dada corresponde con el patrón. Se ha definido un conjunto de reglas para hacer esto de manera eficiente y precisa y aquí mostramos algunos ejemplos. Podemos aprender más sobre estas reglas leyendo tutoriales detallados<sup>1</sup> <sup>2</sup>. La hoja de referencia de RStudio también es muy útil<sup>3</sup>.

Los patrones suministrados a las funciones **stringr** pueden ser una expresión regular en lugar de una cadena estándar. Aprenderemos cómo funciona esto a través de una serie de ejemplos.

A lo largo de esta sección, verán que creamos cadenas para probar nuestra expresión regular. Para hacer esto, definimos patrones que sabemos deben corresponder y también patrones que sabemos no deben corresponder. Los llamaremos **yes** y **no**, respectivamente. Esto nos permite verificar los dos tipos de errores: no corresponder o corresponder incorrectamente.

<sup>1</sup><https://www.regular-expressions.info/tutorial.html>

<sup>2</sup><http://r4ds.had.co.nz/strings.html#matching-patrones-con-expresiones-regulares>

<sup>3</sup>[https://github.com/rstudio/cheatsheets/raw/master/translations/spanish/strings\\_Spanish.pdf](https://github.com/rstudio/cheatsheets/raw/master/translations/spanish/strings_Spanish.pdf)

### 24.5.1 Las cadenas son expresiones regulares

Técnicamente, cualquier cadena es una expresión regular, quizás el ejemplo más simple es un solo carácter. Entonces la coma , usada en el siguiente ejemplo de código, es un ejemplo sencillo de una búsqueda con expresiones regulares.

```
pattern <- ","
str_detect(murders_raw$total, pattern)
```

Suprimimos el *output* que es un vector lógico que nos dice qué entradas tienen comas.

Arriba, notamos que una entrada incluía un cm. Este es también un ejemplo simple de una expresión regular. Podemos mostrar todas las entradas que usaron cm así:

```
str_subset(reported_heights$height, "cm")
#> [1] "165cm" "170 cm"
```

### 24.5.2 Caracteres especiales

Ahora consideremos un ejemplo un poco más complicado. ¿Cuál de las siguientes cadenas contiene el patrón cm o inches?

```
yes <- c("180 cm", "70 inches")
no <- c("180", "70")
s <- c(yes, no)

str_detect(s, "cm") | str_detect(s, "inches")
#> [1] TRUE TRUE FALSE FALSE
```

Sin embargo, no necesitamos hacer esto. La característica principal que distingue el lenguaje de las expresiones regulares de cadenas es que podemos usar caracteres especiales. Estos son caracteres con un significado. Comenzamos presentando | que significa o. Entonces, si queremos saber si cm o inches aparece en las cadenas, podemos usar la expresión regular cm|inches:

```
str_detect(s, "cm|inches")
#> [1] TRUE TRUE FALSE FALSE
```

y obtener la respuesta correcta.

Otro carácter especial que será útil para identificar valores de pies y pulgadas es \d que significa cualquier dígito: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. La barra inversa se utiliza para distinguirlo del carácter d. En R, tenemos que escapar de la barra inversa \ así que tenemos que usar \\d para representar dígitos. Aquí hay un ejemplo:

```
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
```

Aprovechamos esta oportunidad para presentar la función `str_view`, que es útil para la solución de problemas ya que nos muestra la primera ocurrencia que corresponde exactamente para cada cadena:

```
str_view(s, pattern)
```

```
5
6
5'10
5 feet
4'11
.
Five
six
```

y `str_view_all` nos muestra todos los patrones que corresponden. Entonces, `3'2` tiene dos equivalencias y `5'10` tiene tres.

```
str_view_all(s, pattern)
```

```
5
6
5'10
5 feet
4'11
.
Five
six
```

Hay muchos otros caracteres especiales. Aprenderemos algunos mas a continuación, pero pueden ver la mayoría o todos en la hoja de referencia mencionada anteriormente<sup>4</sup>.

### 24.5.3 Clases de caracteres

Las clases de caracteres se utilizan para definir una serie de caracteres que pueden corresponder. Definimos clases de caracteres entre corchetes `[ ]`. Entonces, por ejemplo, si queremos que el patrón corresponda solo si tenemos un 5 o un 6, usamos la expresión regular `[56]`:

```
str_view(s, "[56]")
```

<sup>4</sup>[https://github.com/rstudio/cheatsheets/raw/master/translations/spanish/strings\\_Spanish.pdf](https://github.com/rstudio/cheatsheets/raw/master/translations/spanish/strings_Spanish.pdf)

```

5
6
5'10
5 feet
4'11
.
Five
six

```

Supongan que queremos unir valores entre 4 y 7. Una forma común de definir clases de caracteres es con rangos. Así por ejemplo, [0-9] es equivalente a \d. El patrón que queremos entonces es [4-7].

```

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

```

Sin embargo, es importante saber que en *regex* todo es un carácter; no hay números. Por eso 4 es el carácter 4 y no el número cuatro. Noten, por ejemplo, que [1-20] **no** significa 1 a 20, significa los caracteres 1 a 2 o el carácter 0. Entonces [1-20] simplemente significa la clase de caracteres compuesta por 0, 1 y 2.

Tengan en cuenta que los caracteres tienen un orden y los dígitos siguen el orden numérico. Entonces 0 viene antes de 1 que viene antes de 2 y así. Por la misma razón, podemos definir letras minúsculas como [a-z], letras mayúsculas como [A-Z] y [a-zA-Z] como ambas.

#### 24.5.4 Anclas

¿Y si queremos una correspondencia cuando tenemos exactamente 1 dígito? Esto será útil en nuestro estudio de caso, ya que los pies nunca tienen más de 1 dígito, por lo que esta restricción nos ayudará. Una forma de hacer esto con *regex* es usando *anclas* (*anchors* en inglés), que nos permiten definir patrones que deben comenzar o terminar en un lugar específico. Las dos anclas más comunes son ^ y \$ que representan el comienzo y el final de una cadena, respectivamente. Entonces el patrón ^\d\$ se lee como “inicio de la cadena seguido por un dígito seguido por el final de la cadena”.

Este patrón ahora solo detecta las cadenas con exactamente un dígito:

```

pattern <- "^\d$"
yes <- c("1", "5", "9")
no <- c("12", "123", " 1", "a4", "b")
s <- c(yes, no)
str_view_all(s, pattern)

```

```

1
5
9
12
123
1
a4
b

```

El 1 no corresponde porque no comienza con el dígito sino con un espacio, que no es fácil de ver.

#### 24.5.5 Cuantificadores

Para la parte de pulgadas, podemos tener uno o dos dígitos. Esto se puede especificar en *regex* con *cuantificadores* (*quantifiers* en inglés). Esto se hace siguiendo el patrón con la cantidad de veces que se puede repetir cerrada por llaves. Usemos un ejemplo para ilustrar. El patrón para uno o dos dígitos es:

```

pattern <- "^\d{1,2}$"
yes <- c("1", "5", "9", "12")
no <- c("123", "a4", "b")
str_view(c(yes, no), pattern)

```

```

1
5
9
12
123
a4
b

```

En este caso, 123 **no** corresponde, pero 12 sí. Entonces, para buscar nuestro patrón de pies y pulgadas, podemos agregar los símbolos para pies ' y pulgadas " después de los dígitos.

Con lo que hemos aprendido, ahora podemos construir un ejemplo para el patrón x'y\" con x pies y y pulgadas.

```

pattern <- "^[4-7] '\d{1,2}\"$"

```

El patrón ahora se está volviendo complejo, pero pueden mirarlo cuidadosamente y desglosarlo:

- ^ = inicio de la cadena
- [4-7] = un dígito, ya sea 4, 5, 6 o 7

- ' = símbolo de pies
- \d{1,2} = uno o dos dígitos
- " = símbolo de pulgadas
- \$ = 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
```

Por ahora, estamos permitiendo que las pulgadas sean 12 o más grandes. Agregaremos una restricción más tarde ya que la expresión regular para esto es un poco más compleja de lo que estamos preparados para mostrar.

#### 24.5.6 Espacio en blanco \s

Otro problema que tenemos son los espacios. Por ejemplo, nuestro patrón no corresponde con 5' 4" porque hay un espacio entre ' y 4, que nuestro patrón no permite. Los espacios son caracteres y R no los ignora:

```
identical("Hi", "Hi ")
#> [1] FALSE
```

En *regex*, \s representa el espacio en blanco. Para encontrar patrones como 5' 4, podemos cambiar nuestro patrón a:

```
pattern_2 <- "^[4-7]'\\s\\d{1,2}\"$"
str_subset(problems, pattern_2)
#> [1] "5' 4\"" "5' 11\"" "5' 7\""
```

Sin embargo, esto no encontrará equivalencia con los patrones sin espacio. Entonces, ¿necesitamos más de un patrón *regex*? Resulta que también podemos usar un cuantificador para esto.

#### 24.5.7 Cuantificadores: \*, ?, +

Queremos que el patrón permita espacios pero que no los requiera. Incluso si hay varios espacios, como en este ejemplo 5' 4, todavía queremos que corresponda. Hay un cuantificador para exactamente este propósito. En *regex*, el carácter \* significa cero o más instancias del carácter anterior. Aquí hay un 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
```

El patrón anterior encuentra correspondencia con la primera cadena, que tiene cero 1s, y todas las cadenas con un o más 1. Entonces podemos mejorar nuestro patrón agregando el \* después del carácter espacial \s.

Hay otros dos cuantificadores similares. Para ninguno o una vez, podemos usar ?, y para uno o más, podemos usar +. Pueden ver cómo difieren con este ejemplo:

```
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
```

Usaremos los tres en nuestro ejemplo de alturas reportadas, pero los veremos en una sección posterior.

#### 24.5.8 Todo excepto

Para especificar patrones que **no** queremos detectar, podemos usar el símbolo ^ pero solo **dentro** de corchetes. Recuerden que fuera del corchete ^ significa el inicio de la cadena. Entonces, por ejemplo, si queremos detectar dígitos precedidos por algo que no sea una letra, podemos hacer lo siguiente:

```
pattern <- "[^a-zA-Z]\d"
yes <- c(".3", "+2", "-0", "*4")
no <- c("A3", "B2", "C0", "E4")
str_detect(yes, pattern)
#> [1] TRUE TRUE TRUE TRUE
str_detect(no, pattern)
#> [1] FALSE FALSE FALSE FALSE
```

Otra forma de generar un patrón que busca *todo excepto* es usar la mayúscula del carácter especial. Por ejemplo \\D significa cualquier cosa que no sea un dígito, \\S significa cualquier cosa excepto un espacio y así sucesivamente.

### 24.5.9 Grupos

Los *grupos* son un elemento muy útil de las expresiones regulares que permiten la extracción de valores. Los grupos se definen usando paréntesis. No afectan las equivalencias de patrones. En cambio, permiten que las herramientas identifiquen partes específicas del patrón para que podamos extraerlas.

Queremos cambiar las alturas escritas como 5.6 a 5'6.

Para evitar cambiar patrones como 70.2, requeriremos que el primer dígito esté entre 4 y 7 [4-7] y que el segundo sea ninguno o uno o más dígitos \\d\*. Comencemos definiendo un patrón sencillo que corresponda con lo siguiente:

```
pattern_without_groups <- "^[4-7],\\d*$"
```

Queremos extraer los dígitos para poder formar la nueva versión usando un punto. Estos son nuestros dos grupos, por lo que los encapsulamos con paréntesis:

```
pattern_with_groups <- "^([4-7]),(\\d*)$"
```

Encapsulamos la parte del patrón que corresponde con las partes que queremos conservar para su uso posterior. Agregar grupos no afecta la detección, ya que solo indica que queremos guardar lo que capturan los grupos. Tengan en cuenta que ambos patrones devuelven el mismo resultado cuando se usa `str_detect`:

```
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 FALSE FALSE FALSE FALSE
str_detect(s, pattern_with_groups)
#> [1] TRUE TRUE TRUE TRUE FALSE FALSE FALSE FALSE
```

Una vez que hayamos definido los grupos, podemos usar la función `str_match` para extraer los valores que definen estos grupos:

```
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
```

Observen que las segunda y tercera columnas contienen pies y pulgadas, respectivamente. La primera columna es la parte de la cadena que corresponde con el patrón. Si no se encuentra una equivalencia, vemos un NA.

Ahora podemos entender la diferencia entre las funciones `str_extract` y `str_match`. `str_extract` extrae solo cadenas que corresponden con un patrón, no los valores definidos por grupos:

```
str_extract(s, pattern_with_groups)
#> [1] "5,9" "5,11" "6," "6,1" NA NA NA
```

## 24.6 Buscar y reemplazar con expresiones regulares

Anteriormente definimos el objeto `problems` que contiene las cadenas que no parecen estar en pulgadas. Podemos ver que pocas de nuestras cadenas problemáticas corresponden con el patrón:

```
pattern <- "[4-7][1,2]""
sum(str_detect(problems, pattern))
#> [1] 14
```

Para ver por qué esto es así, mostramos algunos ejemplos que ilustran por qué no tenemos más equivalencias:

```
problems[c(2, 10, 11, 12, 15)] %>% str_view(pattern)
```

```
5' 4"
5'7"
5'3"
5 feet and 8.11 inches
5.5
```

Un problema inicial que vemos de inmediato es que algunos estudiantes escribieron las palabras “feet” e “inches”. Podemos ver las entradas que hicieron esto con la función `str_subset`:

```
str_subset(problems, "inches")
#> [1] "5 feet and 8.11 inches" "Five foot eight inches"
#> [3] "5 feet 7inches" "5ft 9 inches"
#> [5] "5 ft 9 inches" "5 feet 6 inches"
```

También vemos que algunas entradas usan dos comillas simples ‘’ en lugar de una comilla doble ‘“’.

```
str_subset(problems, "''")
#> [1] "5'9''" "5'10''" "5'10''" "5'3''" "5'7''" "5'6''"
#> [7] "5'7.5''" "5'7.5''" "5'10''" "5'11''" "5'10''" "5'5''"
```

```
pattern <- "[4-7] '\d{1,2}"
```

Si hacemos este reemplazo antes de buscar equivalencias, obtendremos muchas más equivalencias:

```
problems %>%
 str_replace("feet|ft|foot", "") %>% # replace feet, ft, foot with '
 str_replace("inches|in|''|\\"", "") %>% # remove all inches symbols
 str_detect(pattern) %>%
 sum()
#> [1] 48
```

Sin embargo, todavía nos faltan muchos casos.

Noten que en el código anterior, aprovechamos la consistencia de **stringr** y usamos el *pipe*.

Por ahora, mejoramos nuestro patrón agregando `\s*` delante y después del símbolo de los pies ' para permitir espacio entre el símbolo de los pies y los números. Ahora encontraremos más equivalencias:

```
pattern <- "[4-7]\s*\s*\d{1,2}"
problems %>%
 str_replace("feet|ft|foot", "") %>% # replace feet, ft, foot with '
 str_replace("inches|in|''|\\"", "") %>% # remove all inches symbols
 str_detect(pattern) %>%
 sum()
#> [1] 53
```

Podríamos estar tentados a evitar esto eliminando todos los espacios con **str\_replace\_all**. Sin embargo, al hacer una operación de este tipo, debemos asegurarnos de evitar efectos indeseados. En nuestros ejemplos de alturas autoreportadas, esto será un problema porque algunas entradas tienen la forma x y con espacio separando los pies de las pulgadas. Si eliminamos todos los espacios, cambiaríamos incorrectamente x y a xy lo que implica que un 6 1 se convertiría en 61 pulgadas en vez de 73 pulgadas.

El segundo grupo grande de entradas problemáticas tienen las formas x.y, x,y o x y. Queremos cambiar todo esto a nuestro formato común x'y. Pero no podemos simplemente buscar y reemplazar porque cambiaríamos valores como 70.5 a 70'5. Por lo tanto, nuestra estrategia será buscar un patrón muy específico que nos asegure que se devuelvan pies y pulgadas y luego, para aquellos que correspondan, se reemplazcan adecuadamente.

### 24.6.1 Buscar y reemplazar usando grupos

Otro aspecto útil de los grupos es que uno puede hacer referencia a los valores extraídos en una expresión regular cuando busca y reemplaza.

El carácter especial *regex* para el grupo en el lugar i es `\i`. Entonces `\1` es el valor extraído del primer grupo, `\2` el valor del segundo y así sucesivamente. Como un ejemplo sencillo, noten 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"
```

Podemos usar esto para convertir casos en nuestras alturas reportadas.

Ahora estamos listos para definir un patrón que nos ayudará a convertir todos los `x.y`, `x,y` y `x y` a nuestro formato preferido. Necesitamos adaptar `pattern_with_groups` para que sea un poco más flexible y capture todos los casos.

```
pattern_with_groups <-"^([4-7])\\s*,\\s+\\.\\s+\\s*(\\d*)$"
```

Examinemos este patrón *regex*:

- `^` = inicio de la cadena
- `[4-7]` = un dígito, ya sea 4, 5, 6 o 7
- `\\s*` = ninguno o más espacios en blanco
- `[,\\.\\s+]` = el símbolo de pies es `,`, `.` o al menos un espacio
- `\\s*` = ninguno o más espacios en blanco
- `\\d*` = ninguno o más dígitos
- `$` = final de la cadena

Podemos ver que parece estar funcionando:

```
str_subset(problems, pattern_with_groups) %>% head()
#> [1] "5.3" "5.25" "5.5" "6.5" "5.8" "5.6"
```

Ahora usamos esto para buscar y reemplazar:

```
str_subset(problems, pattern_with_groups) %>%
 str_replace(pattern_with_groups, "\\\1'\\\\2") %>% head
#> [1] "5'3" "5'25" "5'5" "6'5" "5'8" "5'6"
```

Lidiaremos con el desafío de “pulgadas más grandes que doce” más tarde.

## 24.7 Probar y mejorar

Desarrollar expresiones regulares correctas en el primer intento a menudo es difícil. Probar y mejorar es un enfoque común para encontrar el patrón de expresiones regulares que satisface todas las condiciones deseadas. En las secciones anteriores, hemos desarrollado una potente técnica de procesamiento de cadenas que puede ayudarnos a detectar muchas de las entradas problemáticas. Aquí probaremos nuestro enfoque, buscaremos más problemas

y modificaremos nuestro enfoque para posibles mejoras. Escribamos una función que identifique todas las entradas que no se pueden convertir en números, recordando que algunas están en centímetros (nos encargaremos de arreglar esas más adelante):

```
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)) %>%
 pull(height)
length(problems)
#> [1] 200
```

Veamos qué proporción de estas se ajusta a nuestro patrón después de los pasos de procesamiento que desarrollamos anteriormente:

```
converted <- problems %>%
 str_replace("feet|foot|ft", "") %>% # convert feet symbols to '
 str_replace("inches|in| ''|\"", "") %>% # remove inches symbols
 str_replace("^(\\d{1,2})\\s*,\\s*(\\d{1,2})$", "\\1'\\2")# change format

pattern <- "^(\\d{1,2})\\s*,\\s*(\\d{1,2})$"
index <- str_detect(converted, pattern)
mean(index)
#> [1] 0.615
```

Observen cómo aprovechamos el *pipe*, una de las ventajas de usar **stringr**. Este último fragmento de código muestra que encontramos equivalencias en más de la mitad de las cadenas. Examinemos los casos restantes:

converted[!index]			
#> [1] "6"	"165cm"	"511"	"6"
#> [5] "2"	"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"
```

Surgen cuatro patrones claros:

1. Muchos estudiantes que miden exactamente 5 o 6 pies no ingresaron ninguna pulgada, por ejemplo 6', y nuestro patrón requiere que se incluyan pulgadas.
2. Algunos estudiantes que miden exactamente 5 o 6 pies ingresaron solo ese número.
3. Algunas de las pulgadas se ingresaron con puntos decimales. Por ejemplo 5'7.5''. Nuestro patrón solo busca dos dígitos.
4. Algunas entradas tienen espacios al final, por ejemplo 5' 9'.

Aunque no es tan común, también vemos los siguientes problemas:

5. Algunas entradas están en metros y algunas usan formatos europeos: 1.6, 1,70.
6. Dos estudiantes añadieron cm.
7. Un estudiante deletreó los números: Five foot eight inches.

No está claro que valga la pena escribir código para manejar estos últimos tres casos, ya que son bastante raros. Sin embargo, algunos de ellos nos brindan la oportunidad de aprender algunas técnicas más de expresiones regulares, por lo que crearemos una solución.

Para el caso 1, si agregamos un '0 después del primer dígito, por ejemplo, convertir todo 6 a 6'0, entonces nuestro patrón previamente definido corresponderá. Esto se puede hacer usando grupos:

```
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"
```

El patrón dice que tiene que comenzar (^) con un dígito entre 4 y 7 y terminar ahí (\$). El paréntesis define el grupo que pasamos como \\\1 para generar la cadena *regex* que usamos para el reemplazo.

Podemos adaptar este código ligeramente para manejar también el caso 2, que cubre la entrada 5'. Noten que 5' se deja intacto. Esto es porque el extra ' hace que el patrón no corresponda ya que tenemos que terminar con un 5 o 6. Queremos permitir que el 5 o 6 sea seguido por un signo de 0 o 1 pie. Entonces podemos simplemente agregar '{0,1}' después de la ' para hacer esto. Sin embargo, podemos usar el carácter especial, ?, que significa ninguna o una vez. Como vimos anteriormente, esto es diferente a \*, que significa ninguna o más. Ahora vemos que el cuarto caso también se convierte:

```
str_replace(s, "^(56)??", "\\\1'0")
#> [1] "5'0" "6'0" "5'0" "5'0" "5'" "5'4"
```

Aquí solo permitimos 5 y 6, pero no 4 y 7. Esto se debe a que medir 5 y 6 pies de altura es bastante común, por lo que suponemos que aquellos que escribieron 5 o 6 realmente querían decir 60 o 72 pulgadas. Sin embargo, medir 4 y 7 pies de altura es tan raro que, aunque aceptamos 84 como una entrada válida, suponemos que 7 fue ingresado por error.

Podemos usar cuantificadores para tratar el caso 3. Estas entradas no corresponden porque las pulgadas incluyen decimales y nuestro patrón no lo permite. Necesitamos permitir que el segundo grupo incluya decimales, no solo dígitos. Esto significa que debemos permitir cero o un punto ., entonces cero o más dígitos. Por eso, usaremos ambos ? y \*. También recuerden que, para este caso particular, el punto debe escaparse ya que es un carácter especial (significa cualquier carácter excepto el salto de línea). Aquí hay un ejemplo sencillo de cómo podemos usar \*.

Entonces podemos adaptar nuestro patrón, que actualmente es `^([4-7] \\s*')\\s*\\d{1,2}$` para permitir un decimal al final:

```
pattern <- "^(4-7) \\s*')\\s*(\\d+\\.?\\d*)$"
```

Podemos tratar el caso 4, metros usando comas, de manera similar a cómo convertimos el x.y a 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"
```

Luego verificaremos si las entradas son metros utilizando sus valores numéricos. Regresaremos al estudio de caso después de presentar dos funciones ampliamente utilizadas en el procesamiento de cadenas que serán útiles al desarrollar nuestra solución final para las alturas autoreportadas.

## 24.8 Podar

En general, los espacios al principio o al final de la cadena no son informativos. Estos pueden ser particularmente engañosos porque a veces pueden ser difíciles de ver:

```
s <- "Hi "
cat(s)
#> Hi
identical(s, "Hi")
#> [1] FALSE
```

Este es un problema lo suficientemente general como para que haya una función dedicada a eliminarlos: `str_trim`. El nombre viene de *trim* o *podar* en inglés.

```
str_trim("5 ' 9 ")
#> [1] "5 ' 9"
```

## 24.9 Cómo cambiar de mayúsculas o minúsculas

Tengan en cuenta que *regex* distingue entre mayúsculas y minúsculas. A menudo queremos encontrar una equivalencia de palabra independientemente de si es mayúscula o minúscula. Un acercamiento para hacer esto es primero cambiar todo a minúsculas y luego continuar ignorando mayúsculas y minúsculas. Como ejemplo, noten que una de las entradas escribe números como palabras *Five foot eight inches*. Aunque no es eficiente, podríamos agregar 13 llamadas adicionales a `str_replace` para convertir *zero* a 0, *one* a 1, y así. Para no tener que escribir dos operaciones separadas para *Zero* y *zero*, *One* y *one*, etc., podemos usar la función `str_to_lower` para convertir todas las letras a minúsculas:

```
s <- c("Five feet eight inches")
str_to_lower(s)
#> [1] "five feet eight inches"
```

Otras funciones relacionadas son `str_to_upper` y `str_to_title`. Ahora estamos listos para definir un procedimiento que convierta todos los casos problemáticos a pulgadas.

## 24.10 Estudio de caso 2: alturas autoreportadas (continuación)

Ahora ponemos todo lo que hemos aprendido en una función que toma un vector de cadena e intenta convertir todas las cadenas posibles a un formato. Escribimos una función que reúne lo que hemos hecho anteriormente:

```
convert_format <- function(s){
 s %>%
 str_replace("feet|foot|ft", "") %>%
 str_replace_all("inches|in|'|\"|cm|and", "") %>%
 str_replace("^(4-7)\\s*,\\s+([\\d]*\\.)\\s*(\\d*)$", "\\\\$1\\\\\$2") %>%
 str_replace("^(56)'?\\$?", "\\\\$1'0") %>%
 str_replace("^(12)\\s*,\\s*(\\d*)\\$?", "\\\\$1\\\\.\$2") %>%
 str_trim()
}
```

También podemos escribir una función que convierta palabras en números:

```
library(english)
words_to_numbers <- function(s){
```

```
s <- str_to_lower(s)
for(i in 0:11)
 s <- str_replace_all(s, words(i), as.character(i))
s
}
```

Tengan en cuenta que podemos realizar la operación anterior de manera más eficiente con la función `recode`, que estudiaremos en la Sección 24.13. Ahora podemos ver qué entradas problemáticas permanecen:

```
converted <- problems %>% words_to_numbers() %>% convert_format()
remaining_problems <- converted[not_inches_or_cm(converted)]
pattern <- "^[4-7] \\s* \\d+\\.?\\d*$"
index <- str_detect(remaining_problems, pattern)
remaining_problems[!index]
#> [1] "511" "2" ">9000" "11111" "103.2"
#> [6] "19" "300" "7" "214" "0.7"
#> [11] "2'33" "612" "1.70" "87" "111"
#> [16] "12" "yyy" "89" "34" "25"
#> [21] "22" "684" "1" "1" "6*12"
#> [26] "87" "1.6" "120" "120" "23"
#> [31] "1.7" "86" "708,661" "649,606" "10000"
#> [36] "1" "728,346" "0" "100" "88"
#> [41] "7,283,465" "34"
```

Además de los casos ingresados como metros, que solucionaremos a continuación, todos parecen ser casos imposibles de resolver.

#### 24.10.1 La función `extract`

La función `extract` de `tidyverse` es útil para el procesamiento de cadenas que usaremos en nuestra solución final y, por ende, la presentamos aquí. En una sección anterior, construimos una expresión regular que nos permite identificar qué elementos de un vector de caracteres corresponden con el patrón de pies y pulgadas. Sin embargo, queremos hacer más. Queremos extraer y guardar los pies y los valores numéricos para poder convertirlos en pulgadas cuando sea apropiado.

Consideremos este ejemplo sencillo:

```
s <- c("5'10", "6'1")
tab <- data.frame(x = s)
```

En la Sección 21.3, aprendimos sobre la función `separate`, que se puede utilizar para lograr nuestro objetivo actual:

```
tab %>% separate(x, c("feet", "inches"), sep = "'")
#> feet inches
#> 1 5 10
#> 2 6 1
```

La función `extract` del paquete **tidyR** nos permite usar grupos *regex* para extraer los valores deseados. Aquí está el equivalente al código anterior que usa `separate`, pero ahora usando `extract`:

```
library(tidyR)
tab %>% extract(x, c("feet", "inches"), regex = "(\\d)'(\\d{1,2})")
#> feet inches
#> 1 5 10
#> 2 6 1
```

Entonces, ¿por qué necesitamos la nueva función `extract`? Hemos visto cómo pequeños cambios pueden conducir a no tener equivalencias exactas. Los grupos en *regex* nos dan más flexibilidad. Por ejemplo, si definimos:

```
s <- c("5'10", "6'1\"", "5'8inches")
tab <- data.frame(x = s)
```

y solo queremos los números, `separate` no lo logra:

```
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`. La *regex* aquí es un poco más complicada ya que tenemos que permitir ' con espacios y `feet`. Tampoco queremos el " incluido en el valor, por lo que no lo incluimos en el grupo:

```
tab %>% extract(x, c("feet", "inches"), regex = "(\\d)'(\\d{1,2})")
#> feet inches
#> 1 5 10
#> 2 6 1
#> 3 5 8
```

### 24.10.2 Juntando todas la piezas

Ahora estamos listos para juntar todas las piezas y discutir nuestros datos de alturas reportados para tratar de recuperar todas las alturas posibles. El código es complejo, pero lo dividiremos en partes.

Comenzamos limpiando la columna `height` para que las alturas estén más cerca de un formato de pulgadas y pies. Agregamos una columna con las alturas originales para luego poder comparar.

Ahora estamos listos para *wrangle* nuestro set de datos de alturas reportadas:

```
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) ~ as.numeric(NA),
 between(height, smallest, tallest) ~ height, #inches
 between(height/2.54, smallest, tallest) ~ height/2.54, #cm
 between(height*100/2.54, smallest, tallest) ~ height*100/2.54, #meters
 TRUE ~ as.numeric(NA))) %>%
 mutate(height = ifelse(is.na(height) &
 inches < 12 & between(guess, smallest, tallest),
 guess, height)) %>%
 select(-guess)
```

Podemos verificar todas las entradas que convertimos al escribir:

```
new_heights %>%
 filter(not_inches(original)) %>%
 select(original, height) %>%
 arrange(height) %>%
 View()
```

Una observación final es que si miramos a los estudiantes más bajos de nuestro curso:

```
new_heights %>% arrange(height) %>% head(n=7)
#> time_stamp sex height feet inches original
#> 1 2017-07-04 01:30:25 Male 50.0 NA NA 50
#> 2 2017-09-07 10:40:35 Male 50.0 NA NA 50
#> 3 2014-09-02 15:18:30 Female 51.0 NA NA 51
#> 4 2016-06-05 14:07:20 Female 52.0 NA NA 52
#> 5 2016-06-05 14:07:38 Female 52.0 NA NA 52
#> 6 2014-09-23 03:39:56 Female 53.0 NA NA 53
#> 7 2015-01-07 08:57:29 Male 53.8 NA NA 53.77
```

Vemos alturas de 51, 52 y 53. Estas alturas tan bajas son raras y es probable que los estudiantes realmente querían escribir 5'1, 5'2 y 5'3. Debido a que no estamos completamente seguros, los dejaremos como se reportaron. El objeto `new_heights` contiene nuestra solución final para este caso de estudio.

## 24.11 División de cadenas

Otra operación de *wrangling* de datos muy común es la división de cadenas. Para ilustrar cómo surge esto, comenzaremos con un ejemplo ilustrativo. Supongan que no tenemos la

función `read_csv` o `read.csv` disponible. En cambio, tenemos que leer un archivo csv usando la función de base R `readLines` así:

```
filename <- system.file("extdata/murders.csv", package = "dslabs")
lines <- readLines(filename)
```

Esta función lee los datos línea por línea para crear un vector de cadenas. En este caso, una cadena para cada fila en la hoja de cálculo. Las primeras seis líneas son:

```
lines %>% head()
#> [1] "state,abb,region,population,total"
#> [2] "Alabama,AL,South,4779736,135"
#> [3] "Alaska,AK,West,710231,19"
#> [4] "Arizona,AZ,West,6392017,232"
#> [5] "Arkansas,AR,South,2915918,93"
#> [6] "California,CA,West,37253956,1257"
```

Queremos extraer los valores que están separados por una coma para cada cadena en el vector. El comando `str_split` hace exactamente esto:

```
x <- str_split(lines, ",")
x %>% head(2)
#> [[1]]
#> [1] "state" "abb" "region" "population" "total"
#>
#> [[2]]
#> [1] "Alabama" "AL" "South" "4779736" "135"
```

Tengan en cuenta que la primera entrada representa el nombre de las columnas, por lo que podemos separarlas:

```
col_names <- x[[1]]
x <- x[-1]
```

Para convertir nuestra lista en un *data frame*, podemos usar un atajo ofrecido por la función `map` en el paquete **purrr**. La función `map` aplica la misma función a cada elemento en una lista. Entonces, si queremos extraer la primera entrada de cada elemento en `x`, podemos escribir:

```
library(purrr)
map(x, function(y) y[1]) %>% head(2)
#> [[1]]
#> [1] "Alabama"
#>
#> [[2]]
#> [1] "Alaska"
```

Sin embargo, debido a que esta es una tarea tan común, **purrr** provee un atajo. Si el segundo argumento recibe un número entero en lugar de una función, supondrá que queremos esa entrada. Entonces, el código anterior se puede escribir de manera más eficiente así:

```
map(x, 1)
```

Para forzar `map` a devolver un vector de caracteres en lugar de una lista, podemos usar `map_chr`. Asimismo, `map_int` devuelve enteros. Entonces, para crear nuestro *data frame*, podemos usar:

```
dat <- tibble(map_chr(x, 1),
 map_chr(x, 2),
 map_chr(x, 3),
 map_chr(x, 4),
 map_chr(x, 5)) %>%
 mutate_all(parse_guess) %>%
 setNames(col_names)
dat %>% head
#> # 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
#> # ... with 1 more row
```

Si exploran el paquete **purrr**, aprenderán que es posible realizar lo anterior con el siguiente código, que es más eficiente:

```
dat <- x %>%
 transpose() %>%
 map(~ parse_guess(unlist(.))) %>%
 setNames(col_names) %>%
 as_tibble()
```

Resulta que podemos evitar todo el trabajo que mostramos arriba después de la llamada a `str_split`. Específicamente, si sabemos que los datos que estamos extrayendo se pueden representar como una tabla, podemos usar el argumento `simplify=TRUE` y `str_split` devuelve una matriz en lugar de una lista:

```
x <- str_split(lines, "", simplify = TRUE)
col_names <- x[1,]
x <- x[-1,]
colnames(x) <- col_names
x %>% as_tibble() %>%
 mutate_all(parse_guess) %>%
 head(5)
#> # A tibble: 5 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
```

## 24.12 Estudio de caso 3: extracción de tablas de un PDF

Uno de los sets de datos de **dslabs** muestra las tasas de financiación científica por género en los Países Bajos:

```
library(dslabs)
data("research_funding_rates")
research_funding_rates %>%
 select("discipline", "success_rates_men", "success_rates_women")
#> discipline success_rates_men success_rates_women
#> 1 Chemical sciences 26.5 25.6
#> 2 Physical sciences 19.3 23.1
#> 3 Physics 26.9 22.2
#> 4 Humanities 14.3 19.3
#> 5 Technical sciences 15.9 21.0
#> 6 Interdisciplinary 11.4 21.8
#> 7 Earth/life sciences 24.4 14.3
#> 8 Social sciences 15.3 11.5
#> 9 Medical sciences 18.8 11.2
```

Los datos provienen de un artículo publicado en *Proceedings of the National Academy of Science* (PNAS)<sup>5</sup>, una revista científica ampliamente leída. Sin embargo, los datos no se proveen en una hoja de cálculo, sino en una tabla en un documento PDF. Aquí hay una captura de pantalla de la tabla:

Table S1. Numbers of applications and awarded grants, along with success rates for male and female applicants, by scientific discipline

Discipline	Applications, n			Awards, n			Success rates, %		
	Total	Men	Women	Total	Men	Women	Total	Men	Women
Total	2,823	1,635	1,188	467	290	177	16.5	17.7 <sub>a</sub>	14.9 <sub>b</sub>
Chemical sciences	122	83	39	32	22	10	26.2	26.5 <sub>a</sub>	25.6 <sub>b</sub>
Physical sciences	174	135	39	35	26	9	20.1	19.3 <sub>a</sub>	23.1 <sub>b</sub>
Physics	76	67	9	20	18	2	26.3	26.9 <sub>a</sub>	22.2 <sub>b</sub>
Humanities	396	230	166	65	33	32	16.4	14.3 <sub>a</sub>	19.3 <sub>b</sub>
Technical sciences	251	189	62	43	30	13	17.1	15.9 <sub>a</sub>	21.0 <sub>b</sub>
Interdisciplinary	183	105	78	29	12	17	15.8	11.4 <sub>a</sub>	21.8 <sub>b</sub>
Earth/life sciences	282	156	126	56	38	18	19.9	24.4 <sub>a</sub>	14.3 <sub>b</sub>
Social sciences	834	425	409	112	65	47	13.4	15.3 <sub>a</sub>	11.5 <sub>b</sub>
Medical sciences	505	245	260	75	46	29	14.9	18.8 <sub>a</sub>	11.2 <sub>b</sub>

Success rates for male and female applicants with different subscripts differ reliably from one another ( $P < 0.05$ ).

(Fuente: Romy van der Lee y Naomi Ellemers, PNAS 2015 112 (40) 12349-12353<sup>6</sup>.)

Podríamos extraer los números a mano, pero esto podría conducir a errores humanos. En

<sup>5</sup><http://www.pnas.org/content/112/40/12349.abstract>

<sup>6</sup><http://www.pnas.org/content/112/40/12349>

cambio, podemos intentar cambiar los datos con R. Comenzamos descargando el documento PDF y luego importándolo a R:

```
library("pdftools")
temp_file <- tempfile()
url <- paste0("https://www.pnas.org/content/suppl/2015/09/16/",
 "1510159112.DCSupplemental/pnas.201510159SI.pdf")
download.file(url, temp_file)
txt <- pdf_text(temp_file)
file.remove(temp_file)
```

Si examinamos el objeto `txt`, notamos que es un vector de caracteres con una entrada para cada página. Entonces mantenemos la página que queremos:

```
raw_data_research_funding_rates <- txt[2]
```

Los pasos anteriores se pueden omitir porque también incluimos estos datos sin procesar en el paquete **dslabs**:

```
data("raw_data_research_funding_rates")
```

Al examinar el objeto `raw_data_research_funding_rates`, vemos que es una cadena larga y cada línea de la página, incluyendo las filas de la tabla, están separadas por el símbolo de nueva línea: `\n`. Por lo tanto, podemos crear una lista con las líneas del texto como elementos así:

```
tab <- str_split(raw_data_research_funding_rates, "\n")
```

Debido a que comenzamos con solo un elemento en la cadena, terminamos con una lista con solo una entrada:

```
tab <- tab[[1]]
```

Al examinar `tab`, vemos que la información para los nombres de columna son la tercera y cuarta entrada:

```
the_names_1 <- tab[3]
the_names_2 <- tab[4]
```

La primera de estas filas se ve así:

```
#> Applications, n
#> Awards, n
#> Success rates, %
```

Queremos crear un vector con un nombre para cada columna. Podemos hacerlo usando algunas de las funciones que acabamos de aprender. Empecemos con `the_names_1`, que mostramos arriba. Queremos eliminar el espacio inicial y todo lo que sigue a la coma. Utilizamos *regex* para este último. Entonces podemos obtener los elementos dividiendo cadenas separadas por espacio. Queremos dividir solo cuando hay 2 o más espacios para no dividir `Success rates`. Para esto usamos la expresión regular `\s{2,}`:

```
the_names_1 <- the_names_1 %>%
 str_trim() %>%
 str_replace_all("\\s+", "") %>%
 str_split("\\s{2,}", simplify = TRUE)
the_names_1
#> [1] [2] [3]
#> [1,] "Applications" "Awards" "Success rates"
```

Ahora examinaremos `the_names_2`:

```
#> Discipline Total Men Women
#> n Total Men Women Total Men Women
```

Aquí queremos podar el espacio inicial y buscar usar espacios para dividir las cadenas como lo hicimos para la primera línea:

```
the_names_2 <- the_names_2 %>%
 str_trim() %>%
 str_split("\\s+", simplify = TRUE)
the_names_2
#> [1] [2] [3] [4] [5] [6] [7] [8]
#> [1,] "Discipline" "Total" "Men" "Women" "Total" "Men" "Women" "Total"
#> [9] [10]
#> [1,] "Men" "Women"
```

Entonces podemos unirlos para generar un nombre para cada columna:

```
tmp_names <- str_c(rep(the_names_1, each = 3), the_names_2[-1], sep = "_")
the_names <- c(the_names_2[1], tmp_names) %>%
 str_to_lower() %>%
 str_replace_all("\\s", "_")
the_names
#> [1] "discipline" "applications_total" "applications_men"
#> [4] "applications_women" "awards_total" "awards_men"
#> [7] "awards_women" "success_rates_total" "success_rates_men"
#> [10] "success_rates_women"
```

Ahora estamos listos para obtener los datos reales. Al examinar el objeto `tab`, notamos que la información está en las líneas 6 a 14. Podemos usar `str_split` de nuevo para lograr nuestro objetivo:

```
new_research_funding_rates <- tab[6:14] %>%
 str_trim() %>%
 str_split("\\s{2,}", simplify = TRUE) %>%
 data.frame() %>%
 setNames(the_names) %>%
 mutate_at(-1, parse_number)
new_research_funding_rates %>% as_tibble()
#> # A tibble: 9 x 10
```

```
#> discipline applications_total applications_men applications_wom~
#> <chr> <dbl> <dbl> <dbl>
#> 1 Chemical scienc~ 122 83 39
#> 2 Physical scienc~ 174 135 39
#> 3 Physics 76 67 9
#> 4 Humanities 396 230 166
#> 5 Technical scienc~ 251 189 62
#> # ... with 4 more rows, and 6 more variables: awards_total <dbl>,
#> # awards_men <dbl>, awards_women <dbl>, success_rates_total <dbl>,
#> # success_rates_men <dbl>, success_rates_women <dbl>
```

Podemos ver que los objetos son idénticos:

```
identical(research_funding_rates, new_research_funding_rates)
#> [1] TRUE
```

### 24.13 Recodificación

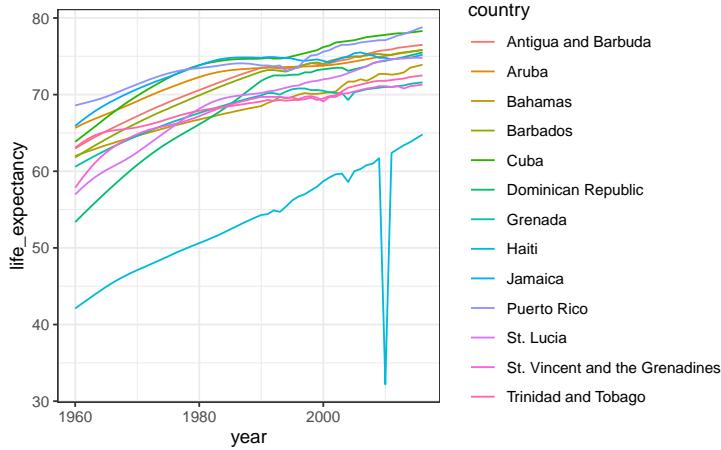
Otra operación común que involucra cadenas es recodificar los nombres de variables categóricas. Supongan que tienen nombres largos para sus niveles y los mostrarán en gráficos. Es posible que quieran utilizar versiones más cortas de estos nombres. Por ejemplo, en los vectores de caracteres con nombres de países, es posible que quieran cambiar “Estados Unidos de América” a “Estados Unidos”, “Reino Unido” a “RU” y así sucesivamente. Podemos hacer esto con `case_when`, aunque `tidyverse` ofrece una opción específicamente diseñada para esta tarea: la función `recode`.

Aquí hay un ejemplo que muestra cómo cambiar el nombre de los países con nombres largos:

```
library(dslabs)
data("gapminder")
```

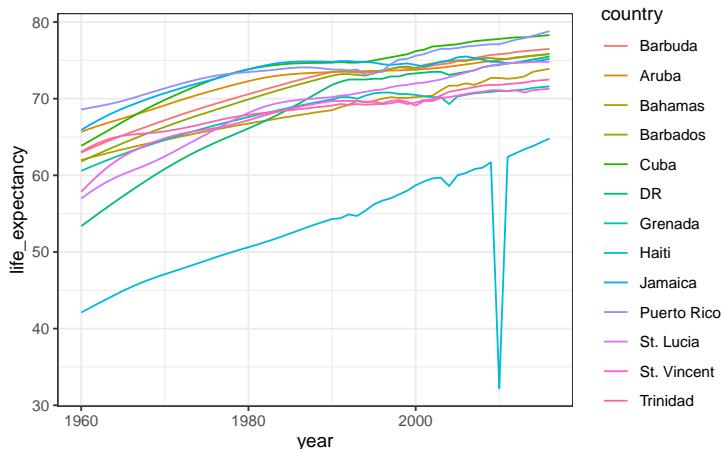
Imaginen que queremos mostrar series temporales de esperanza de vida por país para el Caribe:

```
gapminder %>%
 filter(region == "Caribbean") %>%
 ggplot(aes(year, life_expectancy, color = country)) +
 geom_line()
```



El gráfico es lo que queremos, pero gran parte del espacio se desperdicia para acomodar algunos de los nombres largos de los países. Tenemos cuatro países con nombres de más de 12 caracteres. Estos nombres aparecen una vez por cada año en el set de datos Gapminder. Una vez que elegimos los apodos, debemos cambiarlos todos de manera consistente. La función `recode` se puede utilizar para hacer esto:

```
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()
```



Hay otras funciones similares en otros paquetes de R, como `recode_factor` y `fct_recode` en el paquete `forcats`.

## 24.14 Ejercicios

1. Complete todas las lecciones y ejercicios en el tutorial interactivo en línea <https://regexone.com/>.
2. En el directorio `extdata` del paquete **dslabs**, encontrará un archivo PDF que contiene datos de mortalidad diaria para Puerto Rico del 1 de enero de 2015 al 31 de mayo de 2018. Puede encontrar el archivo así:

```
fn <- system.file("extdata", "RD-Mortality-Report_2015-18-180531.pdf",
 package="dslabs")
```

```
system2("open", args = fn)
```

y en Windows, puede escribir:

```
system("cmd.exe", input = paste("start", fn))
```

¿Cuál de las siguientes opciones describe mejor este archivo?

- a. Es una tabla. Extraer los datos será fácil.
  - b. Es un informe escrito en prosa. Extraer los datos será imposible.
  - c. Es un informe que combina gráficos y tablas. Extraer los datos parece posible.
  - d. Muestra gráficos de los datos. Extraer los datos será difícil.
3. Vamos a crear un set de datos ordenado con cada fila representando una observación. Las variables en este set de datos serán año, mes, día y muertes. Comience instalando y cargando el paquete **pdftools**:

```
install.packages("pdftools")
library(pdftools)
```

Ahora lean `fn` utilizando la función `pdf_text` y almacene los resultados en un objeto llamado `txt`. ¿Cuál de las siguientes opciones describe mejor lo que ve en `txt`?

- a. Una tabla con los datos de mortalidad.
  - b. Una cadena de caracteres de longitud 12. Cada entrada representa el texto en cada página. Los datos de mortalidad están ahí en alguna parte.
  - c. Una cadena de caracteres con una entrada que contiene toda la información en el archivo PDF.
  - d. Un documento HTML.
4. Extraiga la novena página del archivo PDF del objeto `txt`. Luego, use `str_split` del paquete **stringr** para que cada línea esté en una entrada diferente. Llame a este vector de cadena `s`. Entonces mire el resultado y elija el que mejor describa lo que ve.
- a. Es una cadena vacía.

- b. Puedo ver la figura que se muestra en la página 1.
  - c. Es una tabla *tidy*.
  - d. ¡Puedo ver la tabla! Pero hay muchas otras cosas de las que debemos deshacernos.
5. ¿Qué tipo de objeto es `s` y cuántas entradas tiene?
6. Vemos que el `output` es una lista con un componente. Redefina `s` para que sea la primera entrada de la lista. ¿Qué tipo de objeto es `s` y cuántas entradas tiene?
7. Al inspeccionar la cadena que obtuvimos anteriormente, vemos un problema común: espacios en blanco antes y después de los otros caracteres. Podar es un primer paso común en el procesamiento de cadenas. Estos espacios adicionales eventualmente complicarán dividir las cadenas, por lo que comenzamos por eliminarlos. Aprendimos sobre el comando `str_trim` que elimina espacios al principio o al final de las cadenas. Use esta función para podar `s`.
8. Queremos extraer los números de las cadenas almacenadas en `s`. Sin embargo, hay muchos caracteres no numéricos que se interpondrán en el camino. Podemos eliminarlos, pero antes de hacerlo queremos preservar la cadena con el encabezado de la columna, que incluye la abreviatura del mes. Utilice la función `str Which` para encontrar las filas con un encabezado. Guarde estos resultados en `header_index`. Sugerencia: encuentre la primera cadena que corresponda con el patrón 2015 utilizando la función `str Which`.
9. Ahora vamos a definir dos objetos: `month` almacenará el mes y `header` almacenará los nombres de columna. Identifique qué fila contiene el encabezado de la tabla. Guarde el contenido de la fila en un objeto llamado `header`. Luego, use `str_split` para ayudar a definir los dos objetos que necesitamos. Sugerencias: el separador aquí es uno o más espacios. Además, considere usar el argumento `simplify`.
10. Observe que hacia el final de la página verá una fila `totals` seguida de filas con otras estadísticas de resumen. Cree un objeto llamado `tail_index` con el índice de la entrada `totals`.
11. Debido a que nuestra página PDF incluye gráficos con números, algunas de nuestras filas tienen solo un número (desde el eje-y del gráfico). Utilice la función `str_count` para crear un objeto `n` con el número de números en cada fila. Sugerencia: escriba una expresión regular para un número como este `\d+`.
12. Ahora estamos listos para eliminar entradas de filas que sabemos que no necesitamos. La entrada `header_index` y todo lo que viene antes se debe eliminar. Entradas para las cuales `n` es 1 también deben eliminarse. Finalmente, se debe eliminar la entrada `tail_index` y todo lo que viene después.
13. Ahora estamos listos para eliminar todas las entradas no numéricas. Haga esto usando `regex` y la función `str_remove_all`. Sugerencia: recuerde que en `regex`, usar la versión en mayúscula de un carácter especial generalmente significa lo contrario. Entonces `\D` significa “no un dígito”. Recuerde que también quiere mantener espacios.
14. Para convertir las cadenas en una tabla, use la función `str_split_fixed`. Convierta `s` en una matriz de datos con solo la fecha y los datos con los números de muertes. Sugerencias: recuerde que el patrón que separa los valores es “uno o más espacios”. Utilice el argumento `n` de la función `str_split_fixed` para asegurarse de que la matriz de datos tenga las columnas necesarias para contener la información que queremos conservar y una columna más para capturar todo lo que no queremos. Finalmente, conserve sólo las columnas que necesita.
15. Ahora ya casi ha terminado. Agregue nombres de columna a la matriz, incluyendo uno llamado `day`. Además, agregue una columna con el mes. Llame al objeto resultante `dat`.

Finalmente, asegúrese de que el día es un número entero, no un carácter. Sugerencia: use solo las primeras cinco columnas.

16. Ahora termine convirtiendo `tab` a formato `tidy` con la función `pivot_longer`.
17. Haga un diagrama de muertes versus día con color para indicar el año. Excluya 2018 ya que no tenemos datos para todo el año.
18. Ahora que hemos *wrangle* estos datos paso a paso, póngalos todos juntos en un programa de R, usando el *pipe* lo mas posible. Sugerencia: primero defina los índices, luego escriba una línea de código que realice todo el procesamiento de la cadena.
19. Avanzado: volvamos al ejemplo de la nómina MLB de la sección “Extracción de la web”. Use lo que aprendió en los capítulos de extracción de la web y procesamiento de cadenas para extraer la nómina de los Yankees de Nueva York, las Medias Rojas de Boston y los Atléticos de Oakland y graffíquelo como una función del tiempo.

# 25

---

## Cómo leer y procesar fechas y horas

---

### 25.1 Datos tipo fecha

Hemos descrito tres tipos principales de vectores: numéricos, de caracteres y lógicos. En proyectos de ciencia de datos, a menudo encontramos variables que son fechas. Aunque podemos representar una fecha con una cadena, por ejemplo `November 2, 2017`, una vez que elegimos un día de referencia, conocido como la *época Unix* (*epoch* en inglés), se pueden convertir en números calculando el número de días desde la época Unix. Los lenguajes de computadora usualmente usan el 1 de enero de 1970 como la época Unix. Entonces, por ejemplo, el 2 de enero de 2017 es el día 1, el 31 de diciembre de 1969 es el día -1 y el 2 de noviembre de 2017 es el día 17,204.

Ahora, ¿cómo debemos representar fechas y horas al analizar los datos en R? Podríamos usar días desde la *época Unix*, pero es casi imposible de interpretar. Si les digo que es el 2 de noviembre de 2017, saben lo que esto significa inmediatamente. Si les digo que es el día 17,204, estarán bastante confundidos. Problemas similares surgen con la hora e incluso pueden aparecer más complicaciones debido a las zonas horarias.

Por esta razón, R define un tipo de datos solo para fechas y horas. Vimos un ejemplo en los datos de las encuestas:

```
library(tidyverse)
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"
```

Estas parecen cadenas, pero no lo son:

```
class(polls_us_election_2016$startdate)
#> [1] "Date"
```

Miren lo que sucede cuando los convertimos en números:

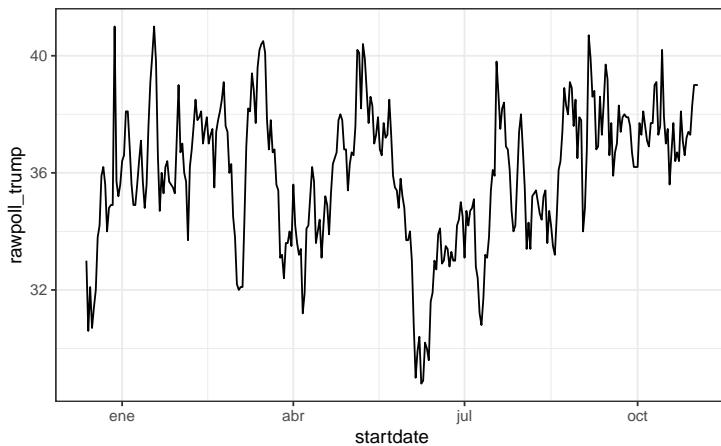
```
as.numeric(polls_us_election_2016$startdate) %>% head
#> [1] 17108 17106 17107 17109 17108 17108
```

Los convierte en días desde la época Unix. La función `as.Date` puede convertir un carácter en una fecha. Entonces, para ver que la época Unix es el día 0, podemos escribir:

```
as.Date("1970-01-01") %>% as.numeric
#> [1] 0
```

Funciones que grafican, como las de **ggplot2**, conocen el formato de fecha. Esto significa que, por ejemplo, un diagrama de dispersión puede usar la representación numérica para decidir la posición del punto, pero incluir la cadena en las etiquetas:

```
polls_us_election_2016 %>% filter(pollster == "Ipsos" & state == "U.S.") %>%
 ggplot(aes(startdate, rawpoll_trump)) +
 geom_line()
```



Noten que se muestran los nombres de los meses, una característica muy conveniente.

## 25.2 El paquete lubridate

El **tidyverse** incluye funcionalidad para manejar fechas a través del paquete **lubridate**.

```
library(lubridate)
```

Tomaremos una muestra aleatoria de fechas para mostrar algunas de las cosas útiles que uno puede hacer:

```
set.seed(2002)
dates <- sample(polls_us_election_2016$startdate, 10) %>% sort
dates
#> [1] "2016-05-31" "2016-08-08" "2016-08-19" "2016-09-22" "2016-09-27"
#> [6] "2016-10-12" "2016-10-24" "2016-10-26" "2016-10-29" "2016-10-30"
```

Las funciones **year**, **month** y **day** extraen esos valores:

```
tibble(date = dates,
 month = month(dates),
 day = day(dates),
 year = year(dates))
#> # A tibble: 10 x 4
#> date month day year
#> <date> <dbl> <int> <dbl>
#> 1 2016-05-31 5 31 2016
#> 2 2016-08-08 8 8 2016
#> 3 2016-08-19 8 19 2016
#> 4 2016-09-22 9 22 2016
#> 5 2016-09-27 9 27 2016
#> # ... with 5 more rows
```

También podemos extraer las etiquetas del mes:

```
month(dates, label = TRUE)
#> [1] may ago ago sep sep oct oct oct oct
#> 12 Levels: ene < feb < mar < abr < may < jun < jul < ago < ... < dic
```

Otro conjunto útil de funciones son las que convierten cadenas en fechas. La función `ymd` supone que las fechas están en el formato AAAA-MM-DD e intenta leer y procesar (*parse* en inglés) lo mejor posible.

```
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 proviene del hecho de que las fechas a menudo vienen en diferentes formatos donde el orden de año, mes y día son diferentes. El formato preferido es mostrar año (con los cuatro dígitos), mes (dos dígitos) y luego día (dos dígitos), o lo que se llama ISO 8601. Específicamente, usamos AAAA-MM-DD para que si ordenamos la cadena, estará ordenado por fecha. Pueden ver que la función `ymd` las devuelve en este formato.

Pero, ¿qué pasa si encuentran fechas como “01/09/02”? Esto podría ser el 1 de septiembre de 2002 o el 2 de enero de 2009 o el 9 de enero de 2002. En estos casos, examinar todo el vector de fechas los ayudará a determinar qué formato es por proceso de eliminación. Una vez que sepan, pueden usar las muchas funciones ofrecidas por **lubridate**.

Por ejemplo, si la cadena es:

```
x <- "09/01/02"
```

La función `ymd` supone que la primera entrada es el año, la segunda es el mes y la tercera es el día, por lo que la convierte a:

```
ymd(x)
#> [1] "2009-01-02"
```

La función `mdy` supone que la primera entrada es el mes, luego el día y entonces el año:

```
mdy(x)
#> [1] "2002-09-01"
```

El paquete **lubridate** ofrece una función para cada posibilidad:

```
ydm(x)
#> [1] "2009-02-01"
myd(x)
#> [1] "2001-09-02"
dmy(x)
#> [1] "2002-01-09"
dym(x)
#> [1] "2001-02-09"
```

El paquete **lubridate** también es útil para lidiar con los tiempos. En base R, pueden obtener la hora actual escribiendo `Sys.time()`. El paquete **lubridate** ofrece una función un poco más avanzada, `now`, que les permite definir la zona horaria:

```
now()
#> [1] "2021-05-24 08:09:06 EDT"
now("GMT")
#> [1] "2021-05-24 12:09:06 GMT"
```

Pueden ver todas las zonas horarias disponibles con la función `OlsonNames()`.

También podemos extraer horas, minutos y segundos:

```
now() %>% hour()
#> [1] 8
now() %>% minute()
#> [1] 9
now() %>% second()
#> [1] 6.67
```

El paquete también incluye una función para convertir cadenas en horas del día, así como funciones que convierten objetos que representan horas del día, que además incluyen las fechas:

```
x <- c("12:34:56")
hms(x)
#> [1] "12H 34M 56S"
x <- "Nov/2/2012 12:34:56"
mdy_hms(x)
#> [1] "2012-11-02 12:34:56 UTC"
```

Este paquete tiene muchas otras funciones útiles. Aquí describimos dos que consideramos particularmente útiles.

La función `make_date` se puede utilizar para rápidamente crear un objeto de fecha. Por ejemplo, para crear un objeto de fecha que represente el 6 de julio de 2019, escribimos:

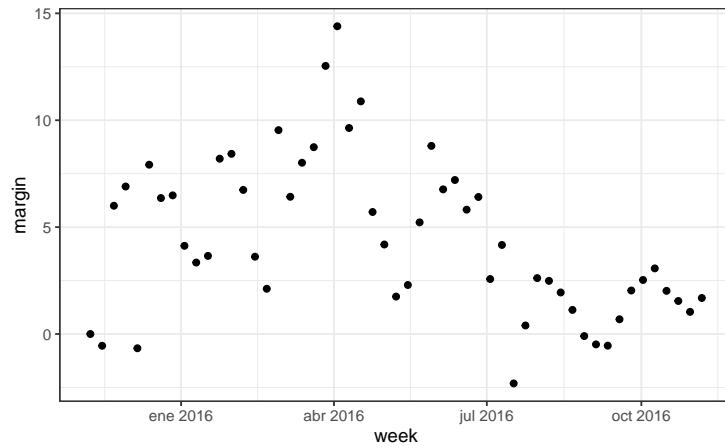
```
make_date(2019, 7, 6)
#> [1] "2019-07-06"
```

Para hacer un vector del 1 de enero para los años 80 escribimos:

```
make_date(1980:1989)
#> [1] "1980-01-01" "1981-01-01" "1982-01-01" "1983-01-01" "1984-01-01"
#> [6] "1985-01-01" "1986-01-01" "1987-01-01" "1988-01-01" "1989-01-01"
```

Otra función muy útil es `round_date`. Se puede utilizar para redondear las fechas al año, trimestre, mes, semana, día, hora, minutos o segundos más cercanos. Entonces, si queremos agrupar todas las encuestas por semana del año, podemos hacer lo siguiente:

```
polls_us_election_2016 %>%
 mutate(week = round_date(startdate, "week")) %>%
 group_by(week) %>%
 summarize(margin = mean(rawpoll_clinton - rawpoll_trump)) %>%
 qplot(week, margin, data = .)
```



## 25.3 Ejercicios

En la sección de ejercicios anterior, depuramos datos de un archivo PDF que contiene estadísticas vitales de Puerto Rico. Hicimos esto para el mes de septiembre. A continuación incluimos un código que lo hace durante los 12 meses.

```

library(tidyverse)
library(lubridate)
library(purrr)
library(pdftools)
library(dslabs)

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_tibble() %>%
 setNames(c("day", header)) %>%
 mutate(month = month, day = as.numeric(day)) %>%
 pivot_longer(-c(day, month), names_to = "year", values_to = "deaths") %>%
 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)) %>%
 filter(date <= "2018-05-01")

```

1. Queremos hacer un gráfico de recuentos de muertes versus fecha. Un primer paso es convertir la variable del mes de caracteres a números. Tenga en cuenta que las abreviaturas de los meses están en espanglés. Utilice la función `recode` para convertir meses en números y redefinir `tab`.
2. Cree una nueva columna `date` con la fecha de cada observación. Sugerencia: use la función `make_date`.
3. Grafique muertes versus fecha.
4. Noten que después del 31 de mayo de 2018, las muertes son todas 0. Los datos probablemente aún no se han ingresado. También vemos una caída a partir del 1 de mayo. Redefina `tab` para excluir observaciones tomadas a partir del 1 de mayo de 2018. Luego, rehaga el gráfico.
5. Vuelva a hacer el gráfico anterior, pero esta vez grafique las muertes versus el día del año; por ejemplo, 12 de enero de 2016 y 12 de enero de 2017, son ambos el día 12. Use el color para indicar los diferentes años. Sugerencia: use la función `yday` de `lubridate`.

6. Vuelva a hacer el gráfico anterior pero, esta vez, use dos colores diferentes para antes y después del 20 de septiembre de 2017.
7. Avanzado: rehaga el gráfico anterior, pero esta vez muestre el mes en el eje-x. Sugerencia: cree una variable con la fecha de un año determinado. Luego, use la función `scale_x_date` para mostrar solo los meses.
8. Rehaga las muertes versus el día pero con promedios semanales. Sugerencia: use la función `round_date`.
9. Rehaga el gráfico pero con promedios mensuales. Sugerencia: use la función `round_date` de nuevo.



# 26

---

## Minería de textos

---

Con la excepción de las etiquetas utilizadas para representar datos categóricos, nos hemos enfocado en los datos numéricos. Pero en muchas aplicaciones, los datos comienzan como texto. Ejemplos bien conocidos son el filtrado de spam, la prevención del delito cibernético, la lucha contra el terrorismo y el análisis de sentimiento (también conocido como minería de opinión). En todos estos casos, los datos sin procesar se componen de texto de forma libre. Nuestra tarea es extraer información de estos datos. En esta sección, aprendemos cómo generar resúmenes numéricos útiles a partir de datos de texto a los que podemos aplicar algunas de las poderosas técnicas de visualización y análisis de datos que hemos aprendido.

---

### 26.1 Estudio de caso: tuits de Trump

Durante las elecciones presidenciales estadounidenses de 2016, el candidato Donald J. Trump usó su cuenta de Twitter como una manera de comunicarse con los posibles votantes. El 6 de agosto de 2016, Todd Vaziri tuiteó sobre Trump y declaró que “Cada tweet no hiperbólico es de iPhone (su personal). Cada tweet hiperbólico es de Android (de él)”<sup>1</sup>. El científico de datos David Robinson realizó un análisis para determinar si los datos respaldan esta afirmación<sup>2</sup>. Aquí, revisamos el análisis de David para aprender algunos de los conceptos básicos de la minería de textos. Para obtener más información sobre la minería de textos en R, recomendamos el libro *Text Mining with R* de Julia Silge y David Robinson<sup>3</sup>.

Utilizaremos los siguientes paquetes:

```
library(tidyverse)
library(lubridate)
library(scales)
```

En general, podemos extraer datos directamente de Twitter usando el paquete **rtweet**. Sin embargo, en este caso, un grupo ya ha compilado datos para nosotros y los ha puesto a disposición en: <https://www.thetrumparchive.com>. Podemos obtener los datos de su API JSON usando un *script* como este:

```
url <- 'http://www.trumptwitterarchive.com/data/realdonaldtrump/%s.json'
trump_tweets <- map(2009:2017, ~sprintf(url, .x)) %>%
 map_df(jsonlite::fromJSON, simplifyDataFrame = TRUE) %>%
```

<sup>1</sup><https://twitter.com/tvaziri/status/762005541388378112/photo/1>

<sup>2</sup><http://varianceexplained.org/r/trump-tweets/>

<sup>3</sup><https://www.tidytextmining.com/>

```
filter(!is_retweet & !str_detect(text, '^!')) %>%
 mutate(created_at = parse_date_time(created_at,
 orders = "a b! d! H!:M!:S! z!* Y!",
 tz="EST"))
```

Para facilitar el análisis, incluimos el resultado del código anterior en el paquete **dslabs**:

```
library(dslabs)
data("trump_tweets")
```

Pueden ver el *data frame* con información sobre los tuits al escribir:

```
head(trump_tweets)
```

con las siguientes variables incluidas:

```
names(trump_tweets)
#> [1] "source"
#> [3] "text"
#> [5] "retweet_count"
#> [7] "favorite_count"
#> [9] "id_str"
#> [11] "created_at"
#> [13] "in_reply_to_user_id_str"
#> [15] "is_retweet"
```

El archivo de ayuda `?trump_tweets` provee detalles sobre lo que representa cada variable. Los tuits están representados por el variable `text`:

```
trump_tweets$text[16413] %>% str_wrap(width = options()$width) %>% cat
#> Great to be back in Iowa! #TBT with @JerryJrFalwell joining me in
#> Davenport- this past winter. #MAGA https://t.co/A5IFOQHnic
```

y la variable `source` nos dice qué dispositivo se usó para componer y cargar cada tuit:

```
trump_tweets %>% count(source) %>% arrange(desc(n)) %>% head(5)
#> source n
#> 1 Twitter Web Client 10718
#> 2 Twitter for Android 4652
#> 3 Twitter for iPhone 3962
#> 4 TweetDeck 468
#> 5 TwitLonger Beta 288
```

Estamos interesados en lo que sucedió durante la campaña, por lo que para este análisis nos enfocaremos en lo que se tuiteó entre el día en que Trump anunció su campaña y el día de las elecciones. Definimos la siguiente tabla que contiene solo los tuits de ese período de tiempo. Tengan en cuenta que usamos `extract` para eliminar la parte `Twitter for` de `source` y filtrar los *retweets*.

```

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) %>%
 as_tibble()

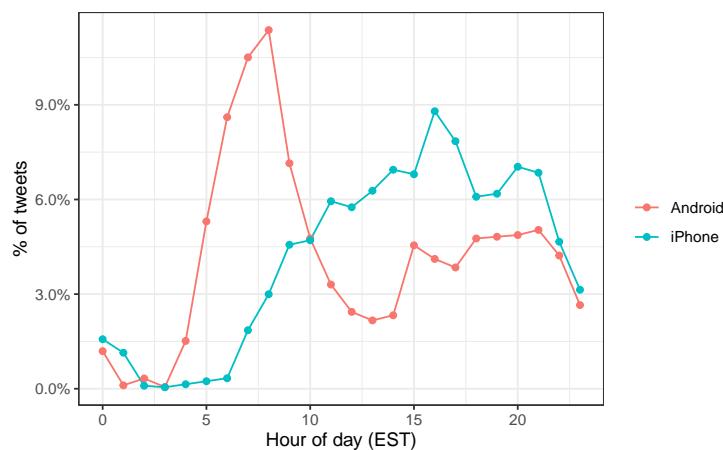
```

Ahora podemos usar la visualización de datos para explorar la posibilidad de que dos grupos diferentes hayan escrito los mensajes desde estos dispositivos. Para cada tuit, extraeremos la hora en que se publicó (hora de la costa este de EE.UU. o EST por sus siglas en inglés), y luego calcularemos la proporción de tuits tuiteada a cada hora para cada dispositivo:

```

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 = "")

```



Notamos un gran pico para Android en las primeras horas de la mañana, entre las 6 y las 8 de la mañana. Parece haber una clara diferencia en estos patrones. Por lo tanto, suponemos que dos entidades diferentes están utilizando estos dos dispositivos.

Ahora estudiaremos cómo difieren los tuits cuando comparamos Android con iPhone. Para hacer esto, utilizaremos el paquete **tidytext**.

## 26.2 Texto como datos

El paquete **tidytext** nos ayuda a convertir texto de forma libre en una tabla ordenada. Tener los datos en este formato facilita enormemente la visualización de datos y el uso de técnicas estadísticas.

```
library(tidytext)
```

La función principal necesaria para lograr esto es `unnest_tokens`. Un *token* se refiere a una unidad que consideramos como un punto de datos. Los *tokens* más comunes son las palabras, pero también pueden ser caracteres individuales, *ngrams*, oraciones, líneas o un patrón definido por una expresión regular. Las funciones tomarán un vector de cadenas y extraerán los *tokens* para que cada uno obtenga una fila en la nueva tabla. Aquí hay un ejemplo sencillo:

```
poem <- c("Roses are red,", "Violets are blue,",
 "Sugar is sweet,", "And so are you.")
example <- tibble(line = c(1, 2, 3, 4),
 text = poem)
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
#> # ... with 8 more rows
```

Ahora consideremos un ejemplo de los tuits. Miren el tuit número 3008 porque luego nos permitirá ilustrar un par de puntos:

```
i <- 3008
campaign_tweets$text[i] %>% str_wrap(width = 65) %>% cat()
#> Great to be back in Iowa! #TBT with @JerryJrFalwell joining me in
#> Davenport- this past winter. #MAGA https://t.co/A5IFOQHnic
campaign_tweets[i,] %>%
 unnest_tokens(word, text) %>%
 pull(word)
```

```
#> [1] "great" "to" "be" "back"
#> [5] "in" "iowa" "tbt" "with"
#> [9] "jerryjrfalwell" "joining" "me" "in"
#> [13] "davenport" "this" "past" "winter"
#> [17] "maga" "https" "t.co" "a5if0qhn1c"
```

Noten que la función intenta convertir *tokens* en palabras. Para hacer esto, sin embargo, elimina los caracteres que son importantes en el contexto de Twitter. Específicamente, la función elimina todos los # y @. Un *token* en el contexto de Twitter no es lo mismo que en el contexto del inglés hablado o escrito. Por esta razón, en lugar de usar el valor pre-determinado, `words`, usamos el *token tweets* que incluye patrones que comienzan con @ y #:

```
campaign_tweets[i,] %>%
 unnest_tokens(word, text, token = "tweets") %>%
 pull(word)
#> [1] "great" "to" "be" "back"
#> [3] "be" "iowa" "tbt" "with"
#> [5] "in" "iowa" "joining" "in"
#> [7] "#tbt" "with" "in" "winter"
#> [9] "@jerryjrfalwell" "joining" "in" "winter"
#> [11] "me" "in" "this" "past"
#> [13] "davenport" "this" "past" "#maga"
#> [15] "past" "winter" "https://t.co/a5if0qhn1c"
```

Otro ajuste menor que queremos hacer es eliminar los enlaces a las imágenes:

```
links <- "https://t.co/[A-Za-z\\d]+|>"
campaign_tweets[i,] %>%
 mutate(text = str_replace_all(text, links, "")) %>%
 unnest_tokens(word, text, token = "tweets") %>%
 pull(word)
#> [1] "great" "to" "be" "back"
#> [4] "back" "in" "iowa" "@jerryjrfalwell"
#> [7] "#tbt" "with" "joining" "in"
#> [10] "joining" "me" "in" "past"
#> [13] "davenport" "this" "past" "#maga"
#> [16] "winter" "#maga" "https://t.co/a5if0qhn1c"
```

Ya estamos listos para extraer las palabras de todos nuestros tuits.

```
tweet_words <- campaign_tweets %>%
 mutate(text = str_replace_all(text, links, "")) %>%
 unnest_tokens(word, text, token = "tweets")
```

Y ahora podemos responder a preguntas como “¿cuáles son las palabras más utilizadas?”:

```
tweet_words %>%
 count(word) %>%
 arrange(desc(n)) %>%
 slice(1:10)
#> # A tibble: 10 x 2
#> word n
#> <chr> <int>
#> 1 the 2329
#> 2 to 1410
#> 3 and 1239
#> 4 in 1185
#> 5 i 1143
#> # ... with 5 more rows
```

No es sorprendente que estas sean las palabras principales. Las palabras principales no son informativas. El paquete *tidytext* tiene una base de datos de estas palabras de uso común, denominadas palabras *stop*, en la minería de textos:

```
stop_words
#> # A tibble: 1,149 x 2
#> word lexicon
#> <chr> <chr>
#> 1 a SMART
#> 2 a's SMART
#> 3 able SMART
#> 4 about SMART
#> 5 above SMART
#> # ... with 1,144 more rows
```

Si filtramos las filas que representan las palabras *stop* con `filter(!word %in% stop_words$word)`:

```
tweet_words <- campaign_tweets %>%
 mutate(text = str_replace_all(text, links, "")) %>%
 unnest_tokens(word, text, token = "tweets") %>%
 filter(!word %in% stop_words$word)
```

terminamos con un conjunto mucho más informativo de las 10 palabras más tuiteadas:

```
tweet_words %>%
 count(word) %>%
 top_n(10, n) %>%
 mutate(word = reorder(word, n)) %>%
 arrange(desc(n))
#> # A tibble: 10 x 2
#> word n
#> <fct> <int>
#> 1 #trump2016 414
#> 2 hillary 405
```

```
#> 3 people 303
#> 4 #makeamericagreatagain 294
#> 5 america 254
#> # ... with 5 more rows
```

Una exploración de las palabras resultantes (que no se muestran aquí) 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+$`. Segundo, algunos de nuestros *tokens* provienen de una cita y comienzan con '`.`'. Queremos eliminar el '`.`' cuando está al comienzo de una palabra, así que simplemente usamos `str_replace`. Agregamos estas dos líneas al código anterior para generar nuestra tabla final:

```
tweet_words <- campaign_tweets %>%
 mutate(text = str_replace_all(text, links, "")) %>%
 unnest_tokens(word, text, token = "tweets") %>%
 filter(!word %in% stop_words$word &
 !str_detect(word, "^\d+")) %>%
 mutate(word = str_replace(word, "^'", ""))
```

Ahora que tenemos las palabras en una tabla e información sobre qué dispositivo se usó para componer el tuit, podemos comenzar a explorar qué palabras son más comunes al comparar Android con iPhone.

Para cada palabra, queremos saber si es más probable que provenga de un tuit de Android o un tuit de iPhone. En la Sección 15.10, discutimos el *riesgo relativo* (*odds ratio* en inglés) como un resumen estadístico útil para cuantificar estas diferencias. Para cada dispositivo y una palabra dada, llamémosla `y`, calculamos el riesgo relativo. Aquí tendremos muchas proporciones que son 0, así que usamos la corrección 0.5 descrita en la Sección 15.10.

```
android_iphone_or <- tweet_words %>%
 count(word, source) %>%
 pivot_wider(names_from = "source", values_from = "n", values_fill = 0) %>%
 mutate(or = (Android + 0.5) / (sum(Android) - Android + 0.5) /
 (iPhone + 0.5) / (sum(iPhone) - iPhone + 0.5)))
```

Aquí están los riesgos relativos más altos para Android:

```
android_iphone_or %>% arrange(desc(or))
#> # A tibble: 5,914 x 4
#> word Android iPhone or
#> <chr> <int> <int> <dbl>
#> 1 poor 13 0 23.1
#> 2 poorly 12 0 21.4
#> 3 turnberry 11 0 19.7
#> 4 @cbsnews 10 0 18.0
#> 5 angry 10 0 18.0
#> # ... with 5,909 more rows
```

y los más altos para iPhone:

```
android_iphone_or %>% arrange(or)
#> # A tibble: 5,914 x 4
#> word Android iPhone or
#> <chr> <int> <int> <dbl>
#> 1 #makeamericagreatagain 0 294 0.00142
#> 2 #americafirst 0 71 0.00595
#> 3 #draintheswamp 0 63 0.00670
#> 4 #trump2016 3 411 0.00706
#> 5 #votetromp 0 56 0.00753
#> # ... with 5,909 more rows
```

Dado que varias de estas palabras son palabras generales de baja frecuencia, podemos imponer un filtro basado en la frecuencia total así:

```
android_iphone_or %>% filter(Android+iPhone > 100) %>%
 arrange(desc(or))
#> # A tibble: 30 x 4
#> word Android iPhone or
#> <chr> <int> <int> <dbl>
#> 1 @cnn 90 17 4.44
#> 2 bad 104 26 3.39
#> 3 crooked 156 49 2.72
#> 4 interviewed 76 25 2.57
#> 5 media 76 25 2.57
#> # ... with 25 more rows

android_iphone_or %>% filter(Android+iPhone > 100) %>%
 arrange(or)
#> # A tibble: 30 x 4
#> word Android iPhone or
#> <chr> <int> <int> <dbl>
#> 1 #makeamericagreatagain 0 294 0.00142
#> 2 #trump2016 3 411 0.00706
#> 3 join 1 157 0.00805
#> 4 tomorrow 24 99 0.209
#> 5 vote 46 67 0.588
#> # ... with 25 more rows
```

Ya vemos un patrón en los tipos de palabras que se tuitean más desde un dispositivo que desde otro. Sin embargo, no estamos interesados en palabras específicas sino en el tono. La afirmación de Vaziri es que los tuits de Android son más hiperbólicos. Entonces, ¿cómo podemos verificar esto con datos? *Hipérbole* es un sentimiento difícil de extraer de las palabras, ya que se basa en la interpretación de frases. No obstante, las palabras pueden asociarse con sentimientos más básicos como la ira, el miedo, la alegría y la sorpresa. En la siguiente sección, demostramos el análisis básico de sentimientos.

### 26.3 Análisis de sentimiento

En el análisis de sentimiento, asignamos una palabra a uno o más “sentimientos”. Aunque este enfoque no siempre indentificará sentimientos que dependen del contexto, como el sarcasmo, cuando se realiza en grandes cantidades de palabras, los resúmenes pueden ofrecer información.

El primer paso en el análisis de sentimiento es asignar un sentimiento a cada palabra. Como demostramos, el paquete **tidytext** incluye varios mapas o léxicos. También usaremos el paquete **textdata**.

```
library(tidytext)
library(textdata)
```

El léxico **bing** divide las palabras en sentimientos **positive** y **negative**. Podemos ver esto usando la función **get\_sentiments** de **tidytext**:

```
get_sentiments("bing")
```

El léxico **AFINN** asigna una puntuación entre -5 y 5, con -5 el más negativo y 5 el más positivo. Tengan en cuenta que este léxico debe descargarse la primera vez que llamen a la función **get\_sentiment**:

```
get_sentiments("afinn")
```

Los léxicos **loughran** y **nrc** ofrecen varios sentimientos diferentes. Noten que estos también deben descargarse la primera vez que los usen.

```
get_sentiments("loughran") %>% count(sentiment)
#> # A tibble: 6 x 2
#> sentiment n
#> <chr> <int>
#> 1 constraining 184
#> 2 litigious 904
#> 3 negative 2355
#> 4 positive 354
#> 5 superfluous 56
#> # ... with 1 more row
```

```
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
#> # ... with 5 more rows
```

Para nuestro análisis, estamos interesados en explorar los diferentes sentimientos de cada tuit, por lo que utilizaremos el léxico `nrc`:

```
nrc <- get_sentiments("nrc") %>%
 select(word, sentiment)
```

Podemos combinar las palabras y los sentimientos usando `inner_join`, que solo mantendrá palabras asociadas con un sentimiento. Aquí tenemos 10 palabras aleatorias extraídas de los tuits:

```
tweet_words %>% inner_join(nrc, by = "word") %>%
 select(source, word, sentiment) %>%
 sample_n(5)
#> # A tibble: 5 x 3
#> source word sentiment
#> <chr> <chr> <chr>
#> 1 iPhone failing fear
#> 2 Android proud trust
#> 3 Android time anticipation
#> 4 iPhone horrible disgust
#> 5 Android failing anger
```

Ahora estamos listos para realizar un análisis cuantitativo comparando los sentimientos de los tuits publicados desde cada dispositivo. Podríamos realizar un análisis tuit por tuit, asignando un sentimiento a cada tuit. Sin embargo, esto sería un desafío ya que cada tuit tendrá varios sentimientos adjuntos, uno para cada palabra que aparezca en el léxico. Con fines ilustrativos, realizaremos un análisis mucho más sencillo: contaremos y compararemos las frecuencias de cada sentimiento que aparece en cada dispositivo.

```
sentiment_counts <- tweet_words %>%
 left_join(nrc, by = "word") %>%
 count(source, sentiment) %>%
 pivot_wider(names_from = "source", values_from = "n") %>%
 mutate(sentiment = replace_na(sentiment, replace = "none"))
sentiment_counts
#> # A tibble: 11 x 3
#> sentiment Android iPhone
#> <chr> <int> <int>
#> 1 anger 958 528
#> 2 anticipation 910 715
#> 3 disgust 638 322
#> 4 fear 795 486
#> 5 joy 688 535
#> # ... with 6 more rows
```

Para cada sentimiento, podemos calcular las probabilidades de estar en el dispositivo: proporción de palabras con sentimiento versus proporción de palabras sin. Entonces calculamos el riesgo relativo comparando los dos dispositivos.

```

sentiment_counts %>%
 mutate(Android = Android/ (sum(Android) - Android) ,
 iPhone = iPhone/ (sum(iPhone) - iPhone),
 or = Android/iPhone) %>%
 arrange(desc(or))
#> # A tibble: 11 x 4
#> sentiment Android iPhone or
#> <chr> <dbl> <dbl> <dbl>
#> 1 disgust 0.0299 0.0186 1.61
#> 2 anger 0.0456 0.0309 1.47
#> 3 negative 0.0807 0.0556 1.45
#> 4 sadness 0.0424 0.0301 1.41
#> 5 fear 0.0375 0.0284 1.32
#> # ... with 6 more rows

```

Sí vemos algunas diferencias y el orden es particularmente interesante: ¡los tres sentimientos más grandes son el asco, la ira y lo negativo! ¿Pero estas diferencias son solo por casualidad? ¿Cómo se compara esto si solo estamos asignando sentimientos al azar? A fin de responder a esta pregunta, para cada sentimiento podemos calcular un riesgo relativo y un intervalo de confianza, como se definen en la Sección 15.10. Agregaremos los dos valores que necesitamos para formar una tabla de dos por dos y el riesgo relativo:

```

library(broom)
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
#> # A tibble: 11 x 7
#> sentiment Android iPhone log_or se conf.low conf.high
#> <chr> <int> <int> <dbl> <dbl> <dbl> <dbl>
#> 1 disgust 638 322 0.474 0.0691 0.338 0.609
#> 2 anger 958 528 0.389 0.0552 0.281 0.497
#> 3 negative 1641 929 0.371 0.0424 0.288 0.454
#> 4 sadness 894 515 0.342 0.0563 0.232 0.452
#> 5 fear 795 486 0.280 0.0585 0.165 0.394
#> # ... with 6 more rows

```

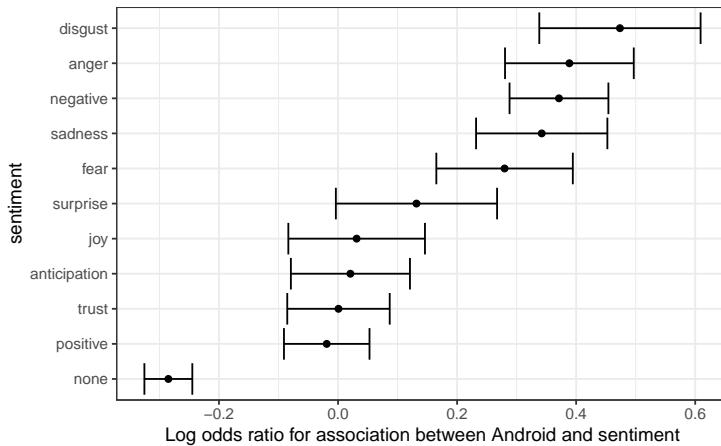
Una visualización gráfica muestra algunos sentimientos que están claramente sobrerepresentados:

```

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()
```



Vemos que el disgusto, la ira, los sentimientos negativos, la tristeza y el miedo están asociados con el Android de una manera que es difícil de explicar solo por casualidad. Las palabras no asociadas con un sentimiento estaban fuertemente asociadas con el iPhone, que está de acuerdo con la afirmación original sobre los tuits hiperbólicos.

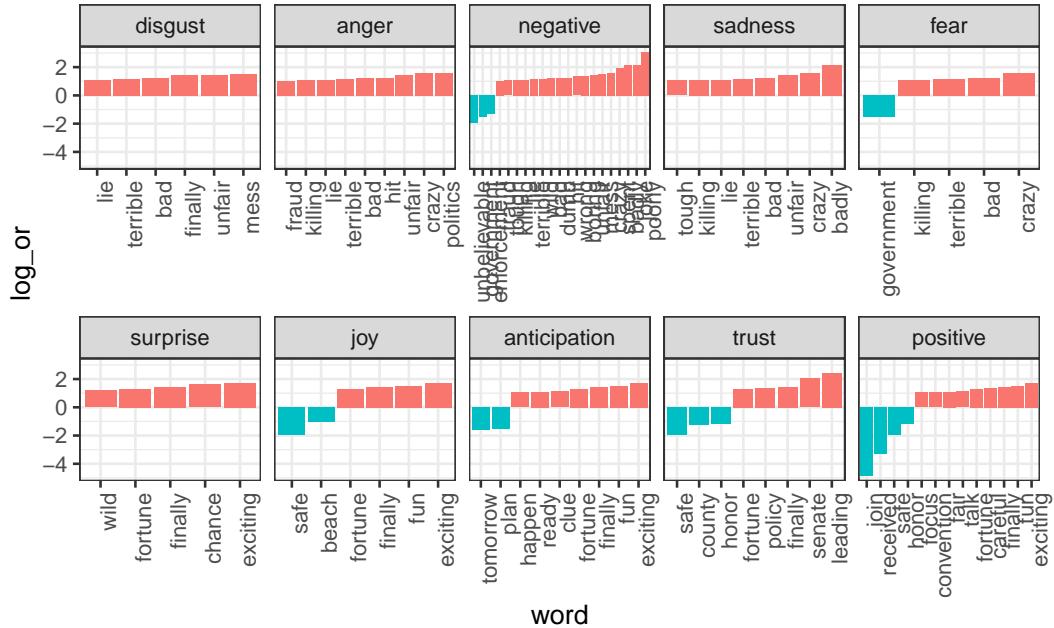
Si estamos interesados en explorar qué palabras específicas están impulsando estas diferencias, podemos referirnos a nuestro objeto `android_iphone_or`:

```
android_iphone_or %>% inner_join(nrc) %>%
 filter(sentiment == "disgust" & Android + iPhone > 10) %>%
 arrange(desc(or))
#> Joining, by = "word"
#> # A tibble: 20 x 5
#> word Android iPhone or sentiment
#> <chr> <int> <int> <dbl> <chr>
#> 1 mess 13 2 4.62 disgust
#> 2 finally 12 2 4.28 disgust
#> 3 unfair 12 2 4.28 disgust
#> 4 bad 104 26 3.39 disgust
#> 5 terrible 31 8 3.17 disgust
#> # ... with 15 more rows
```

y hacer un gráfico:

```
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))
```



Este es solo un ejemplo sencillo de los muchos análisis que uno puede realizar con **tidytext**. Para obtener más información, nuevamente recomendamos el libro *Tidy Text Mining*<sup>4</sup>.

## 26.4 Ejercicios

*Project Gutenberg* es un archivo digital de libros de dominio público. El paquete **gutenbergr** de R facilita la importación de estos textos en R. Puede instalar y cargarlo escribiendo:

```
install.packages("gutenbergr")
library(gutenbergr)
```

Los libros disponibles se pueden ver así:

```
gutenberg_metadata
```

1. Utilice `str_detect` para encontrar la identificación de la novela *Pride and Prejudice*.

<sup>4</sup><https://www.tidytextmining.com/>

2. Observe que hay varias versiones. La función `gutenberg_works()` filtra esta tabla para eliminar réplicas e incluye solo trabajos en inglés. Lea el archivo de ayuda y use esta función para encontrar la identificación de *Pride and Prejudice*.
3. Utilice la función `gutenberg_download` para descargar el texto de *Pride and Prejudice*. Guárdelo en un objeto llamado `book`.
4. Use el paquete `tidytext` para crear una tabla ordenada con todas las palabras en el texto. Guarde la tabla en un objeto llamado `words`.
5. Más adelante haremos un gráfico de sentimiento versus ubicación en el libro. Para esto, será útil agregar una columna a la tabla con el número de palabra.
6. Elimine las palabras `stop` y los números del objeto `words`. Sugerencia: use `anti_join`.
7. Ahora use el léxico AFINN para asignar un valor de sentimiento a cada palabra.
8. Haga un gráfico de puntuación de sentimiento versus ubicación en el libro y agregue un suavizador.
9. Suponga que hay 300 palabras por página. Convierta las ubicaciones en páginas y luego calcule el sentimiento promedio en cada página. Grafique esa puntuación promedio por página. Agregue un suavizador que pase por los datos.

## Part V

# Machine Learning



# 27

---

## Introducción a machine learning

---

Quizás las metodologías de ciencia de datos más populares provienen del campo de *machine learning*. Las historias de éxito de *machine learning* incluyen lectores de códigos postales escritos a mano implementados por el servicio postal, tecnología de reconocimiento de voz como Siri de Apple, sistemas de recomendación de películas, detectores de spam y *malware*, automóviles sin conductor y predictores de precios de viviendas. Aunque hoy en día los términos Inteligencia Artificial y *machine learning* se usan indistintamente, hacemos la siguiente distinción: mientras que los primeros algoritmos de inteligencia artificial, como esos utilizados por las máquinas de ajedrez, implementaron la toma de decisiones según reglas programables derivadas de la teoría o de los primeros principios, en *machine learning* las decisiones de aprendizaje se basan en algoritmos **que se construyen con datos**.

---

### 27.1 Notación

En *machine learning*, los datos se presentan en forma de:

1. el *resultado* (*outcome* en inglés) que queremos predecir y
2. los *atributos* (*features* en inglés) que usaremos para predecir el resultado.

Queremos construir un algoritmo que tome los valores de los atributos como entrada y devuelva una predicción para el resultado cuando no sabemos el resultado. El enfoque de *machine learning* consiste en *entrenar* un algoritmo utilizando un set de datos para el cual conocemos el resultado y luego usar este algoritmo en el futuro para hacer una predicción cuando no sabemos el resultado.

Aquí usaremos  $Y$  para denotar el resultado y  $X_1, \dots, X_p$  para denotar atributos. Tengan en cuenta que los atributos a veces se denominan *predictores* o *covariables*. Consideramos estos sinónimos.

Los problemas de predicción se pueden dividir en resultados categóricos y continuos. Para resultados categóricos,  $Y$  puede ser cualquiera de  $K$  clases. El número de clases puede variar mucho entre distintas aplicaciones. Por ejemplo, en los datos del lector de dígitos,  $K = 10$  con las clases representando los dígitos 0, 1, 2, 3, 4, 5, 6, 7, 8 y 9. En el reconocimiento de voz, los resultados son todas las palabras o frases posibles que estamos tratando de detectar. La detección de spam tiene dos resultados: spam o no spam. En este libro, denotamos las categorías  $K$  con índices  $k = 1, \dots, K$ . Sin embargo, para datos binarios usaremos  $k = 0, 1$  para conveniencias matemáticas que demostraremos más adelante.

La configuración general es la siguiente. Tenemos una serie de características y un resultado desconocido que queremos predecir:

outcome	feature 1	feature 2	feature 3	feature 4	feature 5
?	$X_1$	$X_2$	$X_3$	$X_4$	$X_5$

Para *construir un modelo* que provee una predicción para cualquier conjunto de valores observados  $X_1 = x_1, X_2 = x_2, \dots, X_5 = x_5$ , recolectamos datos para los cuales conocemos el resultado:

outcome	feature 1	feature 2	feature 3	feature 4	feature 5
$y_1$	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{1,4}$	$x_{1,5}$
$y_2$	$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{2,4}$	$x_{2,5}$
:	:	:	:	:	:
$y_n$	$x_{n,1}$	$x_{n,2}$	$x_{n,3}$	$x_{n,4}$	$x_{n,5}$

Cuando el resultado es continuo, nos referimos a la tarea de *machine learning* como *predicción*. El resultado principal del modelo es una función  $f$  que produce automáticamente una predicción, denotada con  $\hat{y}$ , para cualquier conjunto de predictores:  $\hat{y} = f(x_1, x_2, \dots, x_p)$ . Usamos el término *resultado real* (*actual outcome* en inglés) para denotar lo que acabamos observando. Entonces queremos que la predicción  $\hat{y}$  coincida con resultado real  $y$  lo mejor posible. Debido a que nuestro resultado es continuo, nuestras predicciones  $\hat{y}$  no serán exactamente correctas o incorrectas, sino que determinaremos un *error* definido como la diferencia entre la predicción y el resultado real  $y - \hat{y}$ .

Cuando el resultado es categórico, nos referimos a la tarea de *machine learning* como *clasiificación*. El resultado principal de este modelo será una *regla de decisión* (*decision rule* en inglés) que determina cuál de las  $K$  clases debemos predecir. En esta situación, la mayoría de los modelos provee funciones de los predictores para cada clase  $k$ ,  $f_k(x_1, x_2, \dots, x_p)$ , que se utilizan para tomar esta decisión. Cuando los datos son binarios, una regla de decisión típica sería algo como: si  $f_1(x_1, x_2, \dots, x_p) > C$ , pronostique la categoría 1, si no, pronostique la otra categoría, con  $C$  un umbral predeterminado. Debido a que los resultados son categóricos, nuestras predicciones serán correctas o incorrectas.

Tengan en cuenta que estos términos varían entre cursos, libros de texto y otras publicaciones. A menudo, el término *predicción* se usa tanto para resultados categóricos como continuos y el término *regresión* puede usarse para el caso continuo. Aquí no usamos el término *regresión* para evitar confusión con nuestro uso previo del término *regresión lineal*. En la mayoría de los casos, estará claro si nuestros resultados son categóricos o continuos, por lo que evitaremos usar estos términos cuando sea posible.

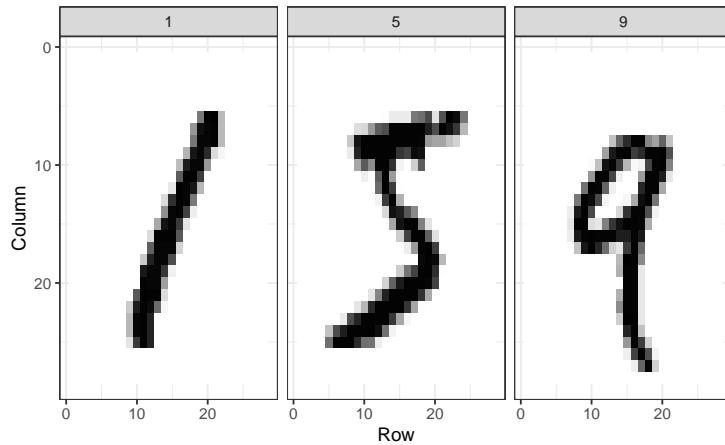
## 27.2 Un ejemplo

Consideremos el ejemplo del lector de código postal. El primer paso para manejar el correo en la oficina de correos es organizar las cartas por código postal:

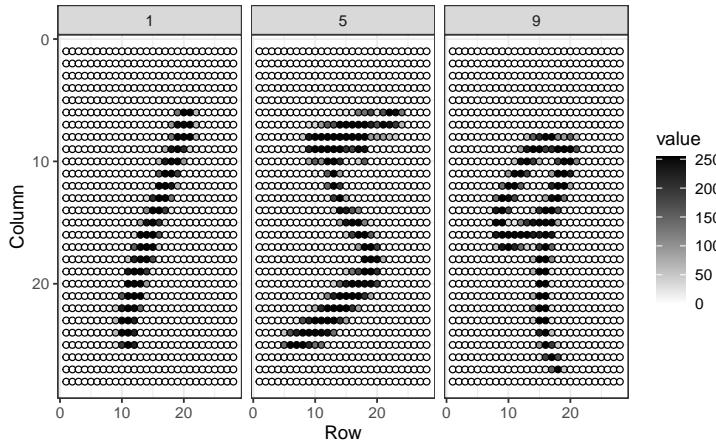


Originalmente, los humanos tenían que clasificarlos a mano. Para hacer esto, tuvieron que leer los códigos postales de cada sobre. Hoy, gracias a los algoritmos de *machine learning*, una computadora puede leer códigos postales y luego un robot clasifica las cartas. En esta parte del libro, aprenderemos cómo construir algoritmos que puedan leer un dígito.

El primer paso para construir un algoritmo es entender cuáles son los resultados y los atributos. A continuación hay tres imágenes de dígitos escritos. Estos ya han sido leídos por un humano y se les ha asignado un resultado  $Y$ . Por lo tanto, se consideran conocidos y sirven como set de entrenamiento.



Las imágenes se convierten en  $28 \times 28 = 784$  píxeles y, para cada píxel, obtenemos una intensidad de escala de grises entre 0 (blanco) y 255 (negro), que consideramos continua por ahora. El siguiente gráfico muestra los atributos individuales de cada imagen:



Para cada imagen digitalizada  $i$ , tenemos un resultado categórico  $Y_i$  que puede ser uno de los 10 valores  $(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)$  y atributos  $X_{i,1}, \dots, X_{i,784}$ . Usamos negrilla  $\mathbf{X}_i = (X_{i,1}, \dots, X_{i,784})$  para distinguir el vector de predictores de los predictores individuales. Cuando nos referimos a un conjunto arbitrario de atributos en lugar de una imagen específica en nuestro set de datos, descartamos el índice  $i$  y usamos  $Y$  y  $\mathbf{X} = (X_1, \dots, X_{784})$ . Utilizamos variables en mayúsculas porque, en general, pensamos en los predictores como variables aleatorias. Usamos minúsculas, por ejemplo  $\mathbf{X} = \mathbf{x}$ , para denotar valores observados. Cuando codificamos usamos minúsculas.

La tarea de *machine learning* es construir un algoritmo que devuelva una predicción para cualquiera de los posibles valores de los atributos. Aquí, aprenderemos varios enfoques para construir estos algoritmos. Aunque en este momento puede parecer imposible lograr esto, comenzaremos con ejemplos sencillos y desarrollaremos nuestro conocimiento hasta que podamos atacar algunos más complejos. De hecho, comenzamos con un ejemplo artificialmente sencillo con un solo predictor y luego pasamos a un ejemplo un poco más realista con dos predictores. Una vez que comprendamos estos, atacaremos algunos retos de *machine learning* del mundo real que involucran muchos predictores.

### 27.3 Ejercicios

1. Para cada uno de los siguientes ejemplos, determine si el resultado es continuo o categórico:
  - a. Lector de dígitos
  - b. Recomendaciones de películas
  - c. Filtro de spam
  - d. Hospitalizaciones
  - e. Siri (reconocimiento de voz)
2. ¿Cuántas funciones tenemos disponibles para la predicción en el set de datos de dígitos?
3. En el ejemplo del lector de dígitos, los resultados se almacenan aquí:

```
library(dslabs)
mnist <- read_mnist()
y <- mnist$train$labels
```

¿Las siguientes operaciones tienen un significado práctico?

```
y[5] + y[6]
y[5] > y[6]
```

Eliga la mejor respuesta:

- a. Sí, porque  $9 + 2 = 11$  y  $9 > 2$ .
- b. No, porque  $y$  no es un vector numérico.
- c. No, porque 11 no es un dígito. Son dos dígitos.
- d. No, porque estas son etiquetas que representan una categoría, no un número. Un 9 representa una clase, no el número 9.

---

## 27.4 Métricas de evaluación

Antes de comenzar a describir enfoques para optimizar la forma en que construimos algoritmos, primero debemos definir a qué nos referimos cuando decimos que un enfoque es mejor que otro. En esta sección, nos centramos en describir las formas en que se evalúan los algoritmos de *machine learning*. Específicamente, necesitamos cuantificar lo que queremos decir con “mejor”.

Para nuestra primera introducción a los conceptos de *machine learning*, comenzaremos con un ejemplo aburrido y sencillo: cómo predecir sexo basado en altura. A medida que explicamos *machine learning* paso a paso, este ejemplo nos permitirá establecer el primer componente básico. Muy pronto, estaremos atacando desafíos más interesantes. Utilizamos el paquete **caret**, que tiene varias funciones útiles para construir y evaluar métodos de *machine learning*. Presentamos los detalles de este paquete en el Capítulo 30.

```
library(tidyverse)
library(caret)
```

Como primer ejemplo, usamos los datos de altura en **dslabs**:

```
library(dslabs)
data(heights)
```

Comenzamos definiendo el resultado y los predictores.

```
y <- heights$sex
x <- heights$height
```

En este caso, solo tenemos un predictor, altura, mientras que  $y$  es claramente un resultado categórico ya que los valores observados son `Male` o `Female`. Sabemos que no podremos predecir  $Y$  de forma precisa basado en  $X$  porque las alturas promedio masculinas y femeninas no son tan diferentes en relación con la variabilidad dentro del grupo. ¿Pero podemos hacerlo mejor que con simplemente adivinar? Para responder a esta pregunta, necesitamos una definición cuantitativa de “mejor”.

#### 27.4.1 Sets de entrenamiento y de evaluación

En última instancia, un algoritmo de *machine learning* se evalúa basado en cómo funciona en el mundo real con sets de datos completamente nuevos. Sin embargo, cuando desarrollamos un algoritmo, generalmente tenemos un set de datos para el cual conocemos los resultados, como lo hacemos con las alturas: sabemos el sexo de cada estudiante en nuestro set de datos. Por lo tanto, para imitar el proceso de evaluación final, generalmente dividimos los datos en dos partes y actuamos como si no supiéramos el resultado de una de estas. Dejamos de fingir que no conocemos el resultado para evaluar el algoritmo, pero solo *después* de haber terminado de construirlo. Nos referimos al grupo para el que conocemos el resultado y que usamos para desarrollar el algoritmo como el *set de entrenamiento* (*training set* en inglés). Nos referimos al grupo para el que aparentamos no conocer el resultado como el *set de evaluación* (*test set* en inglés).

Una forma estándar de generar los sets de entrenamiento y de evaluación es dividiendo aleatoriamente los datos. El paquete `caret` incluye la función `createDataPartition` que nos ayuda a generar índices para dividir aleatoriamente los datos en sets de entrenamiento y de evaluación:

```
set.seed(2007)
test_index <- createDataPartition(y, times = 1, p = 0.5, list = FALSE)
```

El argumento `times` se usa para definir cuántas muestras aleatorias de índices devolver, el argumento `p` se utiliza para definir qué proporción de los datos está representada por el índice y el argumento `list` se usa para decidir si queremos que los índices se devuelvan como una lista o no. Podemos usar el resultado de la llamada a la función `createDataPartition` para definir los sets de entrenamiento y de evaluación de esta manera:

```
test_set <- heights[test_index,]
train_set <- heights[-test_index,]
```

Ahora desarrollaremos un algoritmo usando **solo** el set de entrenamiento. Una vez que hayamos terminado de desarrollar el algoritmo, lo *congelaremos* y lo evaluaremos utilizando el set de evaluación. La forma más sencilla de evaluar el algoritmo cuando los resultados son categóricos es simplemente informar la proporción de casos que se predijeron correctamente en el set de evaluación. Esta métrica generalmente se conoce como *exactitud general* (*overall accuracy* en inglés).

#### 27.4.2 Exactitud general

Para demostrar el uso de la exactitud general, crearemos dos algoritmos diferentes y los compararemos.

Comencemos desarrollando el algoritmo más sencillo posible: adivinar el resultado.

```
y_hat <- sample(c("Male", "Female"), length(test_index), replace = TRUE)
```

Tengan en cuenta que estamos ignorando completamente el predictor y simplemente adivinando el sexo.

En las aplicaciones de *machine learning*, es útil usar factores para representar los resultados categóricos porque las funciones de R desarrolladas para *machine learning*, como las del paquete **caret**, requieren o recomiendan que los resultados categóricos se codifiquen como factores. Para convertir `y_hat` en factores podemos usar la función `factor`:

```
y_hat <- sample(c("Male", "Female"), length(test_index), replace = TRUE) %>%
 factor(levels = levels(test_set$sex))
```

La *exactitud general* se define simplemente como la proporción general que se predice correctamente:

```
mean(y_hat == test_set$sex)
#> [1] 0.51
```

No es sorprendente que nuestra exactitud sea 50%. ¡Estamos adivinando!

¿Podemos mejorarla? El análisis de datos exploratorios sugiere que sí porque, en promedio, los hombres son un poco más altos que las mujeres:

```
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
```

Pero, ¿cómo usamos esta información? Probemos con otro enfoque sencillo: predecir `Male` si la altura está dentro de dos desviaciones estándar del hombre promedio.

```
y_hat <- ifelse(x > 62, "Male", "Female") %>%
 factor(levels = levels(test_set$sex))
```

La exactitud aumenta de 0.50 a aproximadamente 0.80:

```
mean(y == y_hat)
#> [1] 0.793
```

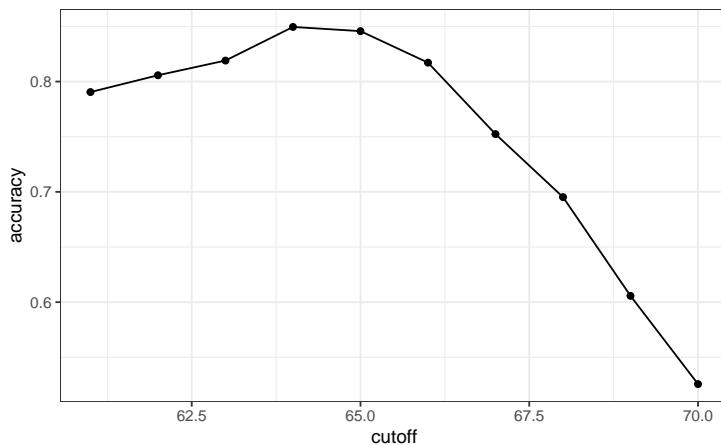
¿Pero podemos mejorarla aún más? En el ejemplo anterior, utilizamos un umbral de 62, pero podemos examinar la exactitud obtenida para otros umbrales y luego elegir el valor que provee los mejores resultados. Sin embargo, recuerden que **es importante que optimicemos el umbral utilizando solo el set de entrenamiento**: el set de evaluación es solo para evaluación. Aunque para este ejemplo sencillo no es un problema, más adelante

aprenderemos que evaluar un algoritmo en el set de entrenamiento puede resultar en un *sobreajuste* (*overfitting* en inglés), que a menudo resulta en evaluaciones peligrosamente sobre optimistas.

Aquí examinamos la exactitud de 10 umbrales diferentes y elegimos el que produce el mejor resultado:

```
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)
})
```

Podemos hacer un gráfico que muestra la exactitud obtenida en el set de entrenamiento para hombres y mujeres:



Vemos que el valor máximo es:

```
max(accuracy)
#> [1] 0.85
```

que es mucho más grande que 0.5. El umbral que resulta en esta exactitud es:

```
best_cutoff <- cutoff[which.max(accuracy)]
best_cutoff
#> [1] 64
```

Ahora podemos evaluar el uso de este umbral en nuestro set de evaluaciones para asegurarnos de que nuestra exactitud no sea demasiado optimista:

```
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] 0.804
```

Vemos que es un poco más baja que la exactitud observada para el set de entrenamiento, pero aún es mejor que adivinar. Y al probar en un set de datos en el que no entrenamos, sabemos que nuestro resultado no se debe a que se haya elegido para dar un buen resultado en el set de evaluación.

### 27.4.3 Matriz de confusión

La regla de predicción que desarrollamos en la sección anterior predice `Male` si el alumno es más alto que 64 pulgadas. Dado que la mujer promedio es aproximadamente 64 pulgadas, esta regla de predicción parece incorrecta. ¿Qué pasó? Si la altura de un estudiante es la de la mujer promedio, ¿no deberíamos predecir `Female`?

En términos generales, la exactitud general puede ser una medida engañosa. Para ver esto, comenzaremos construyendo lo que se conoce como *matriz de confusión* (*confusion matrix* en inglés), que básicamente tabula cada combinación de predicción y valor real. Podemos hacer esto en R usando la función `table`:

```
table(predicted = y_hat, actual = test_set$sex)
#> actual
#> predicted Female Male
#> Female 48 32
#> Male 71 374
```

Si estudiamos esta tabla detenidamente, revela un problema. Si calculamos la exactitud por separado para cada sexo, obtenemos:

```
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 0.403
#> 2 Male 0.921
```

Hay un desequilibrio en la exactitud para hombres y mujeres: se predice que demasiadas mujeres son hombres. ¡Estamos prediciendo que casi la mitad de las mujeres son hombres! ¿Cómo es que nuestra exactitud general sea tan alta? Esto se debe a que la *prevención* de los hombres en este set de datos es alta. Estas alturas se obtuvieron de tres cursos de ciencias de datos, dos de los cuales tenían más hombres matriculados:

```
prev <- mean(y == "Male")
prev
#> [1] 0.773
```

Entonces, al calcular la exactitud general, el alto porcentaje de errores cometidos prediciendo cuáles son mujeres se ve superado por las ganancias en las predicciones acertadas para los hombres. **Esto puede ser un gran problema en machine learning.** Si sus datos

de entrenamiento están sesgados de alguna manera, es probable que también desarrolle algoritmos sesgados. El hecho de que hayamos utilizado un set de evaluación no importa porque también se deriva del set de datos sesgado original. Esta es una de las razones por las que observamos métricas distintas de la exactitud general al evaluar un algoritmo de *machine learning*.

Hay varias métricas que podemos usar para evaluar un algoritmo de manera que la prevalencia no afecte nuestra evaluación y todas estas pueden derivarse de la matriz de confusión. Una forma general de mejorar el uso de la exactitud general es estudiar la *sensibilidad* y la *especificidad* por separado.

#### 27.4.4 Sensibilidad y especificidad

Para definir la sensibilidad y la especificidad, necesitamos un resultado binario. Cuando los resultados son categóricos, podemos definir estos términos para una categoría específica. En el ejemplo de dígitos, podemos pedir la especificidad en el caso de predecir correctamente 2 en lugar de algún otro dígito. Una vez que especifiquemos una categoría de interés, podemos hablar sobre resultados positivos,  $Y = 1$ , y resultados negativos,  $Y = 0$ .

En general, la *sensibilidad* se define como la capacidad de un algoritmo para predecir un resultado positivo cuando el resultado real es positivo:  $\hat{Y} = 1$  cuando  $Y = 1$ . Un algoritmo que predice que todo es positivo ( $\hat{Y} = 1$  pase lo que pase) tiene una sensibilidad perfecta, pero esta métrica por sí sola no es suficiente para evaluar un algoritmo. Por esta razón, también examinamos la *especificidad*, que generalmente se define como la capacidad de un algoritmo para no predecir un resultado positivo  $\hat{Y} = 0$  cuando el resultado real no es positivo  $Y = 0$ . Podemos resumir de la siguiente manera:

- Alta sensibilidad:  $Y = 1 \implies \hat{Y} = 1$
- Alta especificidad:  $Y = 0 \implies \hat{Y} = 0$

Aunque lo anterior a menudo se considera la definición de especificidad, otra forma de pensar en la especificidad es por la proporción de predicciones positivas que realmente son positivas:

- Alta especificidad:  $\hat{Y} = 1 \implies Y = 1$ .

Para ofrecer definiciones precisas, nombramos las cuatro entradas de la matriz de confusión:

	Actually Positive	Actually Negative
Predicted positive	True positives (TP)	False positives (FP)
Predicted negative	False negatives (FN)	True negatives (TN)

Típicamente, la sensibilidad se cuantifica con  $TP/(TP + FN)$ , la proporción de positivos verdaderos (la primera columna =  $TP + FN$ ) que se predicen ser positivos ( $TP$ ). Esta cantidad se conoce como la *tasa de positivos verdaderos* (*true positive rate* o TPR por sus siglas en inglés) o *recall*.

La especificidad se define como  $TN/(TN + FP)$  o la proporción de negativos (la segunda columna =  $FP + TN$ ) que se predicen ser negativos ( $TN$ ). Esta cantidad también se denomina la *tasa de falsos positivos* (*true negative rate* o TNR por sus siglas en inglés). Hay otra forma de cuantificar la especificidad que es  $TP/(TP + FP)$  o la proporción de resultados

que se predicen ser positivos (la primera fila o  $TP + FP$ ) que realmente son positivos ( $TP$ ). Esta cantidad se conoce como *valor predictivo positivo* (*positive predictive value* o PPV por sus siglas en inglés) y también como *precisión*. Tengan en cuenta que, a diferencia de TPR y TNR, la precisión depende de la prevalencia. Por ejemplo, una mayor prevalencia implica que se puede obtener una precisión alta aun cuando están adivinando.

Los diferentes nombres pueden ser confusos, por lo que incluimos una tabla para ayudarnos a recordar los términos. La tabla incluye una columna que muestra la definición si pensamos en las proporciones como probabilidades.

Medida de	Nombre 1	Nombre 2	Definición	Representación de probabilidad
sensibilidad	TPR	Recall	$\frac{TP}{TP+FN}$	$\Pr(\hat{Y} = 1   Y = 1)$
especificidad	TNR	1-FPR	$\frac{TN}{TN+FP}$	$\Pr(\hat{Y} = 0   Y = 0)$
especificidad	PPV	Precisión	$\frac{TP}{TP+FP}$	$\Pr(Y = 1   \hat{Y} = 1)$

Aquí, TPR es la tasa de positivos verdaderos, FPR es la tasa de falsos positivos y PPV es el valor predictivo positivo. La función `confusionMatrix` del paquete `caret` calcula todas estas métricas para nosotros una vez que definamos qué categoría es “positiva”. La función espera factores como entrada y el primer nivel se considera el resultado positivo o  $Y = 1$ . En nuestro ejemplo, `Female` es el primer nivel porque viene antes de `Male` alfabéticamente. Si escriben esto en R, verán varias métricas que incluyen exactitud, sensibilidad, especificidad y PPV.

```
cm <- confusionMatrix(data = y_hat, reference = test_set$sex)
```

Pueden acceder a estos directamente, por ejemplo, así:

```
cm$overall["Accuracy"]
#> Accuracy
#> 0.804
cm$byClass[c("Sensitivity", "Specificity", "Prevalence")]
#> Sensitivity Specificity Prevalence
#> 0.403 0.921 0.227
```

Podemos ver que la alta exactitud general es posible a pesar de la sensibilidad relativamente baja. Como sugerimos anteriormente, la razón por la que esto sucede es debido a la baja prevalencia (0.23): la proporción de mujeres es baja. Como la prevalencia es baja, no predecir mujeres reales como mujeres (baja sensibilidad) no disminuye la exactitud tanto como no predecir hombres reales como hombres (baja especificidad). Este es un ejemplo de por qué es importante examinar la sensibilidad y la especificidad y no solo la exactitud. Antes de aplicar este algoritmo a sets de datos generales, debemos preguntarnos si la prevalencia será la misma.

#### 27.4.5 Exactitud equilibrada y medida $F_1$

Aunque generalmente recomendamos estudiar tanto la especificidad como la sensibilidad, a menudo es útil tener un resumen de un número, por ejemplo, para fines de optimización.

Una medida que se prefiere sobre la exactitud general es el promedio de especificidad y de sensibilidad, conocida como *exactitud equilibrada* (*balanced accuracy* en inglés). Debido a que la especificidad y la sensibilidad son tasas, es más apropiado calcular la *media armónica* (*harmonic average* en inglés). De hecho, la medida  $F_1$  ( $F_1$ -score en inglés), un resumen de un número ampliamente utilizado, es la media armónica de precisión y *recall*:

$$\frac{1}{\frac{1}{2} \left( \frac{1}{\text{recall}} + \frac{1}{\text{precision}} \right)}$$

Dado que es más fácil de escribir, a menudo se ve esta media armónica reescrita como:

$$2 \times \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

cuando se define  $F_1$ .

Recuerden que, según el contexto, algunos tipos de errores son más costosos que otros. Por ejemplo, en el caso de la seguridad de los aviones, es mucho más importante maximizar la sensibilidad sobre la especificidad: no predecir el mal funcionamiento de un avión antes de que se estrelle es un error mucho más costoso que impedir que vuela un avión cuando el avión está en perfectas condiciones. En un caso criminal de asesinato, lo contrario es cierto ya que un falso positivo puede resultar en la ejecución de una persona inocente. La medida  $F_1$  se puede adaptar para pesar la especificidad y la sensibilidad de manera diferente. Para hacer esto, definimos  $\beta$  para representar cuánto más importante es la sensibilidad en comparación con la especificidad y consideramos una media armónica ponderada:

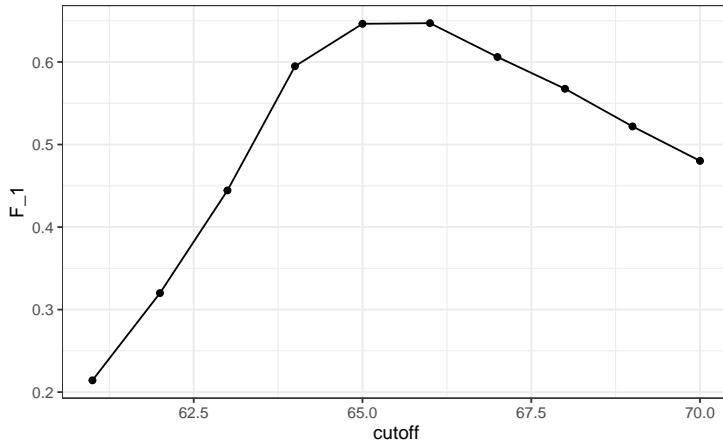
$$\frac{1}{\frac{\beta^2}{1+\beta^2} \frac{1}{\text{recall}} + \frac{1}{1+\beta^2} \frac{1}{\text{precision}}}$$

La función `F_meas` en el paquete `caret` calcula este resumen con un valor de `beta` igual a 1 por defecto.

Reconstruyamos nuestro algoritmo de predicción, pero esta vez maximizando la medida F en lugar de la exactitud 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))
})
```

Como antes, podemos trazar estas medidas  $F_1$  versus los umbrales:



Vemos que el maximo de la medida  $F_1$  es:

```
max(F_1)
#> [1] 0.647
```

Este máximo se logra cuando usamos el siguiente umbral:

```
best_cutoff <- cutoff[which.max(F_1)]
best_cutoff
#> [1] 66
```

Un umbral de 66 tiene más sentido que de 64. Además, equilibra la especificidad y la sensibilidad de nuestra matriz de confusión:

```
y_hat <- ifelse(test_set$height > best_cutoff, "Male", "Female") %>%
 factor(levels = levels(test_set$sex))
sensitivity(data = y_hat, reference = test_set$sex)
#> [1] 0.63
specificity(data = y_hat, reference = test_set$sex)
#> [1] 0.833
```

Ahora vemos que obtenemos mejores resultados que adivinando, que tanto la sensibilidad como la especificidad son relativamente altas y que hemos construido nuestro primer algoritmo de *machine learning*. Este toma altura como predictor y predice mujeres si la persona mide 65 pulgadas o menos.

#### 27.4.6 La prevalencia importa en la práctica

Un algoritmo de *machine learning* con sensibilidad y especificidad muy altas puede ser inútil en la práctica cuando la prevalencia se acerca a 0 o 1. Para ver esto, consideren el caso de una doctora que se especializa en una enfermedad rara y que está interesada en desarrollar un algoritmo para predecir quién tiene la enfermedad. La doctora comparte los datos con ustedes, que entonces desarrollan un algoritmo con una sensibilidad muy alta. Explican que

esto significa que si un paciente tiene la enfermedad, es muy probable que el algoritmo prediga correctamente. También le dicen a la doctora que están preocupados porque, según el set de datos que analizaron, la mitad de los pacientes tienen la enfermedad:  $\Pr(\hat{Y} = 1)$ . La doctora no está preocupada ni impresionada y explica que lo importante es la precisión de la evaluación:  $\Pr(Y = 1|\hat{Y} = 1)$ . Usando el teorema de Bayes, podemos conectar las dos medidas:

$$\Pr(Y = 1 | \hat{Y} = 1) = \Pr(\hat{Y} = 1 | Y = 1) \frac{\Pr(Y = 1)}{\Pr(\hat{Y} = 1)}$$

La doctora sabe que la prevalencia de la enfermedad es de 5 en 1,000, lo que implica que  $\Pr(Y = 1)/\Pr(\hat{Y} = 1) = 1/100$  y, por lo tanto, la precisión de su algoritmo es inferior a 0.01. La doctora no tiene mucho uso para su algoritmo.

#### 27.4.7 Curvas ROC y precision-recall

Al comparar los dos métodos (adivinar versus usar un umbral de altura), comparamos la exactitud y  $F_1$ . El segundo método claramente superó al primero. Sin embargo, si bien consideramos varios umbrales para el segundo método, para el primero solo consideramos un enfoque: adivinar con igual probabilidad. Noten que adivinar `Male` con mayor probabilidad nos daría una mayor exactitud debido al sesgo en 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] 0.739
```

Pero, como se describió anteriormente, esto tendría el costo de una menor sensibilidad. Las curvas que describimos en esta sección nos ayudarán a ver esto.

Recuerden que para cada uno de estos parámetros, podemos obtener una sensibilidad y especificidad diferente. Por esta razón, un enfoque muy común para evaluar métodos es compararlos gráficamente trazando ambos.

Un gráfico ampliamente utilizado que hace esto es la curva *característica operativa del receptor* (*Receiver Operating Characteristic* o ROC por sus siglas en inglés). Para aprender más sobre el origen del nombre, pueden consultar la página de Wikipedia Curva ROC<sup>1</sup>.

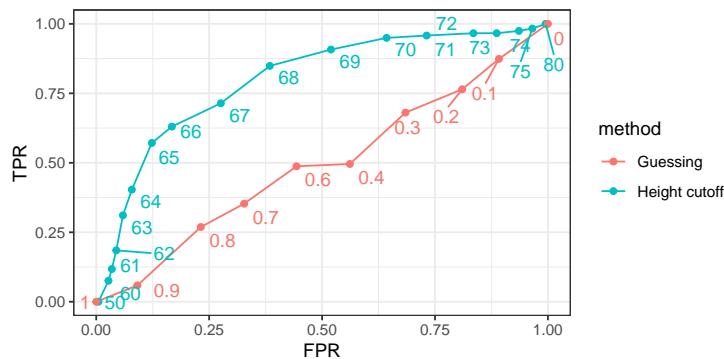
La curva ROC representa la sensibilidad (TPR) frente a la especificidad 1 o la tasa de falsos positivos (FPR). Aquí calculamos el TPR y el FPR necesarios para diferentes probabilidades de adivinar `Male`:

```
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"))
```

<sup>1</sup>[https://es.wikipedia.org/wiki/Curva\\_ROC](https://es.wikipedia.org/wiki/Curva_ROC)

```
list(method = "Guessing",
 FPR = 1 - specificity(y_hat, test_set$sex),
 TPR = sensitivity(y_hat, test_set$sex))
})
```

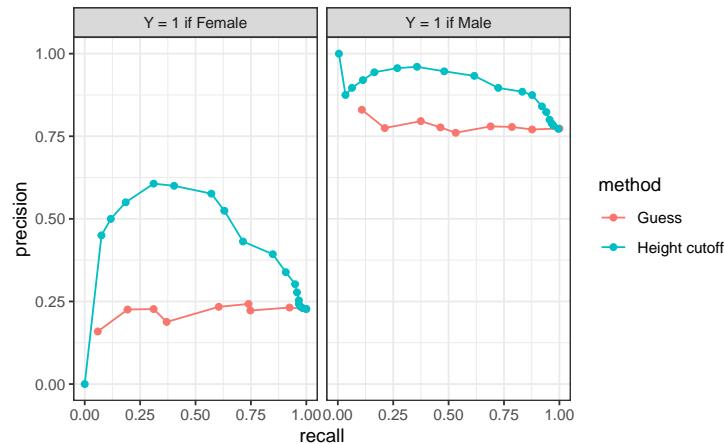
Podemos usar un código similar para calcular estos valores para nuestro segundo enfoque. Al graficar ambas curvas juntas, podemos comparar la sensibilidad para diferentes valores de especificidad:



Vemos que obtenemos una mayor sensibilidad con este enfoque para todos los valores de especificidad, lo que implica que es un método mejor. Tengan en cuenta que si simplemente adivinamos, las curvas ROC caen en la línea de identidad. También noten que cuando hacemos curvas ROC, a veces ayuda agregar el umbral asociado con cada punto al gráfico.

Los paquetes **pROC** y **plotROC** son útiles para generar estos gráficos.

Las curvas ROC tienen una debilidad y es que ninguna de las medidas graficadas depende de la prevalencia. En los casos en que la prevalencia es importante, en su lugar podemos hacer un gráfico *precision-recall*. La idea es similar, pero en cambio graficamos la precisión versus el *recall*:



En este gráfico inmediatamente vemos que la precisión de adivinar no es alta. Esto se

debe a que la prevalencia es baja. También vemos que si cambiamos los positivos para que representen “Male” en lugar de “Female”, la curva ROC permanece igual, pero el gráfico *precision-recall* cambia.

#### 27.4.8 Función de pérdida

Hasta ahora hemos descrito métricas de evaluación que se aplican exclusivamente a datos categóricos. Específicamente, para los resultados binarios, hemos descrito cómo la sensibilidad, especificidad, exactitud y  $F_1$  se pueden utilizar como cuantificación. Sin embargo, estas métricas no son útiles para resultados continuos. En esta sección, describimos cómo el enfoque general para definir “mejor” en *machine learning* es definir una *función de pérdida* (*loss function* en inglés), que puede aplicarse tanto a datos categóricos como continuos.

La función de pérdida más utilizada es la función de pérdida al cuadrado. Si  $\hat{y}$  es nuestro predictor e  $y$  es el resultado observado, la función de pérdida al cuadrado es simplemente:

$$(\hat{y} - y)^2$$

Debido a que frecuentemente tenemos un set de evaluaciones con muchas observaciones, digamos  $N$ , usamos el *error cuadrático medio* (*mean squared error* o MSE por sus siglas en inglés):

$$\text{MSE} = \frac{1}{N} \text{RSS} = \frac{1}{N} \sum_{i=1}^N (\hat{y}_i - y_i)^2$$

En la práctica, a menudo indicamos la *raíz de la desviación cuadrática media* (*root mean squared error* o RMSE por sus siglas en inglés), que es  $\sqrt{\text{MSE}}$ , porque está en las mismas unidades que los resultados. Pero hacer las matemáticas muchas veces es más fácil con el MSE y, por lo tanto, se usa más en los libros de texto, ya que estos generalmente describen las propiedades teóricas de los algoritmos.

Si los resultados son binarios, tanto RMSE como MSE son equivalentes a la exactitud menos uno, ya que  $(\hat{y} - y)^2$  es 0 si la predicción fue correcta y 1 en caso contrario. En general, nuestro objetivo es construir un algoritmo que minimice la pérdida para que esté lo más cerca posible a 0.

Debido a que nuestros datos son generalmente una muestra aleatoria, podemos pensar en el MSE como una variable aleatoria y el MSE observado puede considerarse como una estimación del MSE esperado, que en notación matemática escribimos así:

$$\mathbb{E} \left\{ \frac{1}{N} \sum_{i=1}^N (\hat{Y}_i - Y_i)^2 \right\}$$

Este es un concepto teórico porque en la práctica solo tenemos un set de datos con el cual trabajar. Una forma de pensar en lo que es esta expectativa teórica es la siguiente: tomamos un gran número (llámelo  $B$ ) de muestras aleatorias, aplicamos nuestro algoritmo a cada muestra aleatoria, calculamos el MSE observado y tomamos el promedio. Podemos escribir la siguiente fórmula para esta cantidad:

$$\frac{1}{B} \sum_{b=1}^B \frac{1}{N} \sum_{i=1}^N (\hat{y}_i^b - y_i^b)^2$$

con  $y_i^b$  denotando la observación  $i$  en la muestra aleatoria  $b$  e  $\hat{y}_i^b$  denotando la predicción resultante obtenida de aplicar exactamente el mismo algoritmo a la muestra aleatoria  $b$ . De nuevo, en la práctica solo observamos una muestra aleatoria, por lo que el MSE esperado es solo teórico. Sin embargo, en el Capítulo 29, describimos un enfoque para estimar el MSE que trata de imitar esta cantidad teórica.

Tengan en cuenta que hay funciones de pérdida distintas de la función de pérdida cuadrática. Por ejemplo, el *error medio absoluto* (*mean absolute error* en inglés) utiliza valores absolutos,  $|\hat{Y}_i - Y_i|$  en lugar de cuadrar los errores  $(\hat{Y}_i - Y_i)^2$ . Sin embargo, en este libro nos enfocamos en minimizar la función de pérdida cuadrática ya que es la más utilizada.

## 27.5 Ejercicios

Los sets de datos `reported_height` y `height` se recopilaron de tres clases impartidas en los Departamentos de Ciencias Computacionales y Bioestadística, así como de forma remota a través de la Escuela de Extensión. La clase de bioestadística se impartió en 2016 junto con una versión en línea ofrecida por la Escuela de Extensión. El 25 de enero de 2016 a las 8:15 a.m., durante una de las clases, los instructores le pidieron a los estudiantes que completaran el cuestionario de sexo y altura que poblaba el set de datos `reported_height`. Los estudiantes en línea completaron la encuesta durante los próximos días, después de que la conferencia se publicara en línea. Podemos usar esta información para definir una variable, llamarla `type`, para denotar el tipo de estudiante: `inclass` (presenciales) o `online` (en línea):

```
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),
 "inclass", "online")) %>% select(sex, type)
x <- dat$type
y <- factor(dat$sex, c("Female", "Male"))
```

1. Muestre estadísticas de resumen que indican que el `type` es predictivo del sexo.
2. En lugar de usar la altura para predecir el sexo, use la variable `type`.
3. Muestre la matriz de confusión.
4. Utilice la función `confusionMatrix` en el paquete `caret` para indicar la exactitud.
5. Ahora use las funciones `sensitivity` y `specificity` para indicar especificidad y sensibilidad.
6. ¿Cuál es la prevalencia (%) de mujeres) en el set de datos `dat` definido anteriormente?

## 27.6 Probabilidades y expectativas condicionales

En las aplicaciones de *machine learning*, rara vez podemos predecir los resultados perfectamente. Por ejemplo, los detectores de spam a menudo no detectan correos electrónicos que son claramente spam, Siri no siempre entiende las palabras que estamos diciendo y su banco a veces piensa que su tarjeta fue robada cuando no fue así. La razón más común para no poder construir algoritmos perfectos es que es imposible. Para entender esto, noten que la mayoría de los sets de datos incluirán grupos de observaciones con los mismos valores exactos observados para todos los predictores, pero con diferentes resultados. Debido a que nuestras reglas de predicción son funciones, entradas iguales (los predictores) implican que los resultados (los atributos/las predicciones) tienen que ser iguales. Por lo tanto, para un set de datos en el que los mismos predictores se asocian con diferentes resultados en diferentes observaciones individuales, es imposible predecir correctamente para todos estos casos. Vimos un ejemplo sencillo de esto en la sección anterior: para cualquier altura dada  $x$ , tendrán hombres y mujeres que son  $x$  pulgadas de alto.

Sin embargo, nada de esto significa que no podamos construir algoritmos útiles que sean mucho mejores que adivinar y que en algunos casos sean mejores que las opiniones de expertos. Para lograr esto de manera óptima, hacemos uso de representaciones probabilísticas del problema basadas en las ideas presentadas en la Sección 17.3. Las observaciones con los mismos valores observados para los predictores pueden ser desiguales, pero podemos suponer que todas tienen la misma probabilidad de esta clase o de esa clase. Escribiremos esta idea matemáticamente para el caso de datos categóricos.

### 27.6.1 Probabilidades condicionales

Usamos la notación  $(X_1 = x_1, \dots, X_p = x_p)$  para representar el hecho de que hemos observado valores  $x_1, \dots, x_p$  para covariables  $X_1, \dots, X_p$ . Esto no implica que el resultado  $Y$  tomará un valor específico. En cambio, implica una probabilidad específica. En particular, denotamos las *probabilidades condicionales* para cada clase  $k$ :

$$\Pr(Y = k | X_1 = x_1, \dots, X_p = x_p), \text{ for } k = 1, \dots, K$$

Para evitar escribir todos los predictores, utilizamos letras en negrilla así:  $\mathbf{X} \equiv (X_1, \dots, X_p)$  y  $\mathbf{x} \equiv (x_1, \dots, x_p)$ . También usaremos la siguiente notación para la probabilidad condicional de ser clase  $k$ :

$$p_k(\mathbf{x}) = \Pr(Y = k | \mathbf{X} = \mathbf{x}), \text{ for } k = 1, \dots, K$$

**Ojo:** Utilizaremos la notación  $p(x)$  para representar probabilidades condicionales como funciones de los predictores. No lo confundan con el  $p$  que representa el número de predictores.

Estas probabilidades guían la construcción de un algoritmo que mejora la predicción: para cualquier  $\mathbf{x}$ , vamos a predecir la clase  $k$  con la mayor probabilidad entre  $p_1(x), p_2(x), \dots, p_K(x)$ . En notación matemática, lo escribimos así:  $\hat{Y} = \max_k p_k(\mathbf{x})$ .

En *machine learning*, nos referimos a esto como la *Regla de Bayes*. Pero recuerden que esta es una regla teórica ya que en la práctica no sabemos  $p_k(\mathbf{x}), k = 1, \dots, K$ . De hecho, estimar

estas probabilidades condicionales puede considerarse como el principal desafío de *machine learning*. Cuanto mejores sean nuestros estimadores de la probabilidad  $\hat{p}_k(\mathbf{x})$ , mejor será nuestro predictor:

$$\hat{Y} = \max_k \hat{p}_k(\mathbf{x})$$

Entonces, lo que predeciremos depende de dos cosas: 1) cuán cerca están las  $\max_k p_k(\mathbf{x})$  a 1 o 0 (certeza perfecta) y 2) cuán cerca están nuestros estimadores de  $\hat{p}_k(\mathbf{x})$  a  $p_k(\mathbf{x})$ . No podemos hacer nada con respecto a la primera restricción, ya que está determinada por la naturaleza del problema y, por lo tanto, nos dedicaremos a encontrar buenas formas de estimar las probabilidades condicionales. La primera restricción implica que tenemos límites en cuanto a cuán bien puede funcionar hasta el mejor algoritmo posible. Deberían acostumbrarse a la idea de que, si bien en algunos retos podremos lograr una exactitud casi perfecta, por ejemplo con lectores de dígitos, en otros nuestro éxito está restringido por la aleatoriedad del proceso, como con recomendaciones de películas.

Antes de continuar, es importante recordar que definir nuestra predicción maximizando la probabilidad no siempre es óptimo en la práctica y depende del contexto. Como se discutió anteriormente, la sensibilidad y la especificidad pueden diferir en importancia. Pero incluso en estos casos, tener un buen estimador de la  $p_k(x)$ ,  $k = 1, \dots, K$  nos bastará para construir modelos de predicción óptimos, ya que podemos controlar el equilibrio entre especificidad y sensibilidad como queramos. Por ejemplo, simplemente podemos cambiar los umbrales utilizados para predecir un resultado u otro. En el ejemplo del avión, podemos evitar que vuela un avión en cualquier momento en que la probabilidad de mal funcionamiento sea superior a 1 en un millón, en lugar del 1/2 predeterminado que se usa cuando los tipos de error son igualmente indeseados.

### 27.6.2 Expectativas condicionales

Para datos binarios, pueden pensar en la probabilidad  $\Pr(Y = 1 | \mathbf{X} = \mathbf{x})$  como la proporción de 1s en el estrato de la población para la cual  $\mathbf{X} = \mathbf{x}$ . Muchos de los algoritmos que aprenderemos se pueden aplicar tanto a datos categóricos como continuos debido a la conexión entre las *probabilidades condicionales* y las *expectativas condicionales*.

Porque la expectativa es el promedio de los valores  $y_1, \dots, y_n$  en la población, en el caso en que las  $y_s$  son 0 o 1, la expectativa es equivalente a la probabilidad de elegir aleatoriamente un 1 ya que el promedio es simplemente la proporción de 1s:

$$E(Y | \mathbf{X} = \mathbf{x}) = \Pr(Y = 1 | \mathbf{X} = \mathbf{x}).$$

Como resultado, a menudo solo usamos la expectativa para denotar tanto la probabilidad condicional como la expectativa condicional.

Al igual que con los resultados categóricos, en la mayoría de las aplicaciones, los mismos predictores observados no garantizan los mismos resultados continuos. En cambio, suponemos que el resultado sigue la misma distribución condicional. Ahora explicaremos por qué usamos la expectativa condicional para definir nuestros predictores.

### 27.6.3 La expectativa condicional minimiza la función de pérdida cuadrática

¿Por qué nos importa la expectativa condicional en *machine learning*? Se debe a que el valor esperado tiene una propiedad matemática atractiva: minimiza el MSE. Específicamente, de todas las posibles predicciones  $\hat{Y}$ ,

$$\hat{Y} = E(Y | \mathbf{X} = \mathbf{x}) \text{ minimizes } E\{(\hat{Y} - Y)^2 | \mathbf{X} = \mathbf{x}\}$$

Debido a esta propiedad, una descripción sucinta de la tarea principal de *machine learning* es que utilizamos datos para estimar:

$$f(\mathbf{x}) \equiv E(Y | \mathbf{X} = \mathbf{x})$$

para cualquier conjunto de características  $\mathbf{x} = (x_1, \dots, x_p)$ . Por supuesto, esto es más fácil decirlo que hacerlo, ya que esta función puede tomar cualquier forma y  $p$  puede ser muy grande. Consideren un caso en el que solo tenemos un predictor  $x$ . La expectativa  $E\{Y | X = x\}$  puede ser cualquier función de  $x$ : una línea, una parábola, una onda sinusoidal, una función escalón, etc. Se vuelve aún más complicado cuando consideramos instancias con grandes  $p$ , en cuyo caso  $f(\mathbf{x})$  es una función de un vector multidimensional  $\mathbf{x}$ . ¡Por ejemplo, en nuestro ejemplo de lector de dígitos  $p = 784$ ! **La principal forma en que los algoritmos competitivos de machine learning difieren es en su enfoque para estimar esta expectativa.**

## 27.7 Ejercicios

1. Calcule las probabilidades condicionales de ser hombre para el set de datos `heights`. Redondee las alturas a la pulgada más cercana. Grafique la probabilidad condicional estimada  $P(x) = \Pr(\text{Male}|\text{height} = x)$  para cada  $x$ .

2. En el gráfico que acabamos de hacer, vemos una gran variabilidad para valores bajos de altura. Esto se debe a que tenemos pocos puntos de datos en estos estratos. Use la función `quantile` para cuantiles  $0.1, 0.2, \dots, 0.9$  y la función `cut` para asegurar que cada grupo tenga el mismo número de puntos. Sugerencia: para cualquier vector numérico  $\mathbf{x}$ , puede crear grupos basados en cuantiles como este:

```
cut(x, quantile(x, seq(0, 1, 0.1)), include.lowest = TRUE)
```

3. Genere datos a partir de una distribución normal de dos variables utilizando el paquete **MASS** como este:

```
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"))
```

Pueden hacer un gráfico rápido de los datos usando `plot(dat)`. Use un enfoque similar al ejercicio anterior para estimar las expectativas condicionales y haga un gráfico.

## 27.8 Estudio de caso: ¿es un 2 o un 7?

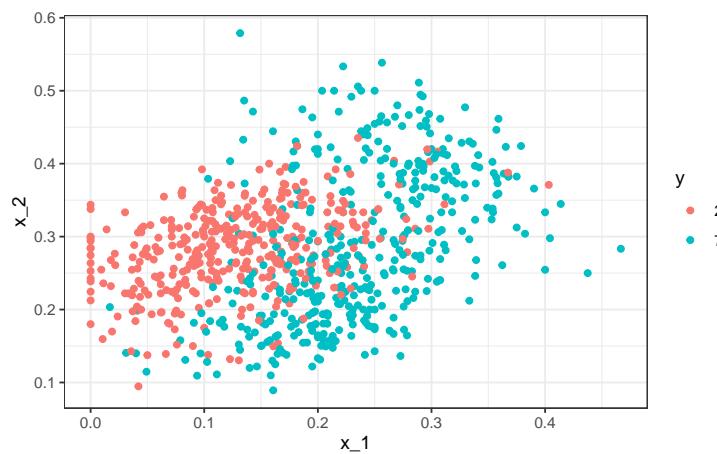
En los dos ejemplos anteriores, solo teníamos un predictor. Realmente no consideramos estos retos de *machine learning*, que se caracterizan por casos con muchos predictores. Volvamos al ejemplo de dígitos en el que teníamos 784 predictores. Para fines ilustrativos, comenzaremos simplificando este problema a uno con dos predictores y dos clases. Específicamente, definimos el desafío como construir un algoritmo que pueda determinar si un dígito es un 2 o 7 de los predictores. No estamos del todo listos para construir algoritmos con 784 predictores, por lo que extraeremos dos predictores sencillos de los 784: la proporción de píxeles oscuros que están en el cuadrante superior izquierdo ( $X_1$ ) y el cuadrante inferior derecho ( $X_2$ ).

Entonces seleccionamos una muestra aleatoria de 1,000 dígitos, 500 en el set de entrenamiento y 500 en el set de evaluación. Proveemos este set de datos en el paquete `dslabs`:

```
library(tidyverse)
library(dslabs)
data("mnist_27")
```

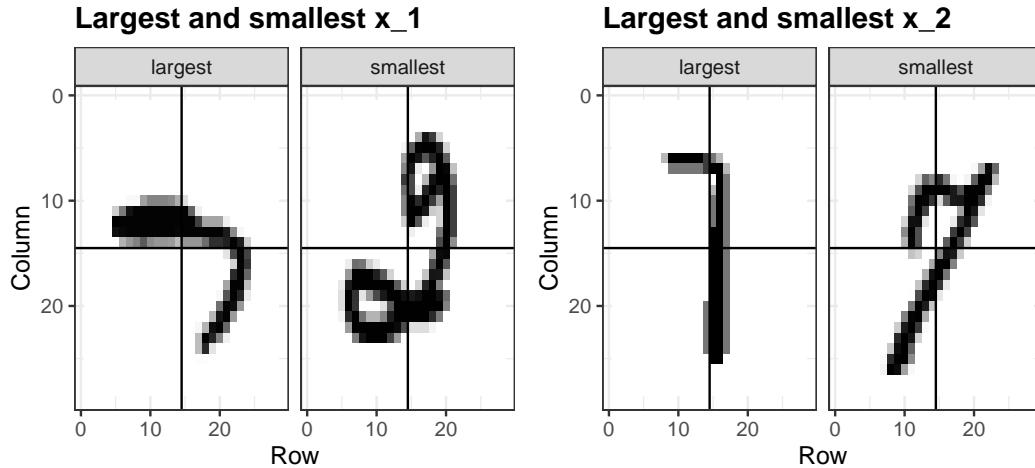
Podemos explorar los datos graficando los dos predictores y usando colores para denotar las etiquetas:

```
mnist_27$train %>% ggplot(aes(x_1, x_2, color = y)) + geom_point()
```



Inmediatamente vemos algunos patrones. Por ejemplo, si  $X_1$  (el panel superior izquierdo) es muy grande, entonces el dígito es probablemente un 7. Además, para valores más pequeños de  $X_1$ , los 2s parecen estar en los valores de rango medio de  $X_2$ .

Para ilustrar como interpretar  $X_1$  y  $X_2$ , incluimos cuatro imágenes como ejemplo. A la izquierda están las imágenes originales de los dos dígitos con los valores más grandes y más pequeños para  $X_1$  y a la derecha tenemos las imágenes correspondientes a los valores más grandes y más pequeños de  $X_2$ :



Comenzamos a tener una idea de por qué estos predictores son útiles, pero también por qué el problema será algo desafiante.

Realmente no hemos aprendido ningún algoritmo todavía, así que intentemos construir un algoritmo usando regresión. El modelo es simplemente:

$$p(x_1, x_2) = \Pr(Y = 1 \mid X_1 = x_1, X_2 = x_2) = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

Lo ajustamos así:

```
fit <- mnist_27$train %>%
 mutate(y = ifelse(y==7, 1, 0)) %>%
 lm(y ~ x_1 + x_2, data = .)
```

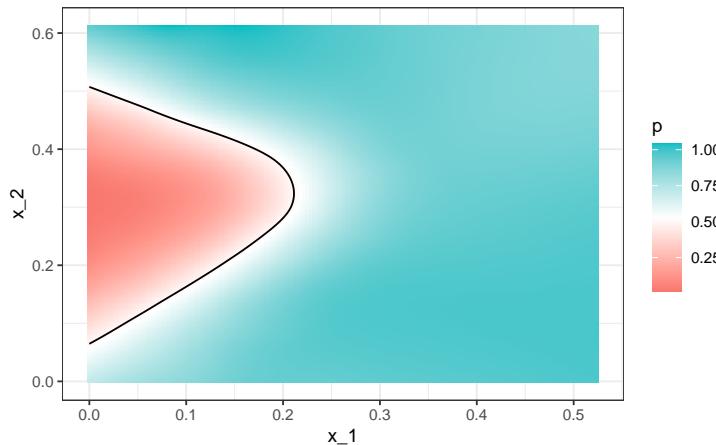
Ahora podemos construir una regla de decisión basada en el estimador  $\hat{p}(x_1, x_2)$ :

```
library(caret)
p_hat <- predict(fit, newdata = mnist_27$test)
y_hat <- factor(ifelse(p_hat > 0.5, 7, 2))
confusionMatrix(y_hat, mnist_27$test$y)$overall[["Accuracy"]]
#> [1] 0.75
```

Obtenemos una exactitud muy superior al 50%. No está mal para nuestro primer intento. ¿Pero podemos mejorar?

Como construimos el ejemplo `mnist_27` y tuvimos a nuestra disposición 60,000 dígitos solo en el set de datos MNIST, lo usamos para construir la distribución condicional *verdadera*  $p(x_1, x_2)$ . Recuerden que esto es algo a lo que no tenemos acceso en la práctica, pero lo incluimos en este ejemplo porque permite comparar  $\hat{p}(x_1, x_2)$  con la verdadera  $p(x_1, x_2)$ . Esta comparación nos enseña las limitaciones de diferentes algoritmos. Hagamos eso aquí. Hemos almacenado el verdadero  $p(x_1, x_2)$  en el objeto `mnist_27` y podemos graficar la imagen usando la función `geom_raster()` de `ggplot2`. Elegimos mejores colores y usamos la función `stat_contour` para dibujar una curva que separa pares  $(x_1, x_2)$  para cual  $p(x_1, x_2) > 0.5$  y pares para cual  $p(x_1, x_2) < 0.5$ :

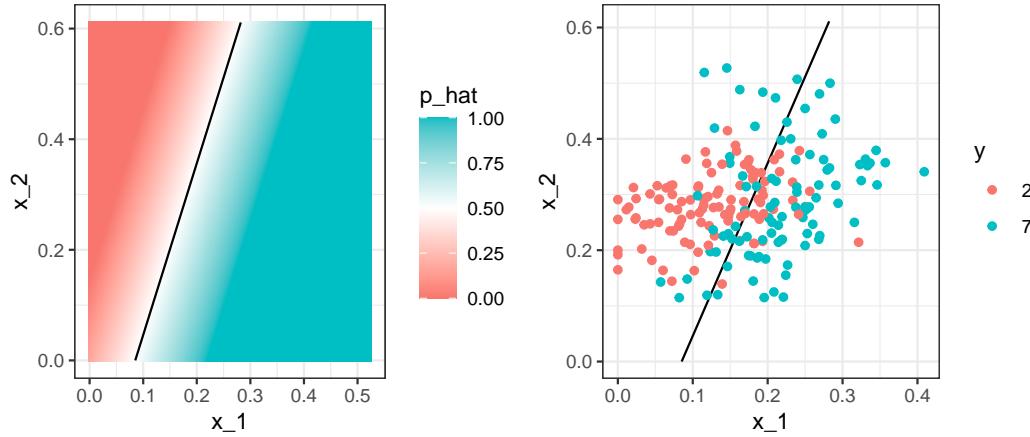
```
mnist_27$true_p %>% ggplot(aes(x_1, x_2, z = p, fill = p)) +
 geom_raster() +
 scale_fill_gradientn(colors=c("#F8766D", "white", "#00BFC4")) +
 stat_contour(breaks=c(0.5), color="black")
```



Arriba vemos un gráfico del verdadero  $p(x, y)$ . Para comenzar a entender las limitaciones de la regresión logística aquí, primero tengan en cuenta que con la regresión logística  $\hat{p}(x, y)$  tiene que ser un plano y, como resultado, el umbral definido por la regla de decisión lo da:  $\hat{p}(x, y) = 0.5$ , lo que implica que el umbral no puede ser otra cosa que una línea recta:

$$\hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 = 0.5 \implies \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 = 0.5 \implies x_2 = (0.5 - \hat{\beta}_0)/\hat{\beta}_2 - \hat{\beta}_1/\hat{\beta}_2 x_1$$

Noten que, para este umbral,  $x_2$  es una función lineal de  $x_1$ . Esto implica que nuestro enfoque de regresión logística no tiene posibilidades de capturar la naturaleza no lineal de la verdadera  $p(x_1, x_2)$ . A continuación se muestra una representación visual de  $\hat{p}(x_1, x_2)$ . Utilizamos la función `squish` del paquete `scales` para restringir los estimados entre 0 y 1. Podemos ver dónde se cometieron los errores al mostrar también los datos y el umbral. Principalmente provienen de valores bajos  $x_1$  que tienen un valor alto o bajo de  $x_2$ . La regresión no puede detectar esto.



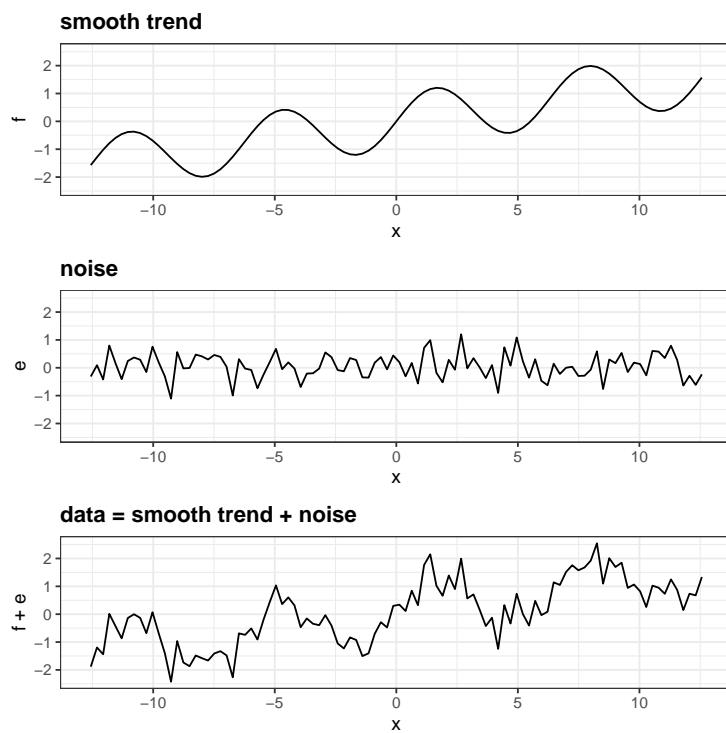
Necesitamos algo más flexible: un método que permita estimadores con formas distintas a un plano.

Vamos a aprender algunos algoritmos nuevos basados en diferentes ideas y conceptos. Pero lo que todos tienen en común es que permiten enfoques más flexibles. Comenzaremos describiendo algoritmos basados en *nearest neighbor* o *kernels*. Para introducir los conceptos detrás de estos enfoques, comenzaremos nuevamente con un ejemplo unidimensional sencillo y describiremos el concepto de *suavización* (*smoothing* en inglés).

# 28

## Suavización

Antes de continuar aprendiendo sobre algoritmos de *machine learning*, presentamos el importante concepto de *suavización* (*smoothing* en inglés). La suavización es una técnica muy poderosa comúnmente usada en el análisis de datos. Otros nombres dados a esta técnica son *ajustamiento de curvas* y *filtro de paso bajo* (*curve fitting* y *low pass filtering* en inglés). La suavización está diseñada para detectar tendencias en presencia de datos ruidosos cuando se desconoce la forma real de la tendencia. El nombre *suavización* proviene del hecho de que para lograr esta hazaña, suponemos que la tendencia es *suave*, como una superficie lisa. En cambio, el ruido (*noise* en inglés), o la desviación de la tendencia, es impredeciblemente ondulante:

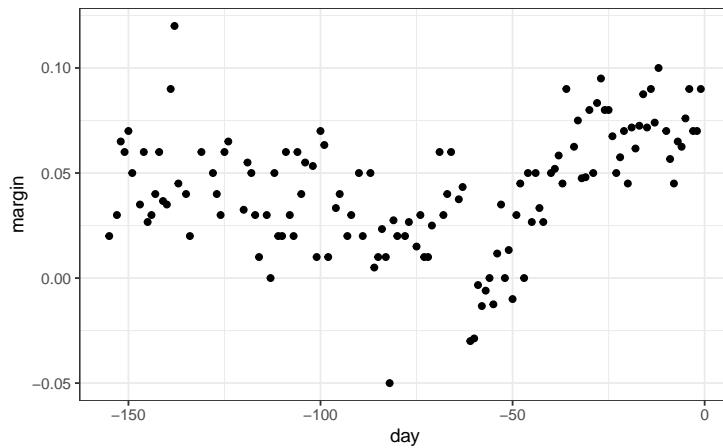


Parte de lo que explicamos en esta sección son los supuestos que nos permiten extraer la tendencia del ruido.

Para entender por qué cubrimos este tema, recuerden que los conceptos detrás de las técnicas de suavización son extremadamente útiles en *machine learning* porque las expectativas/probabilidades condicionales que necesitamos estimar pueden considerarse como tendencias de formas desconocidas afectadas por incertidumbre.

Para explicar estos conceptos, nos enfocaremos primero en un problema con un solo predictor. Específicamente, tratamos de estimar la tendencia temporal en el margen de la encuesta de votación popular de 2008 en Estados Unidos (la diferencia entre Obama y McCain).

```
library(tidyverse)
library(dslabs)
data("polls_2008")
qplot(day, margin, data = polls_2008)
```

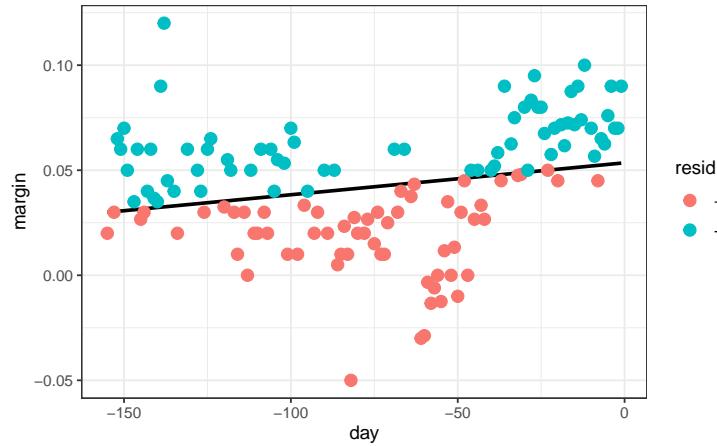


Para los fines de este ejemplo, no lo piensen como un problema de pronóstico. En cambio, simplemente estamos interesados en entender la forma de la tendencia *después* de que terminen las elecciones.

Suponemos que para cualquier día  $x$ , hay una verdadera preferencia entre el electorado  $f(x)$ , pero debido a la incertidumbre introducida por el sondeo, cada punto de datos viene con un error  $\varepsilon$ . Un modelo matemático para el margen de encuesta observado  $Y_i$  es:

$$Y_i = f(x_i) + \varepsilon_i$$

Para pensar en esto como un problema de *machine learning*, recuerden que queremos predecir  $Y$  dado un día  $x$ . Si supiéramos la expectativa condicional  $f(x) = E(Y | X = x)$ , la usaríamos. Pero como no conocemos esta expectativa condicional, tenemos que estimarla. Usemos la regresión, ya que es el único método que hemos aprendido hasta ahora.



La línea que vemos no parece describir muy bien la tendencia. Por ejemplo, el 4 de septiembre (día -62), se celebró la Convención Republicana y los datos sugieren que este evento le dio a John McCain un impulso en las encuestas. Sin embargo, la línea de regresión no captura esta tendencia potencial. Para ver más claramente la *falta de ajuste*, observamos que los puntos por encima de la línea ajustada (azul) y los de abajo (rojo) no se distribuyen uniformemente entre los días. Por lo tanto, necesitamos un enfoque alternativo más flexible.

## 28.1 Suavización de comportamientos

La idea general de la suavización es agrupar los puntos de datos en estratos en los que el valor de  $f(x)$  se puede suponer que es constante. Podemos hacer esta suposición porque pensamos que  $f(x)$  cambia lentamente y, como resultado,  $f(x)$  es casi constante en pequeñas ventanas de tiempo. Un ejemplo de esta idea para los datos `poll_2008` es suponer que la opinión pública se mantuvo aproximadamente igual en el plazo de una semana. Con este supuesto, tenemos varios puntos de datos con el mismo valor esperado.

Si fijamos un día para estar en el centro de nuestra semana, llámelo  $x_0$ , entonces para cualquier otro día  $x$  tal que  $|x - x_0| \leq 3.5$ , suponemos que  $f(x)$  es una constante  $f(x) = \mu$ . Esta suposición implica que:

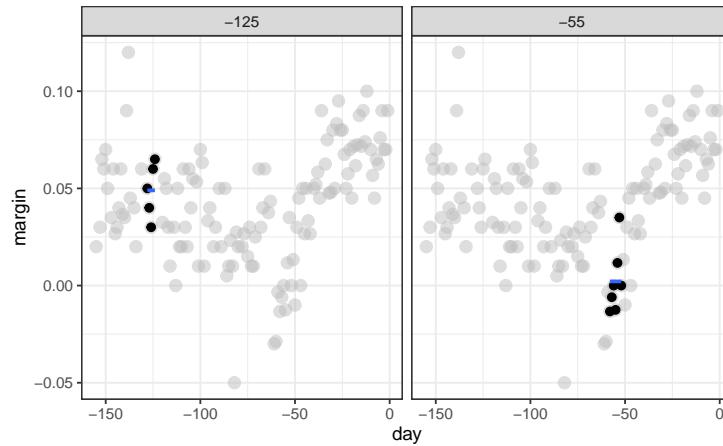
$$E[Y_i | X_i = x_i] \approx \mu \text{ if } |x_i - x_0| \leq 3.5$$

En la suavización, llamamos el tamaño del intervalo que satisface  $|x_i - x_0| \leq 3.5$ , el *tamaño de la ventana, parámetro de suavizado o span*. Más adelante, aprenderemos como intentamos optimizar este parámetro.

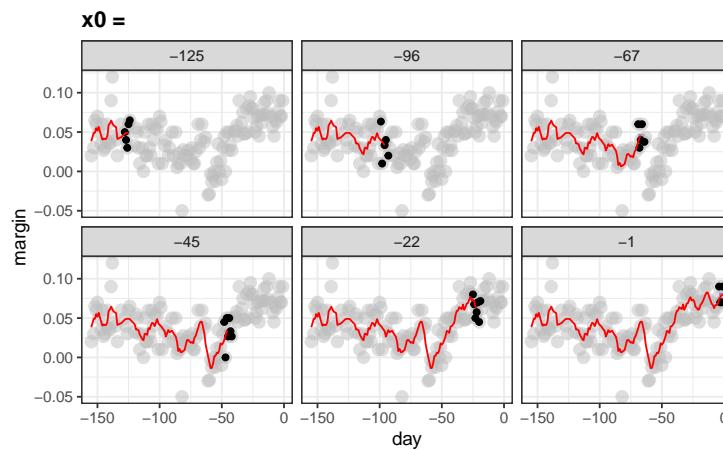
Esta suposición implica que un buen estimador de  $f(x)$  es el promedio de  $Y_i$  valores en la ventana. Si definimos  $A_0$  como el conjunto de índices  $i$  tal que  $|x_i - x_0| \leq 3.5$  y  $N_0$  como el número de índices en  $A_0$ , entonces nuestro estimador es:

$$\hat{f}(x_0) = \frac{1}{N_0} \sum_{i \in A_0} Y_i$$

La idea detrás de la *suavización de compartimientos* (*bin smoothing* en inglés) es hacer este cálculo con cada valor de  $x$  como el centro. En el ejemplo de la encuesta, para cada día, calcularíamos el promedio de los valores dentro de una semana con ese día en el centro. Aquí hay dos ejemplos:  $x_0 = -125$  y  $x_0 = -55$ . El segmento azul representa el promedio resultante.



Al calcular esta media para cada punto, formamos un estimador de la curva subyacente  $f(x)$ . A continuación, mostramos el procedimiento que ocurre a medida que avanzamos de -155 a 0. En cada valor de  $x_0$ , mantenemos el estimador  $\hat{f}(x_0)$  y continuamos al siguiente punto:

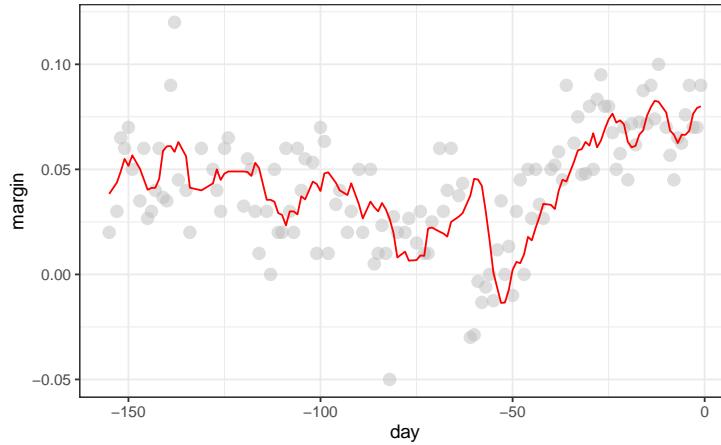


El código final y el estimador resultante se ven así:

```
span <- 7
fit <- with(polls_2008,
 ksmooth(day, margin, kernel = "box", bandwidth = span))

polls_2008 %>% mutate(smooth = fit$y) %>%
 ggplot(aes(day, margin)) +
```

```
geom_point(size = 3, alpha = .5, color = "grey") +
geom_line(aes(day, smooth), color="red")
```



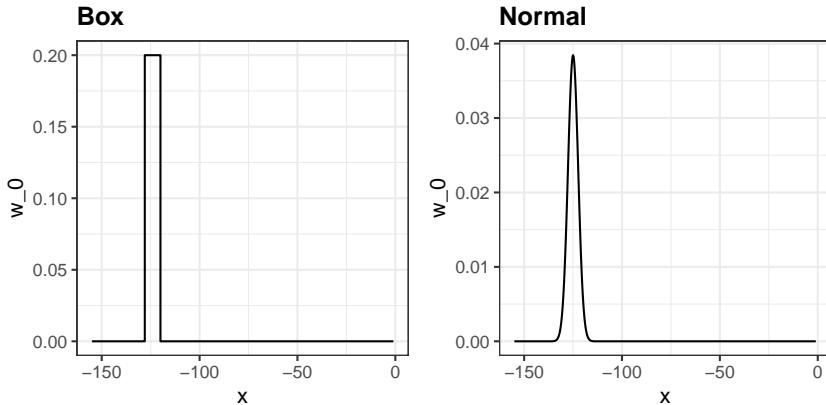
## 28.2 Kernels

El resultado final de la suavización de compartimientos es bastante ondulante. Una de las razones es que cada vez que la ventana se mueve, cambian dos puntos. Podemos atenuar esto algo tomando promedios ponderados que le dan al punto central más peso que a los puntos lejanos, con los dos puntos en los bordes recibiendo muy poco peso.

Pueden pensar en el enfoque de suavización de compartimiento como un promedio ponderado:

$$\hat{f}(x_0) = \sum_{i=1}^N w_0(x_i) Y_i$$

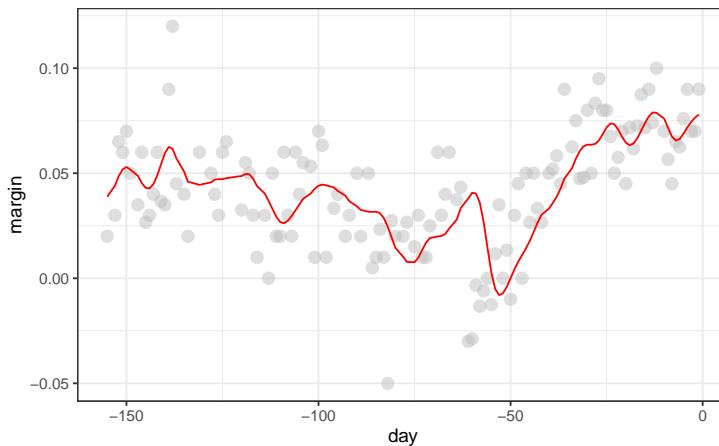
en el que cada punto recibe un peso de 0 o  $1/N_0$ , con  $N_0$  el número de puntos en la semana. En el código anterior, usamos el argumento `kernel="box"` en nuestra llamada a la función `ksmooth`. Esto se debe a que la *función de peso*  $w_0(x)$  parece una caja. La función `ksmooth` ofrece una opción “más suave” que utiliza la densidad normal para asignar pesos.



El código final y el gráfico resultante para el *kernel* normal se ven así:

```
span <- 7
fit <- with(polls_2008,
 ksmooth(day, margin, kernel = "normal", bandwidth = span))

polls_2008 %>% mutate(smooth = fit$y) %>%
 ggplot(aes(day, margin)) +
 geom_point(size = 3, alpha = .5, color = "grey") +
 geom_line(aes(day, smooth), color="red")
```



Observen que el estimador final ahora se ve más suave.

Hay varias funciones en R que implementan suavizadores de comportamientos. Un ejemplo es `ksmooth`, que mostramos arriba. En la práctica, sin embargo, generalmente preferimos métodos que usan modelos ligeramente más complejos que ajustar una constante. El resultado final arriba, por ejemplo, todavía es algo ondulado en partes que no esperamos que sea (entre -125 y -75, por ejemplo). Métodos como `loess`, que explicamos a continuación, mejoran esto.

### 28.3 Regresión ponderada local (loess)

Una limitación del enfoque de suavización de compartimientos que acabamos de describir es que necesitamos ventanas pequeñas para que se cumpla el supuesto de que la función es aproximadamente constante. Como resultado, terminamos con un pequeño número de puntos de datos para promediar y obtener estimaciones imprecisas  $\hat{f}(x)$ . Aquí describimos cómo la *regresión ponderada local* (loess o *local weighted regression* en inglés) nos permite considerar tamaños de ventana más grandes. Para hacer esto, usaremos un resultado matemático, conocido como el teorema de Taylor, que dice que si examinamos muy de cerca cualquier función suave  $f(x)$ , parecerá una línea. Para ver por qué esto tiene sentido, consideren los bordes curvos que hacen los jardineros con palas rectas:



(“Downing Street garden path edge”<sup>1</sup> del usuario de Flickr Número 10<sup>2</sup>. Licencia CC-BY 2.0<sup>3</sup>.)

En lugar de suponer que la función es aproximadamente constante en una ventana, suponemos que la función es localmente lineal. Podemos considerar tamaños de ventana más grandes cuando suponemos que la función es localmente lineal que cuando suponemos que es localmente constante. En lugar de la ventana de una semana, consideraremos una ventana más grande en la que la tendencia es aproximadamente lineal. Comenzamos con una ventana de tres semanas y luego consideramos y evaluamos otras opciones:

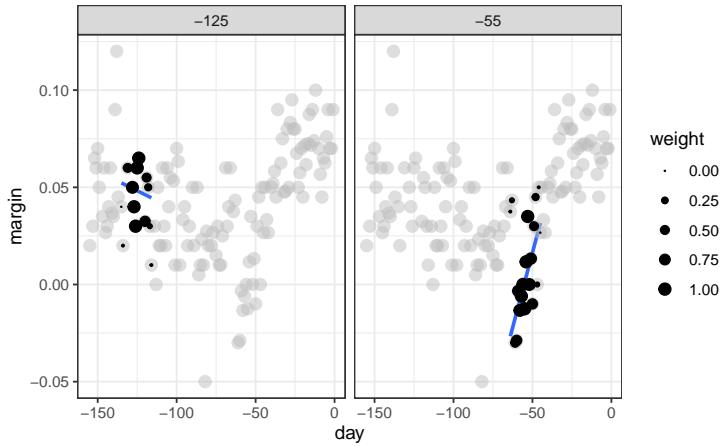
$$E[Y_i|X_i = x_i] = \beta_0 + \beta_1(x_i - x_0) \text{ if } |x_i - x_0| \leq 21$$

Para cada punto  $x_0$ , loess define una ventana y ajusta una línea dentro de esa ventana. Aquí hay un ejemplo que muestra los ajustes para  $x_0 = -125$  y  $x_0 = -55$ :

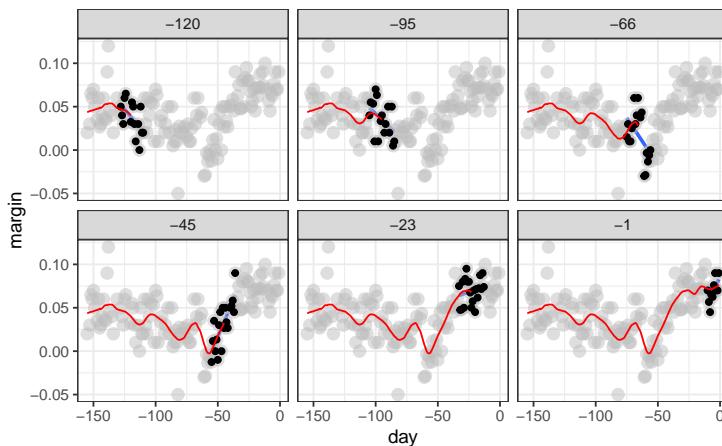
<sup>1</sup><https://www.flickr.com/photos/49707497@N06/7361631644>

<sup>2</sup><https://www.flickr.com/photos/number10gov/>

<sup>3</sup><https://creativecommons.org/licenses/by/2.0/>



El valor ajustado en  $x_0$  se convierte en nuestro estimador  $\hat{f}(x_0)$ . A continuación, mostramos el procedimiento que ocurre mientras cambiamos de -155 a 0.

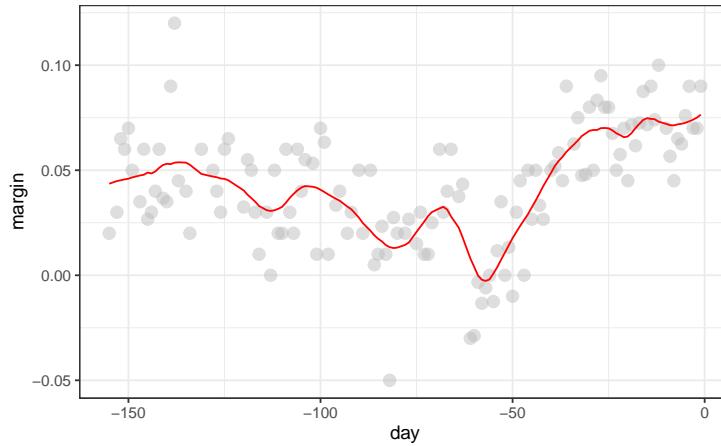


El resultado final es un ajuste más suave que ese producido por la suavización de comparamiento porque utilizamos tamaños de muestra más grandes para estimar nuestros parámetros locales:

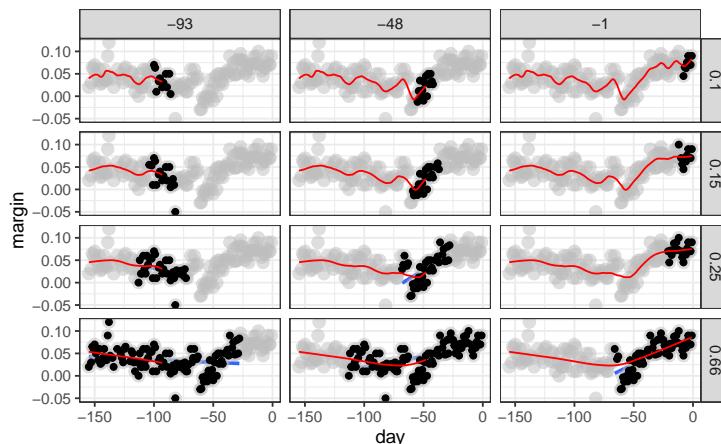
```
total_days <- diff(range(polls_2008$day))
span <- 21/total_days

fit <- loess(margin ~ day, degree=1, span = span, data=polls_2008)

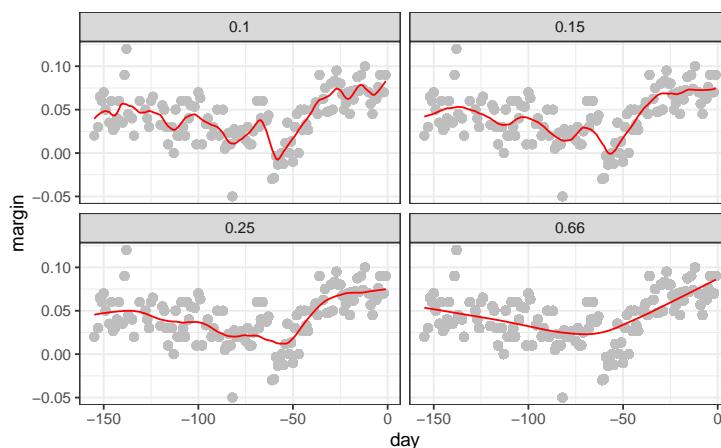
polls_2008 %>% mutate(smooth = fit$fitted) %>%
 ggplot(aes(day, margin)) +
 geom_point(size = 3, alpha = .5, color = "grey") +
 geom_line(aes(day, smooth), color="red")
```



Podemos ver cómo diferentes tamaños de ventanas, *spans*, conducen a diferentes estimadores:



Aquí están los estimadores finales:



Hay otras tres diferencias entre `loess` y el típico suavizador de compartimiento.

1. En vez de mantener el tamaño del compartimiento igual, `loess` mantiene el mismo número de puntos utilizados en el ajuste local. Este número se controla a través del argumento `span`, que espera una proporción. Por ejemplo, si `N` es el número de puntos de datos y `span=0.5`, entonces para un determinado  $x$ , `loess` usará los  $0.5 * N$  puntos más cercanos a  $x$  para el ajuste.

2. Al ajustar una línea localmente, `loess` utiliza un enfoque *ponderado*. Básicamente, en lugar de usar mínimos cuadrados, minimizamos una versión ponderada:

$$\sum_{i=1}^N w_0(x_i) [Y_i - \{\beta_0 + \beta_1(x_i - x_0)\}]^2$$

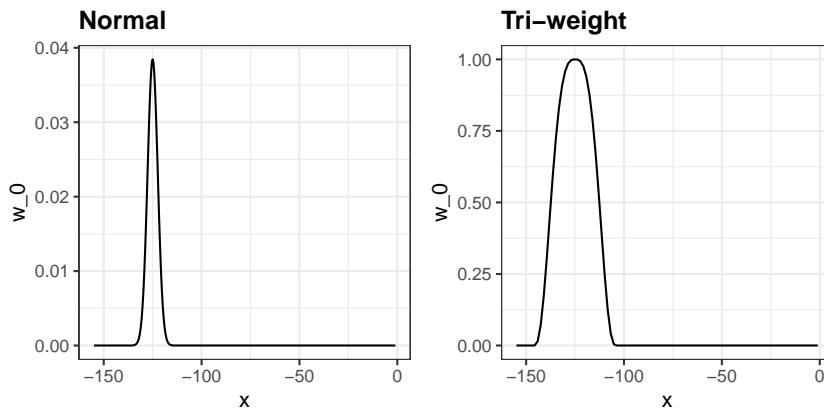
Sin embargo, en lugar del *kernel* gaussiano, `loess` usa una función llamada el *Tukey tri-weight*:

$$W(u) = (1 - |u|^3)^3 \text{ if } |u| \leq 1 \text{ and } W(u) = 0 \text{ if } |u| > 1$$

Para definir los pesos, denotamos  $2h$  como el tamaño de la ventana y definimos:

$$w_0(x_i) = W\left(\frac{x_i - x_0}{h}\right)$$

Este *kernel* difiere del *kernel* gaussiano en que más puntos obtienen valores más cercanos al máximo:



3. `loess` tiene la opción de ajustar el modelo local *robustamente*. Se implementa un algoritmo iterativo en el que, después de ajustar un modelo en una iteración, se detectan valores atípicos y se ponderan hacia abajo para la siguiente iteración. Para usar esta opción, usamos el argumento `family="symmetric"`.

### 28.3.1 Ajustando con paráolas

El teorema de Taylor también nos dice que si miramos cualquier función matemática lo suficientemente cerca, parece una parábola. El teorema además establece que no tienen que

mirar tan de cerca cuando se aproxima con paráolas como cuando se aproxima con líneas. Esto significa que podemos hacer que nuestras ventanas sean aún más grandes y ajustar paráolas en lugar de líneas.

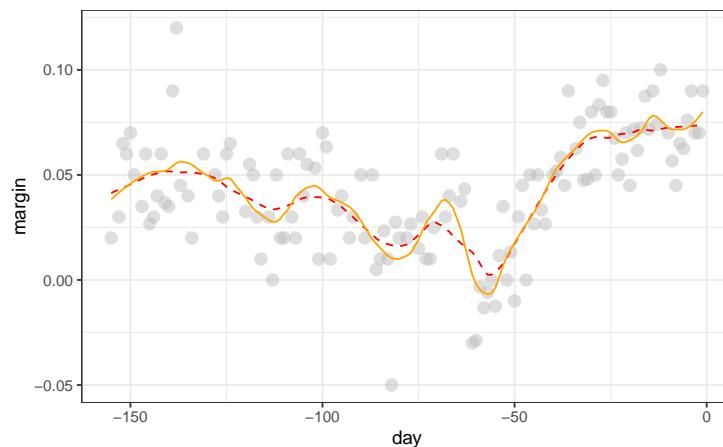
$$E[Y_i|X_i = x_i] = \beta_0 + \beta_1(x_i - x_0) + \beta_2(x_i - x_0)^2 \text{ if } |x_i - x_0| \leq h$$

Este es el procedimiento por defecto de la función `loess`. Es posible que hayan notado que cuando mostramos el código para usar `loess`, configuramos `degree = 1`. Esto le dice a `loess` que se ajuste a polinomios de grado 1, un nombre elegante para líneas. Si leen la página de ayuda para `loess`, verán que, para el argumento `degree`, el valor predeterminado es 2. Por defecto, `loess` se ajusta a paráolas, no a líneas. Aquí hay una comparación de las líneas de ajuste (guiones rojos) y las paráolas de ajuste (naranja sólido):

```
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)
```

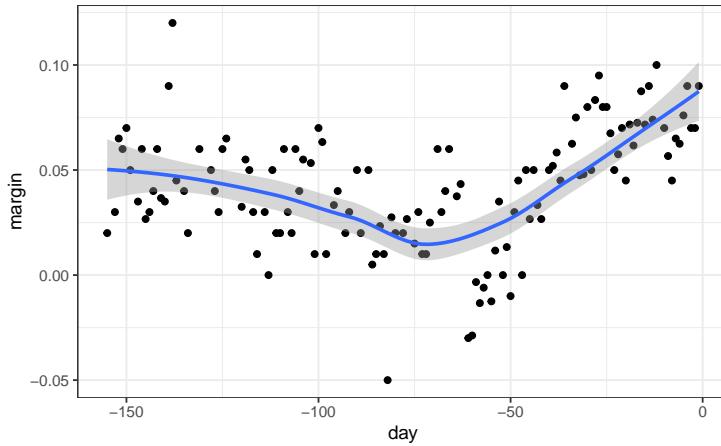


`degree = 2` nos da resultados más ondulantes. Por eso, preferimos `degree = 1` ya que es menos propenso a este tipo de ruido.

### 28.3.2 Cuidado con los parámetros de suavización predeterminados

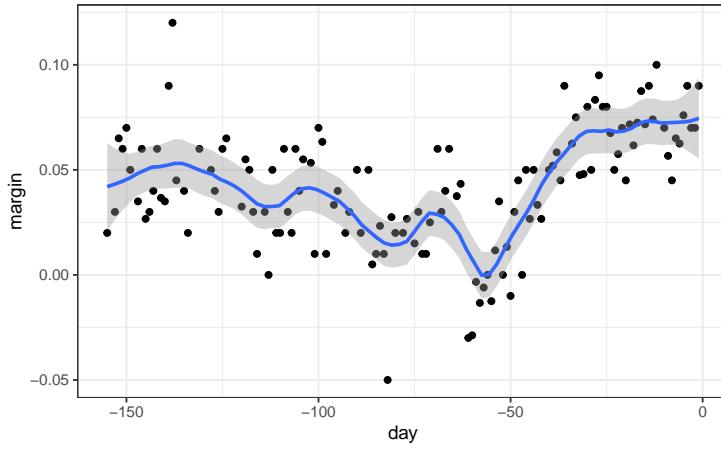
`ggplot2` utiliza `loess` en su función `geom_smooth`:

```
polls_2008 %>% ggplot(aes(day, margin)) +
 geom_point() +
 geom_smooth()
```



Pero tengan cuidado con los parámetros predeterminados ya que rara vez son óptimos. Afortunadamente, pueden cambiarlos fácilmente:

```
polls_2008 %>% ggplot(aes(day, margin)) +
 geom_point() +
 geom_smooth(method = "loess", span = 0.15, method.args = list(degree=1))
```



## 28.4 Conectando la suavización al *machine learning*

Para ver cómo la suavización se relaciona con el *machine learning* usando un ejemplo concreto, consideren nuestro ejemplo introducido en la Sección 27.8. Si definimos el resultado

$Y = 1$  para dígitos que son siete e  $Y = 0$  para dígitos que son 2, entonces estamos interesados en estimar la probabilidad condicional:

$$p(x_1, x_2) = \Pr(Y = 1 \mid X_1 = x_1, X_2 = x_2).$$

con  $X_1$  y  $X_2$  los dos predictores definidos en la Sección 27.8. En este ejemplo, los 0s y 1s que observamos son “ruidosos” porque para algunas regiones las probabilidades  $p(x_1, x_2)$  no están tan cerca de 0 o 1. Por lo tanto, necesitamos estimar  $p(x_1, x_2)$ . La suavización es una alternativa para lograr esto. En la Sección 27.8, vimos que la regresión lineal no era lo suficientemente flexible como para capturar la naturaleza no lineal de  $p(x_1, x_2)$ ; los enfoques de suavización, por ende, pueden proveer una mejora. En el siguiente capítulo, describimos un algoritmo popular de *machine learning*, *k vecinos más cercanos*, que se basa en la suavización por compartimientos.

## 28.5 Ejercicios

1. En la parte de *wrangling* de este libro, utilizamos el siguiente código para obtener recuentos de mortalidad para Puerto Rico para 2015-2018.

```
"JUL" = 7, "AGO" = 8, "SEP" = 9,
"OCT" = 10, "NOV" = 11, "DEC" = 12)) %>%
 mutate(date = make_date(year, month, day)) %>%
 filter(date <= "2018-05-01")
```

Utilice la función `loess` para obtener un estimador uniforme del número esperado de muertes como función de la fecha. Grafique la función suave que resulta. Use un *span* de dos meses.

2. Grafique los estimadores suaves como función del día del año, todas en el mismo gráfico pero con diferentes colores.

3. Suponga que queremos predecir 2s y 7s en nuestro set de datos `mnist_27` con solo la segunda covariable. ¿Podemos hacer esto? A primera vista parece que los datos no tienen mucho poder predictivo. De hecho, si ajustamos una regresión logística regular, el coeficiente para `x_2` no es significativo!

```
library(broom)
library(dslabs)
data("mnist_27")
mnist_27$train %>%
 glm(y ~ x_2, family = "binomial", data = .) %>%
 tidy()
```

Graficar un diagrama de dispersión aquí no es útil ya que y es binario:

```
qplot(x_2, y, data = mnist_27$train)
```

Ajuste una línea loess a los datos anteriores y grafique los resultados. Observe que hay poder predictivo, excepto que la probabilidad condicional no es lineal.

# 29

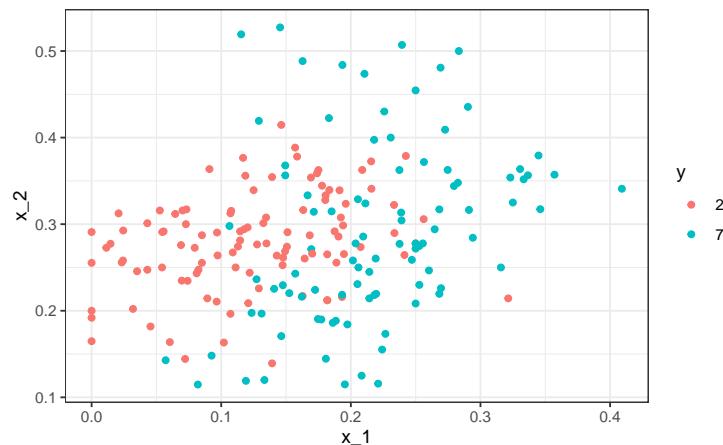
## Validación cruzada

En este capítulo, presentamos la validación cruzada (*cross validation* en inglés), una de las ideas más importantes de *machine learning*. Aquí nos centramos en los aspectos conceptuales y matemáticos. Describiremos cómo implementar la validación cruzada en la práctica con el paquete **caret** en la Sección 30.2 del próximo capítulo. Para motivar el concepto, utilizaremos los dos datos de dígitos predictores presentados en la Sección 27.8 y cubriremos, por primera vez, un algoritmo real de *machine learning*: *k vecinos más cercanos* (kNN o *k-nearest neighbors* en inglés).

### 29.1 Motivación con k vecinos más cercanos

Empecemos cargando los datos y mostrando un gráfico de los predictores con resultados representados con color.

```
library(tidyverse)
library(dslabs)
data("mnist_27")
mnist_27$test %>% ggplot(aes(x_1, x_2, color = y)) + geom_point()
```



Utilizaremos estos datos para estimar la función de probabilidad condicional:

$$p(x_1, x_2) = \Pr(Y = 1 \mid X_1 = x_1, X_2 = x_2)$$

como se define en la Sección 28.4. Con kNN estimamos  $p(x_1, x_2)$  de manera similar a la suavización de compartimientos. Sin embargo, como veremos, kNN es más fácil de adaptar a múltiples dimensiones. Primero, definimos la distancia entre todas las observaciones según los atributos. Entonces, para cualquier punto  $(x_1, x_2)$  para el cual queremos un estimador de  $p(x_1, x_2)$ , buscamos los  $k$  puntos más cercanos a  $(x_1, x_2)$  y calculamos el promedio de los 0s y 1s asociados con estos puntos. Nos referimos al conjunto de puntos utilizados para calcular el promedio como el *vecindario* (*neighborhood* en inglés). Debido a la conexión que describimos anteriormente entre las expectativas condicionales y las probabilidades condicionales, esto nos da un estimador  $\hat{p}(x_1, x_2)$ , al igual que el suavizador de compartimiento nos dio un estimador de una tendencia. Como en el caso de los suavizadores de compartimientos, podemos controlar la flexibilidad de nuestro estimador, en este caso a través del parámetro  $k$ :  $ks$  más grandes resultan en estimadores más suaves, mientras que  $ks$  más pequeñas resultan en estimadores más flexibles y más ondulados.

Para implementar el algoritmo, podemos usar la función `knn3` del paquete **caret**. Mirando el archivo de ayuda para este paquete, vemos que podemos llamarlo de una de dos maneras. Utilizaremos el primero en el que especificamos una *formula* y un *data frame*. El *data frame* contiene todos los datos que se utilizarán. La fórmula tiene la forma `outcome ~ predictor_1 + predictor_2 + predictor_3` y así sucesivamente. Por lo tanto, escribiríamos `y ~ x_1 + x_2`. Si vamos a usar todos los predictores, podemos usar el `.` así `y ~ ..`. La llamada final se ve así:

```
library(caret)
knn_fit <- knn3(y ~ ., data = mnist_27$train)
```

Para esta función, también debemos elegir un parámetro: el número de *vecinos* para incluir. Comencemos con el valor predeterminado  $k = 5$ .

```
knn_fit <- knn3(y ~ ., data = mnist_27$train, k = 5)
```

En este caso, dado que nuestro set de datos es equilibrado y nos preocupamos tanto por la sensibilidad como por la especificidad, utilizaremos la exactitud para cuantificar el rendimiento.

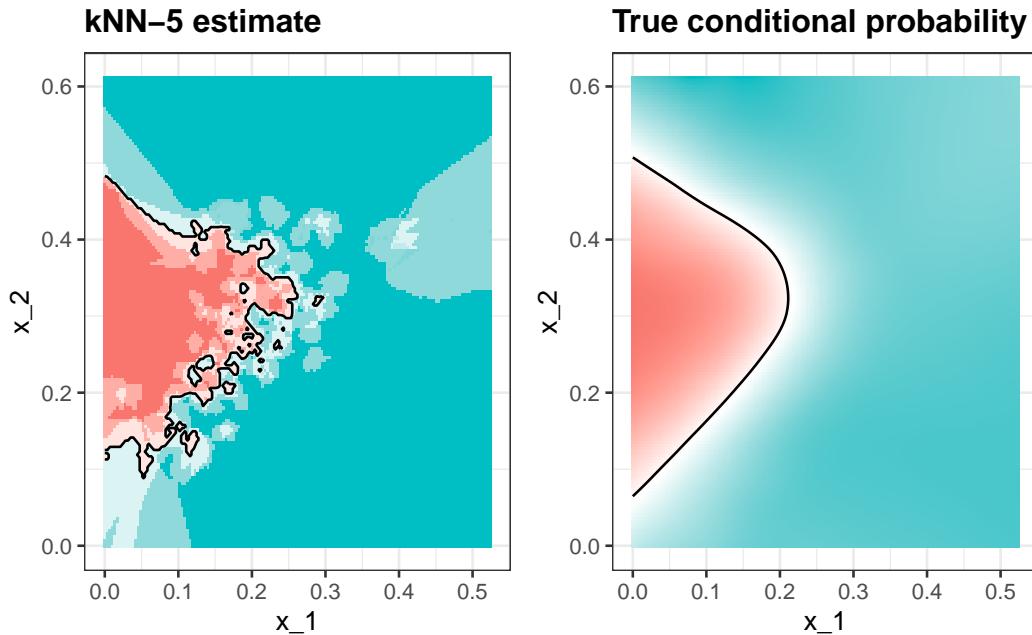
La función `predict` para `knn` produce una probabilidad para cada clase. Mantenemos la probabilidad de ser un 7 como el estimador  $\hat{p}(x_1, x_2)$ :

```
y_hat_knn <- predict(knn_fit, mnist_27$test, type = "class")
confusionMatrix(y_hat_knn, mnist_27$test$y)$overall["Accuracy"]
#> Accuracy
#> 0.815
```

En la Sección 27.8, utilizamos la regresión lineal para generar un estimador.

```
fit_lm <- mnist_27$train %>%
 mutate(y = ifelse(y == 7, 1, 0)) %>%
 lm(y ~ x_1 + x_2, data = .)
p_hat_lm <- predict(fit_lm, mnist_27$test)
y_hat_lm <- factor(ifelse(p_hat_lm > 0.5, 7, 2))
confusionMatrix(y_hat_lm, mnist_27$test$y)$overall["Accuracy"]
#> Accuracy
#> 0.75
```

Y vemos que kNN, con el parámetro predeterminado, ya supera la regresión. Para ver por qué este es el caso, graficaremos  $\hat{p}(x_1, x_2)$  y lo compararemos con la probabilidad condicional verdadera  $p(x_1, x_2)$ :



Vemos que kNN se adapta mejor a la forma no lineal de  $p(x_1, x_2)$ . Sin embargo, nuestro estimador tiene algunas islas de azul en el área roja, lo que intuitivamente no tiene mucho sentido. Esto se debe a lo que llamamos *sobreentrenamiento* (*overtraining* en inglés). Describimos el *sobreentrenamiento* en detalle a continuación. Este es la razón por la que tenemos una mayor exactitud en el set de entrenamiento en comparación con el set de evaluación:

```
y_hat_knn <- predict(knn_fit, mnist_27$train, type = "class")
confusionMatrix(y_hat_knn, mnist_27$train$y)$overall["Accuracy"]
#> Accuracy
#> 0.882

y_hat_knn <- predict(knn_fit, mnist_27$test, type = "class")
confusionMatrix(y_hat_knn, mnist_27$test$y)$overall["Accuracy"]
#> Accuracy
#> 0.815
```

### 29.1.1 Sobreentrenamiento

El *sobreentrenamiento* es peor cuando fijamos  $k = 1$ . Con  $k = 1$ , el estimador para cada  $(x_1, x_2)$  en el set de entrenamiento se obtiene solo con la  $y$  correspondiente a ese punto. En este caso, si  $(x_1, x_2)$  son únicos, obtendremos una exactitud perfecta en el set de entrenamiento porque cada punto se usa para predecirse a sí mismo. Recuerden que si los

predictores no son únicos y tienen resultados diferentes para al menos un set de predictores, entonces es imposible predecir perfectamente.

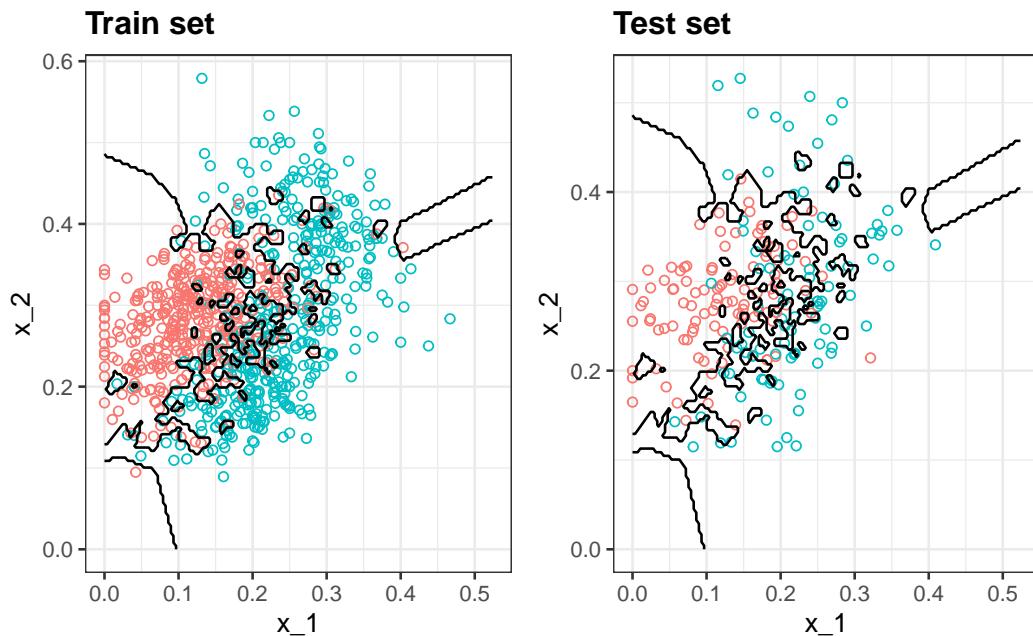
Aquí ajustamos un modelo kNN con  $k = 1$ :

```
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(y_hat_knn_1, mnist_27$train$y)$overall[["Accuracy"]]
#> [1] 0.999
```

Sin embargo, la exactitud del set de evaluación es peor que la regresión logística:

```
y_hat_knn_1 <- predict(knn_fit_1, mnist_27$test, type = "class")
confusionMatrix(y_hat_knn_1, mnist_27$test$y)$overall[["Accuracy"]]
#> Accuracy
#> 0.74
```

Podemos ver el problema de sobreajuste en estos gráficos.



Las curvas negras denotan los límites de la regla de decisión.

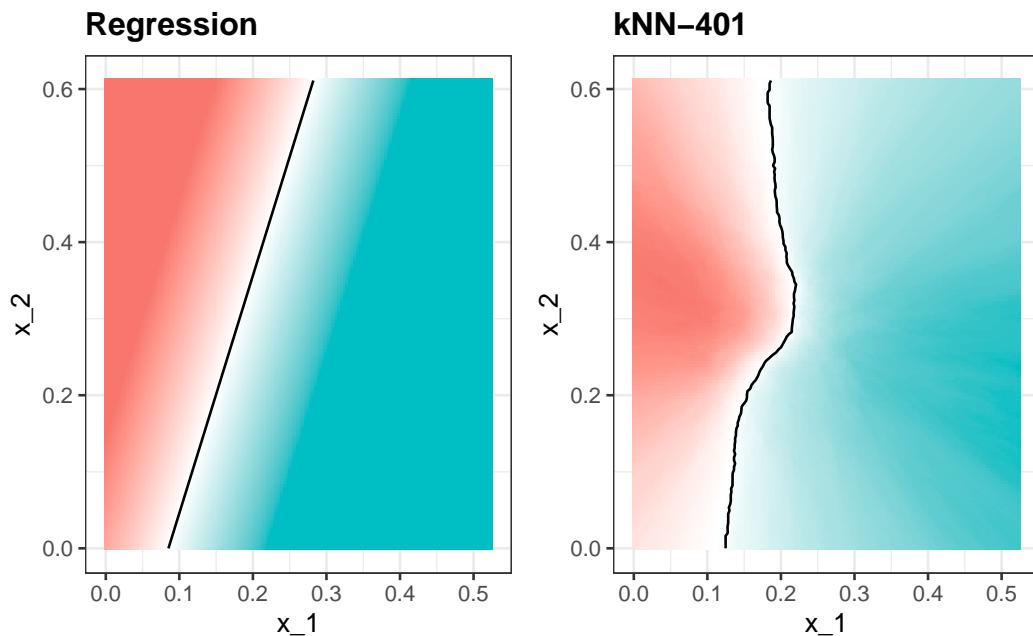
El estimador  $\hat{p}(x_1, x_2)$  sigue los datos de entrenamiento muy de cerca (izquierda). Pueden ver que en el set de entrenamiento, los límites se han trazado para rodear perfectamente un punto rojo único en un mar azul. Como la mayoría de los puntos  $(x_1, x_2)$  son únicos, la predicción es 1 o 0 y la predicción para ese punto es la etiqueta asociada. Sin embargo, tan pronto introducimos el set de entrenamiento (derecha), vemos que muchas de estas pequeñas islas ahora tienen el color opuesto y terminamos haciendo varias predicciones incorrectas.

### 29.1.2 Sobre-suavización

Aunque no tan mal como con los ejemplos anteriores, vimos que con  $k = 5$  también hemos sobreentrenado. Por lo tanto, debemos considerar una  $k$  más grande. Probemos, por ejemplo, un número mucho mayor:  $k = 401$ .

```
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(y_hat_knn_401, mnist_27$test$y)$overall["Accuracy"]
#> Accuracy
#> 0.79
```

Esto resulta ser similar a la regresión:



Este tamaño de  $k$  es tan grande que no permite suficiente flexibilidad. A esto lo llamamos *sobre-suavización* (*over-smoothing* en inglés).

### 29.1.3 Escogiendo la $k$ en kNN

Entonces, ¿cómo elegimos  $k$ ? En principio, queremos escoger la  $k$  que maximiza la exactitud o minimiza el MSE esperado como se define en la Sección 27.4.8. El objetivo de la validación cruzada es estimar estas cantidades para cualquier algoritmo y escoger un set de parámetros de ajuste como  $k$ . Para entender por qué necesitamos un método especial para hacer esto, repetimos lo que hicimos anteriormente pero para diferentes valores de  $k$ :

```
ks <- seq(3, 251, 2)
```

Hacemos esto usando la función `map_df` para repetir lo anterior para cada uno.

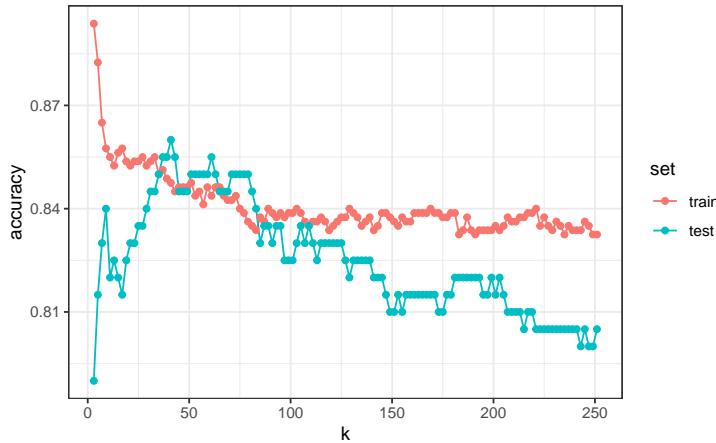
```
library(purrr)
accuracy <- map_df(ks, function(k){
 fit <- knn3(y ~ ., data = mnist_27$train, k = k)

 y_hat <- predict(fit, mnist_27$train, type = "class")
 cm_train <- confusionMatrix(y_hat, mnist_27$train$y)
 train_error <- cm_train$overall["Accuracy"]

 y_hat <- predict(fit, mnist_27$test, type = "class")
 cm_test <- confusionMatrix(y_hat, mnist_27$test$y)
 test_error <- cm_test$overall["Accuracy"]

 tibble(train = train_error, test = test_error)
})
```

Tengan en cuenta que estimamos la exactitud utilizando tanto el set de entrenamiento como el set de evaluación. Ahora podemos graficar las estimadores de exactitud para cada valor de  $k$ :



Primero, noten que el estimador obtenido en el set de entrenamiento es generalmente menor que el estimador obtenido con el set de evaluación, con una diferencia mayor para valores más pequeños de  $k$ . Esto se debe al sobreentrenamiento. También, observen que el gráfico de exactitud versus  $k$  es bastante irregular. No esperamos esto porque pequeños cambios en  $k$  no deberían afectar demasiado cuán bien predice el algoritmo. La irregularidad se explica por el hecho de que la exactitud se calcula en una muestra  $y$ , y, por lo tanto, es una variable aleatoria. Esto demuestra por qué preferimos minimizar la pérdida esperada en lugar de la pérdida que observamos con un solo set de datos.

Si tuviéramos que usar estos estimadores para elegir la  $k$  que maximiza la exactitud, usaríamos los estimadores basados en los datos de evaluación:

```
ks[which.max(accuracy$test)]
#> [1] 41
max(accuracy$test)
#> [1] 0.86
```

Otra razón por la cual necesitamos un mejor estimador de exactitud es que si usamos el set de evaluación para elegir esta  $k$ , no debemos esperar que el estimador de exactitud que acompaña se extrapole al mundo real. Esto se debe a que incluso aquí rompimos una regla de oro de *machine learning*: seleccionamos  $k$  utilizando el set de evaluación. La validación cruzada también ofrece un estimador que considera esto.

## 29.2 Descripción matemática de validación cruzada

En la Sección 27.4.8, describimos que un objetivo común de *machine learning* es encontrar un algoritmo que produzca predictores  $\hat{Y}$  para un resultado  $Y$  que minimiza el MSE:

$$\text{MSE} = \mathbb{E} \left\{ \frac{1}{N} \sum_{i=1}^N (\hat{Y}_i - Y_i)^2 \right\}$$

Cuando todo lo que tenemos a nuestra disposición es un set de datos, podemos estimar el MSE con el MSE observado de esta manera:

$$\hat{\text{MSE}} = \frac{1}{N} \sum_{i=1}^N (\hat{y}_i - y_i)^2$$

Estos dos a menudo se conocen como el *error verdadero* y el *error aparente* (*true error* y *apparent error* en inglés), respectivamente.

Hay dos características importantes del error aparente que siempre debemos considerar:

1. Debido a que nuestros datos son aleatorios, el error aparente es una variable aleatoria. Por ejemplo, el set de datos que tenemos puede ser una muestra aleatoria de una población más grande. Un algoritmo puede tener un error aparente menor que otro algoritmo debido a la suerte.
2. Si entrenamos un algoritmo en el mismo set de datos que usamos para calcular el error aparente, podríamos estar sobreentrenando. En general, cuando hacemos esto, el error aparente será una subestimación del error verdadero. Veremos un ejemplo extremo de esto con  *$k$  vecinos más cercanos*.

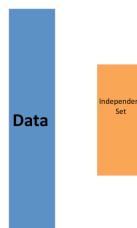
La validación cruzada es una técnica que nos permite aliviar estos dos problemas. Para entender la validación cruzada, es útil pensar en el error verdadero, una cantidad teórica, como el promedio de muchos errores aparentes obtenidos al aplicar el algoritmo a  $B$  nuevas muestras aleatorias de los datos, ninguna de ellas utilizada para entrenar el algoritmo. Como se muestra en un capítulo anterior, pensamos en el verdadero error como:

$$\frac{1}{B} \sum_{b=1}^B \frac{1}{N} \sum_{i=1}^N (\hat{y}_i^b - y_i^b)^2$$

con  $B$  un número grande que puede considerarse prácticamente infinito. Como ya se mencionó, esta es una cantidad teórica porque solo tenemos disponible un set de resultados:  $y_1, \dots, y_n$ . La validación cruzada se basa en la idea de imitar la configuración teórica anterior de la mejor manera posible con los datos que tenemos. Para hacer esto, tenemos que generar una serie de diferentes muestras aleatorias. Hay varios enfoques que podemos usar, pero la idea general para todos ellos es generar aleatoriamente sets de datos más pequeños que no se usan para el entrenamiento y, en cambio, se usan para estimar el error verdadero.

### 29.3 Validación cruzada K-fold

El primero que vamos a describir es *validación cruzada K-fold* (*K-fold cross validation* en inglés). En términos generales, un desafío de *machine learning* comienza con un set de datos (azul en la imagen a continuación). Necesitamos construir un algoritmo usando este set de datos que eventualmente se usará en sets de datos completamente independientes (amarillo).



Pero no podemos ver estos sets de datos independientes.



Entonces, para imitar esta situación, separamos una parte de nuestro set de datos y nos imaginamos que es un set de datos independiente: dividimos el set de datos en un *set de entrenamiento* (azul) y un *set de evaluación* (rojo). Entrenaremos nuestro algoritmo exclusivamente en el set de entrenamiento y usaremos el set de evaluación solo para fines de evaluación.

Por lo general, intentamos seleccionar una pequeña parte del set de datos para tener la mayor cantidad de datos posible para entrenar. Sin embargo, también queremos que el set de evaluación sea grande para que podamos obtener un estimador estable de la pérdida sin ajustar un número poco práctico de modelos. Las opciones típicas son usar del 10% al 20% de los datos para la evaluación.



Queremos reiterar que es indispensable que no usemos el set de evaluación para nada: no para filtrar filas, no para seleccionar características, ¡para nada!

Ahora, esto presenta un nuevo problema porque para la mayoría de los algoritmos de *machine learning*, necesitamos seleccionar parámetros, por ejemplo, el número de vecinos  $k$  en  $k$  vecinos más cercanos. Aquí, nos referiremos al set de parámetros como  $\lambda$ . Necesitamos optimizar los parámetros del algoritmo sin usar nuestro set de evaluación y sabemos que si optimizamos y evaluamos en el mismo set de datos, sobreentrenaremos. Aquí es donde la validación cruzada es más útil.

Para cada set de parámetros de algoritmo que se considera, queremos un estimador del MSE y luego elegiremos los parámetros con el MSE más pequeño. La validación cruzada nos da este estimador.

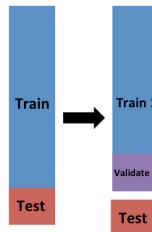
Primero, antes de comenzar el procedimiento de validación cruzada, es importante fijar todos los parámetros del algoritmo. Aunque entrenaremos el algoritmo en el conjunto de sets de entrenamiento, los parámetros  $\lambda$  serán los mismos en todos los sets de entrenamiento. Usaremos  $\hat{y}_i(\lambda)$  para denotar los predictores obtenidos cuando usamos parámetros  $\lambda$ .

Entonces, si vamos a imitar esta definición:

$$\text{MSE}(\lambda) = \frac{1}{B} \sum_{b=1}^B \frac{1}{N} \sum_{i=1}^N (\hat{y}_i^b(\lambda) - y_i^b)^2$$

queremos considerar sets de datos que puedan considerarse una muestra aleatoria independiente y queremos hacerlo varias veces. Con la validación cruzada  $K$ -fold, lo hacemos  $K$  veces. En los dibujos animados, mostramos un ejemplo que usa  $K = 5$ .

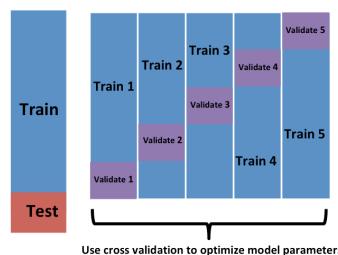
Eventualmente, terminaremos con  $K$  muestras, pero comencemos describiendo cómo construir la primera: simplemente elegimos  $M = N/K$  observaciones al azar (redondeamos si  $M$  no es un número redondo) y piensen en esto como una muestra aleatoria  $y_1^b, \dots, y_M^b$ , con  $b = 1$ . Llamamos a esto el set de validación:



Ahora podemos ajustar el modelo en el set de entrenamiento, luego calcular el error aparente en el set independiente:

$$\hat{MSE}_b(\lambda) = \frac{1}{M} \sum_{i=1}^M (\hat{y}_i^b(\lambda) - y_i^b)^2$$

Tengan en cuenta que esta es solo una muestra y, por consiguiente, devolverá un estimador ruidoso del error verdadero. Por eso tomamos  $K$  muestras, no solo una. En la validación cruzada  $k$ -fold, dividimos aleatoriamente las observaciones en  $K$  sets no superpuestos:

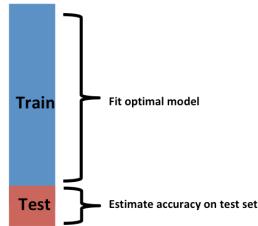


Ahora repetimos el cálculo anterior para cada uno de estos sets  $b = 1, \dots, K$  y obtenemos  $\hat{MSE}_1(\lambda), \dots, \hat{MSE}_K(\lambda)$ . Luego, para nuestro estimador final, calculamos el promedio:

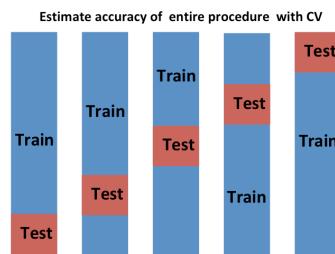
$$\hat{MSE}(\lambda) = \frac{1}{B} \sum_{b=1}^K \hat{MSE}_b(\lambda)$$

y obtenemos un estimador de nuestra pérdida. Un paso final sería seleccionar el  $\lambda$  que minimiza el MSE.

Hemos descrito cómo usar la validación cruzada para optimizar los parámetros. Sin embargo, ahora tenemos que considerar el hecho de que la optimización se produjo en los datos de entrenamiento y, por lo tanto, necesitamos un estimador de nuestro algoritmo final basado en datos que no se utilizaron para optimizar la elección. Aquí es donde usamos el set de evaluación que separamos al principio:

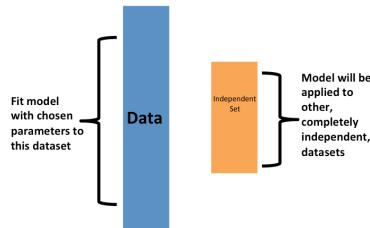


Podemos hacer una validación cruzada nuevamente:



y obtener un estimador final de nuestra pérdida esperada. Sin embargo, noten que esto significa que todo nuestro tiempo de cálculo se multiplica por  $K$ . Pronto aprenderán que realizar esta tarea toma tiempo porque estamos realizando muchos cálculos complejos. Por eso, siempre estamos buscando formas de reducir este tiempo. Para la evaluación final, a menudo solo usamos un set de evaluación.

Una vez que estemos satisfechos con este modelo y queramos ponerlo a disposición de otros, podríamos reajustar el modelo en todo el set de datos, sin cambiar los parámetros optimizados.



Ahora, ¿cómo elegimos la validación cruzada  $K$ ? Grandes valores de  $K$  son preferibles porque los datos de entrenamiento imitan mejor el set de datos original. Sin embargo, los cálculos para valores más grandes de  $K$  serán mucho más lentos: por ejemplo, la validación cruzada con 100 pliegues (*folds* en inglés) será 10 veces más lenta que la validación cruzada con 10 pliegues. Por esta razón, los valores de  $K = 5$  y  $K = 10$  son opciones populares.

Una forma de mejorar la varianza de nuestro estimador final es tomar más muestras. Para hacer esto, ya no necesitaríamos que el set de entrenamiento se divida en sets no superpuestos. En cambio, simplemente elegiríamos  $K$  sets de algún tamaño al azar.

Una versión popular de esta técnica selecciona observaciones al azar con reemplazo en cada pliegue (lo que significa que la misma observación puede aparecer dos veces). Este enfoque tiene algunas ventajas (que no se discuten aquí) y generalmente se conoce como el *bootstrap*. De hecho, este es el enfoque predeterminado en el paquete **caret**. Describimos cómo implementar la validación cruzada con el paquete **caret** en el próximo capítulo. En la siguiente sección, incluimos una explicación de cómo funciona el *bootstrap* en general.

## 29.4 Ejercicios

Genere un set de predictores aleatorios y resultados como este:

```
set.seed(1996)
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)]
```

1. Como  $x$  e  $y$  son completamente independientes, no debemos poder predecir  $y$  con  $x$  con exactitud mayor que 0.5. Confirme esto ejecutando validación cruzada utilizando regresión logística para ajustar el modelo. Debido a que tenemos tantos predictores, seleccionamos una muestra aleatoria  $x_{\text{subset}}$ . Use el subconjunto cuando entrene el modelo. Sugerencia: use la función `train` de **caret**. El componente `results` de `train` le muestra la exactitud. Ignore las advertencias.

2. Ahora, en lugar de una selección aleatoria de predictores, vamos a buscar aquellos que sean más predictivos del resultado. Podemos hacer esto comparando los valores para el grupo  $y = 1$  con esos en el grupo  $y = 0$ , para cada predictor, utilizando una prueba t. Puede realizar este paso así:

```
devtools::install_bioc("genefilter")
install.packages("genefilter")
library(genefilter)
tt <- colttests(x, y)
```

Cree un vector de los valores-p y llámelo `pvals`.

3. Cree un índice `ind` con los números de columna de los predictores que eran “estadísticamente significativos” asociados con  $y$ . Defina “estadísticamente significativo” como un valor-p menor que 0.01. ¿Cuántos predictores sobreviven este umbral?

4. Vuelva a ejecutar la validación cruzada pero después de redefinir `x_subset` como el subconjunto de  $x$  definido por las columnas que muestran una asociación “estadísticamente significativa” con  $y$ . ¿Cuál es la exactitud ahora?

5. Vuelva a ejecutar la validación cruzada nuevamente, pero esta vez usando kNN. Pruebe los

siguientes parámetros de ajuste: `k = seq(101, 301, 25)`. Haga un gráfico de la exactitud resultante.

6. En los ejercicios 3 y 4, vemos que a pesar del hecho de que `x` e `y` son completamente independientes, pudimos predecir `y` con una exactitud superior al 70%. Debemos estar haciendo algo mal entonces. ¿Qué es?

- a. La función `train` estima la exactitud usando los mismos datos que usa para entrenar el algoritmo.
- b. Estamos sobreajustando el modelo al incluir 100 predictores.
- c. Utilizamos todo el set de datos para seleccionar las columnas utilizadas en el modelo. Este paso debe incluirse como parte del algoritmo. La validación cruzada se realizó **después de** esta selección.
- d. La alta exactitud se debe solo a la variabilidad aleatoria.

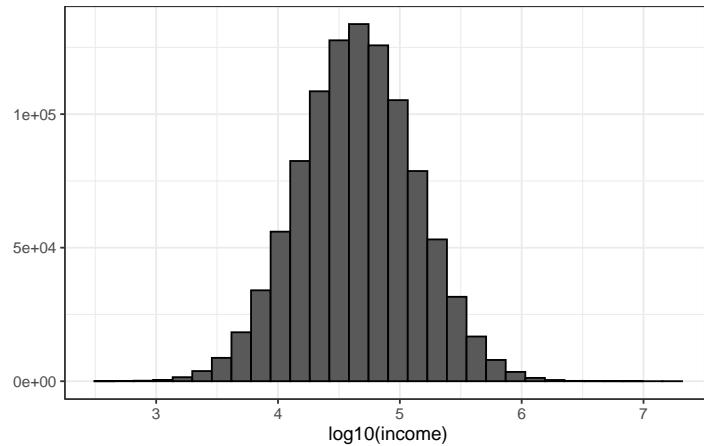
7. **Avanzado:** Vuelva a realizar la validación cruzada, pero esta vez incluya el paso de selección en la validación cruzada. La exactitud ahora debería estar cerca de 50%.

8. Cargue el set de datos `tissue_gene_expression`. Utilice la función `train` para predecir el tejido a partir de la expresión génica. Use kNN. ¿Qué `k` funciona mejor?

## 29.5 Bootstrap

Supongan que la distribución del ingreso de su población es la siguiente:

```
set.seed(1995)
n <- 10^6
income <- 10^(rnorm(n, log10(45000), log10(3)))
qplot(log10(income), bins = 30, color = I("black"))
```



La mediana de la población es:

```
m <- median(income)
m
#> [1] 44939
```

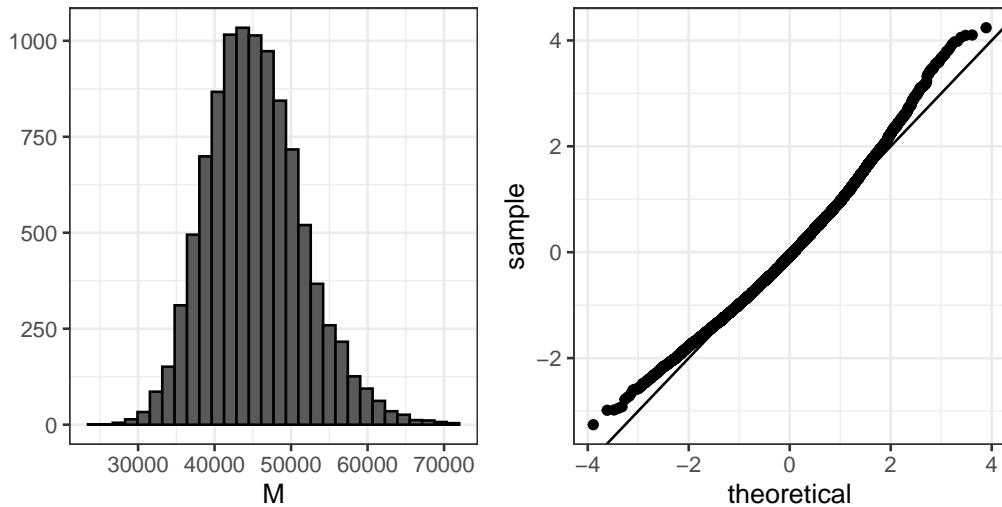
Supongan que no tenemos acceso a toda la población, pero queremos estimar la mediana  $m$ . Tomamos una muestra de 100 y estimamos la mediana de la población  $m$  con la mediana de la muestra  $M$ :

```
N <- 100
X <- sample(income, N)
median(X)
#> [1] 38461
```

¿Podemos construir un intervalo de confianza? ¿Cuál es la distribución de  $M$ ?

Debido a que estamos simulando los datos, podemos usar una simulación de Monte Carlo para aprender la distribución de  $M$ .

```
library(gridExtra)
B <- 10^4
M <- replicate(B, {
 X <- sample(income, N)
 median(X)
})
p1 <- qplot(M, bins = 30, color = I("black"))
p2 <- qplot(sample = scale(M), xlab = "theoretical", ylab = "sample") +
 geom_abline()
grid.arrange(p1, p2, ncol = 2)
```



Si conocemos esta distribución, podemos construir un intervalo de confianza. El problema aquí es que, como ya hemos descrito, en la práctica no sabemos la distribución. En el pasado,

hemos usado el teorema del límite central (CLT), pero el CLT que estudiamos aplica a los promedios y aquí estamos interesados en la mediana. Podemos ver que el intervalo de confianza del 95% basado en CLT:

```
median(X) + 1.96 * sd(X) / sqrt(N) * c(-1, 1)
#> [1] 21018 55905
```

es bastante diferente del intervalo de confianza que generaríamos si conocemos la distribución real de  $M$ :

```
quantile(M, c(0.025, 0.975))
#> 2.5% 97.5%
#> 34438 59050
```

El *bootstrap* nos permite aproximar una simulación de Monte Carlo sin tener acceso a toda la distribución. La idea general es relativamente sencilla. Actuamos como si la muestra observada fuera la población. Luego muestreamos, con reemplazo, sets de datos del mismo tamaño de muestra del set de datos original. Entonces calculamos la estadística de resumen, en este caso la mediana, en estas *muestras de bootstrap*.

La teoría nos dice que, en muchas situaciones, la distribución de las estadísticas obtenidas con muestras de *bootstrap* se aproxima a la distribución de nuestra estadística real. Así es como construimos muestras de *bootstrap* y una distribución aproximada:

```
B <- 10^4
M_star <- replicate(B, {
 X_star <- sample(X, N, replace = TRUE)
 median(X_star)
})
```

Noten que un intervalo de confianza construido con *bootstrap* está mucho más cerca a uno construido con la distribución teórica:

```
quantile(M_star, c(0.025, 0.975))
#> 2.5% 97.5%
#> 30253 56909
```

Para obtener más información sobre *bootstrap*, incluyendo las correcciones que se pueden aplicar para mejorar estos intervalos de confianza, consulte el libro *An Introduction to the Bootstrap* de Efron y Tibshirani.

Tengan en cuenta que en la validación cruzada podemos usar ideas similares a las que utilizamos con el *bootstrap*: en lugar de dividir los datos en particiones iguales, simplemente usamos muestras *bootstrap* repetidas veces.

## 29.6 Ejercicios

1. La función `createResample` se puede utilizar para crear muestras de *bootstrap*. Por ejemplo, podemos crear 10 muestras de *bootstrap* para el set de datos `mnist_27` así:

```
set.seed(1995)
indexes <- createResample(mnist_27$train$y, 10)
```

¿Cuántas veces aparecen 3, 4 y 7 en el primer índice re-muestreado?

2. Vemos que algunos números aparecen más de una vez y otros no aparecen ninguna vez. Esto tiene que ser así para que cada set de datos sea independiente. Repita el ejercicio para todos los índices re-muestreados.

3. Genere un set de datos aleatorio como este:

```
y <- rnorm(100, 0, 1)
```

Estime el 75o cuantil, que sabemos es:

```
qnorm(0.75)
```

con el cuantil de muestra:

```
quantile(y, 0.75)
```

Ejecute una simulación de Monte Carlo para saber el valor esperado y el error estándar de esta variable aleatoria.

4. En la práctica, no podemos ejecutar una simulación de Monte Carlo porque no sabemos si `rnorm` se está utilizando para simular los datos. Use el `bootstrap` para estimar el error estándar usando solo la muestra inicial `y`. Use 10 muestras de `bootstrap`.

5. Repita el ejercicio 4, pero con 10,000 muestras de `bootstrap`.

# 30

---

## *El paquete caret*

---

Ya hemos aprendido sobre regresión y kNN como algoritmos *machine learning* y, en secciones posteriores, aprenderemos sobre varios otros. Todos estos son solo un pequeño subconjunto del total de los algoritmos disponibles. Muchos de estos algoritmos se implementan en R. Sin embargo, se distribuyen a través de diferentes paquetes, desarrollados por diferentes autores, y a menudo usan una sintaxis diferente. El paquete **caret** intenta consolidar estas diferencias y dar consistencia. Actualmente incluye 237 métodos diferentes que se resumen en el manual del paquete **caret**<sup>1</sup>. Tengan en cuenta que **caret** no incluye los paquetes necesarios y, para implementar un paquete a través de **caret**, tendrán que instalar el paquete. Los paquetes requeridos para cada método se describen en el manual del paquete.

El paquete **caret** también provee una función que realiza la validación cruzada para nosotros. Aquí ofrecemos algunos ejemplos que muestran cómo utilizamos este paquete increíblemente útil. Usaremos el ejemplo “2 o 7” para ilustrar:

```
library(tidyverse)
library(dslabs)
data("mnist_27")
```

---

### 30.1 La función **train** de **caret**

La función **train** de **caret** nos permite entrenar diferentes algoritmos utilizando una sintaxis similar. Entonces, por ejemplo, podemos escribir:

```
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 usar el resultado de esta función directamente sin necesidad de mirar los detalles de **predict.glm** y **predict.knn**. En cambio, podemos aprender cómo obtener predicciones de **predict.train**.

El código se ve igual para ambos métodos:

```
y_hat_glm <- predict(train_glm, mnist_27$test, type = "raw")
y_hat_knn <- predict(train_knn, mnist_27$test, type = "raw")
```

---

<sup>1</sup><https://topepo.github.io/caret/available-models.html>

Esto nos permite comparar rápidamente los algoritmos. Por ejemplo, podemos comparar la exactitud de esta manera:

```
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
```

## 30.2 Validación cruzada

Cuando un algoritmo incluye parámetros de ajuste, `train` automáticamente utiliza la validación cruzada para decidir entre algunos valores predeterminados. Para saber qué parámetro o parámetros están optimizados, pueden leer el manual<sup>2</sup> o estudiar lo que devuelve:

```
getModelInfo("knn")
```

También podemos usar una búsqueda rápida como esta:

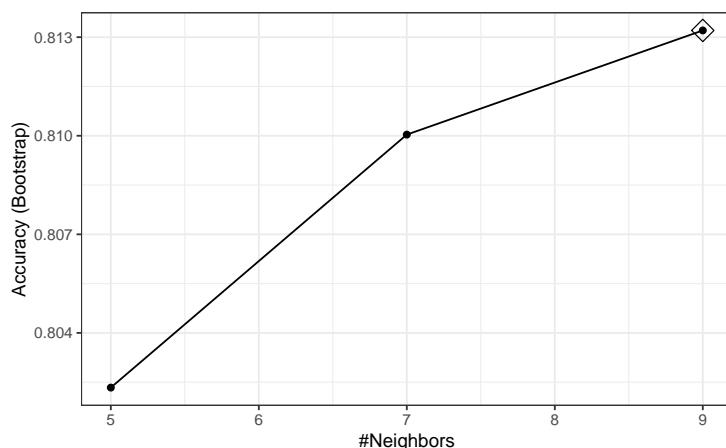
```
modelLookup("knn")
```

Si lo ejecutamos con valores predeterminados:

```
train_knn <- train(y ~ ., method = "knn", data = mnist_27$train)
```

pueden ver rápidamente los resultados de la validación cruzada utilizando la función `ggplot`. El argumento `highlight` destaca el máximo:

```
ggplot(train_knn, highlight = TRUE)
```



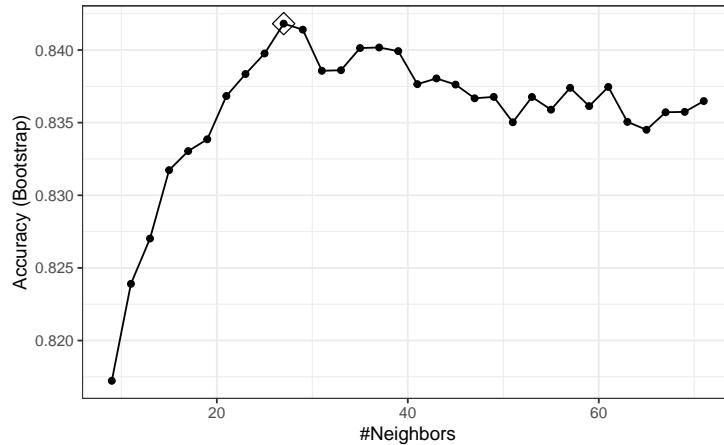
<sup>2</sup><http://topepo.github.io/caret/available-models.html>

Por defecto, la validación cruzada se realiza tomando 25 muestras de *bootstrap* que comprenden el 25% de las observaciones. Para el método kNN, por defecto se intenta  $k = 5, 7, 9$ . Cambiamos esto usando el parámetro `tuneGrid`. La cuadrícula de valores debe ser suministrada por un *data frame* con los nombres de los parámetros como se especifica en el output de `modelLookup`.

Aquí, presentamos un ejemplo donde probamos 30 valores entre 9 y 67. Para hacer esto con `caret`, necesitamos definir una columna llamada `k` de la siguiente manera: `data.frame(k = seq(9, 67, 2))`.

Noten que al ejecutar este código, estamos ajustando 30 versiones de kNN a 25 muestras de *bootstrap*. Como estamos ajustando  $30 \times 25 = 750$  kNN modelos, ejecutar este código tomará varios segundos. Fijamos la semilla porque la validación cruzada es un procedimiento aleatorio y queremos asegurarnos de que el resultado aquí sea reproducible.

```
set.seed(2008)
train_knn <- train(y ~ ., method = "knn",
 data = mnist_27$train,
 tuneGrid = data.frame(k = seq(9, 71, 2)))
ggplot(train_knn, highlight = TRUE)
```



Para acceder al parámetro que maximiza la exactitud, pueden usar esto:

```
train_knn$bestTune
#> k
#> 10 27
```

y para acceder el mejor modelo, pueden usar esto:

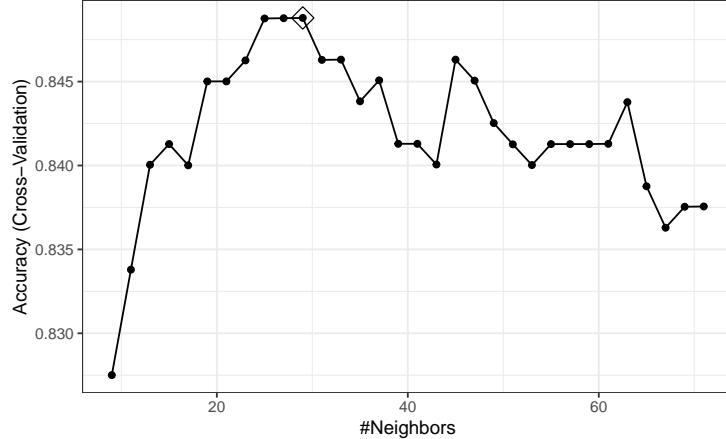
```
train_knn$finalModel
#> 27-nearest neighbor model
#> Training set outcome distribution:
#>
#> 2 7
#> 379 421
```

La función `predict` utilizará este modelo de mejor rendimiento. Aquí está la exactitud del mejor modelo cuando se aplica al set de evaluación, que todavía no hemos utilizado porque la validación cruzada se realizó en el set de entrenamiento:

```
confusionMatrix(predict(train_knn, mnist_27$test, type = "raw"),
 mnist_27$test$y)$overall[["Accuracy"]]
#> Accuracy
#> 0.835
```

Si queremos cambiar la forma en que realizamos la validación cruzada, podemos usar la función `trainControl`. Podemos hacer que el código anterior sea un poco más rápido mediante, por ejemplo, la validación cruzada *10 fold*. Esto significa que tenemos 10 muestras y cada una usa el 10% de las observaciones. Logramos esto usando el siguiente código:

```
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)
ggplot(train_knn_cv, highlight = TRUE)
```



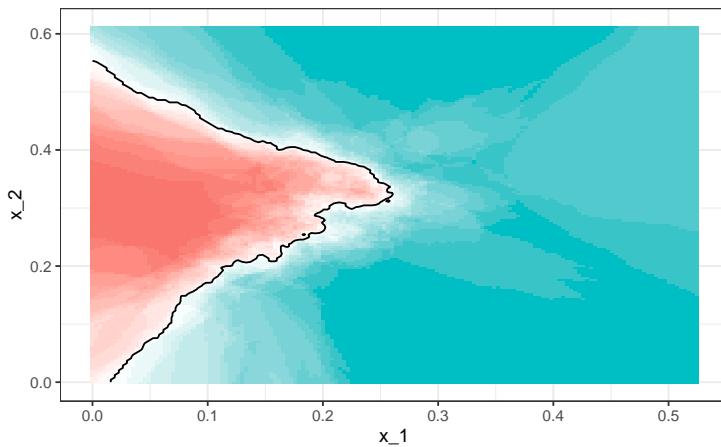
Observamos que los estimadores de exactitud son más variables, algo esperado ya que cambiamos el número de muestras utilizadas para estimar la exactitud.

Tengan en cuenta que el componente `results` de lo que devuelve `train` incluye varias estadísticas de resumen relacionadas con la variabilidad de los estimadores de validación cruzada:

```
names(train_knn$results)
#> [1] "k" "Accuracy" "Kappa" "AccuracySD" "KappaSD"
```

### 30.3 Ejemplo: ajuste con loess

El modelo kNN que da el mejor resultado también da una buena aproximación de la probabilidad condicional verdadera:



Sin embargo, sí vemos que la frontera es algo ondulada. Esto se debe a que kNN, igual que el suavizador de compartimiento básico, no utiliza un *kernel*. Para mejorar esto, podríamos tratar loess. Al leer la parte sobre los modelos disponibles en el manual<sup>3</sup>, vemos que podemos usar el método `gamLoess`. En el manual<sup>4</sup>, también vemos que necesitamos instalar el paquete `gam` si aún no lo hemos hecho:

```
install.packages("gam")
```

Luego, vemos que tenemos dos parámetros para optimizar:

```
modelLookup("gamLoess")
#> model parameter label forReg forClass probModel
#> 1 gamLoess span Span TRUE TRUE TRUE
#> 2 gamLoess degree Degree TRUE TRUE TRUE
```

Nos mantendremos en un grado de 1. Pero para intentar diferentes valores para *span*, aún tenemos que incluir una columna en la tabla con el nombre `degree`:

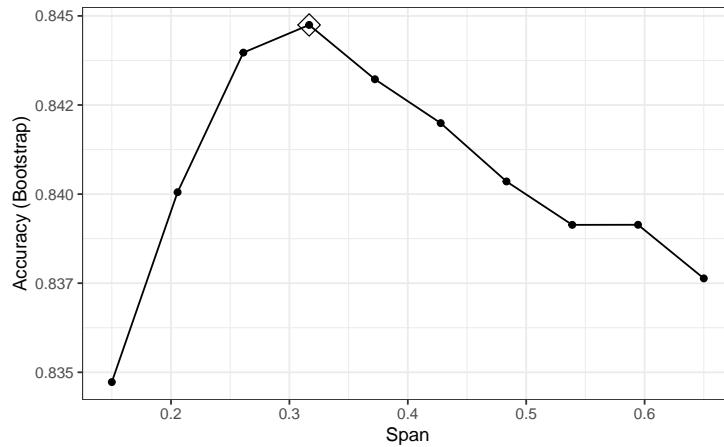
```
grid <- expand.grid(span = seq(0.15, 0.65, len = 10), degree = 1)
```

Utilizaremos los parámetros de control de validación cruzada predeterminados.

<sup>3</sup><https://topepo.github.io/caret/available-models.html>

<sup>4</sup><https://topepo.github.io/caret/train-models-by-tag.html>

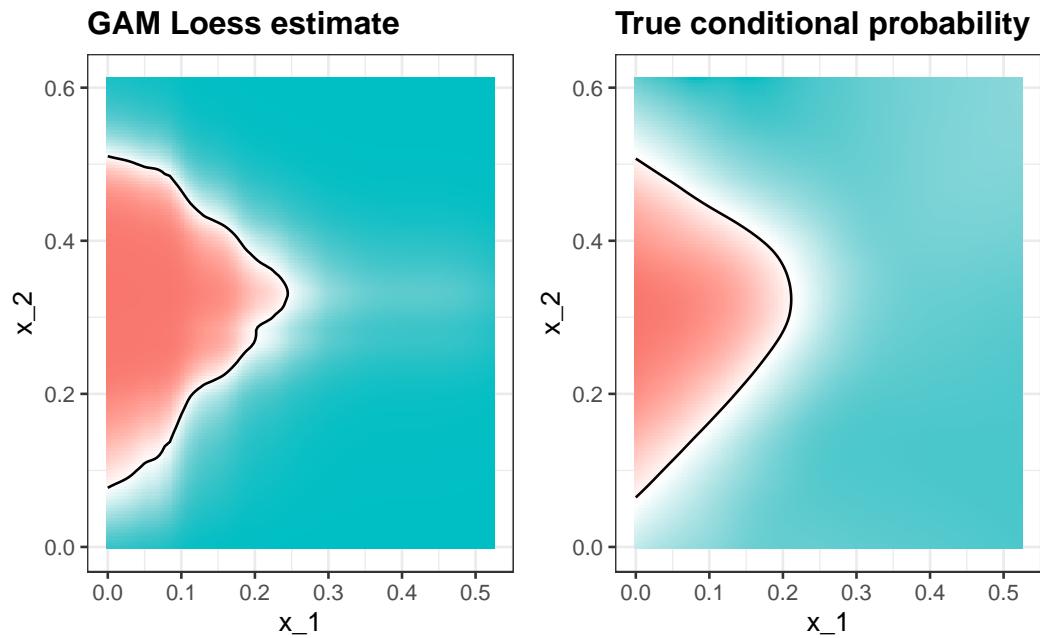
```
train_loess <- train(y ~ .,
 method = "gamLoess",
 tuneGrid=grid,
 data = mnist_27$train)
ggplot(train_loess, highlight = TRUE)
```



Podemos ver que el método funciona de manera similar a kNN:

```
confusionMatrix(data = predict(train_loess, mnist_27$test),
 reference = mnist_27$test$y)$overall[["Accuracy"]]
#> Accuracy
#> 0.85
```

y produce un estimador más suave de la probabilidad condicional:





# 31

---

## *Ejemplos de algoritmos*

---

Hay docenas de algoritmos de *machine learning*. Aquí ofrecemos algunos ejemplos que abarcan enfoques bastante diferentes. A lo largo del capítulo, usaremos los dos datos de dígitos predictores presentados en la Sección 27.8 para demostrar cómo funcionan los algoritmos.

```
library(tidyverse)
library(dslabs)
library(caret)
data("mnist_27")
```

---

### 31.1 Regresión lineal

La regresión lineal puede considerarse un algoritmo de *machine learning*. En la Sección 27.8, demostramos cómo la regresión lineal a veces es demasiada rígida para ser útil. Esto es generalmente cierto, pero para algunos desafíos funciona bastante bien. También sirve como enfoque de partida: si no podemos mejorarlo con un enfoque más complejo, probablemente querremos continuar con la regresión lineal. Para establecer rápidamente la conexión entre la regresión y el *machine learning*, reformularemos el estudio de Galton con alturas, una variable continua.

```
library(HistData)

set.seed(1983)
galton_heights <- GaltonFamilies %>%
 filter(gender == "male") %>%
 group_by(family) %>%
 sample_n(1) %>%
 ungroup() %>%
 select(father, childHeight) %>%
 rename(son = childHeight)
```

Supongan que tienen la tarea de construir un algoritmo de *machine learning* que prediga la altura del hijo  $Y$  usando la altura del padre  $X$ . Generaremos sets de evaluación y de entrenamiento:

```
y <- galton_heights$son
test_index <- createDataPartition(y, times = 1, p = 0.5, list = FALSE)
```

```
train_set <- galton_heights %>% slice(-test_index)
test_set <- galton_heights %>% slice(test_index)
```

En este caso, si solo estuviéramos ignorando la altura del padre y adivinando la altura del hijo, adivinaríamos la altura promedio de los hijos.

```
m <- mean(train_set$son)
m
#> [1] 69.2
```

La raíz cuadrada de nuestra perdida cuadrática es:

```
sqrt(mean((m - test_set$son)^2))
#> [1] 2.77
```

¿Podemos mejorar? En el capítulo de regresión, aprendimos que si el par  $(X, Y)$  sigue una distribución normal de dos variables, la expectativa condicional (lo que queremos estimar) es equivalente a la línea de regresión:

$$f(x) = E(Y | X = x) = \beta_0 + \beta_1 x$$

En la Sección 18.3, presentamos los mínimos cuadrados como método para estimar la pendiente  $\beta_0$  y el intercepto  $\beta_1$ :

```
fit <- lm(son ~ father, data = train_set)
fit$coef
#> (Intercept) father
#> 35.976 0.482
```

Esto nos da una estimador de la expectativa condicional:

$$\hat{f}(x) = 35 + 0.25x$$

Podemos ver que esto realmente provee una mejora sobre adivinar.

```
y_hat <- fit$coef[1] + fit$coef[2]*test_set$father
sqrt(mean((y_hat - test_set$son)^2))
#> [1] 2.54
```

### 31.1.1 La función predict

La función `predict` es muy útil para aplicaciones de *machine learning*. Esta función toma como argumentos el resultado de funciones que ajustan modelos como `lm` o `glm` (aprendaremos sobre `glm` pronto) y un *data frame* con los nuevos predictores para los cuales predecir. Entonces, en nuestro ejemplo actual, usaríamos `predict` así:

```
y_hat <- predict(fit, test_set)
```

Utilizando `predict`, podemos obtener los mismos resultados que obtuvimos anteriormente:

```
y_hat <- predict(fit, test_set)
sqrt(mean((y_hat - test_set$son)^2))
#> [1] 2.54
```

`predict` no siempre devuelve objetos del mismo tipo; depende del tipo de objeto que se le envíe. Para conocer los detalles, deben consultar el archivo de ayuda específico para el tipo de objeto de ajuste. `predict` es un tipo de función especial en R (denominada *función genérica*) que llama a otras funciones según el tipo de objeto que recibe. Así que si `predict` recibe un objeto producido por la función `lm`, llamará `predict.lm`. Si recibe un objeto producido por la función `glm`, llamará `predict.glm`. Estas dos funciones son similares pero con algunas diferencias. Pueden obtener más información sobre las diferencias leyendo los archivos de ayuda:

```
?predict.lm
?predict.glm
```

Hay muchas otras versiones de `predict` y muchos algoritmos de *machine learning* tienen una función `predict`.

## 31.2 Ejercicios

1. Cree un set de datos con el siguiente código.

```
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"))
```

Use el paquete `caret` para dividirlo en un set de evaluación y uno de entrenamiento del mismo tamaño. Entrene un modelo lineal e indique el RMSE. Repita este ejercicio 100 veces y haga un histograma de los RMSE e indique el promedio y la desviación estándar. Sugerencia: adapte el código mostrado anteriormente como demostramos aquí.

```
y <- dat$y
test_index <- createDataPartition(y, times = 1, p = 0.5, list = FALSE)
train_set <- dat %>% slice(-test_index)
test_set <- dat %>% slice(test_index)
fit <- lm(y ~ x, data = train_set)
y_hat <- fit$coef[1] + fit$coef[2]*test_set$x
mean((y_hat - test_set$y)^2)
```

y póngalo dentro de una llamada a `replicate`.

2. Ahora repetiremos lo anterior pero usando sets de datos más grandes. Repita el ejercicio 1 pero para sets de datos con `n <- c(100, 500, 1000, 5000, 10000)`. Guarde el promedio y la desviación estándar de RMSE de estas 100 repeticiones para cada `n`. Sugerencia: use las funciones `sapply` o `map`.

3. Describa lo que observa con el RMSE a medida que aumenta el tamaño del set de datos.

- a. En promedio, el RMSE no cambia mucho ya que `n` se hace más grande, mientras que la variabilidad de RMSE disminuye.
- b. Debido a la ley de los grandes números, el RMSE disminuye: más datos significa estimadores más precisos.
- c. `n = 10000` no es lo suficientemente grande. Para ver una disminución en RMSE, necesitamos hacerla más grande.
- d. El RMSE no es una variable aleatoria.

4. Ahora repita el ejercicio 1, pero esta vez haga la correlación entre `x` e `y` más grande cambiando `Sigma` así:

```
n <- 100
Sigma <- 9*matrix(c(1, 0.95, 0.95, 1), 2, 2)
dat <- MASS::mvrnorm(n = 100, c(69, 69), Sigma) %>%
 data.frame() %>% setNames(c("x", "y"))
```

Repita el ejercicio y observe lo que le sucede al RMSE ahora.

5. ¿Cuál de las siguientes explica mejor por qué el RMSE en el ejercicio 4 es mucho más bajo que en el ejercicio 1?

- a. Es solo suerte. Si lo hacemos nuevamente, será más grande.
- b. El teorema del límite central nos dice que el RMSE es normal.
- c. Cuando aumentamos la correlación entre `x` e `y`, `x` tiene más poder predictivo y, por lo tanto, provee un mejor estimador de `y`. Esta correlación tiene un efecto mucho mayor en RMSE que `n`. `n` grande simplemente ofrece estimadores más precisos de los coeficientes del modelo lineal.
- d. Ambos son ejemplos de regresión, por lo que el RMSE tiene que ser el mismo.

6. Cree un set de datos con el siguiente código:

```
n <- 1000
Sigma <- matrix(c(1, 3/4, 3/4, 3/4, 1, 0, 3/4, 0, 1), 3, 3)
dat <- MASS::mvrnorm(n = 100, c(0, 0, 0), Sigma) %>%
 data.frame() %>% setNames(c("y", "x_1", "x_2"))
```

Tenga en cuenta que `y` está correlacionado con ambos `x_1` y `x_2`, pero los dos predictores son independientes entre sí.

```
cor(dat)
```

Use el paquete `caret` para dividir en un set de evaluación y un set de entrenamiento del

mismo tamaño. Compare el RMSE al usar solo  $x_1$ , sólo  $x_2$ , y ambos  $x_1$  y  $x_2$ . Entrene un modelo lineal e indique el RMSE.

7. Repita el ejercicio 6, pero ahora cree un ejemplo en el que  $x_1$  y  $x_2$  están altamente correlacionados:

```
n <- 1000
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"))
```

Use el paquete **caret** para dividir en un set de evaluación y uno de entrenamiento del mismo tamaño. Compare el RMSE al usar solo  $x_1$ , sólo  $x_2$  y ambos  $x_1$  y  $x_2$ . Entrene un modelo lineal e indique el RMSE.

8. Compare los resultados del ejercicio 6 y 7 y elija la declaración con la que está de acuerdo:

- Agregar predictores adicionales puede mejorar sustancialmente RMSE, pero no cuando están altamente correlacionados con otro predictor.
- Agregar predictores adicionales mejora las predicciones por igual en ambos ejercicios.
- Agregar predictores adicionales da como resultado un ajuste excesivo.
- A menos que incluyamos todos los predictores, no tenemos poder de predicción.

### 31.3 Regresión logística

El enfoque de regresión puede extenderse a datos categóricos. En esta sección, primero ilustramos cómo, para datos binarios, simplemente se pueden asignar valores numéricos de 0 y 1 a los resultados  $y$ . Entonces, se aplica la regresión como si los datos fueran continuos. Más tarde, señalaremos una limitación de este enfoque y presentaremos la *regresión logística* como una solución. La regresión logística es un caso específico de un set de *modelos lineales generalizados*. Para ilustrar la regresión logística, la aplicaremos a nuestro ejemplo anterior de predicción de sexo basado en altura definido en la Sección 27.4.1.

Si definimos el resultado  $Y$  como 1 para mujeres y 0 para hombres, y  $X$  como la altura, nos interesa la probabilidad condicional:

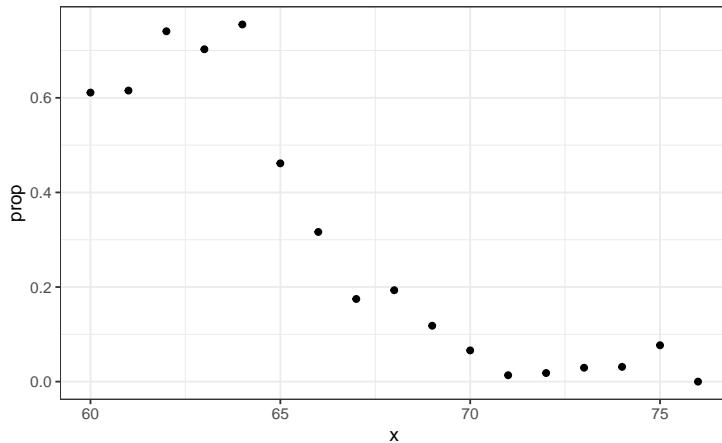
$$\Pr(Y = 1 \mid X = x)$$

Como ejemplo, ofrecemos una predicción para un estudiante que mide 66 pulgadas de alto. ¿Cuál es la probabilidad condicional de ser mujer si mide 66 pulgadas de alto? En nuestro set de datos, podemos estimar esto redondeando a la pulgada más cercana y calculando:

```
train_set %>%
 filter(round(height) == 66) %>%
 summarize(y_hat = mean(sex=="Female"))
#> y_hat
#> 1 0.347
```

Para construir un algoritmo de predicción, queremos estimar la proporción de la población femenina para cualquier altura dada  $X = x$ , que escribimos como la probabilidad condicional descrita anteriormente:  $\Pr(Y = 1|X = x)$ . Veamos cómo se ve esto para varios valores de  $x$  (eliminaremos estratos de  $x$  con pocos puntos de datos):

```
heights %>%
 mutate(x = round(height)) %>%
 group_by(x) %>%
 filter(n() >= 10) %>%
 summarize(prop = mean(sex == "Female")) %>%
 ggplot(aes(x, prop)) +
 geom_point()
```



Dado que los resultados del gráfico anterior son casi lineal y que es el único enfoque que hemos aprendido hasta ahora, intentaremos la regresión lineal. Suponemos que:

$$p(x) = \Pr(Y = 1|X = x) = \beta_0 + \beta_1 x$$

Noten: como  $p_0(x) = 1 - p_1(x)$ , solo estimaremos  $p_1(x)$  y eliminaremos el índice  $_1$ .

Si convertimos los factores a 0s y 1s, podemos estimar  $\beta_0$  y  $\beta_1$  usando mínimos cuadrados.

```
lm_fit <- mutate(train_set, y = as.numeric(sex == "Female")) %>%
 lm(y ~ height, data = .)
```

Una vez que tengamos estimadores  $\hat{\beta}_0$  y  $\hat{\beta}_1$ , podemos obtener una predicción real. Nuestro estimador de la probabilidad condicional  $p(x)$  es:

$$\hat{p}(x) = \hat{\beta}_0 + \hat{\beta}_1 x$$

Para formar una predicción, definimos una *regla de decisión*: predecir mujer si  $\hat{p}(x) > 0.5$ . Podemos comparar nuestras predicciones con los resultados usando:

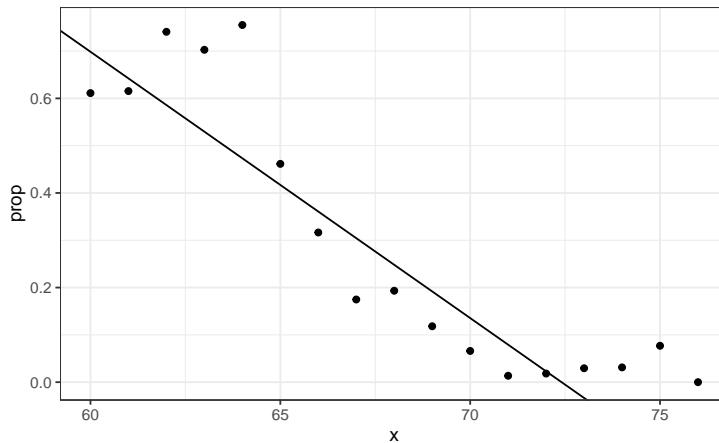
```
p_hat <- predict(lm_fit, test_set)
y_hat <- ifelse(p_hat > 0.5, "Female", "Male") %>% factor()
confusionMatrix(y_hat, test_set$sex)$overall[["Accuracy"]]
#> [1] 0.798
```

Vemos que este método funciona mucho mejor que adivinar.

### 31.3.1 Modelos lineales generalizados

La función  $\beta_0 + \beta_1 x$  puede tomar cualquier valor, incluyendo negativos y valores mayores que 1. De hecho, el estimador  $\hat{p}(x)$  calculado en la sección de regresión lineal se vuelve negativo.

```
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])
```



El rango es:

```
range(p_hat)
#> [1] -0.578 1.262
```

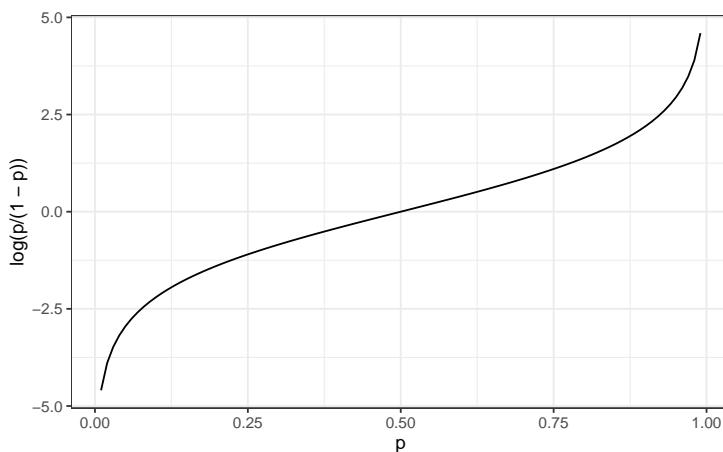
Pero estamos estimando una probabilidad:  $\Pr(Y = 1 | X = x)$  que está restringida entre 0 y 1.

La idea de los *modelos lineales generalizados* (*generalized linear models* o GLM por sus siglas en inglés) es 1) definir una distribución de  $Y$  que sea consistente con sus posibles resultados y 2) encontrar una función  $g$  tal que  $g(\Pr(Y = 1 | X = x))$  se pueda modelar como una combinación lineal de predictores. La regresión logística es el GLM más utilizado. Es una

extensión de regresión lineal que asegura que el estimador de  $\Pr(Y = 1 | X = x)$  esté entre 0 y 1. Este enfoque utiliza la transformación *logística* que presentamos en la Sección 9.8.1:

$$g(p) = \log \frac{p}{1-p}$$

Esta transformación logística convierte probabilidad en logaritmo del riesgo relativo. Como se discutió en la sección de visualización de datos, el riesgo relativo nos dice cuánto más probable es que algo suceda en comparación con no suceder.  $p = 0.5$  significa que las probabilidades son de 1 a 1; por lo tanto, el riesgo relativo es 1. Si  $p = 0.75$ , las probabilidades son de 3 a 1. Una buena característica de esta transformación es que convierte las probabilidades en simétricas alrededor de 0. Aquí hay un gráfico de  $g(p)$  versus  $p$ :



Con la *regresión logística*, modelamos la probabilidad condicional directamente con:

$$g\{\Pr(Y = 1 | X = x)\} = \beta_0 + \beta_1 x$$

Con este modelo, ya no podemos usar mínimos cuadrados. En su lugar, calculamos el *estimador de máxima verosimilitud* (*maximum likelihood estimate* o MLE por sus siglas en inglés). Pueden aprender más sobre este concepto en un libro de texto de teoría estadística<sup>1</sup>.

En R, podemos ajustar el modelo de regresión logística con la función `glm`: modelos lineales generalizados. Esta función puede ajustar varios modelos, no solo regresión logística, por lo cual tenemos que especificar el modelo que queremos a través del argumento `family`:

```
glm_fit <- train_set %>%
 mutate(y = as.numeric(sex == "Female")) %>%
 glm(y ~ height, data = ., family = "binomial")
```

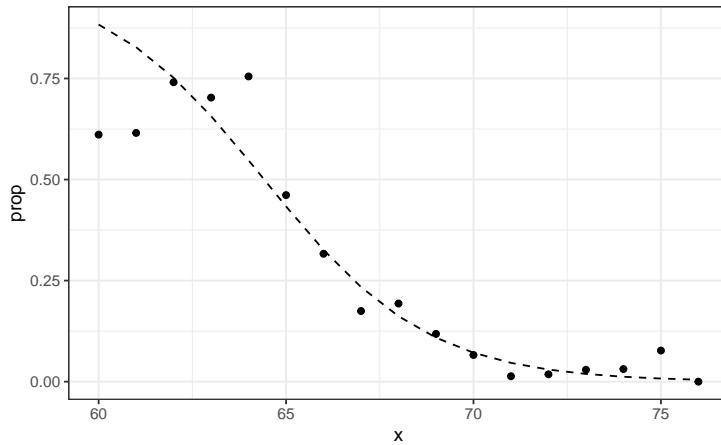
Podemos obtener predicciones usando la función `predict`:

<sup>1</sup><http://www.amazon.com/Mathematical-Statistics-Analysis-Available-Enhanced/dp/0534399428>

```
p_hat_logit <- predict(glm_fit, newdata = test_set, type = "response")
```

Cuando usamos `predict` con un objeto `glm`, tenemos que especificar que queremos `type="response"` si queremos las probabilidades condicionales, ya que por defecto la función devuelve los valores luego de la transformación logística.

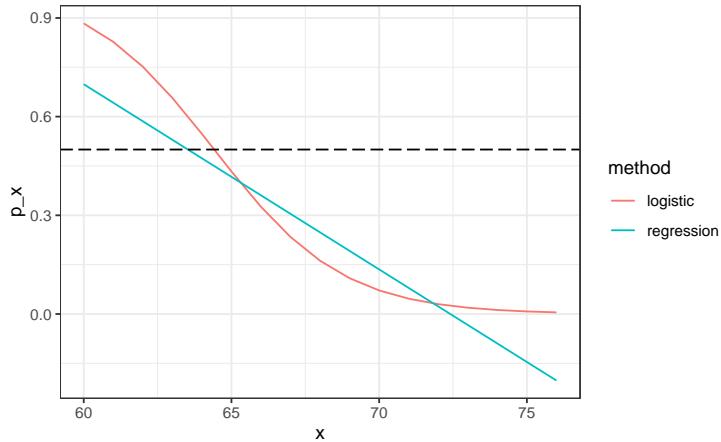
Este modelo se ajusta a los datos un poco mejor que la línea:



Como tenemos un estimador  $\hat{p}(x)$ , podemos obtener predicciones:

```
y_hat_logit <- ifelse(p_hat_logit > 0.5, "Female", "Male") %>% factor
confusionMatrix(y_hat_logit, test_set$sex)$overall[["Accuracy"]]
#> [1] 0.808
```

Las predicciones resultantes son similares. Esto se debe a que los dos estimadores de  $p(x)$  mayores que 1/2 en aproximadamente la misma región de  $x$ :



Las regresiones lineales y logísticas proveen un estimador de la expectativa condicional:

$$\mathrm{E}(Y \mid X = x)$$

que en el caso de datos binarios es equivalente a la probabilidad condicional:

$$\mathrm{Pr}(Y = 1 \mid X = x)$$

### 31.3.2 Regresión logística con más de un predictor

En esta sección, aplicamos la regresión logística a los datos “2 o 7” presentados en la Sección 27.8. En este caso, estamos interesados en estimar una probabilidad condicional que depende de dos variables. El modelo de regresión logística estándar en este caso supondrá que:

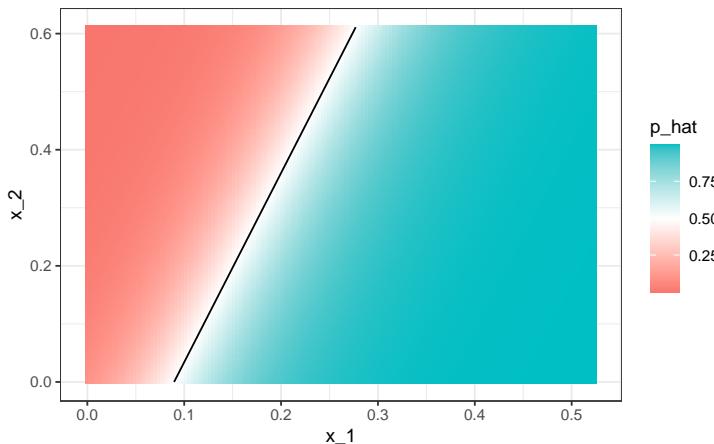
$$g\{p(x_1, x_2)\} = g\{\mathrm{Pr}(Y = 1 \mid X_1 = x_1, X_2 = x_2)\} = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

con  $g(p) = \log \frac{p}{1-p}$ , la función logística descrita en la sección anterior. Para ajustar el modelo, usamos el siguiente código:

```
fit_glm <- glm(y ~ x_1 + x_2, data=mnist_27$train, family = "binomial")
p_hat_glm <- predict(fit_glm, mnist_27$test, type = "response")
y_hat_glm <- factor(ifelse(p_hat_glm > 0.5, 7, 2))
confusionMatrix(y_hat_glm, mnist_27$test$y)$overall["Accuracy"]
#> Accuracy
#> 0.75
```

Comparando con los resultados que obtuvimos en la Sección 27.8, vemos que la regresión logística funciona de manera similar a la regresión. Esto no es sorprendente dado que el estimador de  $\hat{p}(x_1, x_2)$  se ve similar también:

```
p_hat <- predict(fit_glm, newdata = mnist_27$true_p, type = "response")
mnist_27$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")
```



Al igual que con la regresión lineal, la regla de decisión es una línea, un hecho que puede corroborarse matemáticamente ya que:

$$g^{-1}(\hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2) = 0.5 \implies \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 = g(0.5) = 0 \implies x_2 = -\hat{\beta}_0/\hat{\beta}_2 - \hat{\beta}_1/\hat{\beta}_2 x_1$$

Por eso,  $x_2$  es una función lineal de  $x_1$ . Esto implica que, al igual que la regresión, nuestro enfoque de regresión logística no tiene ninguna posibilidad de capturar la naturaleza no lineal de la verdadera  $p(x_1, x_2)$ . Una vez que pasemos a ejemplos más complejos, veremos que la regresión lineal y la regresión lineal generalizada son limitadas y no lo suficientemente flexibles como para ser útiles para la mayoría de los desafíos de *machine learning*. Las nuevas técnicas que aprendemos son esencialmente enfoques para estimar la probabilidad condicional de una manera más flexible.

## 31.4 Ejercicios

- Defina el siguiente set de datos:

```
make_data <- function(n = 1000, p = 0.5,
 mu_0 = 0, mu_1 = 2,
 sigma_0 = 1, sigma_1 = 1){
 y <- rbinom(n, 1, p)
 f_0 <- rnorm(n, mu_0, sigma_0)
 f_1 <- rnorm(n, mu_1, sigma_1)
 x <- ifelse(y == 1, f_1, f_0)
 test_index <- createDataPartition(y, times = 1, p = 0.5, list = FALSE)
 list(train = data.frame(x = x, y = as.factor(y)) %>%
 slice(-test_index),
 test = data.frame(x = x, y = as.factor(y)) %>%
 slice(test_index))
}
```

dat <- make\_data()

Noten que hemos definido una variable  $x$  que es predictiva de un resultado binario  $y$ .

```
dat$train %>% ggplot(aes(x, color = y)) + geom_density()
```

Compare la exactitud de la regresión lineal y la regresión logística.

- Repita la simulación del primer ejercicio 100 veces y compare la exactitud promedio para cada método. Observe como dan prácticamente la misma respuesta.
- Genere 25 sets de datos diferentes cambiando la diferencia entre las dos clases: `delta <- seq(0, 3, len = 25)`. Grafique exactitud versus `delta`.

### 31.5 k vecinos más cercanos (kNN)

Presentamos el algoritmo kNN en la Sección 29.1 y demostramos cómo usamos la validación cruzada para elegir  $k$  en la Sección 30.2. Aquí revisamos rápidamente cómo ajustamos un modelo kNN usando el paquete **caret**. En la Sección 30.2, presentamos el siguiente código para ajustar un modelo kNN:

```
train_knn <- train(y ~ ., method = "knn",
 data = mnist_27$train,
 tuneGrid = data.frame(k = seq(9, 71, 2)))
```

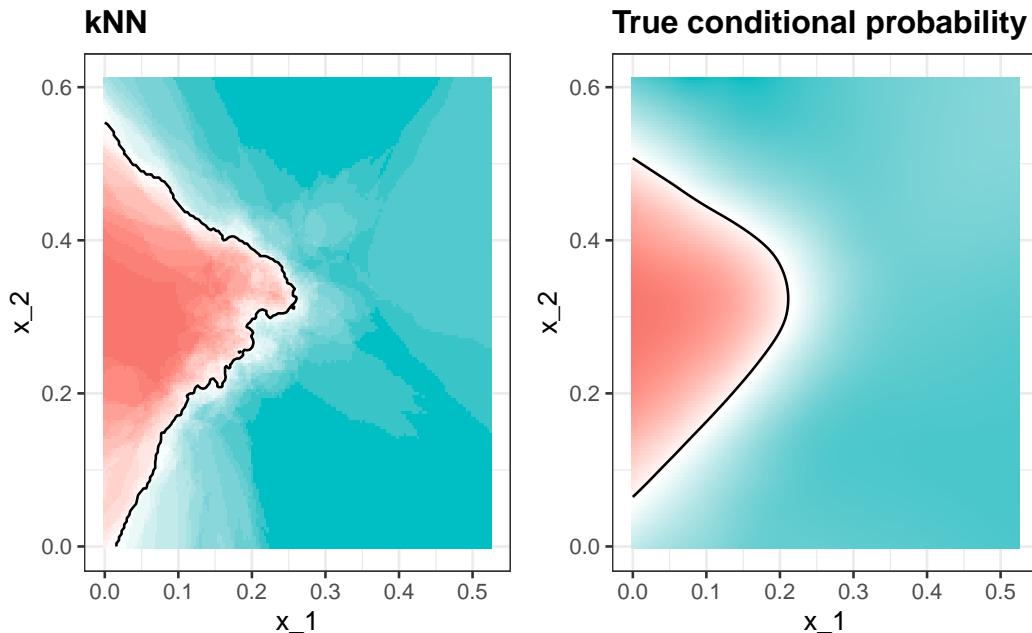
Vimos que el parámetro que maximizaba la exactitud estimada era:

```
train_knn$bestTune
#> k
#> 10 27
```

Este modelo resulta en una exactitud mejor que la de regresión y de regresión logística:

```
confusionMatrix(predict(train_knn, mnist_27$test, type = "raw"),
 mnist_27$test$y)$overall["Accuracy"]
#> Accuracy
#> 0.835
```

Un gráfico de la probabilidad condicional estimada muestra que el estimador de kNN es lo suficientemente flexible para capturar la forma de la probabilidad condicional verdadera.



## 31.6 Ejercicios

1. Anteriormente utilizamos regresión logística para predecir el sexo basado en la altura. Use kNN para hacer lo mismo. Use el código descrito en este capítulo para seleccionar la medida  $F_1$  y graficarla contra  $k$ . Compare con el  $F_1$  de aproximadamente 0.6 que obtuvimos con regresión.
2. Cargue el siguiente set de datos:

```
data("tissue_gene_expression")
```

Este set de datos incluye una matriz  $\mathbf{x}$ :

```
dim(tissue_gene_expression$x)
```

con la expresión génica medida en 500 genes para 189 muestras biológicas que representan siete tejidos diferentes. El tipo de tejido se almacena en  $y$ :

```
table(tissue_gene_expression$y)
```

Divida los datos en sets de entrenamiento y de evaluación. Luego, use kNN para predecir el tipo de tejido y ver qué exactitud obtiene. Pruébelo para  $k = 1, 3, \dots, 11$ .

---

## 31.7 Modelos generativos

Hemos explicado cómo, cuando se usa la función de pérdida cuadrática, las expectativas/probabilidades condicionales ofrecen el mejor enfoque para desarrollar una regla de decisión. En un caso binario, el error verdadero más pequeño que podemos lograr está determinado por la regla de Bayes, que es una regla de decisión basada en la probabilidad condicional verdadera:

$$p(\mathbf{x}) = \Pr(Y = 1 \mid \mathbf{X} = \mathbf{x})$$

Hemos descrito varios enfoques para estimar  $p(\mathbf{x})$ . En todos estos, estimamos la probabilidad condicional directamente y no consideramos la distribución de los predictores. En *machine learning*, estos se denominan enfoques *discriminativos*.

Sin embargo, el teorema de Bayes nos dice que conocer la distribución de los predictores  $\mathbf{X}$  puede ser útil. Métodos que modelan la distribución conjunta de  $Y$  y  $\mathbf{X}$  se denominan *modelos generativos* (modelamos cómo todos los datos,  $\mathbf{X}$  e  $Y$ , se generan). Comenzamos describiendo el modelo generativo más general, *Naive Bayes*, y luego describimos dos casos específicos: el *análisis discriminante cuadrático* (*quadratic discriminant analysis* o QDA por sus siglas en inglés) y el *análisis discriminante lineal* (*linear discriminant analysis* o LDA por sus siglas en inglés).

### 31.7.1 Naive Bayes

Recordemos que la regla de Bayes nos dice que podemos reescribir  $p(\mathbf{x})$  así:

$$p(\mathbf{x}) = \Pr(Y = 1 | \mathbf{X} = \mathbf{x}) = \frac{f_{\mathbf{X}|Y=1}(\mathbf{x})\Pr(Y = 1)}{f_{\mathbf{X}|Y=0}(\mathbf{x})\Pr(Y = 0) + f_{\mathbf{X}|Y=1}(\mathbf{x})\Pr(Y = 1)}$$

con  $f_{\mathbf{X}|Y=1}$  y  $f_{\mathbf{X}|Y=0}$  representando las funciones de distribución del predictor  $\mathbf{X}$  para las dos clases  $Y = 1$  y  $Y = 0$ . La fórmula implica que si podemos estimar estas distribuciones condicionales de los predictores, podemos desarrollar una poderosa regla de decisión. Sin embargo, esto es un gran “si”. A medida que avancemos, encontraremos ejemplos en los que  $\mathbf{X}$  tiene muchas dimensiones y no tenemos mucha información sobre la distribución. En estos casos, *Naive Bayes* será prácticamente imposible de implementar. Sin embargo, hay casos en los que tenemos un pequeño número de predictores (no más de 2) y muchas categorías en las que los modelos generativos pueden ser bastante poderosos. Describimos dos ejemplos específicos y utilizamos nuestros estudios de caso descritos anteriormente para ilustrarlos.

Comencemos con un caso muy sencillo y poco interesante, pero ilustrativo: el ejemplo relacionado con la predicción del sexo basado en la altura.

```
library(tidyverse)
library(caret)

library(dslabs)
data("heights")

y <- heights$height
set.seed(1995)
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, el enfoque *Naive Bayes* es particularmente apropiado porque sabemos que la distribución normal es una buena aproximación para las distribuciones condicionales de altura dado el sexo para ambas clases  $Y = 1$  (mujer) y  $Y = 0$  (hombre). Esto implica que podemos aproximar las distribuciones condicionales  $f_{X|Y=1}$  y  $f_{X|Y=0}$  al simplemente estimar los promedios y las desviaciones estándar de los 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 64.8 4.14
#> 2 Male 69.2 3.57
```

La prevalencia, que denotaremos con  $\pi = \Pr(Y = 1)$ , puede estimarse a partir de los datos con:

```
pi <- train_set %>% summarize(pi=mean(sex=="Female")) %>% pull(pi)
pi
#> [1] 0.212
```

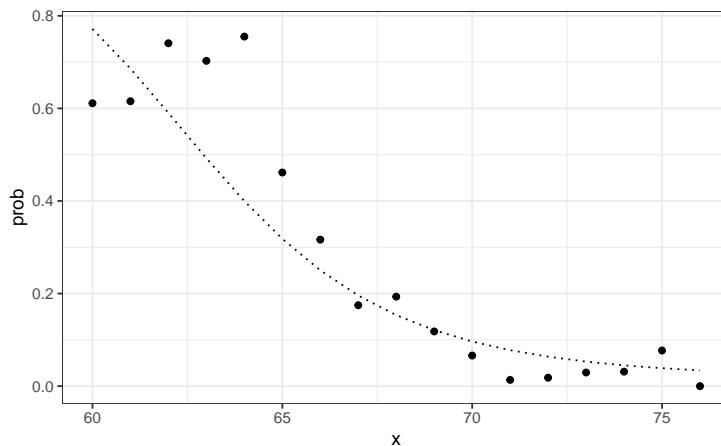
Ahora podemos usar nuestros estimadores de promedio y desviación estándar para obtener una regla:

```
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))
```

Nuestro estimador de *Naive Bayes*  $\hat{p}(x)$  se parece mucho a nuestro estimador de regresión logística:



De hecho, podemos mostrar que el enfoque de *Naive Bayes* es matemáticamente similar a la predicción de regresión logística. Sin embargo, dejamos la demostración a un texto más avanzado, como *Elements of Statistical Learning*<sup>2</sup>. Podemos ver que son similares empíricamente al comparar las dos curvas resultantes.

### 31.7.2 Controlando la prevalencia

Una característica útil del enfoque *Naive Bayes* es que incluye un parámetro para tomar en cuenta las diferencias en la prevalencia. Usando nuestra muestra, estimamos  $f_{X|Y=1}$ ,  $f_{X|Y=0}$  y  $\pi$ . Si usamos sombreros para denotar los estimadores, podemos escribir  $\hat{p}(x)$  como:

$$\hat{p}(x) = \frac{\hat{f}_{X|Y=1}(x)\hat{\pi}}{\hat{f}_{X|Y=0}(x)(1 - \hat{\pi}) + \hat{f}_{X|Y=1}(x)\hat{\pi}}$$

<sup>2</sup><https://web.stanford.edu/~hastie/Papers/ESLII.pdf>

Como discutimos anteriormente, nuestra muestra tiene una prevalencia mucho menor, 0.21, que la población general. Entonces si usamos la regla  $\hat{p}(x) > 0.5$  para predecir mujeres, nuestra exactitud se verá afectada debido a 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] 0.213
```

Nuevamente, esto se debe a que el algoritmo da más peso a la especificidad para tomar en cuenta la baja prevalencia:

```
specificity(data = factor(y_hat_bayes), reference = factor(test_set$sex))
#> [1] 0.967
```

Esto se debe principalmente al hecho de que  $\hat{\pi}$  es sustancialmente menor que 0.5, por lo que tendemos a predecir `Male` más a menudo. Tiene sentido que un algoritmo de *machine learning* haga esto en nuestra muestra porque tenemos un mayor porcentaje de hombres. Pero si tuviéramos que extrapolar esto a una población general, nuestra exactitud general se vería afectada por la baja sensibilidad.

El enfoque *Naive Bayes* nos da una forma directa de corregir esto, ya que simplemente podemos forzar  $\hat{\pi}$  a ser el valor que queremos. Entonces, para equilibrar especificidad y sensibilidad, en lugar de cambiar el umbral en la regla de decisión, simplemente podríamos cambiar  $\hat{\pi}$  a 0.5 así:

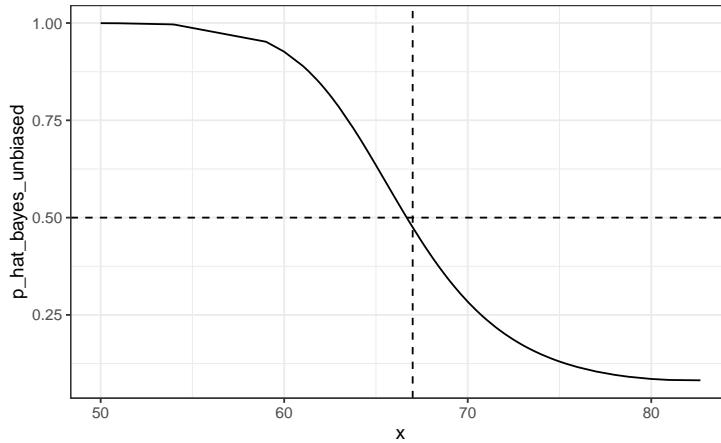
```
p_hat_bayes_unbiased <- f1 * 0.5 / (f1 * 0.5 + f0 * (1 - 0.5))
y_hat_bayes_unbiased <- ifelse(p_hat_bayes_unbiased > 0.5, "Female", "Male")
```

Tengan en cuenta la diferencia de sensibilidad con un mejor equilibrio:

```
sensitivity(factor(y_hat_bayes_unbiased), factor(test_set$sex))
#> [1] 0.693
specificity(factor(y_hat_bayes_unbiased), factor(test_set$sex))
#> [1] 0.832
```

La nueva regla también nos da un umbral muy intuitivo entre 66-67, que es aproximadamente la mitad de las alturas promedio de hombres y mujeres:

```
qplot(x, p_hat_bayes_unbiased, geom = "line") +
 geom_hline(yintercept = 0.5, lty = 2) +
 geom_vline(xintercept = 67, lty = 2)
```



### 31.7.3 Análisis discriminante cuadrático

El *análisis discriminante cuadrático* (QDA) es una versión de *Naive Bayes* en la cual suponemos que las distribuciones  $p_{\mathbf{X}|Y=1}(x)$  y  $p_{\mathbf{X}|Y=0}(x)$  siguen una distribución normal de múltiples variables. El ejemplo sencillo que describimos en la sección anterior es QDA. Veamos ahora un caso un poco más complicado: el ejemplo “2 o 7”.

```
data("mnist_27")
```

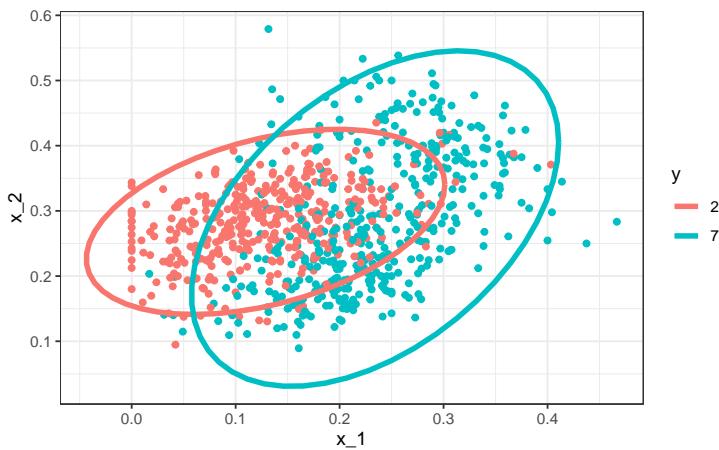
En este caso, tenemos dos predictores, por lo que suponemos que cada uno sigue una distribución normal de dos variables. Esto implica que necesitamos estimar dos promedios, dos desviaciones estándar y una correlación para cada caso  $Y = 1$  y  $Y = 0$ . Una vez que tengamos estos, podemos aproximar las distribuciones  $f_{X_1, X_2|Y=1}$  y  $f_{X_1, X_2|Y=0}$ . Podemos estimar fácilmente los parámetros a partir de los datos:

```
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
```

Aquí ofrecemos una forma visual de mostrar el enfoque. Graficamos los datos y usamos gráficos de contorno (*contour plots* en inglés) para dar una idea de cómo son las dos densidades normales estimadas (mostramos la curva que representa una región que incluye el 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)
```



Esto define el siguiente estimador de  $f(x_1, x_2)$ .

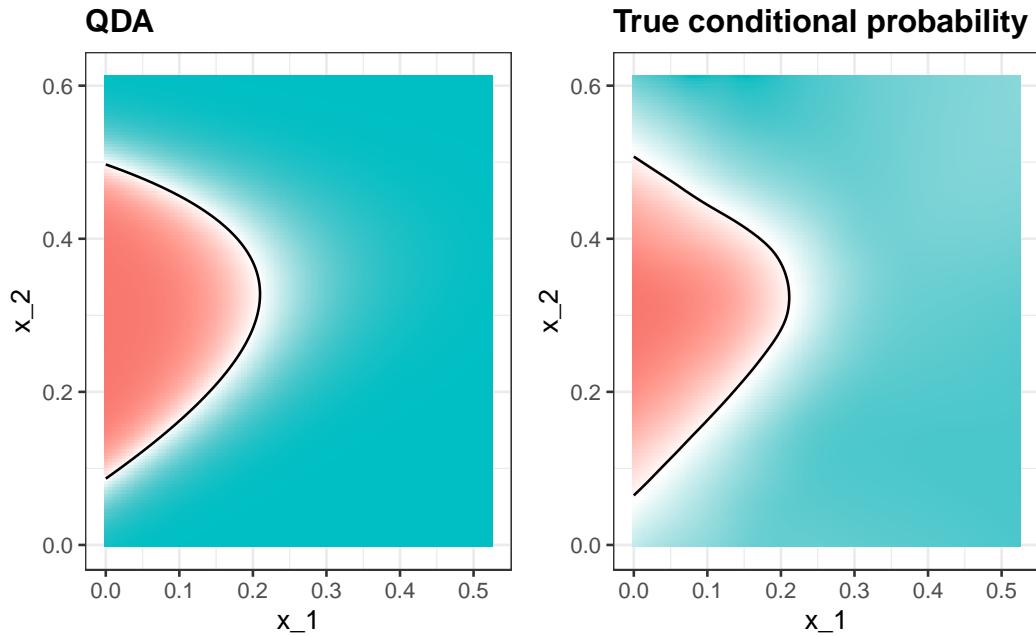
Podemos usar la función `train` del paquete `caret` para ajustar el modelo y obtener predicciones:

```
library(caret)
train_qda <- train(y ~ ., method = "qda", data = mnist_27$train)
```

Vemos que obtenemos una exactitud relativamente buena:

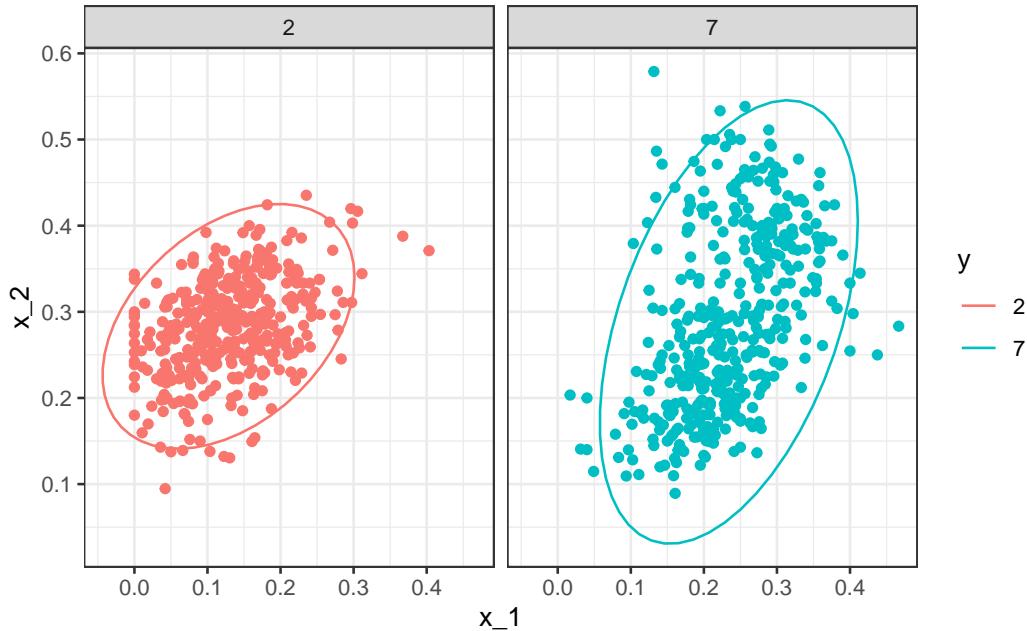
```
y_hat <- predict(train_qda, mnist_27$test)
confusionMatrix(y_hat, mnist_27$test$y)$overall["Accuracy"]
#> Accuracy
#> 0.82
```

La probabilidad condicional estimada se ve relativamente bien, aunque no se ajusta tan bien como los suavizadores de *kernel*:



Una razón por la que QDA no funciona tan bien como los métodos de *kernel* es quizás porque la presunción de normalidad no es válida. Aunque para los 2s parece razonable, para los 7s no lo parece. Observen la ligera curvatura en los puntos para los 7s:

```
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)
```

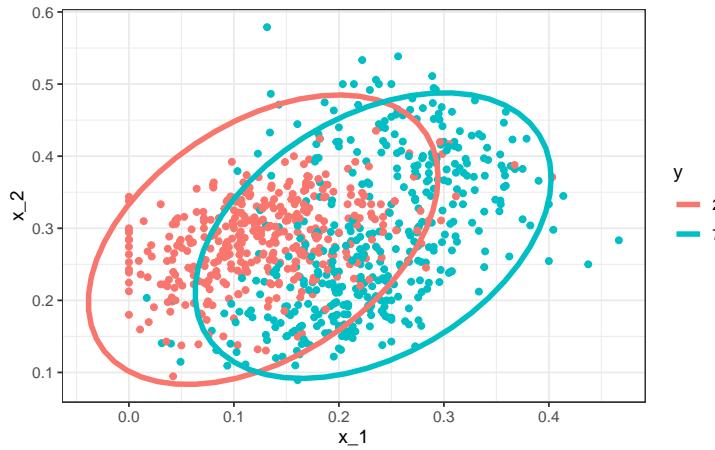


QDA puede funcionar bien aquí, pero se vuelve más difícil de usar a medida que aumente el número de predictores. Aquí tenemos 2 predictores y tuvimos que calcular 4 medias, 4 desviaciones estándar y 2 correlaciones. ¿Cuántos parámetros tendríamos si en lugar de 2 predictores tuviéramos 10? El principal problema proviene del estimador de correlaciones para 10 predictores. Con 10, tenemos 45 correlaciones para cada clase. En general, la fórmula es  $K \times p(p - 1)/2$ , que se hace grande rápidamente. Una vez el número de parámetros se acerca al tamaño de nuestros datos, el método deja de ser práctico debido al sobreajuste.

#### 31.7.4 Análisis discriminante lineal

Una solución relativamente sencilla para el problema de tener demasiados parámetros es suponer que la estructura de correlación es la misma para todas las clases, lo que reduce el número de parámetros que necesitamos estimar.

En este caso, calcularíamos solo un par de desviaciones estándar y una correlación, y las distribuciones se ven así:

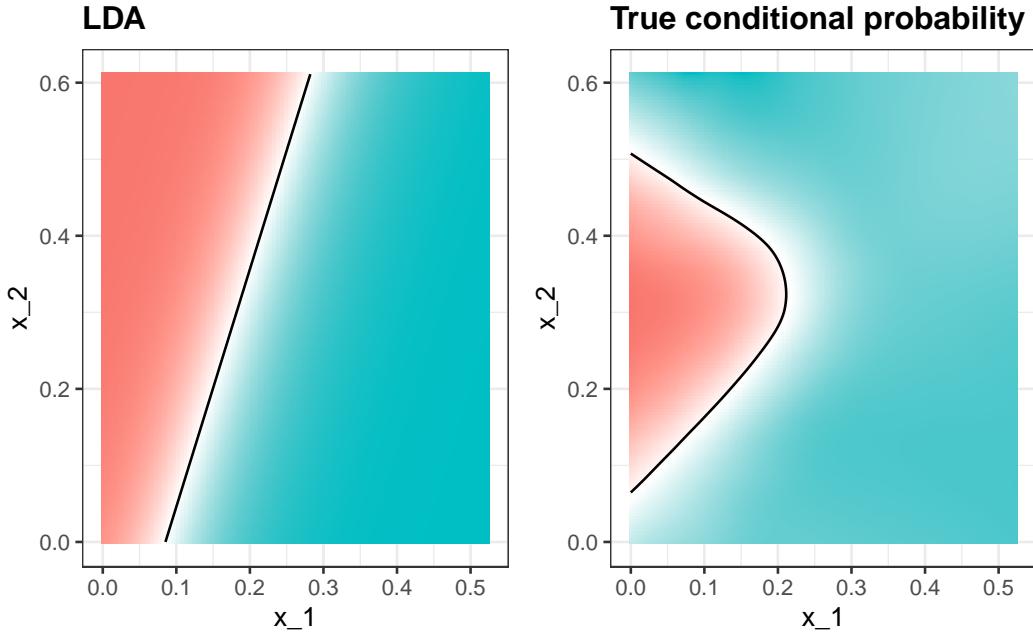


Ahora el tamaño de las elipses y el ángulo son iguales. Esto se debe a que tienen las mismas desviaciones estándar y correlaciones.

Podemos ajustar el modelo LDA usando **caret**:

```
train_lda <- train(y ~ ., method = "lda", data = mnist_27$train)
y_hat <- predict(train_lda, mnist_27$test)
confusionMatrix(y_hat, mnist_27$test$y)$overall[["Accuracy"]]
#> Accuracy
#> 0.75
```

Cuando forzamos esta suposición, podemos mostrar matemáticamente que el umbral es una línea, al igual que con la regresión logística. Por esta razón, llamamos al método *análisis lineal discriminante* (LDA). Del mismo modo, para QDA, podemos mostrar que el umbral debe ser una función cuadrática.



En el caso de LDA, la falta de flexibilidad no nos permite capturar la no linealidad en la verdadera función de probabilidad condicional.

### 31.7.5 Conexión a distancia

La densidad normal es:

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{-\frac{(x-\mu)^2}{\sigma^2}\right\}$$

Si eliminamos la constante  $1/(\sqrt{2\pi}\sigma)$  y luego tomamos el logaritmo, obtenemos:

$$-\frac{(x-\mu)^2}{\sigma^2}$$

que es el negativo de una distancia al cuadrado escalada por la desviación estándar. Para dimensiones mayores, lo mismo es cierto, excepto que la escala es más compleja e implica correlaciones.

## 31.8 Estudio de caso: más de tres clases

Podemos generar un ejemplo con tres categorías así:

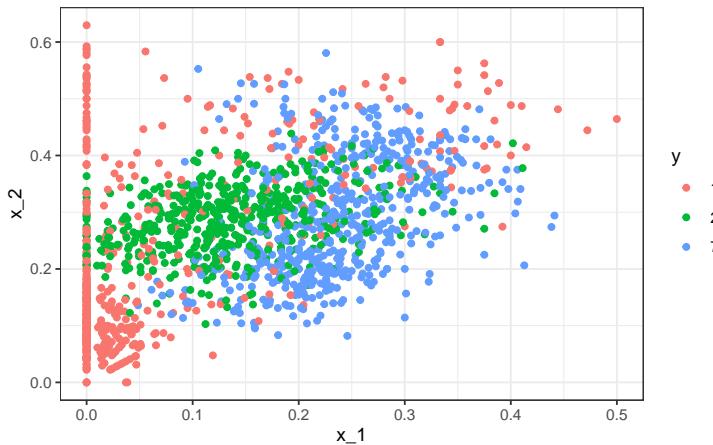
```

if(!exists("mnist")) mnist <- read_mnist()
set.seed(3456)
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)
get the quadrants
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)
binarize the values. Above 200 is ink, below is no ink
x <- x > 200
proportion of pixels in lower right quadrant
x <- cbind(rowSums(x[,upper_left_ind])/rowSums(x),
 rowSums(x[,lower_right_ind])/rowSums(x))
##save data
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])

```

Aquí están los datos de entrenamiento:

```
train_set %>% ggplot(aes(x_1, x_2, color=y)) + geom_point()
```



Podemos usar el paquete **caret** para entrenar el modelo QDA:

```
train_qda <- train(y ~ ., method = "qda", data = train_set)
```

Ahora estimamos tres probabilidades condicionales (aunque tienen que sumar a 1):

```

predict(train_qda, test_set, type = "prob") %>% head()
#> 1 2 7

```

```
#> 1 0.7655 0.23043 0.00405
#> 2 0.2031 0.72514 0.07175
#> 3 0.5396 0.45909 0.00132
#> 4 0.0393 0.09419 0.86655
#> 5 0.9600 0.00936 0.03063
#> 6 0.9865 0.00724 0.00623
```

Nuestras predicciones son una de las tres clases:

```
predict(train_qda, test_set) %>% head()
#> [1] 1 2 1 7 1 1
#> Levels: 1 2 7
```

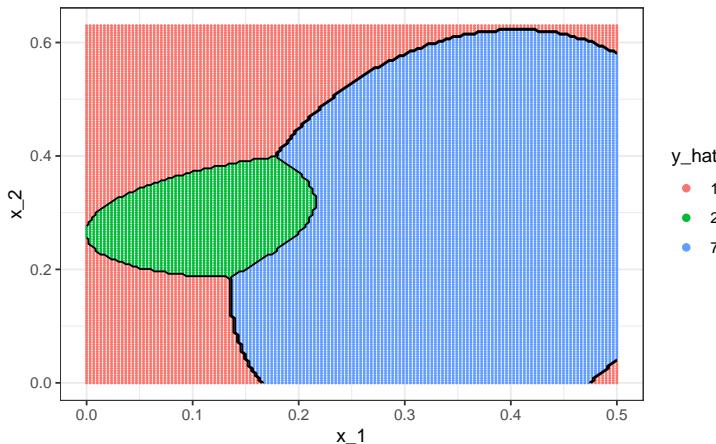
La matriz de confusión es, por lo tanto, una tabla de 3 por 3:

```
confusionMatrix(predict(train_qda, test_set), test_set$y)$table
#>
#> Reference
#> Prediction 1 2 7
#> 1 111 9 11
#> 2 10 86 21
#> 7 21 28 102
```

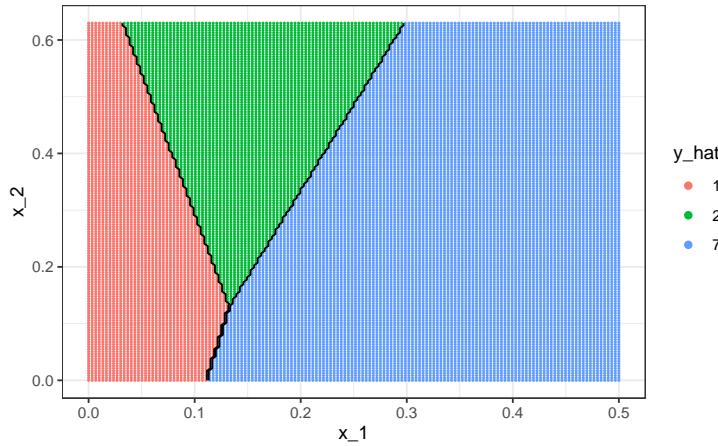
La exactitud es 0.749.

Tengan en cuenta que para la sensibilidad y especificidad, tenemos un par de valores para **cada** clase. Para definir estos términos, necesitamos un resultado binario. Por lo tanto, tenemos tres columnas: una para cada clase como positivos y las otras dos como negativas.

Para visualizar qué partes de la región se llaman 1, 2 y 7, ahora necesitamos tres colores:



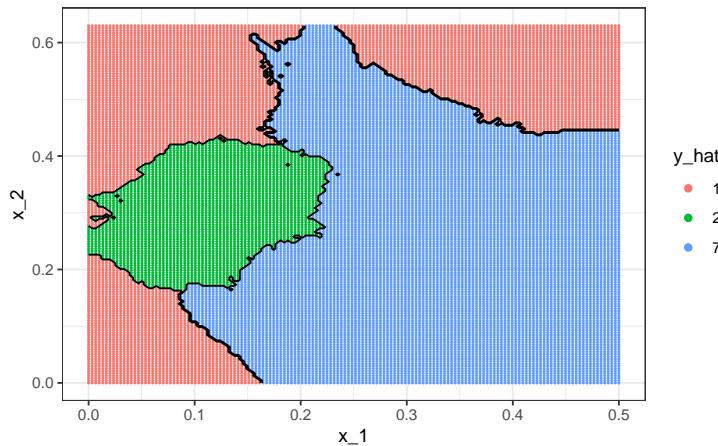
La exactitud para LDA, 0.629, es mucho peor porque el modelo es más rígido. Aquí vemos como se ve la regla de decisión:



Los resultados para kNN:

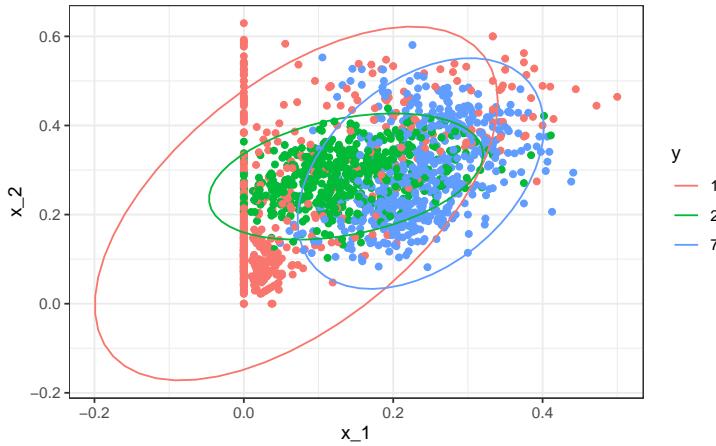
```
train_knn <- train(y ~ ., method = "knn", data = train_set,
 tuneGrid = data.frame(k = seq(15, 51, 2)))
```

son mucho mejores con una exactitud de 0.749. La regla de decisión se ve así:



Noten que una de las limitaciones de los modelos generativos mostrados aquí se debe a la falta de ajuste del supuesto normal, en particular para la clase 1.

```
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")
```



Los modelos generativos pueden ser muy útiles, pero solo cuando somos capaces de aproximar con éxito la distribución de predictores condicionados en cada clase.

### 31.9 Ejercicios

Vamos a aplicar LDA y QDA al set de datos `tissue_gene_expression`. Comenzaremos con ejemplos sencillos basados en este set de datos y luego desarrollaremos un ejemplo realista.

1. Cree un set de datos con solo las clases “cerebellum” e “hippocampus” (dos partes del cerebro) y una matriz de predicción con 10 columnas seleccionadas al azar.

```
set.seed(1993)
data("tissue_gene_expression")
tissues <- c("cerebellum", "hippocampus")
ind <- which(tissue_gene_expression$y %in% tissues)
y <- droplevels(tissue_gene_expression$y[ind])
x <- tissue_gene_expression$x[ind,]
x <- x[, sample(ncol(x), 10)]
```

Utilice la función `train` para estimar la exactitud de LDA.

2. En este caso, LDA se ajusta a dos distribuciones normales de 10 dimensiones. Mire el modelo ajustado mirando el componente `finalModel` del resultado de `train`. Observe que hay un componente llamado `means` que incluye el estimador de los promedios de ambas distribuciones. Grafique este vector de promedios uno contra el otro y determine qué predictores (genes) parecen estar impulsando el algoritmo.

3. Repita el ejercicio 1 con QDA. ¿Tiene mejor exactitud que LDA?

4. ¿Los mismos predictores (genes) impulsan el algoritmo? Haga un gráfico como en el ejercicio 2.

5. Algo que vemos en el gráfico anterior es que el valor de los predictores se correlaciona en ambos grupos: algunos predictores son bajos en ambos grupos, mientras que otros son

altos en ambos grupos. El valor medio de cada predictor, `colMeans(x)`, no es informativo ni útil para la predicción, y para fines de interpretación, a menudo es útil centrar o escalar cada columna. Esto se puede lograr con el argumento `preProcessing` en `train`. Vuelva a ejecutar LDA con `preProcessing = "scale"`. Tenga en cuenta que la exactitud no cambia, pero vea cómo es más fácil identificar los predictores que difieren más entre los grupos en el gráfico realizado en el ejercicio 4.

6. En los ejercicios anteriores, vimos que ambos enfoques funcionaron bien. Grafique los valores predictores para los dos genes con las mayores diferencias entre los dos grupos en un diagrama de dispersión para ver cómo parecen seguir una distribución normal de dos variables como se supone para los enfoques LDA y QDA. Coloree los puntos por el resultado.
7. Ahora vamos a aumentar un poco la complejidad del desafío: consideraremos todos los tipos de tejidos.

```
set.seed(1993)
data("tissue_gene_expression")
y <- tissue_gene_expression$y
x <- tissue_gene_expression$x
x <- x[, sample(ncol(x), 10)]
```

¿Qué exactitud obtiene con LDA?

8. Vemos que los resultados son ligeramente peores. Utilice la función `confusionMatrix` para aprender qué tipo de errores estamos cometiendo.
9. Grafique una imagen de los centros de las siete distribuciones normales de 10 dimensiones.

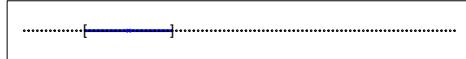
## 31.10 Árboles de clasificación y regresión (CART)

### 31.10.1 La maldición de la dimensionalidad

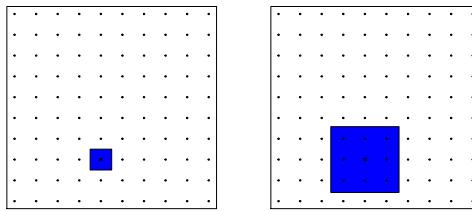
Describimos cómo métodos como LDA y QDA no deben usarse con muchos predictores  $p$  porque el número de parámetros que necesitamos estimar se vuelve demasiado grande. Por ejemplo, con el ejemplo de dígitos  $p = 784$ , tendríamos más de 600,000 parámetros con LDA y lo multiplicaríamos por el número de clases para QDA. Los métodos de *kernel*, como kNN o regresión local, no tienen parámetros de modelo para estimar. Sin embargo, también se enfrentan a un desafío cuando se utilizan predictores múltiples debido a lo que se conoce como la *maldición de la dimensionalidad*. La *dimensión* aquí se refiere al hecho de que cuando tenemos  $p$  predictores, la distancia entre dos observaciones se calcula en un espacio  $p$ -dimensional.

Una manera útil de entender la maldición de la dimensionalidad es considerar cuán grande tenemos que hacer un *span/vecindario/ventana* para incluir un porcentaje dado de los datos. Recuerden que con vecindarios más grandes, nuestros métodos pierden flexibilidad.

Por ejemplo, supongan que tenemos un predictor continuo con puntos igualmente espaciados en el intervalo  $[0,1]$  y queremos crear ventanas que incluyen 1/10 de datos. Entonces es fácil ver que nuestras ventanas tienen que ser de tamaño 0.1:

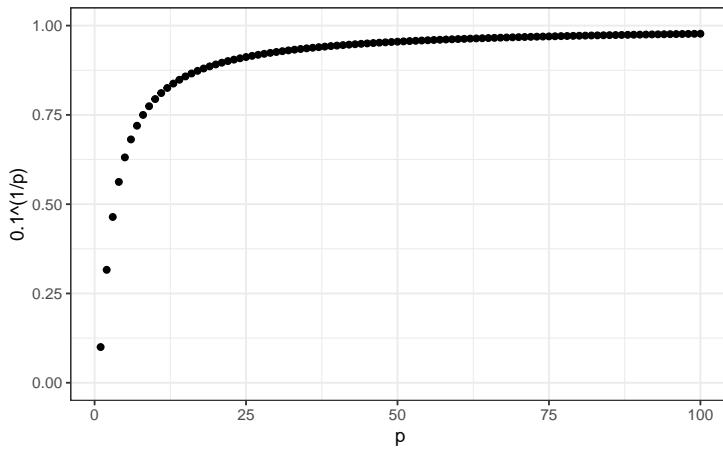


Ahora, para dos predictores, si decidimos mantener el vecindario igual de pequeño, 10% para cada dimensión, incluimos solo 1 punto. Si queremos incluir el 10% de los datos, entonces necesitamos aumentar el tamaño de cada lado del cuadrado a  $\sqrt{10} \approx .316$ :



Usando la misma lógica, si queremos incluir el 10% de los datos en un espacio tridimensional, entonces el lado de cada cubo es  $\sqrt[3]{10} \approx 0.464$ . En general, para incluir el 10% de los datos en un caso con  $p$  dimensiones, necesitamos un intervalo con cada lado del tamaño  $\sqrt[p]{10}$  del total. Esta proporción se acerca a 1 rápidamente y, si la proporción es 1, significa que incluimos todos los datos y ya no estamos suavizando.

```
library(tidyverse)
p <- 1:100
qplot(p, .1^(1/p), ylim = c(0,1))
```



Cuando llegamos a 100 predictores, el vecindario ya no es muy local, puesto que cada lado cubre casi todo el set de datos.

Aquí observamos un conjunto de métodos elegantes y versátiles que se adaptan a dimensiones más altas y también permiten que estas regiones tomen formas más complejas mientras producen modelos que son interpretables. Estos son métodos muy populares, conocidos y estudiados. Nos concentraremos en los árboles de regresión y decisión y su extensión a bosques aleatorios.

### 31.10.2 Motivación CART

Para motivar esta sección, utilizaremos un nuevo set de datos que incluye el desglose de la composición del aceite de oliva en 8 ácidos grasos:

```
library(tidyverse)
library(dslabs)
data("olive")
names(olive)
#> [1] "region" "area" "palmitic" "palmitoleic"
#> [5] "stearic" "oleic" "linoleic" "linolenic"
#> [9] "arachidic" "eicosenoic"
```

Con fines ilustrativos, intentaremos predecir la región utilizando los valores de composición de ácidos grasos como predictores.

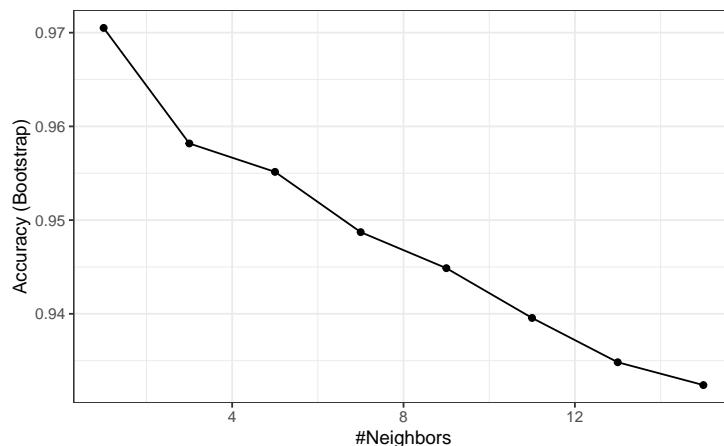
```
table(olive$region)
#>
#> Northern Italy Sardinia Southern Italy
#> 151 98 323
```

Quitamos la columna `area` porque no la usaremos como predictor.

```
olive <- select(olive, -area)
```

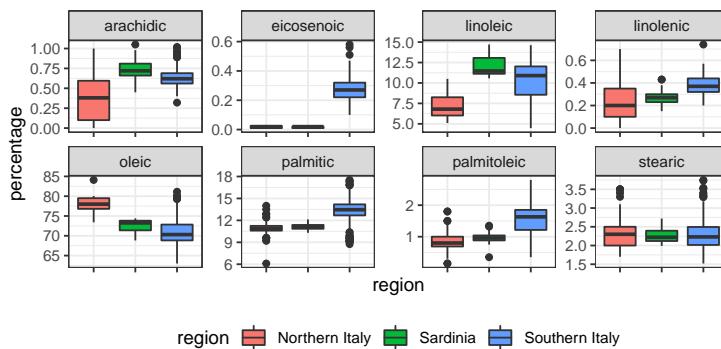
Intentemos rápidamente predecir la región usando kNN:

```
library(caret)
fit <- train(region ~ ., method = "knn",
 tuneGrid = data.frame(k = seq(1, 15, 2)),
 data = olive)
ggplot(fit)
```

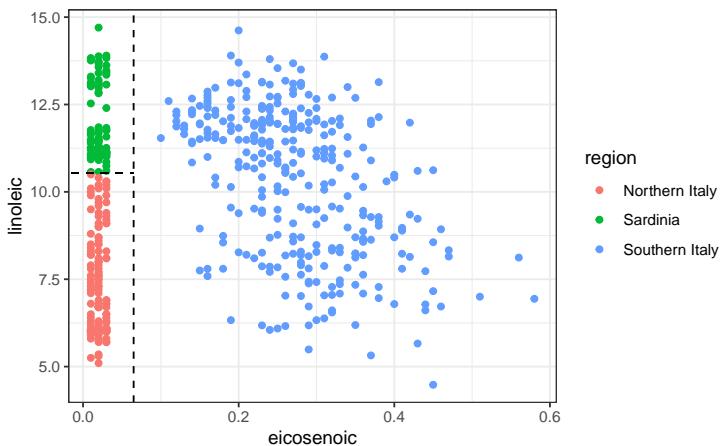


Vemos que usando solo un vecino, podemos predecir relativamente bien. Sin embargo, un

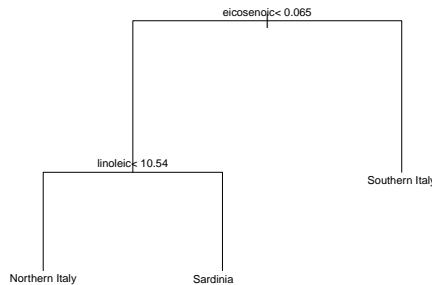
poco de exploración de datos revela que deberíamos poder hacerlo aún mejor. Por ejemplo, si observamos la distribución de cada predictor estratificado por región, vemos que el *eicosenoic* solo está presente en el sur de Italia y que el *linoleic* separa el norte de Italia de Cerdeña.



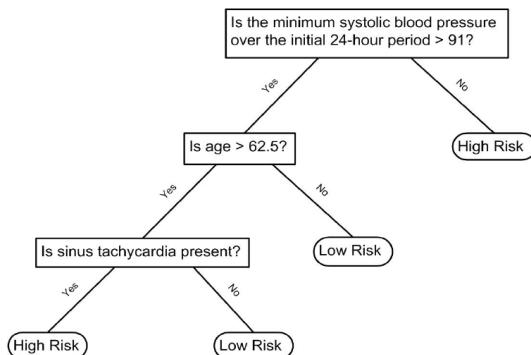
¡Esto implica que deberíamos ser capaces de construir un algoritmo que prediga perfectamente! Podemos ver esto claramente al graficar los valores para *eicosenoic* y *linoleic*.



En la Sección 33.3.4, definimos espacios predictores. El espacio predictor aquí consiste en puntos de ocho dimensiones con valores entre 0 y 100. En el gráfico anterior, mostramos el espacio definido por los dos predictores *eicosenoic* y *linoleic* y, a simple vista, podemos construir una regla de predicción que divida el espacio del predictor para que cada partición contenga solo resultados de una categoría. Esto a su vez se puede utilizar para definir un algoritmo con una exactitud perfecta. Específicamente, definimos la siguiente regla de decisión. Si el *eicosenoic* es mayor que 0.065, predecimos el sur de Italia. Si no, entonces si *linoleic* es más grande que 10.535, predecimos Cerdeña, y si es más bajo, predecimos el norte de Italia. Podemos dibujar este árbol de decisión así:



Los árboles de decisión como este se usan a menudo en la práctica. Por ejemplo, para determinar el riesgo de una persona de tener un mal resultado después de un ataque cardíaco, los médicos usan lo siguiente:



(Fuente: Walton 2010 Informal Logic, Vol. 30, No. 2, pp. 159-184<sup>3</sup>.)

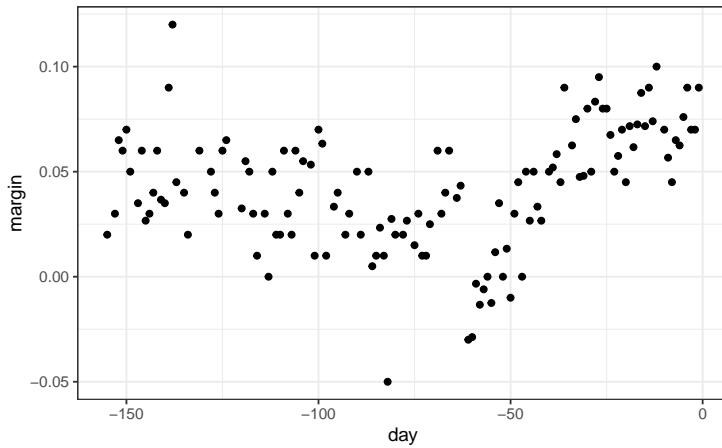
Un árbol es básicamente un diagrama de flujo con preguntas de sí o no. La idea general de los métodos que estamos describiendo es definir un algoritmo que use datos para crear estos árboles con predicciones en los extremos, conocidos como *nodos* (*nodes* en inglés). Los árboles de regresión y de decisión operan prediciendo una variable de resultado  $Y$  al dividir los predictores.

### 31.10.3 Árboles de regresión

Cuando el resultado es continuo, llamamos al método un árbol de *regresión*. Para introducir árboles de regresión, utilizaremos los datos de la encuesta de 2008 que usamos en secciones anteriores para describir la idea básica de cómo construimos estos algoritmos. Al igual que con otros algoritmos de *machine learning*, intentaremos estimar la expectativa condicional  $f(x) = E(Y|X = x)$  con  $Y$  el margen de la encuesta y  $x$  el día.

<sup>3</sup>[https://papers.ssrn.com/sol3/Delivery.cfm/SSRN\\_ID1759289\\_code1486039.pdf?abstractid=1759289&zmirid=1&ztype=2](https://papers.ssrn.com/sol3/Delivery.cfm/SSRN_ID1759289_code1486039.pdf?abstractid=1759289&zmirid=1&ztype=2)

```
data("polls_2008")
qplot(day, margin, data = polls_2008)
```



La idea general aquí es construir un árbol de decisión y, al final de cada *nodo*, obtener un predictor  $\hat{y}$ . Una forma matemática de describir esto es decir que estamos dividiendo el espacio predictivo en  $J$  regiones no superpuestas,  $R_1, R_2, \dots, R_J$ , y luego para cualquier predictor  $x$  que caiga dentro de la región  $R_j$ , estimar  $f(x)$  con el promedio de las observaciones de entrenamiento  $y_i$  para el cual el predictor asociado  $x_i$  también está en  $R_j$ .

¿Pero cómo decidimos la partición  $R_1, R_2, \dots, R_J$  y como elegimos  $J$ ? Aquí es donde el algoritmo se vuelve un poco complicado.

Los árboles de regresión crean particiones de manera recursiva. Comenzamos el algoritmo con una partición, el espacio predictor completo. En nuestro primer ejemplo sencillo, este espacio es el intervalo  $[-155, 1]$ . Pero después del primer paso, tendremos dos particiones. Después del segundo paso, dividiremos una de estas particiones en dos y tendremos tres particiones, luego cuatro, entonces cinco, y así sucesivamente. Describimos cómo elegimos la partición para una partición adicional, y cuándo parar, más adelante.

Después de seleccionar una partición  $\mathbf{x}$  para dividir a fin de crear las nuevas particiones, encontramos un predictor  $j$  y un valor  $s$  que definen dos nuevas particiones, que llamaremos  $R_1(j, s)$  y  $R_2(j, s)$  y que dividen nuestras observaciones en la partición actual al preguntar si  $x_j$  es mayor que  $s$ :

$$R_1(j, s) = \{\mathbf{x} \mid x_j < s\} \text{ and } R_2(j, s) = \{\mathbf{x} \mid x_j \geq s\}$$

En nuestro ejemplo actual, solo tenemos un predictor, por lo que siempre elegiremos  $j = 1$ , pero en general este no será el caso. Ahora, después de definir las nuevas particiones  $R_1$  y  $R_2$  y parar el proceso de particionar, calculamos predictores tomando el promedio de todas las observaciones  $y$  para el cual el  $\mathbf{x}$  asociado está en  $R_1$  y  $R_2$ . Nos referimos a estos dos como  $\hat{y}_{R_1}$  y  $\hat{y}_{R_2}$  respectivamente.

¿Pero cómo elegimos  $j$  y  $s$ ? Básicamente, encontramos el par que minimiza la suma de errores cuadrados (*residual sum of squares* o RSS por sus siglas en inglés):

$$\sum_{i: x_i \in R_1(j, s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i: x_i \in R_2(j, s)} (y_i - \hat{y}_{R_2})^2$$

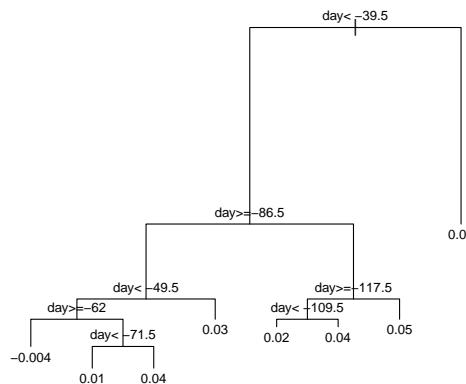
Esto se aplica de manera recursiva a las nuevas regiones  $R_1$  y  $R_2$ . Describimos cómo paramos más tarde, pero una vez que terminemos de dividir el espacio del predictor en regiones, en cada región se realiza una predicción utilizando las observaciones en esa región.

Echemos un vistazo a lo que hace este algoritmo en los datos de la encuesta de las elecciones presidenciales de 2008. Utilizaremos la función `rpart` del paquete `rpart`.

```
library(rpart)
fit <- rpart(margin ~ ., data = polls_2008)
```

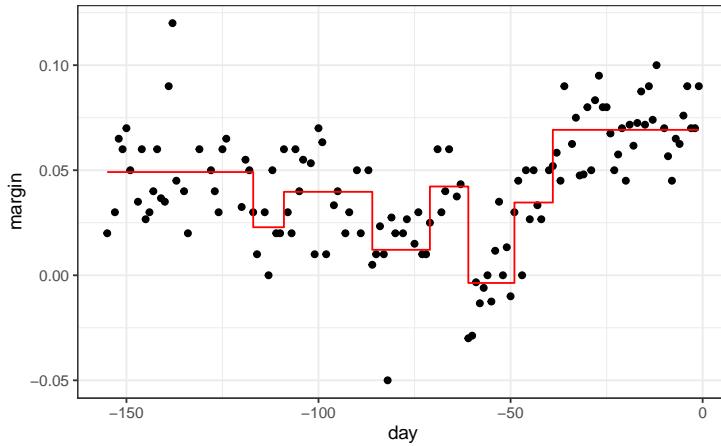
Aquí, solo hay un predictor  $y$ , por lo tanto, no tenemos que decidir cuál dividir. Simplemente tenemos que decidir qué valor  $s$  utilizaremos para dividir. Podemos ver visualmente dónde se hicieron las divisiones:

```
plot(fit, margin = 0.1)
text(fit, cex = 0.75)
```



La primera división se realiza el día 39.5. Una de esas regiones se divide en el día 86.5. Las dos nuevas particiones que resultan se dividen en los días 49.5 y 117.5, respectivamente, y así sucesivamente. Terminamos con 8 particiones. El estimador final  $\hat{f}(x)$  se ve así:

```
polls_2008 %>%
 mutate(y_hat = predict(fit)) %>%
 ggplot() +
 geom_point(aes(day, margin)) +
 geom_step(aes(day, y_hat), col="red")
```



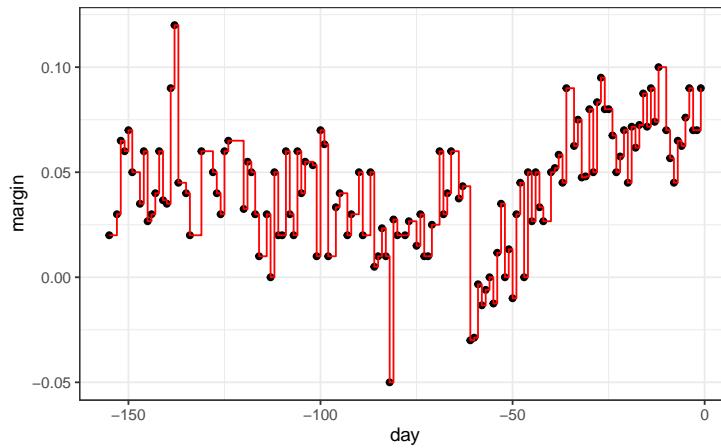
Observen que el algoritmo paró luego de 8 particiones. Ahora explicamos cómo se toma esa decisión.

Primero, necesitamos definir el término *parámetro de complejidad* (*complexity parameter* o *cp* por sus siglas en inglés). Cada vez que dividimos y definimos dos nuevas particiones, nuestro set de entrenamiento RSS disminuye. Esto se debe a que con más particiones, nuestro modelo tiene más flexibilidad para adaptarse a los datos de entrenamiento. De hecho, si se divide hasta que cada punto sea su propia partición, entonces RSS baja hasta 0 ya que el promedio de un valor es el mismo valor. Para evitar esto, el algoritmo establece un mínimo de cuánto debe mejorar el RSS para que se agregue otra partición. Este parámetro se conoce como *parámetro de complejidad*. El RSS debe mejorar por un factor de *cp* para que se agregue la nueva partición. Por lo tanto, los valores grandes de *cp* obligarán al algoritmo a detenerse antes, lo que resulta en menos nodos.

Sin embargo, *cp* no es el único parámetro utilizado para decidir si debemos dividir una partición existente. Otro parámetro común es el número mínimo de observaciones requeridas en una partición antes de dividirla más. El argumento que se usa en la función *rpart* es *minsplit* y el valor predeterminado es 20. La implementación *rpart* de árboles de regresión también permite a los usuarios determinar un número mínimo de observaciones en cada nodo. El argumento es *minbucket* y por defecto usa el valor *round(minsplit/3)*.

Como se esperaba, si establecemos *cp* = 0 y *minsplit* = 2, nuestra predicción es lo más flexible posible y nuestros predictores son nuestros datos originales:

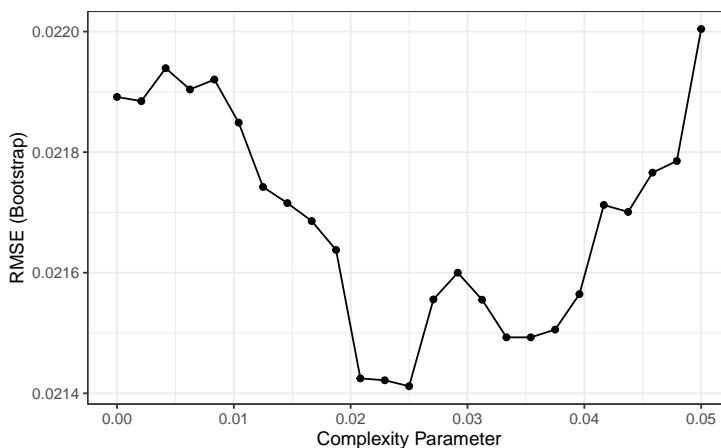
```
fit <- rpart(margin ~ ., data = polls_2008,
 control = rpart.control(cp = 0, minsplit = 2))
polls_2008 %>%
 mutate(y_hat = predict(fit)) %>%
 ggplot() +
 geom_point(aes(day, margin)) +
 geom_step(aes(day, y_hat), col="red")
```



Intuitivamente, sabemos que este no es un buen enfoque, ya que generalmente dará como resultado un entrenamiento excesivo. Estos tres parámetros, `cp`, `minsplit` y `minbucket`, se pueden usar para controlar la variabilidad de los predictores finales. Entre más grandes sean estos valores, más datos se promedian para calcular un predictor y, por lo tanto, reducir la variabilidad. El inconveniente es que restringe la flexibilidad.

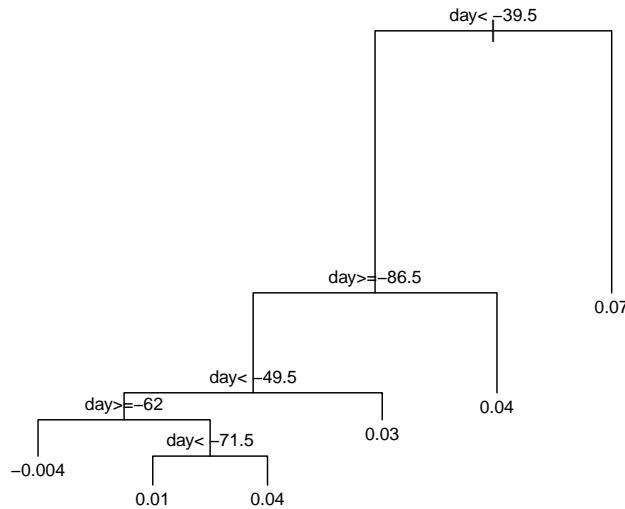
Entonces, ¿cómo elegimos estos parámetros? Podemos usar validación cruzada, descrita en el Capítulo 29, como con cualquier parámetro de ajuste. Aquí tenemos un ejemplo del uso de validación cruzada para elegir `cp`:

```
library(caret)
train_rpart <- train(margin ~ .,
 method = "rpart",
 tuneGrid = data.frame(cp = seq(0, 0.05, len = 25)),
 data = polls_2008)
ggplot(train_rpart)
```



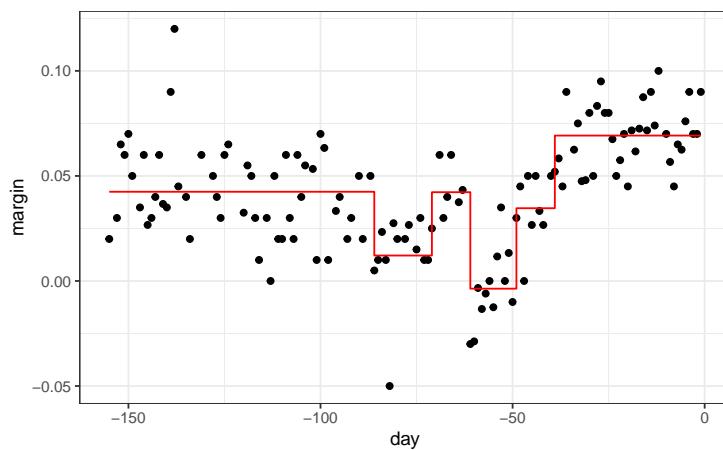
Para ver el árbol que resulta, accedemos `finalModel` y lo graficamos:

```
plot(train_rpart$finalModel, margin = 0.1)
text(train_rpart$finalModel, cex = 0.75)
```



Y debido a que solo tenemos un predictor, podemos graficar  $\hat{f}(x)$ :

```
polls_2008 %>%
 mutate(y_hat = predict(train_rpart)) %>%
 ggplot() +
 geom_point(aes(day, margin)) +
 geom_step(aes(day, y_hat), col="red")
```



Tengan en cuenta que si ya tenemos un árbol y queremos aplicar un valor de `cp` más alto, podemos usar la función `prune`. Llamamos a esto *podar* (*pruning* en inglés) un árbol porque estamos cortando particiones que no cumplen con un criterio `cp`. Anteriormente creamos un árbol que usaba un `cp = 0` y lo guardamos en `fit`. Podemos podarlo así:

```
pruned_fit <- prune(fit, cp = 0.01)
```

### 31.10.4 Árboles de clasificación (decisión)

Los árboles de clasificación, o árboles de decisión, se usan en problemas de predicción donde el resultado es categórico. Utilizamos el mismo principio de partición con algunas diferencias para tomar en cuenta el hecho de que ahora estamos trabajando con un resultado categórico.

La primera diferencia es que formamos predicciones calculando qué clase es la más común entre las observaciones del set de entrenamiento dentro de la partición, en lugar de tomar el promedio en cada partición (ya que no podemos tomar el promedio de las categorías).

La segunda es que ya no podemos usar RSS para elegir la partición. Si bien podríamos utilizar el enfoque simplista de buscar particiones que minimicen el error de entrenamiento, los enfoques de mejor desempeño utilizan métricas más sofisticadas. Dos de los más populares son el *índice de Gini* (*Gini Index* en inglés) y *entropía* (*entropy* en inglés).

En una situación perfecta, los resultados en cada una de nuestras particiones son todos de la misma categoría, ya que esto permitirá una exactitud perfecta. El *índice de Gini* será 0 en este caso y se hará más grande a medida que nos desviamos de este escenario. Para definir el índice de Gini, definimos  $\hat{p}_{j,k}$  como la proporción de observaciones en partición  $j$  que son de clase  $k$ . Específicamente, el índice de Gini se define como:

$$\text{Gini}(j) = \sum_{k=1}^K \hat{p}_{j,k} (1 - \hat{p}_{j,k})$$

Si estudian la fórmula cuidadosamente, verán que es 0 en la situación perfecta descrita anteriormente.

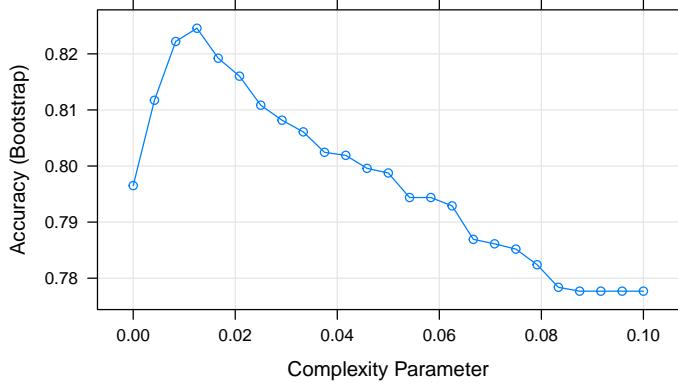
*Entropía* es una cantidad muy similar, definida como:

$$\text{entropy}(j) = - \sum_{k=1}^K \hat{p}_{j,k} \log(\hat{p}_{j,k}), \text{ with } 0 \times \log(0) \text{ defined as 0}$$

Veamos cómo funciona un árbol de clasificación en el ejemplo de dígitos que examinamos antes utilizando este código para ejecutar el algoritmo y trazar la exactitud resultante:

```
library(dslabs)
data("mnist_27")

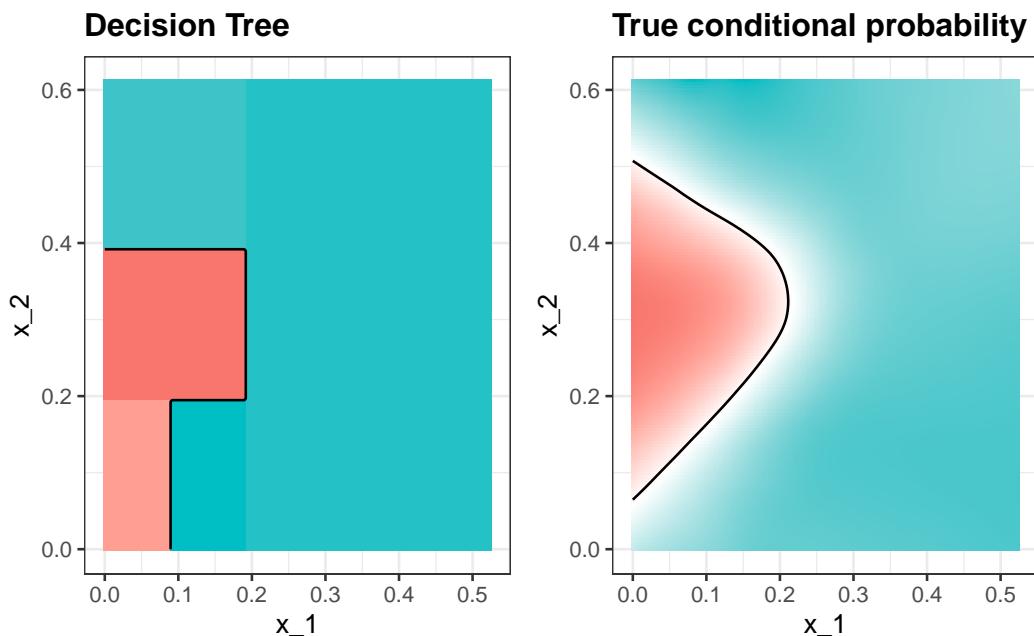
train_rpart <- train(y ~ .,
 method = "rpart",
 tuneGrid = data.frame(cp = seq(0.0, 0.1, len = 25)),
 data = mnist_27$train)
plot(train_rpart)
```



La exactitud que logramos con este enfoque es mejor que la que obtuvimos con la regresión, pero no es tan buena como la que obtuvimos con los métodos *kernel*:

```
y_hat <- predict(train_rpart, mnist_27$test)
confusionMatrix(y_hat, mnist_27$test$y)$overall["Accuracy"]
#> Accuracy
#> 0.805
```

El gráfico de la probabilidad condicional estimada nos muestra las limitaciones de los árboles de clasificación:



Tengan en cuenta que con los árboles de decisión, es difícil suavizar los límites ya que cada partición crea una discontinuidad.

Los árboles de clasificación tienen ciertas ventajas que los hacen muy útiles. Son altamente interpretables, incluso más que los modelos lineales. Son fáciles de visualizar (si son lo suficientemente pequeños). Finalmente, pueden modelar procesos de decisión humana y no requieren el uso de predictores ficticios para variables categóricas. Por otro lado, si usamos particiones recursivas es muy posible que sobreentrenemos y, por lo tanto, es un poco más difícil de entrenar que, por ejemplo, la regresión lineal o kNN. Además, en términos de exactitud, rara vez es el método de mejor rendimiento, ya que no es muy flexible y es muy inestable a los cambios en los datos de entrenamiento. Los bosques aleatorios, explicados a continuación, mejoran varias de estas deficiencias.

### 31.11 Bosques aleatorios

Los bosques aleatorios son un enfoque de *machine learning* **muy popular** que abordan las deficiencias de los árboles de decisión utilizando una idea inteligente. El objetivo es mejorar la predicción y reducir la inestabilidad mediante *el promedio* de múltiples árboles de decisión (un bosque de árboles construido con aleatoriedad). Tienen dos atributos que ayudan a lograr esto.

El primer paso es *bootstrap aggregation* o *bagging*. La idea general es generar muchos predictores, cada uno utilizando árboles de regresión o de clasificación, y luego formar una predicción final basada en la predicción promedio de todos estos árboles. Para asegurar que los árboles individuales no sean iguales, utilizamos el *bootstrap* para inducir aleatoriedad. Estos dos atributos combinados explican el nombre: el *bootstrap* hace que los árboles individuales sean **aleatorios** y la combinación de árboles es el **bosque**. Los pasos específicos son los siguientes.

1. Construyan  $B$  árboles de decisión utilizando el set de entrenamiento. Nos referimos a los modelos ajustados como  $T_1, T_2, \dots, T_B$ . Entonces explicamos cómo nos aseguramos de que sean diferentes.
2. Para cada observación en el set de evaluación, formen una predicción  $\hat{y}_j$  usando el árbol  $T_j$ .
3. Para resultados continuos, formen una predicción final con el promedio  $\hat{y} = \frac{1}{B} \sum_{j=1}^B \hat{y}_j$ . Para la clasificación de datos categóricos, predigan  $\hat{y}$  con voto mayoritario (clase más frecuente entre  $\hat{y}_1, \dots, \hat{y}_T$ ).

Entonces, ¿cómo obtenemos diferentes árboles de decisión de un solo set de entrenamiento? Para esto, usamos la aleatoriedad en dos maneras que explicamos en los pasos a continuación. Dejen que  $N$  sea el número de observaciones en el set de entrenamiento. Para crear  $T_j$ ,  $j = 1, \dots, B$  del set de entrenamiento, hagan lo siguiente:

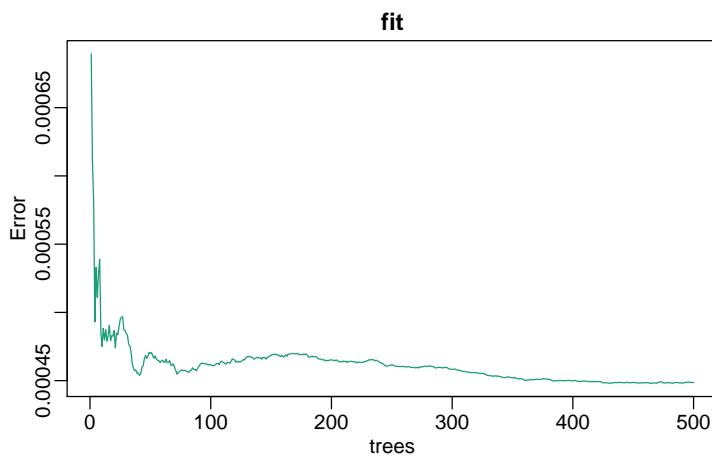
1. Crean un set de entrenamiento de *bootstrap* al mostrar  $N$  observaciones del set de entrenamiento **con reemplazo**. Esta es la primera forma de inducir aleatoriedad.
2. Una gran cantidad de atributos es típico en los desafíos de *machine learning*. A menudo, muchos atributos pueden ser informativos, pero incluirlos todos en el modelo puede resultar en un sobreajuste. La segunda forma en que los bosques aleatorios inducen aleatoriedad es seleccionando al azar los atributos que se incluirán en la construcción de cada árbol. Se selecciona un subconjunto aleatorio diferente para cada árbol. Esto reduce la correlación entre los árboles en el bosque, mejorando así la exactitud de la predicción.

Para ilustrar cómo los primeros pasos pueden dar como resultado estimadores más uniformes, demostraremos ajustando un bosque aleatorio a los datos de las encuestas de 2008. Utilizaremos la función `randomForest` en el paquete `randomForest`:

```
library(randomForest)
fit <- randomForest(margin~., data = polls_2008)
```

Noten que si aplicamos la función `plot` al objeto resultante, almacenado en `fit`, vemos cómo cambia la tasa de error de nuestro algoritmo a medida que agregamos árboles.

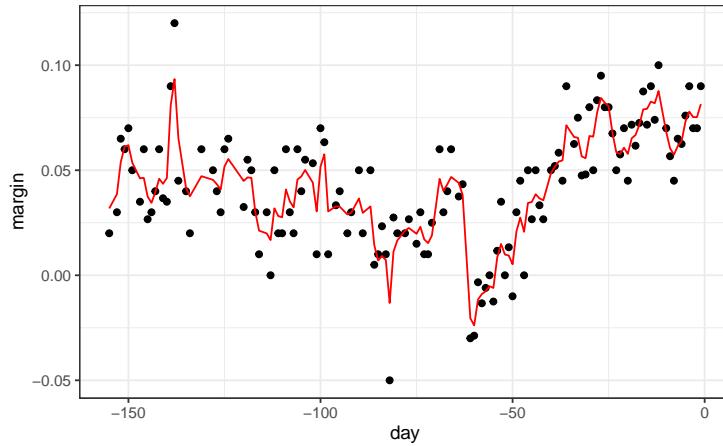
```
rafaelib::mypar()
plot(fit)
```



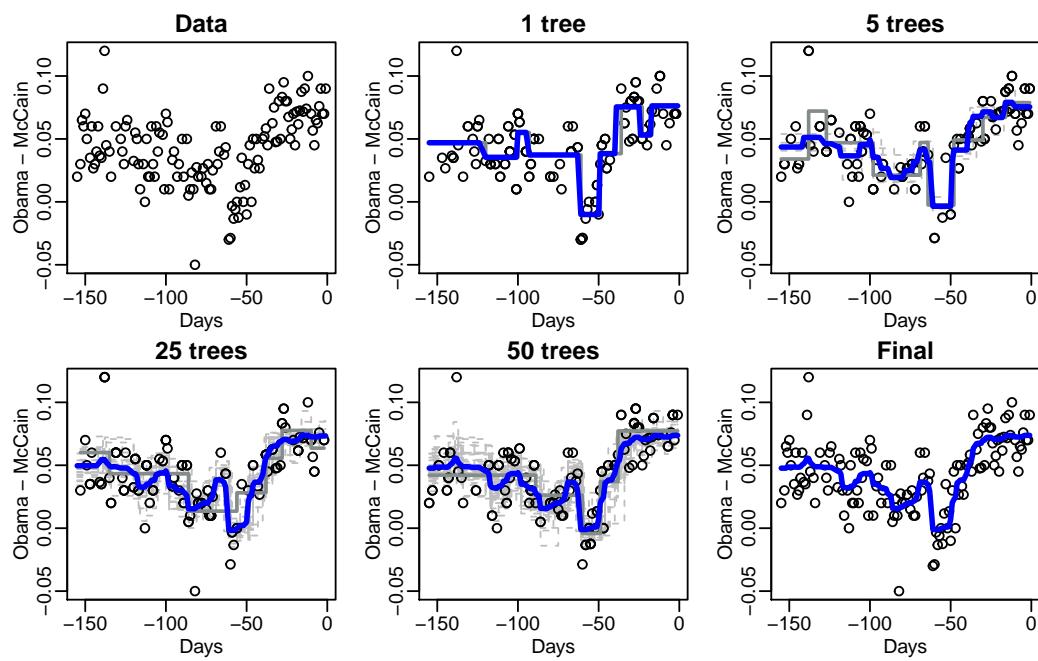
Podemos ver que en este caso, la exactitud mejora a medida que agregamos más árboles hasta unos 30 árboles donde la exactitud se estabiliza.

El estimador resultante para este bosque aleatorio puede verse así:

```
polls_2008 %>%
 mutate(y_hat = predict(fit, newdata = polls_2008)) %>%
 ggplot() +
 geom_point(aes(day, margin)) +
 geom_line(aes(day, y_hat), col="red")
```



Observen que el estimador del bosque aleatorio es mucho más uniforme que lo que logramos con el árbol de regresión en la sección anterior. Esto es posible porque el promedio de muchas funciones de escalón puede ser suave. Podemos ver esto examinando visualmente cómo cambia el estimador a medida que agregamos más árboles. En el siguiente gráfico, pueden ver cada una de las muestras de *bootstrap* para varios valores de  $b$  y para cada una vemos el árbol que se ajusta en gris, los árboles anteriores que se ajustaron en gris más claro y el resultado de tomar el promedio de todos los árboles estimadores hasta ese punto.

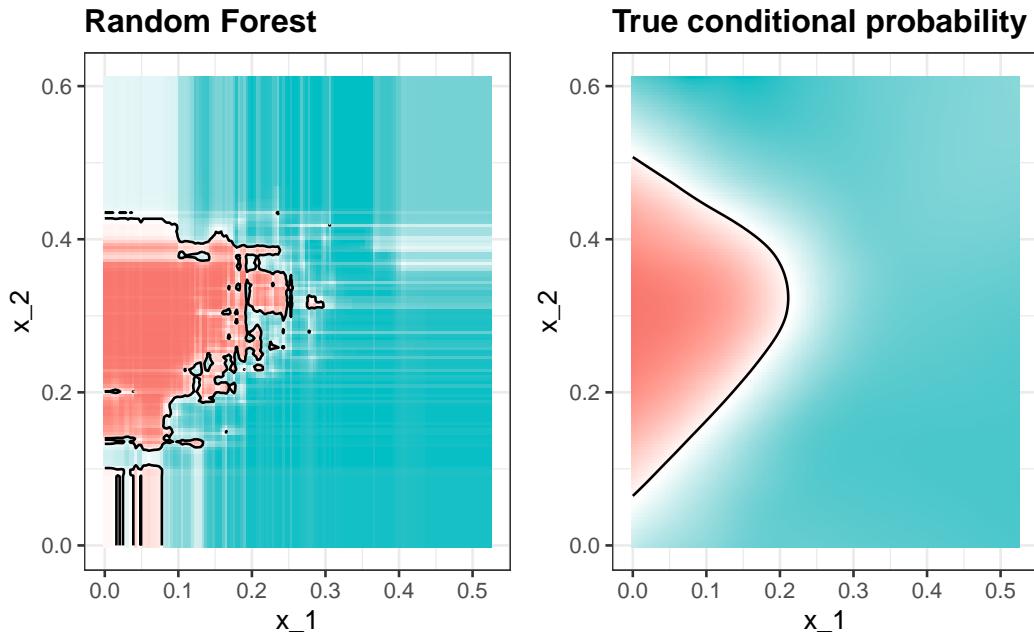


Aquí está el ajuste del bosque aleatorio para nuestro ejemplo de dígitos basado en dos predictores:

```
library(randomForest)
train_rf <- randomForest(y ~ ., data=mnist_27$train)

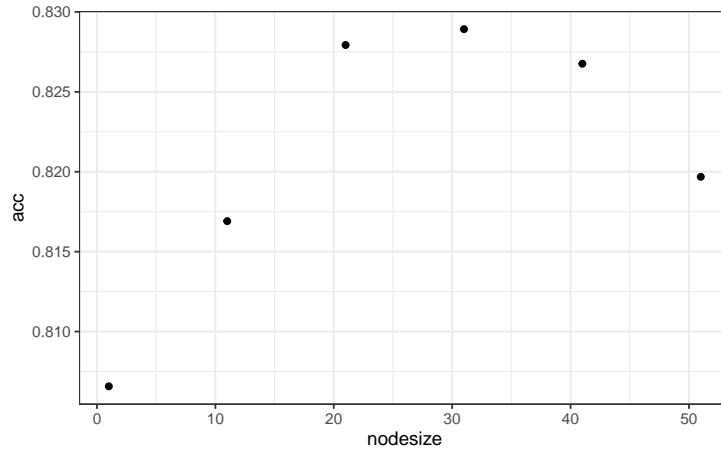
confusionMatrix(predict(train_rf, mnist_27$test),
 mnist_27$test$y)$overall["Accuracy"]
#> Accuracy
#> 0.79
```

Así es como se ven las probabilidades condicionales:



La visualización del estimador muestra que, aunque obtenemos una alta exactitud, parece que podemos mejorar al hacer que el estimador sea más uniforme. Esto podría lograrse cambiando el parámetro que controla el número mínimo de puntos de datos en los nodos del árbol. Mientras más grande sea este mínimo, más suave será el estimador final. Podemos entrenar los parámetros del bosque aleatorio. A continuación, utilizamos el paquete **caret** para optimizar el tamaño mínimo del nodo. Debido a que este no es uno de los parámetros que el paquete **caret** optimiza por defecto, escribiremos nuestro propio código:

```
nodesize <- seq(1, 51, 10)
acc <- sapply(nodesize, function(ns){
 train(y ~ ., method = "rf", data = mnist_27$train,
 tuneGrid = data.frame(mtry = 2),
 nodesize = ns)$results$Accuracy
})
qplot(nodesize, acc)
```

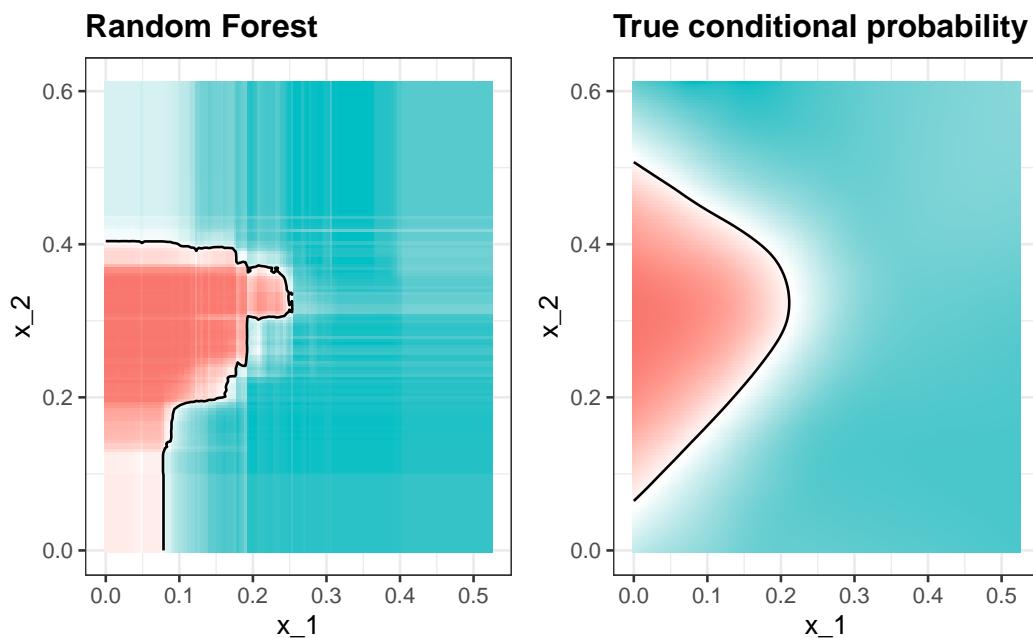


Ahora podemos ajustar el bosque aleatorio con el tamaño de nodo mínimo optimizado a todos los datos de entrenamiento y evaluar el rendimiento en los datos de evaluación.

```
train_rf_2 <- randomForest(y ~ ., data=mnist_27$train,
 nodesize = nodesize[which.max(acc)])

confusionMatrix(predict(train_rf_2, mnist_27$test),
 mnist_27$test$y)$overall[["Accuracy"]]
#> Accuracy
#> 0.82
```

El modelo seleccionado mejora la exactitud y provee un estimador más uniforme.



Tengan en cuenta que podemos evitar escribir nuestro propio código utilizando otras implementaciones de bosques aleatorios como se describe en el manual **caret**<sup>4</sup>.

El bosque aleatorio funciona mejor en todos los ejemplos que hemos considerado. Sin embargo, una desventaja de los bosques aleatorios es que perdemos interpretabilidad. Un enfoque que ayuda con la interpretabilidad es examinar la *importancia de la variable* (*variable importance* en inglés). Para definir importancia, contamos cuán frecuentemente se usa el predictor en los árboles individuales. Pueden obtener más información sobre *importancia* en un libro de *machine learning* avanzado<sup>5</sup>. El paquete **caret** incluye la función `varImp` que extrae la importancia de cada variable de cualquier modelo en el que se implementa el cálculo. Ofecemos un ejemplo de cómo usamos la importancia en la siguiente sección.

### 31.12 Ejercicios

1. Cree un set de datos sencillo donde el resultado crece 0.75 unidades en promedio por cada aumento en un predictor:

```
n <- 1000
sigma <- 0.25
x <- rnorm(n, 0, 1)
y <- 0.75 * x + rnorm(n, 0, sigma)
dat <- data.frame(x = x, y = y)
```

Utilice `rpart` para ajustar un árbol de regresión y guarde el resultado en `fit`.

2. Grafique el árbol final para que pueda ver dónde ocurrieron las particiones.
3. Haga un diagrama de dispersión de `y` versus `x` junto con los valores predichos basados en el ajuste.
4. Ahora modele con un bosque aleatorio en lugar de un árbol de regresión usando `randomForest` del paquete **randomForest** y rehaga el diagrama de dispersión con la línea de predicción.
5. Use la función `plot` para ver si el bosque aleatorio ha convergido o si necesitamos más árboles.
6. Parece que los valores predeterminados para el bosque aleatorio dan como resultado un estimador demasiado flexible (no uniforme). Vuelva a ejecutar el bosque aleatorio pero esta vez con `nodesize` fijado en 50 y `maxnodes` fijado en 25. Rehaga el gráfico.
7. Vemos que esto produce resultados más suaves. Usemos la función `train` para ayudarnos a elegir estos valores. Del manual **caret**<sup>6</sup> vemos que no podemos ajustar el parámetro `maxnodes` ni el argumento `nodesize` con la función `randomForest`, así que usaremos el paquete **Rborist** y ajustaremos el argumento `minNode`. Utilice la función `train` para probar valores `minNode <- seq(5, 250, 25)`. Vea qué valor minimiza el estimador RMSE.
8. Haga un diagrama de dispersión junto con la predicción del modelo mejor ajustado.

<sup>4</sup><http://topepo.github.io/caret/available-models.html>

<sup>5</sup><https://web.stanford.edu/~hastie/Papers/ESLII.pdf>

<sup>6</sup><https://topepo.github.io/caret/available-models.html>

9. Utilice la función `rpart` para ajustar un árbol de clasificación al set de datos `tissue_gene_expression`. Utilice la función `train` para estimar la exactitud. Pruebe valores `cp` de `seq(0, 0.05, 0.01)`. Grafique la exactitud para indicar los resultados del mejor modelo.
10. Estudie la matriz de confusión para el árbol de clasificación de mejor ajuste. ¿Qué observa que sucede con la placenta?
11. Tenga en cuenta que las placas se llaman endometrio con más frecuencia que placenta. Además, noten que la cantidad de placas es solo seis y que, de forma predeterminada, `rpart` requiere 20 observaciones antes de dividir un nodo. Por lo tanto, no es posible con estos parámetros tener un nodo en el que las placas sean la mayoría. Vuelva a ejecutar el análisis anterior, pero esta vez permita que `rpart` divida cualquier nodo usando el argumento `control = rpart.control(minsplit = 0)`. ¿Aumenta la exactitud? Mire la matriz de confusión de nuevo.
12. Grafique el árbol del modelo de mejor ajuste obtenido en el ejercicio 11.
13. Podemos ver que con solo seis genes, podemos predecir el tipo de tejido. Ahora veamos si podemos hacerlo aún mejor con un bosque aleatorio. Utilice la función `train` y el método `rf` para entrenar un bosque aleatorio. Pruebe valores de `mtry` que van desde, al menos, `seq(50, 200, 25)`. ¿Qué valor de `mtry` maximiza la exactitud? Para permitir que pequeños `nodesize` crezcan como lo hicimos con los árboles de clasificación, use el siguiente argumento: `nodesize = 1`. Esto tardará varios segundos en ejecutarse. Si desea probarlo, intente usar valores más pequeños con `ntree`. Fije la semilla en 1990.
14. Use la función `varImp` en el resultado de `train` y guárdelo en un objeto llamado `imp`.
15. El modelo `rpart` que ejecutamos anteriormente produjo un árbol que utilizaba solo seis predictores. Extraer los nombres de los predictores no es sencillo, pero se puede hacer. Si el resultado de la llamada a `train` fue `fit_rpart`, podemos extraer los nombres así:

```
ind <- !(fit_rpart$finalModel$frame$var == "<leaf>")
tree_terms <-
 fit_rpart$finalModel$frame$var[ind] %>%
 unique() %>%
 as.character()
tree_terms
```

¿Cuál es la importancia de variable para estos predictores? ¿Cuáles son sus rangos?

16. **Avanzado:** extraiga los 50 predictores principales según la importancia, tome un subconjunto de `x` con solo estos predictores y aplique la función `heatmap` para ver cómo se comportan estos genes a través de los tejidos. Presentaremos la función `heatmap` en el Capítulo 34.



# 32

---

## *Machine learning en la práctica*

---

Ahora que hemos aprendido varios métodos y los hemos explorado con ejemplos ilustrativos, los aplicaremos a un ejemplo real: los dígitos MNIST.

Podemos cargar estos datos usando el siguiente paquete de **dslabs**:

```
library(tidyverse)
library(dslabs)
mnist <- read_mnist()
```

El set de datos incluye dos componentes, un set de entrenamiento y un set de evaluación:

```
names(mnist)
#> [1] "train" "test"
```

Cada uno de estos componentes incluye una matriz con atributos en las columnas:

```
dim(mnist$train$images)
#> [1] 60000 784
```

y un vector con las clases como enteros:

```
class(mnist$train$labels)
#> [1] "integer"
table(mnist$train$labels)
#>
#> 0 1 2 3 4 5 6 7 8 9
#> 5923 6742 5958 6131 5842 5421 5918 6265 5851 5949
```

Como queremos que este ejemplo se ejecute en una computadora portátil pequeña y en menos de una hora, consideraremos un subconjunto del set de datos. Tomaremos muestras de 10,000 filas aleatorias del set de entrenamiento y 1,000 filas aleatorias del set de evaluación:

```
set.seed(1990)
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])
```

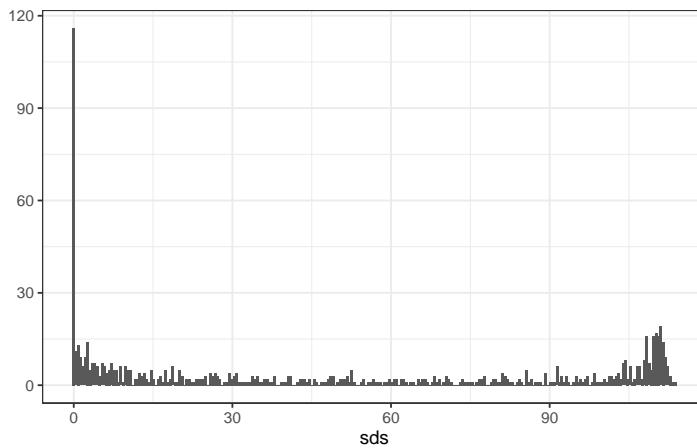
### 32.1 Preprocesamiento

En *machine learning*, a menudo transformamos predictores antes de ejecutar el algoritmo. También eliminamos predictores que claramente no son útiles. Llamamos a estos pasos *preprocesamiento* (*preprocessing* en inglés).

Ejemplos de preprocesamiento incluyen estandarizar los predictores, transformar logarítmicamente algunos predictores, eliminar los predictores que están altamente correlacionados con otros y eliminar los predictores con muy pocos valores no únicos o una variación cercana a cero. Mostramos un ejemplo a continuación.

Podemos ejecutar la función `nearZero` del paquete **caret** para ver que muchos atributos no varían mucho de una observación a otra. Podemos ver que hay una gran cantidad de atributos con variabilidad 0:

```
library(matrixStats)
sds <- colSds(x)
qplot(sds, bins = 256)
```



Esto se espera porque hay partes de la imagen que raras veces contienen escritura (píxeles oscuros).

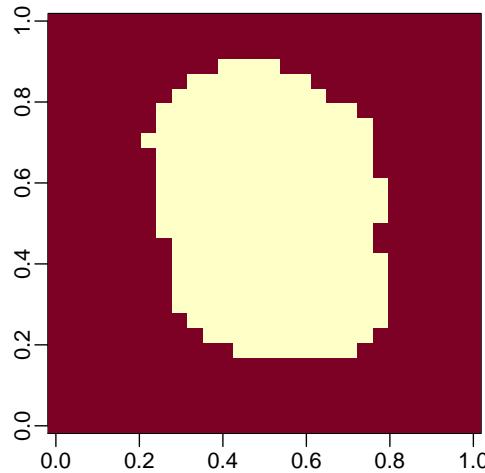
El paquete **caret** incluye una función que recomienda que se eliminen estos atributos debido a que la variación es casi cero:

```
library(caret)
nzv <- nearZeroVar(x)
```

Podemos ver las columnas que se recomiendan eliminar:

```
image(matrix(1:784 %in% nzv, 28, 28))
```

```
rafalib::mynpar()
image(matrix(1:784 %in% nzv, 28, 28))
```



Entonces nos quedamos con este número de columnas:

```
col_index <- setdiff(1:ncol(x), nzv)
length(col_index)
#> [1] 252
```

Ahora estamos listos para adaptarnos a algunos modelos. Antes de comenzar, debemos agregar nombres de columna a las matrices de predictores, ya que **caret** los requiere:

```
colnames(x) <- 1:ncol(mnist$train$images)
colnames(x_test) <- colnames(x)
```

## 32.2 *k*-vecino más cercano y bosque aleatorio

Comencemos con kNN. El primer paso es optimizar para  $k$ . Tengan en cuenta que cuando ejecutamos el algoritmo, tendremos que calcular una distancia entre cada observación en el set de evaluación y cada observación en el set de entrenamiento. Hay muchos cálculos. Por lo tanto, utilizaremos la validación cruzada *k-fold* para mejorar la velocidad.

Si ejecutamos el siguiente código, el tiempo de computación en una computadora portátil estándar será de varios minutos.

```
control <- trainControl(method = "cv", number = 10, p = .9)
train_knn <- train(x[, col_index], y,
 method = "knn",
```

```
tuneGrid = data.frame(k = c(3,5,7)),
trControl = control)
train_knn
```

En general, es una buena idea hacer una prueba con un subconjunto de datos para tener una idea del tiempo antes de comenzar a ejecutar un código que puede tardar horas en completarse. Podemos hacer esto de la siguiente manera:

```
n <- 1000
b <- 2
index <- sample(nrow(x), n)
control <- trainControl(method = "cv", number = b, p = .9)
train_knn <- train(x[index, col_index], y[index],
method = "knn",
tuneGrid = data.frame(k = c(3,5,7)),
trControl = control)
```

Entonces podemos aumentar `n` y `b` e intentar establecer un patrón de cómo afectan el tiempo de computación para tener una idea de cuánto tiempo tomará el proceso de ajuste para valores mayores de `n` y `b`. Quieren saber si una función tomará horas, o incluso días, antes de ejecutarla.

Una vez que optimicemos nuestro algoritmo, podemos aplicarlo a todo el set de datos:

```
fit_knn <- knn3(x[, col_index], y, k = 3)
```

¡La exactitud es casi 0.95!

```
y_hat_knn <- predict(fit_knn, x_test[, col_index], type="class")
cm <- confusionMatrix(y_hat_knn, factor(y_test))
cm$overall["Accuracy"]
#> Accuracy
#> 0.953
```

Ahora logramos una exactitud de aproximadamente 0.95. De la especificidad y sensibilidad, también vemos que los 8 son los más difíciles de detectar y que el dígito pronosticado incorrectamente con mas frecuencia es el 7.

```
cm$byClass[,1:2]
#> Sensitivity Specificity
#> Class: 0 0.990 0.996
#> Class: 1 1.000 0.993
#> Class: 2 0.965 0.997
#> Class: 3 0.950 0.999
#> Class: 4 0.930 0.997
#> Class: 5 0.921 0.993
#> Class: 6 0.977 0.996
#> Class: 7 0.956 0.989
#> Class: 8 0.887 0.999
#> Class: 9 0.951 0.990
```

Ahora veamos si podemos hacerlo aún mejor con el algoritmo de bosque aleatorio.

Con bosque aleatorio, el tiempo de cálculo es un reto. Para cada bosque, necesitamos construir cientos de árboles. También tenemos varios parámetros que podemos ajustar.

Debido a que con el bosque aleatorio el ajuste es la parte más lenta del procedimiento en lugar de la predicción (como con kNN), usaremos solo una validación cruzada de cinco pliegues (*folds* en inglés). Además, reduciremos la cantidad de árboles que se ajustan ya que aún no estamos construyendo nuestro modelo final.

Finalmente, para calcular en un set de datos más pequeño, tomaremos una muestra aleatoria de las observaciones al construir cada árbol. Podemos cambiar este número con el argumento `nSamp`.

```
library(randomForest)
control <- trainControl(method="cv", number = 5)
grid <- data.frame(mtry = c(1, 5, 10, 25, 50, 100))

train_rf <- train(x[, col_index], y,
 method = "rf",
 ntree = 150,
 trControl = control,
 tuneGrid = grid,
 nSamp = 5000)
```

Ahora que hemos optimizado nuestro algoritmo, estamos listos para ajustar nuestro modelo final:

```
fit_rf <- randomForest(x[, col_index], y,
 minNode = train_rf$bestTune$mtry)
```

Para verificar que ejecutamos suficientes árboles, podemos usar la función `plot`:

```
plot(fit_rf)
```

Vemos que logramos una alta exactitud:

```
y_hat_rf <- predict(fit_rf, x_test[, col_index])
cm <- confusionMatrix(y_hat_rf, y_test)
cm$overall["Accuracy"]
#> Accuracy
#> 0.956
```

Con algunos ajustes adicionales, podemos obtener una exactitud aún mayor.

### 32.3 Importancia variable

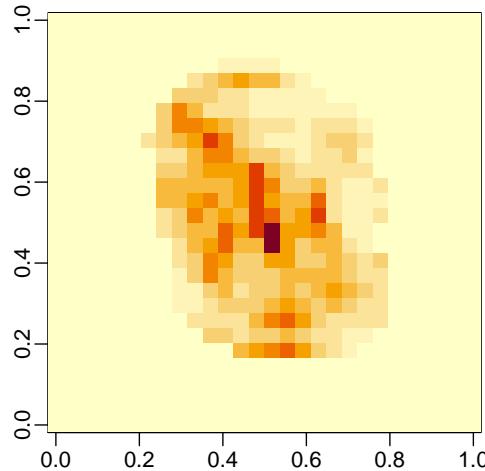
La siguiente función calcula la importancia de cada atributo:

```
imp <- importance(fit_rf)
```

Podemos ver qué atributos se utilizan más al graficar una imagen:

```
mat <- rep(0, ncol(x))
mat[col_index] <- imp
image(matrix(mat, 28, 28))
```

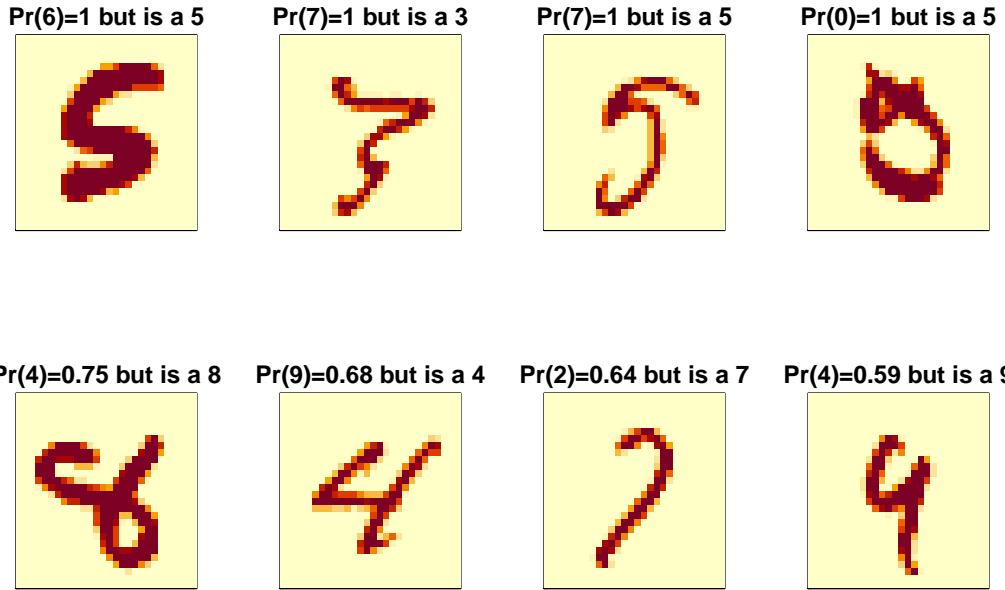
```
rafalib::mpar()
mat <- rep(0, ncol(x))
mat[col_index] <- imp
image(matrix(mat, 28, 28))
```



### 32.4 Evaluaciones visuales

Una parte importante del análisis de datos es visualizar los resultados para determinar por qué estamos fallando. Cómo hacemos esto depende de la aplicación. A continuación mostramos las imágenes de dígitos para los cuales hicimos una predicción incorrecta. Podemos comparar lo que obtenemos con kNN a los resultados de bosque aleatorio.

Aquí vemos unos errores para el bosque aleatorio:



Al examinar errores como este, a menudo encontramos debilidades específicas en los algoritmos o en las opciones de parámetros que podemos intentar corregir.

## 32.5 Conjuntos

La idea de un conjunto (*ensemble* en inglés) es similar a la idea de combinar datos de diferentes encuestadores para obtener un mejor estimador del verdadero apoyo para cada candidato.

En *machine learning*, generalmente se pueden mejorar los resultados finales combinando los resultados de diferentes algoritmos.

Aquí hay un ejemplo sencillo donde calculamos nuevas probabilidades de clase tomando el promedio de bosque aleatorio y kNN. Podemos ver que la exactitud mejora a 0.96:

```
p_rf <- predict(fit_rf, x_test[,col_index], type = "prob")
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)$overall["Accuracy"]
#> Accuracy
#> 0.961
```

En los ejercicios, vamos a construir varios modelos de *machine learning* para el set de datos `mnist_27` y luego construir un conjunto.

## 32.6 Ejercicios

1. Utilice el set de entrenamiento `mnist_27` para construir un modelo con varios de los modelos disponibles del paquete `caret`. Por ejemplo, puede tratar estos:

```
models <- c("glm", "lda", "naive_bayes", "svmLinear", "gamboost",
 "gamLoess", "qda", "knn", "kknn", "loclda", "gam", "rf",
 "ranger", "wsrf", "Rborist", "avNNet", "mlp", "monmlp", "gbm",
 "adaboost", "svmRadial", "svmRadialCost", "svmRadialSigma")
```

Aunque no hemos explicado muchos de estos algoritmos, aplíquelos usando `train` con todos los parámetros predeterminados. Guarde los resultados en una lista. Es posible que tenga que instalar algunos paquetes. Es posible que probablemente recibirá algunas advertencias.

2. Ahora que tiene todos los modelos entrenados en una lista, use `sapply` o `map` para crear una matriz de predicciones para el set de evaluación. Debería terminar con una matriz con `length(mnist_27$test$y)` filas y `length(models)` columnas.
3. Ahora calcule la exactitud para cada modelo en el set de evaluación.
4. Ahora construya una predicción de conjunto para el voto mayoritario y calcule su exactitud.
5. Anteriormente calculamos la exactitud de cada método en el set de entrenamiento y notamos que variaban. ¿Qué métodos individuales funcionan mejor que el conjunto?
6. Es tentador eliminar los métodos que no funcionan bien y volver a hacer el conjunto. El problema con este acercamiento es que estamos utilizando los datos de evaluación para tomar una decisión. Sin embargo, podríamos usar los estimadores de exactitud obtenidos de la validación cruzada con los datos de entrenamiento. Obtenga estos estimadores y guárdelos en un objeto.
7. Ahora solo consideremos los métodos con una exactitud estimada de 0.8 al construir el conjunto. ¿Cuál es la exactitud ahora?
8. **Avanzado:** Si dos métodos dan resultados que son iguales, unirlos no cambiará los resultados en absoluto. Para cada par de métricas, compare cuán frequentemente predicen lo mismo. Entonces use la función `heatmap` para visualizar los resultados. Sugerencia: use el argumento `method = "binary"` en la función `dist`.
9. **Avanzado:** Tenga en cuenta que cada método también puede producir una probabilidad condicional estimada. En lugar del voto mayoritario, podemos tomar el promedio de estas probabilidades condicionales estimadas. Para la mayoría de los métodos, podemos usar el `type = "prob"` en la función `train`. Sin embargo, algunos de los métodos requieren que use el argumento `trControl=trainControl(classProbs=TRUE)` al llamar `train`. Además, estos métodos no funcionan si las clases tienen números como nombres. Sugerencia: cambie los niveles de esta manera:

```
dat$train$y <- recode_factor(dat$train$y, "2"="two", "7"="seven")
dat$test$y <- recode_factor(dat$test$y, "2"="two", "7"="seven")
```

10. En este capítulo, ilustramos un par de algoritmos de *machine learning* en un subconjunto del set de datos MNIST. Intente ajustar un modelo a todo el set de datos.

# 33

---

## Sets grandes de datos

---

Los problemas de *machine learning* a menudo implican sets de datos que son tan o más grandes que el set de datos MNIST. Existe una variedad de técnicas computacionales y conceptos estadísticos que son útiles para análisis de grandes sets de datos. En este capítulo, exploramos brevemente estas técnicas y conceptos al describir álgebra matricial, reducción de dimensiones, regularización y factorización de matrices. Utilizamos sistemas de recomendación relacionados con las clasificaciones de películas como un ejemplo motivador.

---

### 33.1 Álgebra matricial

En *machine learning*, las situaciones en las que todos los predictores son numéricos, o pueden representarse como números son comunes. El set de datos de dígitos es un ejemplo: cada píxel registra un número entre 0 y 255. Carguemos los datos:

```
library(tidyverse)
library(dslabs)
if(!exists("mnist")) mnist <- read_mnist()
```

En estos casos, a menudo es conveniente guardar los predictores en una matriz y el resultado en un vector en lugar de utilizar un *data frame*. Pueden ver que los predictores se guardan en una matriz:

```
class(mnist$train$images)
#> [1] "matrix" "array"
```

Esta matriz representa 60,000 dígitos, así que para los ejemplos en este capítulo, usaremos un subconjunto más manejable. Tomaremos los primeros 1,000 predictores x y etiquetas y:

```
x <- mnist$train$images[1:1000,]
y <- mnist$train$labels[1:1000]
```

La razón principal para usar matrices es que ciertas operaciones matemáticas necesarias para desarrollar código eficiente se pueden realizar usando técnicas de una rama de las matemáticas llamada álgebra lineal. De hecho, álgebra lineal y notación matricial son elementos claves del lenguaje utilizado en trabajos académicos que describen técnicas de *machine learning*. No cubriremos álgebra lineal en detalle aquí, pero demostraremos cómo usar matrices en R para que puedan aplicar las técnicas de álgebra lineal ya implementadas en la base R u otros paquetes.

Para motivar el uso de matrices, plantearemos cinco preguntas/desafíos:

1. ¿Algunos dígitos requieren más tinta que otros? Estudien la distribución de la oscuridad total de píxeles y cómo varía según los dígitos.
2. ¿Algunos píxeles no son informativos? Estudien la variación de cada píxel y eliminen los predictores (columnas) asociados con los píxeles que no cambian mucho y, por lo tanto, no proveen mucha información para la clasificación.
3. ¿Podemos eliminar las manchas? Primero, observen la distribución de todos los valores de píxeles. Usen esto para elegir un umbral para definir el espacio no escrito. Luego, cambien cualquier valor por debajo de ese umbral a 0.
4. Binaricen los datos. Primero, observen la distribución de todos los valores de píxeles. Usen esto para elegir un umbral para distinguir entre escritura y no escritura. Luego, conviertan todas las entradas en 1 o 0, respectivamente.
5. Escalen cada uno de los predictores en cada entrada para tener el mismo promedio y desviación estándar.

Para completar esto, tendremos que realizar operaciones matemáticas que involucran varias variables. El **tidyverse** no está desarrollado para realizar este tipo de operaciones matemáticas. Para esta tarea, es conveniente usar matrices.

Antes de hacer esto, presentaremos la notación matricial y el código R básico para definir y operar en matrices.

### 33.1.1 Notación

En álgebra matricial, tenemos tres tipos principales de objetos: escalares, vectores y matrices. Un escalar es solo un número, por ejemplo  $a = 1$ . Para denotar escalares en notación matricial, generalmente usamos una letra minúscula no en negrilla.

Los vectores son como los vectores numéricos que definimos en R: incluyen varias entradas escalares. Por ejemplo, la columna que contiene el primer píxel:

```
length(x[,1])
#> [1] 1000
```

tiene 1,000 entradas. En álgebra matricial, utilizamos la siguiente notación para un vector que representa un atributo/predictor:

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix}$$

Del mismo modo, podemos usar la notación matemática para representar diferentes atributos matemáticamente agregando un índice:

$$\mathbf{X}_1 = \begin{pmatrix} x_{1,1} \\ \vdots \\ x_{N,1} \end{pmatrix} \text{ and } \mathbf{X}_2 = \begin{pmatrix} x_{1,2} \\ \vdots \\ x_{N,2} \end{pmatrix}$$

Si estamos escribiendo una columna, como  $\mathbf{X}_1$ , en una oración, a menudo usamos la notación:  $\mathbf{X}_1 = (x_{1,1}, \dots x_{N,1})^\top$  con  $^\top$  la operación de transposición que convierte las columnas en filas y las filas en columnas.

Una matriz se puede definir como una serie de vectores del mismo tamaño unidos como columnas:

```
x_1 <- 1:5
x_2 <- 6:10
cbind(x_1, x_2)
#> x_1 x_2
#> [1,] 1 6
#> [2,] 2 7
#> [3,] 3 8
#> [4,] 4 9
#> [5,] 5 10
```

Matemáticamente, los representamos con letras mayúsculas en negrilla:

$$\mathbf{X} = [\mathbf{X}_1 \mathbf{X}_2] = \begin{pmatrix} x_{1,1} & x_{1,2} \\ \vdots & \\ x_{N,1} & x_{N,2} \end{pmatrix}$$

La *dimensión* de una matriz a menudo es un atributo importante necesario para asegurar que se puedan realizar ciertas operaciones. La dimensión es un resumen de dos números definido como el número de filas  $\times$  el número de columnas. En R, podemos extraer la dimensión de una matriz con la función `dim`:

```
dim(x)
#> [1] 1000 784
```

Los vectores pueden considerarse  $N \times 1$  matrices. Sin embargo, en R, un vector no tiene dimensiones:

```
dim(x_1)
#> NULL
```

No obstante, explícitamente convertimos un vector en una matriz usando la función `as.matrix`:

```
dim(as.matrix(x_1))
#> [1] 5 1
```

Podemos usar esta notación para denotar un número arbitrario de predictores con la siguiente matriz  $N \times p$ , por ejemplo, con  $p = 784$ :

$$\mathbf{X} = \begin{pmatrix} x_{1,1} & \dots & x_{1,p} \\ x_{2,1} & \dots & x_{2,p} \\ \vdots & & \\ x_{N,1} & \dots & x_{N,p} \end{pmatrix}$$

Almacenamos esta matriz en x:

```
dim(x)
#> [1] 1000 784
```

Ahora aprenderemos varias operaciones útiles relacionadas con el álgebra matricial. Utilizamos tres de las preguntas motivadoras mencionadas anteriormente.

### 33.1.2 Convertir un vector en una matriz

A menudo es útil convertir un vector en una matriz. Por ejemplo, debido a que las variables son píxeles en una cuadrícula, podemos convertir las filas de intensidades de píxeles en una matriz que representa esta cuadrícula.

Podemos convertir un vector en una matriz con la función `matrix` y especificando el número de filas y columnas que debe tener la matriz resultante. La matriz se llena **por columna**: la primera columna se llena primero, luego la segunda y así sucesivamente. Este ejemplo ayuda a ilustrar:

```
my_vector <- 1:15
mat <- matrix(my_vector, 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
```

Podemos llenar por fila usando el argumento `byrow`. Entonces, por ejemplo, para *transponer* la matriz `mat`, podemos usar:

```
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
```

Cuando convertimos las columnas en filas, nos referimos a las operaciones como *transponer* la matriz. La función `t` se puede usar para transponer directamente una matriz:

```
identical(t(mat), mat_t)
#> [1] TRUE
```

**Advertencia:** La función `matrix` recicla valores en el vector **sin advertencia** si el producto de las columnas y las filas no coincide con la longitud del vector:

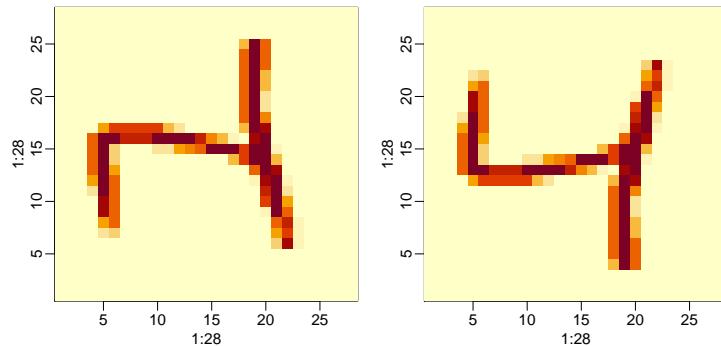
```
matrix(my_vector, 4, 5)
#> Warning in matrix(my_vector, 4, 5): la longitud de los datos [15] no es
#> un submúltiplo o múltiplo del número de filas [4] en la matriz
#> [,1] [,2] [,3] [,4] [,5]
#> [1,] 1 5 9 13 2
#> [2,] 2 6 10 14 3
#> [3,] 3 7 11 15 4
#> [4,] 4 8 12 1 5
```

Para poner las intensidades de píxeles de nuestra, digamos, tercera entrada, que es 4 en la cuadrícula, podemos usar:

```
grid <- matrix(x[3,], 28, 28)
```

Para confirmar que lo hemos hecho correctamente, podemos usar la función `image`, que muestra una imagen de su tercer argumento. La parte superior de este gráfico es el píxel 1, que se muestra en la parte inferior para que la imagen se volteé. A continuación incluimos el código que muestra cómo voltearlo:

```
image(1:28, 1:28, grid)
image(1:28, 1:28, grid[, 28:1])
```



### 33.1.3 Resúmenes de filas y columnas

Para la primera tarea, relacionada con la oscuridad total de píxeles, queremos sumar los valores de cada fila y luego visualizar cómo estos valores varían por dígito.

La función `rowSums` toma una matriz como entrada y calcula los valores deseados:

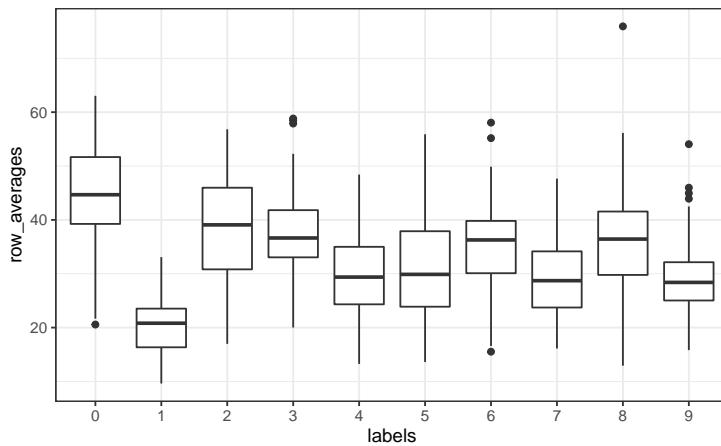
```
sums <- rowSums(x)
```

También podemos calcular los promedios con `rowMeans` si queremos que los valores permanezcan entre 0 y 255:

```
avg <- rowMeans(x)
```

Una vez que tengamos esto, podemos generar un diagrama de caja:

```
tibble(labels = as.factor(y), row_averages = avg) %>%
 qplot(labels, row_averages, data = ., geom = "boxplot")
```



De este gráfico vemos que, como era de esperar, los 1s usan menos tinta que los otros dígitos.

Podemos calcular las sumas y los promedios de la columna usando la función `colSums` y `colMeans`, respectivamente.

El paquete `matrixStats` agrega funciones que realizan operaciones en cada fila o columna de manera muy eficiente, incluyendo las funciones `rowSds` y `colSds`.

### 33.1.4 `apply`

Las funciones que acabamos de describir están realizando una operación similar a la que hacen `sapply` y la función `map` de `purrr`: aplicar la misma función a una parte de su objeto. En este caso, la función se aplica a cada fila o cada columna. La función `apply` les permite aplicar cualquier función, no solo `sum` o `mean`, a una matriz. El primer argumento es la matriz, el segundo es la dimensión (1 para las filas y 2 para las columnas) y el tercero es la función. Así, por ejemplo, `rowMeans` se puede escribir como:

```
avgs <- apply(x, 1, mean)
```

Pero noten que al igual que con `sapply` y `map`, podemos ejecutar cualquier función. Entonces, si quisieramos la desviación estándar para cada columna, podríamos escribir:

```
sds <- apply(x, 2, sd)
```

La desventaja de esta flexibilidad es que estas operaciones no son tan rápidas como las funciones dedicadas, como `rowMeans`.

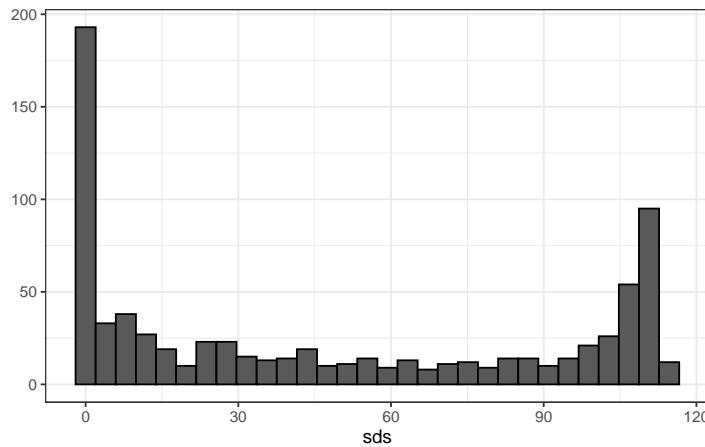
### 33.1.5 Filtrar columnas basado en resúmenes

Ahora pasamos a la tarea 2: estudiar la variación de cada píxel y eliminar las columnas asociadas con píxeles que no cambian mucho y, por lo tanto, no informan la clasificación. Aunque es un enfoque simplista, cuantificaremos la variación de cada píxel con su desviación estándar en todas las entradas. Como cada columna representa un píxel, utilizamos la función `colSds` del paquete **matrixStats**:

```
library(matrixStats)
sds <- colSds(x)
```

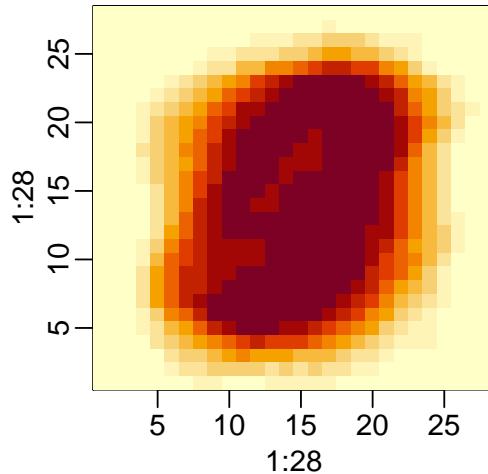
Un vistazo rápido a la distribución de estos valores muestra que algunos píxeles tienen una variabilidad muy baja de entrada a entrada:

```
qplot(sds, bins = "30", color = I("black"))
```



Esto tiene sentido ya que no escribimos en algunas partes del cuadro. Aquí está la variación graficada por ubicación:

```
image(1:28, 1:28, matrix(sds, 28, 28)[, 28:1])
```



Vemos que hay poca variación en las esquinas.

Podríamos eliminar atributos que no tienen variación ya que estos no nos ayuda a predecir. En la Sección 2.4.7, describimos las operaciones utilizadas para extraer columnas:

```
x[,c(351,352)]
```

y para extraer filas:

```
x[c(2,3),]
```

Además, podemos usar índices lógicos para determinar qué columnas o filas mantener. Entonces, si quisieramos eliminar predictores no informativos de nuestra matriz, podríamos escribir esta línea de código:

```
new_x <- x[, colSds(x) > 60]
dim(new_x)
#> [1] 1000 314
```

Solo se mantienen las columnas para las que la desviación estándar es superior a 60, lo que elimina más de la mitad de los predictores.

Aquí agregamos una advertencia importante relacionada con el subconjunto de matrices: si seleccionan una columna o una fila, el resultado ya no es una matriz sino un vector.

```
class(x[,1])
#> [1] "integer"
dim(x[1,])
#> NULL
```

Sin embargo, podemos preservar la clase de matriz usando el argumento `drop=FALSE`:

```
class(x[, 1, drop=FALSE])
#> [1] "matrix" "array"
dim(x[, 1, drop=FALSE])
#> [1] 1000 1
```

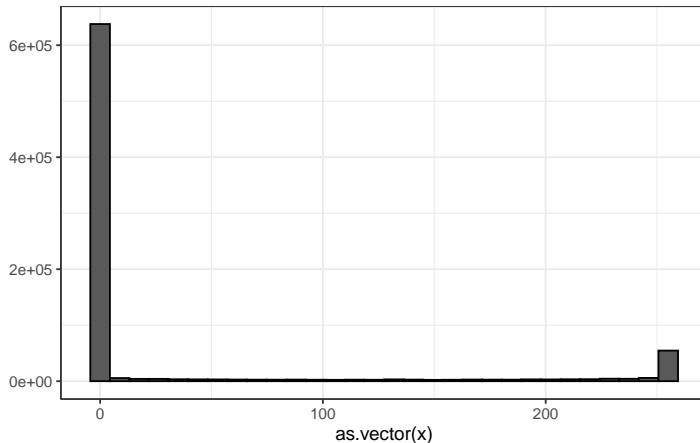
### 33.1.6 Indexación con matrices

Podemos hacer rápidamente un histograma de todos los valores en nuestro set de datos. Vimos cómo podemos convertir vectores en matrices. También podemos deshacer esto y convertir matrices en vectores. La operación se realiza por fila:

```
mat <- matrix(1:15, 5, 3)
as.vector(mat)
#> [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

Para ver un histograma de todos nuestros datos predictores, podemos usar:

```
qplot(as.vector(x), bins = 30, color = I("black"))
```



Notamos una clara dicotomía que se explica como partes de la imagen con tinta y partes sin ella. Si creemos que los valores menor que, digamos, 50 son manchas, podemos cambiarlos a cero usando:

```
new_x <- x
new_x[new_x < 50] <- 0
```

Para ver un ejemplo de cómo esto ocurre, usamos una matriz más pequeña:

```
mat <- matrix(1:15, 5, 3)
mat[mat < 3] <- 0
mat
#> [,1] [,2] [,3]
```

```
#> [1,] 0 6 11
#> [2,] 0 7 12
#> [3,] 3 8 13
#> [4,] 4 9 14
#> [5,] 5 10 15
```

También podemos usar operadores lógicas con una matriz de valores lógicos:

```
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
```

### 33.1.7 Binarizar los datos

El histograma anterior parece sugerir que estos datos son principalmente binarios. Un píxel tiene tinta o no. Usando lo que hemos aprendido, podemos binarizar los datos usando solo operaciones de matrices:

```
bin_x <- x
bin_x[bin_x < 255/2] <- 0
bin_x[bin_x > 255/2] <- 1
```

También podemos convertir a una matriz de valores lógicos y luego forzar una conversión a números como este:

```
bin_X <- (x > 255/2)*1
```

### 33.1.8 Vectorización para matrices

En R, si restamos un vector de una matriz, el primer elemento del vector se resta de la primera fila, el segundo elemento de la segunda fila, y así sucesivamente. Usando notación matemática, lo escribiríamos de la siguiente manera:

$$\begin{pmatrix} X_{1,1} & \dots & X_{1,p} \\ X_{2,1} & \dots & X_{2,p} \\ \vdots & & \\ X_{N,1} & \dots & X_{N,p} \end{pmatrix} - \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_N \end{pmatrix} = \begin{pmatrix} X_{1,1} - a_1 & \dots & X_{1,p} - a_1 \\ X_{2,1} - a_2 & \dots & X_{2,p} - a_2 \\ \vdots & & \\ X_{N,1} - a_n & \dots & X_{N,p} - a_n \end{pmatrix}$$

Lo mismo es válido para otras operaciones aritméticas. Esto implica que podemos escalar cada fila de una matriz así:

```
(x - rowMeans(x))/ rowSds(x)
```

Si desean escalar cada columna, tengan cuidado ya que este enfoque no funciona para las columnas. Para realizar una operación similar, convertimos las columnas en filas usando la transposición `t`, procedemos como se indica arriba y volvemos a transponer:

```
t(t(X) - colMeans(X))
```

También podemos usar la función `sweep` que funciona de manera similar a `apply`. Toma cada entrada de un vector y la resta de la fila o columna correspondiente.

```
X_mean_0 <- sweep(x, 2, colMeans(x))
```

La función `sweep` tiene otro argumento que les permite definir la operación aritmética. Entonces, para dividir por la desviación estándar, hacemos lo siguiente:

```
x_mean_0 <- sweep(x, 2, colMeans(x))
x_standardized <- sweep(x_mean_0, 2, colSds(x), FUN = "/")
```

### 33.1.9 Operaciones de álgebra matricial

Finalmente, aunque no discutimos las operaciones de álgebra matricial, como la multiplicación de matrices, compartimos aquí los comandos relevantes para aquellos que conocen las matemáticas y quieren aprender el código:

1. La multiplicación de matrices se realiza con `%*%`. Por ejemplo, el producto cruzado es:

```
t(x) %*% x
```

2. Podemos calcular el producto cruzado directamente con la función:

```
crossprod(x)
```

3. Para calcular el inverso de una función, usamos `solve`. Aquí se aplica al producto cruzado:

```
solve(crossprod(x))
```

4. La descomposición QR está fácilmente disponible mediante el uso de la función `qr`:

```
qr(x)
```

---

## 33.2 Ejercicios

1. Cree una matriz de 100 por 10 de números normales generados aleatoriamente. Ponga el resultado en `x`.

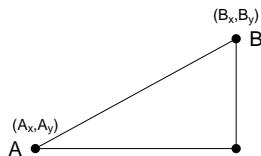
2. Aplique las tres funciones de R que le dan la dimensión de  $x$ , el número de filas de  $x$  y el número de columnas de  $x$ , respectivamente.
  3. Agregue el escalar 1 a la fila 1, el escalar 2 a la fila 2 y así sucesivamente, a la matriz  $x$ .
  4. Agregue el escalar 1 a la columna 1, el escalar 2 a la columna 2 y así sucesivamente, a la matriz  $x$ . Sugerencia: use `sweep` con `FUN = "+"`.
  5. Calcule el promedio de cada fila de  $x$ .
  6. Calcule el promedio de cada columna de  $x$ .
  7. Para cada dígito en los datos de entrenamiento MNIST, calcule la proporción de píxeles que se encuentran en un área gris, definida como valores entre 50 y 205. Haga un diagrama de caja basado en clase de dígitos. Sugerencia: utilice operadores lógicos y `rowMeans`.
- 

### 33.3 Distancia

Muchos de los análisis que realizamos con datos de alta dimensión se relacionan directa o indirectamente con la distancia. La mayoría de las técnicas de agrupamiento y *machine learning* se basan en la capacidad de definir la distancia entre observaciones, utilizando atributos (*features* en inglés) o predictores.

#### 33.3.1 Distancia euclíadiana

Como repaso, definamos la distancia entre dos puntos,  $A$  y  $B$ , en un plano cartesiano.



La distancia euclíadiana entre  $A$  y  $B$  es simplemente:

$$\text{dist}(A, B) = \sqrt{(A_x - B_x)^2 + (A_y - B_y)^2}$$

Esta definición se aplica al caso de una dimensión, en la que la distancia entre dos números es el valor absoluto de su diferencia. Entonces, si nuestros dos números unidimensionales son  $A$  y  $B$ , la distancia es:

$$\text{dist}(A, B) = \sqrt{(A - B)^2} = |A - B|$$

### 33.3.2 Distancia en dimensiones superiores

Anteriormente presentamos un set de datos de entrenamiento con matriz para 784 atributos. Con fines ilustrativos, veremos una muestra aleatoria de 2s y 7s.

```
library(tidyverse)
library(dslabs)

if(!exists("mnist")) mnist <- read_mnist()

set.seed(1995)
ind <- which(mnist$train$labels %in% c(2,7)) %>% sample(500)
x <- mnist$train$images[ind,]
y <- mnist$train$labels[ind]
```

Los predictores están en `x` y las etiquetas en `y`.

Para el propósito de, por ejemplo, suavizamiento, estamos interesados en describir la distancia entre observaciones; en este caso, dígitos. Más adelante, con el fin de seleccionar atributos, también podríamos estar interesados en encontrar píxeles que se comporten *de manera similar* en todas las muestras.

Para definir la distancia, necesitamos saber qué son *puntos*, ya que la distancia matemática se calcula entre puntos. Con datos de alta dimensión, los puntos ya no están en el plano cartesiano. En cambio, los puntos están en dimensiones más altas. Ya no podemos visualizarlos y necesitamos pensar de manera abstracta. Por ejemplo, predictores  $\mathbf{X}_i$  se definen como un punto en el espacio dimensional 784:  $\mathbf{X}_i = (x_{i,1}, \dots, x_{i,784})^\top$ .

Una vez que definamos los puntos de esta manera, la distancia euclíadiana se define de manera muy similar a la de dos dimensiones. Por ejemplo, la distancia entre los predictores para dos observaciones, digamos observaciones  $i = 1$  y  $i = 2$ , es:

$$\text{dist}(1, 2) = \sqrt{\sum_{j=1}^{784} (x_{1,j} - x_{2,j})^2}$$

Este es solo un número no negativo, tal como lo es para dos dimensiones.

### 33.3.3 Ejemplo de distancia euclíadiana

Las etiquetas para las tres primeras observaciones son:

```
y[1:3]
#> [1] 7 2 7
```

Los vectores de predictores para cada una de estas observaciones son:

```
x_1 <- x[1,]
x_2 <- x[2,]
x_3 <- x[3,]
```

El primer y tercer número son 7s y el segundo es un 2. Esperamos que las distancias entre el mismo número:

```
sqrt(sum((x_1 - x_2)^2))
#> [1] 3273
```

sean más pequeñas que entre diferentes números:

```
sqrt(sum((x_1 - x_3)^2))
#> [1] 2311
sqrt(sum((x_2 - x_3)^2))
#> [1] 2636
```

Como se esperaba, los 7s están más cerca uno del otro.

Una forma más rápida de calcular esto es usar álgebra matricial:

```
sqrt(crossprod(x_1 - x_2))
#> [,1]
#> [1,] 3273
sqrt(crossprod(x_1 - x_3))
#> [,1]
#> [1,] 2311
sqrt(crossprod(x_2 - x_3))
#> [,1]
#> [1,] 2636
```

También podemos calcular **todas** las distancias a la vez de manera relativamente rápida utilizando la función **dist**, que calcula la distancia entre cada fila y produce un objeto de clase **dist**:

```
d <- dist(x)
class(d)
#> [1] "dist"
```

Hay varias funciones relacionadas con *machine learning* en R que toman objetos de clase **dist** como entrada. Para acceder a las entradas usando índices de fila y columna, necesitamos forzar **d** a ser una matriz. Podemos ver la distancia que calculamos arriba de esta manera:

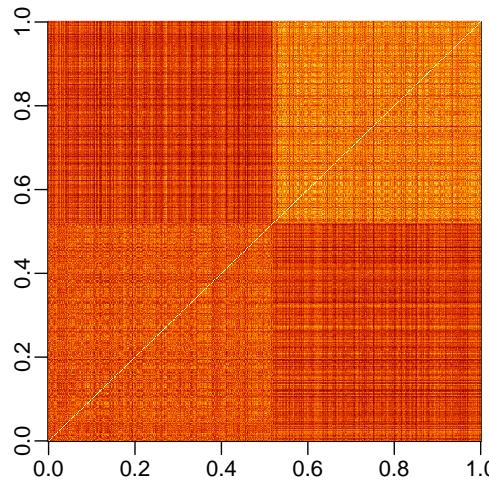
```
as.matrix(d)[1:3,1:3]
#> 1 2 3
#> 1 0 3273 2311
#> 2 3273 0 2636
#> 3 2311 2636 0
```

Rápidamente vemos una imagen de estas distancias usando este código:

```
image(as.matrix(d))
```

Si ordenamos esta distancia por las etiquetas, podemos ver que, en general, los 2s están más cerca uno del otro y los 7s están más cerca el uno del otro:

```
image(as.matrix(d)[order(y), order(y)])
```



Algo que notamos aquí es que parece haber más uniformidad en la forma en que se dibujan los 7s, ya que parecen estar más cercas (más rojos) a otros 7s que los 2s a otros 2s.

#### 33.3.4 Espacio predictor

El *espacio predictor* (*predictor space* en inglés) es un concepto que a menudo se usa para describir algoritmos de *machine learning*. El término *espacio* se refiere a una definición matemática que no describimos en detalle aquí. En cambio, ofrecemos una explicación simplificada para ayudar a entender el término espacio predictor cuando se usa en el contexto de algoritmos de *machine learning*.

El espacio predictor puede considerarse como la colección de todos los posibles vectores de predictores que deben considerarse para el reto en cuestión de *machine learning*. Cada miembro del espacio se conoce como un *punto*. Por ejemplo, en el set de datos 2 o 7, el espacio predictor consta de todos los pares  $(x_1, x_2)$  tal que ambos  $x_1$  y  $x_2$  están dentro de 0 y 1. Este espacio en particular puede representarse gráficamente como un cuadrado. En el set de datos MNIST, el espacio predictor consta de todos los vectores de 784 dimensiones, con cada elemento vectorial un número entero entre 0 y 256. Un elemento esencial de un espacio predictor es que necesitamos definir una función que nos de la distancia entre dos puntos. En la mayoría de los casos, usamos la distancia euclídea, pero hay otras posibilidades. Un caso particular en el que no podemos simplemente usar la distancia euclídea es cuando tenemos predictores categóricos.

Definir un espacio predictor es útil en *machine learning* porque hacemos cosas como definir vecindarios de puntos, como lo requieren muchas técnicas de suavización. Por ejemplo, podemos definir un vecindario como todos los puntos que están dentro de 2 unidades de un centro predefinido. Si los puntos son bidimensionales y usamos la distancia euclídea, este vecindario se representa gráficamente como un círculo con radio 2. En tres dimensiones,

el vecindario es una esfera. Pronto aprenderemos sobre algoritmos que dividen el espacio en regiones que no se superponen y luego hacen diferentes predicciones para cada región utilizando los datos de la región.

### 33.3.5 Distancia entre predictores

También podemos calcular distancias entre predictores. Si  $N$  es el número de observaciones, la distancia entre dos predictores, digamos 1 y 2, es:

$$\text{dist}(1, 2) = \sqrt{\sum_{i=1}^N (x_{i,1} - x_{i,2})^2}$$

Para calcular la distancia entre todos los pares de los 784 predictores, primero podemos transponer la matriz y luego usar `dist`:

```
d <- dist(t(x))
dim(as.matrix(d))
#> [1] 784 784
```

## 33.4 Ejercicios

1. Cargue el siguiente set de datos:

```
data("tissue_gene_expression")
```

Este set de datos incluye una matriz `x`:

```
dim(tissue_gene_expression$x)
```

con la expresión génica medida en 500 genes para 189 muestras biológicas que representan siete tejidos diferentes. El tipo de tejido se almacena en `y`:

```
table(tissue_gene_expression$y)
```

Calcule la distancia entre cada observación y almacénela en un objeto `d`.

2. Compare la distancia entre las dos primeras observaciones (ambos cerebros), las observaciones 39 y 40 (ambos colones) y las observaciones 73 y 74 (ambos endometrios). Vea si las observaciones del mismo tipo de tejido están más cercanas entre sí.

3. Vemos que, de hecho, las observaciones del mismo tipo de tejido están más cercanas entre sí en los seis ejemplos de tejido que acabamos de examinar. Haga un diagrama de todas las distancias usando la función `image` para ver si este patrón es general. Sugerencia: primero convierta `d` en una matriz.

## 33.5 Reducción de dimensiones

Un reto típico de *machine learning* incluirá una gran cantidad de predictores, lo que hace que la visualización sea algo retante. Hemos mostrado métodos para visualizar datos univariados y emparejados, pero los gráficos que revelan relaciones entre muchas variables son más complicados en dimensiones más altas. Por ejemplo, para comparar cada uno de las 784 atributos en nuestro ejemplo de predicción de dígitos, tendríamos que crear, por ejemplo, 306,936 diagramas de dispersión. La creación de un único diagrama de dispersión de los datos es imposible debido a la alta dimensionalidad.

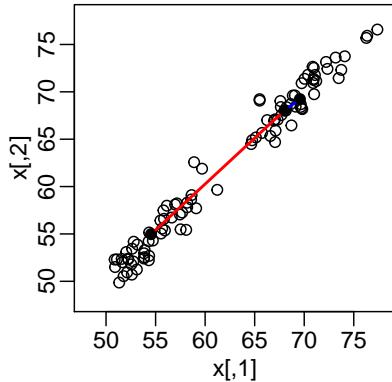
Aquí describimos técnicas eficaces y útiles para el análisis exploratorio de datos, entre otras cosas, generalmente conocidas como *reducción de dimensiones*. La idea general es reducir la dimensión del set de datos mientras se conservan características importantes, como la distancia entre atributos u observaciones. Con menos dimensiones, la visualización es más factible. La técnica detrás de todo, la descomposición de valores singulares, también es útil en otros contextos. El *análisis de componentes principales (principal component analysis o PCA por sus siglas en inglés)* es el enfoque que mostraremos. Antes de aplicar PCA a sets de datos de alta dimensión, motivaremos las ideas con un ejemplo sencillo.

### 33.5.1 Preservando la distancia

Consideremos un ejemplo con alturas de gemelos. Algunas parejas son adultas, otras son niños. Aquí simulamos 100 puntos bidimensionales que representan el número de desviaciones estándar que cada individuo tiene respecto a la altura media. Cada punto es un par de gemelos. Utilizamos la función `mvrnorm` del paquete **MASS** para simular datos de distribución normal de dos variables.

```
set.seed(1988)
library(MASS)
n <- 100
Sigma <- matrix(c(9, 9 * 0.9, 9 * 0.92, 9 * 1), 2, 2)
x <- rbind(mvrnorm(n/ 2, c(69, 69), Sigma),
 mvrnorm(n/ 2, c(55, 55), Sigma))
```

Un diagrama de dispersión revela que la correlación es alta y que hay dos grupos de gemelos, los adultos (puntos superiores derechos) y los niños (puntos inferiores izquierdos):



Nuestros atributos son  $N$  puntos bidimensionales, las dos alturas y, con fines ilustrativos, finjiremos como si visualizar dos dimensiones es demasiado difícil. Por lo tanto, queremos reducir las dimensiones de dos a una, pero todavía poder entender atributos importantes de los datos, por ejemplo, que las observaciones se agrupan en dos grupos: adultos y niños.

Consideremos un desafío específico: queremos un resumen unidimensional de nuestros predictores a partir del cual podamos aproximar la distancia entre dos observaciones. En el gráfico anterior, mostramos la distancia entre la observación 1 y 2 (azul) y la observación 1 y 51 (rojo). Noten que la línea azul es más corta, que implica que 1 y 2 están más cerca.

Podemos calcular estas distancias usando `dist`:

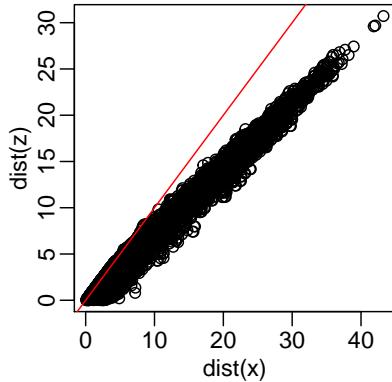
```
d <- dist(x)
as.matrix(d)[1, 2]
#> [1] 1.98
as.matrix(d)[2, 51]
#> [1] 18.7
```

Esta distancia se basa en dos dimensiones y necesitamos una aproximación de distancia basada en solo una.

Comencemos con un enfoque simplista de solo eliminar una de las dos dimensiones. Comparemos las distancias reales con la distancia calculada solo con la primera dimensión:

```
z <- x[,1]
```

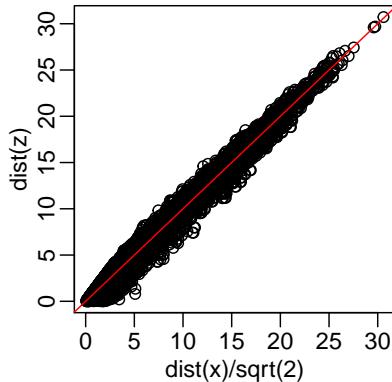
Aquí están las distancias aproximadas versus las distancias originales:



El gráfico se ve casi igual si usamos la segunda dimensión. Obtenemos una subestimación general. Esto no sorprende porque estamos añadiendo más cantidades positivas al cálculo de la distancia a medida que aumentamos el número de dimensiones. Si, en cambio, usamos un promedio, como este:

$$\sqrt{\frac{1}{2} \sum_{j=1}^2 (X_{1,j} - X_{2,j})^2},$$

entonces la subestimación desaparece. Dividimos la distancia por  $\sqrt{2}$  para lograr la corrección.



Esto funciona bastante bien y obtenemos una diferencia típica de:

```
sd(dist(x) - dist(z)*sqrt(2))
#> [1] 1.21
```

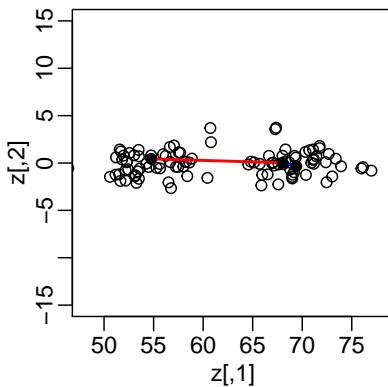
Ahora, ¿podemos elegir un resumen unidimensional que haga que esta aproximación sea aún mejor?

Si consideramos el diagrama de dispersión anterior y visualizamos una línea entre cualquier

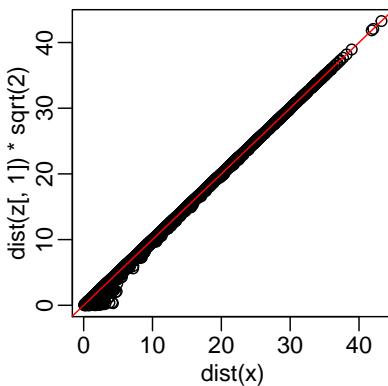
par de puntos, la longitud de esta línea es la distancia entre los dos puntos. Estas líneas tienden a ir a lo largo de la dirección de la diagonal. Noten que si en su lugar graficamos la diferencia versus el promedio:

```
z <- cbind((x[,2] + x[,1])/2, x[,2] - x[,1])
```

podemos ver cómo la distancia entre puntos se explica principalmente por la primera dimensión: el promedio.



Esto significa que podemos ignorar la segunda dimensión y no perder demasiada información. Si la línea es completamente plana, no perdemos ninguna información. Usando la primera dimensión de esta matriz transformada, obtenemos una aproximación aún mejor:



con la diferencia típica mejorada aproximadamente un 35%:

```
sd(dist(x) - dist(z[,1])*sqrt(2))
#> [1] 0.315
```

Más tarde, aprendemos que  $z[,1]$  es el primer componente principal de la matriz  $x$ .

### 33.5.2 Transformaciones lineales (avanzado)

Tengan en cuenta que cada fila de  $X$  se transformó usando una transformación lineal. Para cualquier fila  $i$ , la primera entrada fue:

$$Z_{i,1} = a_{1,1}X_{i,1} + a_{2,1}X_{i,2}$$

con  $a_{1,1} = 0.5$  y  $a_{2,1} = 0.5$ .

La segunda entrada también fue una transformación lineal:

$$Z_{i,2} = a_{1,2}X_{i,1} + a_{2,2}X_{i,2}$$

con  $a_{1,2} = 1$  y  $a_{2,2} = -1$ .

Además, podemos usar la transformación lineal para obtener  $X$  de  $Z$ :

$$X_{i,1} = b_{1,1}Z_{i,1} + b_{2,1}Z_{i,2}$$

con  $b_{1,2} = 1$  y  $b_{2,1} = 0.5$  y

$$X_{i,2} = b_{2,1}Z_{i,1} + b_{2,2}Z_{i,2}$$

con  $b_{2,1} = 1$  y  $a_{1,2} = -0.5$ .

Si saben álgebra lineal, pueden escribir la operación que acabamos de realizar de esta manera:

$$Z = XA \text{ with } A = \begin{pmatrix} 1/2 & 1 \\ 1/2 & -1 \end{pmatrix}.$$

Y que podemos transformar de vuelta a  $X$  multiplicando por  $A^{-1}$  así:

$$X = ZA^{-1} \text{ with } A^{-1} = \begin{pmatrix} 1 & 1 \\ 1/2 & -1/2 \end{pmatrix}.$$

La reducción de dimensiones frecuentemente se puede describir como la aplicación de una transformación  $A$  a una matriz  $X$  con muchas columnas que mueven la información contenida en  $X$  a las primeras columnas de  $Z = AX$ , manteniendo solo estas pocas columnas informativas y reduciendo así la dimensión de los vectores contenidos en las filas.

### 33.5.3 Transformaciones ortogonales (avanzado)

Noten que dividimos lo anterior por  $\sqrt{2}$  para tomar en cuenta las diferencias en las dimensiones al comparar una distancia de 2 dimensiones con una distancia de 1 dimensión. De hecho, podemos garantizar que las escalas de distancia sigan siendo las mismas si volvemos a escalar las columnas de  $A$  para asegurar que la suma de cuadrados es 1:

$$a_{1,1}^2 + a_{2,1}^2 = 1 \text{ and } a_{1,2}^2 + a_{2,2}^2 = 1,$$

y que la correlación de las columnas es 0:

$$a_{1,1}a_{1,2} + a_{2,1}a_{2,2} = 0.$$

Recuerden que si las columnas están centradas para tener un promedio de 0, entonces la suma de los cuadrados es equivalente a la varianza o desviación estándar al cuadrado.

En nuestro ejemplo, para lograr ortogonalidad, multiplicamos el primer set de coeficientes (primera columna de  $A$ ) por  $\sqrt{2}$  y el segundo por  $1/\sqrt{2}$ . Entonces obtenemos la misma distancia exacta cuando usamos ambas dimensiones:

```
z[,1] <- (x[,1] + x[,2])/ sqrt(2)
z[,2] <- (x[,2] - x[,1])/ sqrt(2)
```

Esto nos da una transformación que preserva la distancia entre dos puntos:

```
max(dist(z) - dist(x))
#> [1] 3.24e-14
```

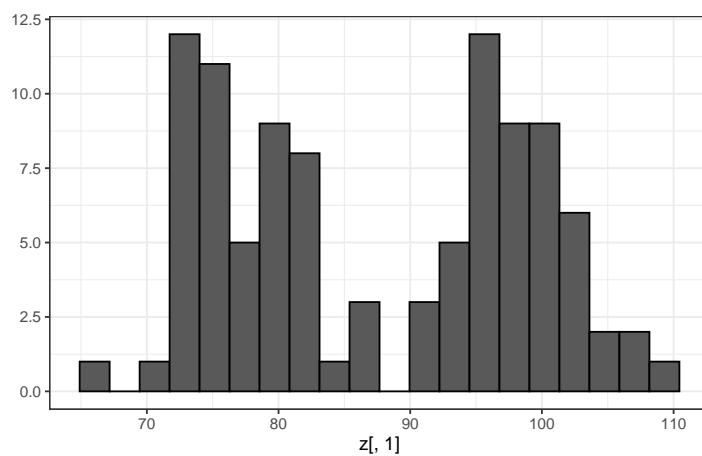
y una aproximación mejorada si usamos solo la primera dimensión:

```
sd(dist(x) - dist(z[,1]))
#> [1] 0.315
```

En este caso,  $Z$  se llama una rotación ortogonal de  $X$ : conserva las distancias entre filas.

Tengan en cuenta que al usar la transformación anterior, podemos resumir la distancia entre cualquier par de gemelos con una sola dimensión. Por ejemplo, una exploración de datos unidimensionales de la primera dimensión de  $Z$  muestra claramente que hay dos grupos, adultos y niños:

```
library(tidyverse)
qplot(z[,1], bins = 20, color = I("black"))
```



Reducimos exitosamente el número de dimensiones de dos a uno con muy poca pérdida de información.

La razón por la que pudimos hacer esto es porque las columnas de  $X$  estaban muy correlacionadas:

```
cor(x[,1], x[,2])
#> [1] 0.988
```

y la transformación produjo columnas no correlacionadas con información “independiente” en cada columna:

```
cor(z[,1], z[,2])
#> [1] 0.0876
```

Una forma en que esta información puede ser útil en una aplicación de *machine learning* es que podemos reducir la complejidad de un modelo utilizando solo  $Z_1$  en lugar de ambos  $X_1$  y  $X_2$ .

Es común obtener datos con varios predictores altamente correlacionados. En estos casos, PCA, que describimos a continuación, puede ser bastante útil para reducir la complejidad del modelo que se está ajustando.

### 33.5.4 Análisis de componentes principales

En el cálculo anterior, la variabilidad total en nuestros datos puede definirse como la suma de la suma de los cuadrados de las columnas. Suponemos que las columnas están centradas, por lo que esta suma es equivalente a la suma de las varianzas de cada columna:

$$v_1 + v_2, \text{ with } v_1 = \frac{1}{N} \sum_{i=1}^N X_{i,1}^2 \text{ and } v_2 = \frac{1}{N} \sum_{i=1}^N X_{i,2}^2$$

Podemos calcular  $v_1$  y  $v_2$  al utilizar:

```
colMeans(x^2)
#> [1] 3904 3902
```

y podemos mostrar matemáticamente que si aplicamos una transformación ortogonal como la anterior, la variación total sigue siendo la misma:

```
sum(colMeans(x^2))
#> [1] 7806
sum(colMeans(z^2))
#> [1] 7806
```

Sin embargo, mientras que la variabilidad en las dos columnas de  $X$  es casi la misma, en la versión transformada  $Z$ , el 99% de la variabilidad se incluye solo en la primera dimensión:

```
v <- colMeans(z^2)
v/sum(v)
#> [1] 1.00e+00 9.93e-05
```

El primer *componente principal* (*principal component* o PC por sus siglas en inglés) de una matriz  $X$  es la transformación ortogonal lineal de  $X$  que maximiza esta variabilidad. La función `prcomp` provee esta información:

```
pca <- prcomp(x)
pca$rotation
#> PC1 PC2
#> [1,] -0.702 0.712
#> [2,] -0.712 -0.702
```

Tengan en cuenta que el primer PC es casi el mismo que ese proporcionado por el  $(X_1 + X_2)/\sqrt{2}$  que utilizamos anteriormente (excepto quizás por un cambio de signo que es arbitrario).

La función `prcomp` devuelve la rotación necesaria para transformar  $X$  para que la variabilidad de las columnas disminuya de más variable a menos (se accede con `$rotation`) así como la nueva matriz resultante (que se accede con `$x`). Por defecto, `prcomp` centra las columnas de  $X$  antes de calcular las matrices.

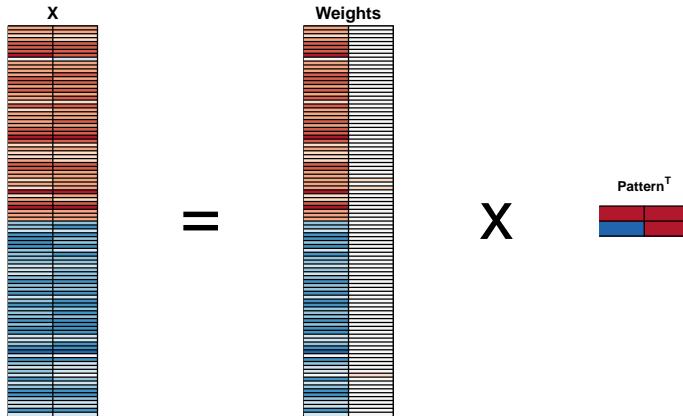
Entonces, usando la multiplicación de matrices que se muestra arriba, notamos que las dos siguientes operaciones dan el mismo resultado (demostrado por una diferencia entre elementos de prácticamente cero):

```
a <- sweep(x, 2, colMeans(x))
b <- pca$x %*% t(pca$rotation)
max(abs(a - b))
#> [1] 3.55e-15
```

La rotación es ortogonal, lo que significa que el inverso es su transposición. Entonces también tenemos que estos dos son idénticos:

```
a <- sweep(x, 2, colMeans(x)) %*% pca$rotation
b <- pca$x
max(abs(a - b))
#> [1] 0
```

Podemos visualizarlos para ver cómo el primer componente resume los datos. En el gráfico a continuación, el rojo representa valores altos y el azul representa valores negativos. Más adelante, en la Sección 33.11.1, aprendemos por qué llamamos a estos pesos (*weights* en inglés) y patrones (*patterns* en inglés):



Resulta que podemos encontrar esta transformación lineal no solo para dos dimensiones, sino también para matrices de cualquier dimensión  $p$ .

Para una matriz multidimensional  $X$  con  $p$  columnas, se puede encontrar una transformación  $Z$  que conserva la distancia entre filas, pero con la varianza de las columnas en orden decreciente. La segunda columna es el segundo componente principal, la tercera columna es el tercer componente principal y así sucesivamente. Como en nuestro ejemplo, si después de un cierto número de columnas, digamos  $k$ , las variaciones de las columnas de  $Z_j$  con  $j > k$  son muy pequeñas, significa que estas dimensiones tienen poco que contribuir a la distancia y podemos aproximar la distancia entre dos puntos con solo  $k$  dimensiones. Si  $k$  es mucho más pequeño que  $p$ , entonces podemos lograr un resumen muy eficiente de nuestros datos.

### 33.5.5 Ejemplo de lirios

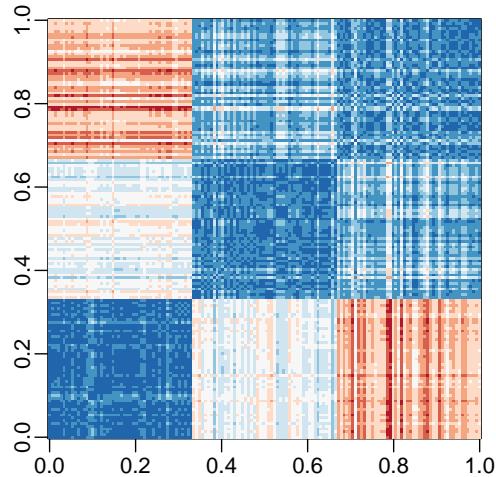
Los datos del lirio (*iris* en inglés) son un ejemplo ampliamente utilizado en los cursos de análisis de datos. Incluye cuatro medidas botánicas relacionadas con tres especies de flores:

```
names(iris)
#> [1] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width"
#> [5] "Species"
```

Si imprimen `iris$Species`, verán que los datos están ordenados por especie.

Calculemos la distancia entre cada observación. Pueden ver claramente las tres especies con una especie muy diferente de las otras dos:

```
x <- iris[,1:4] %>% as.matrix()
d <- dist(x)
image(as.matrix(d), col = rev(RColorBrewer::brewer.pal(9, "RdBu")))
```



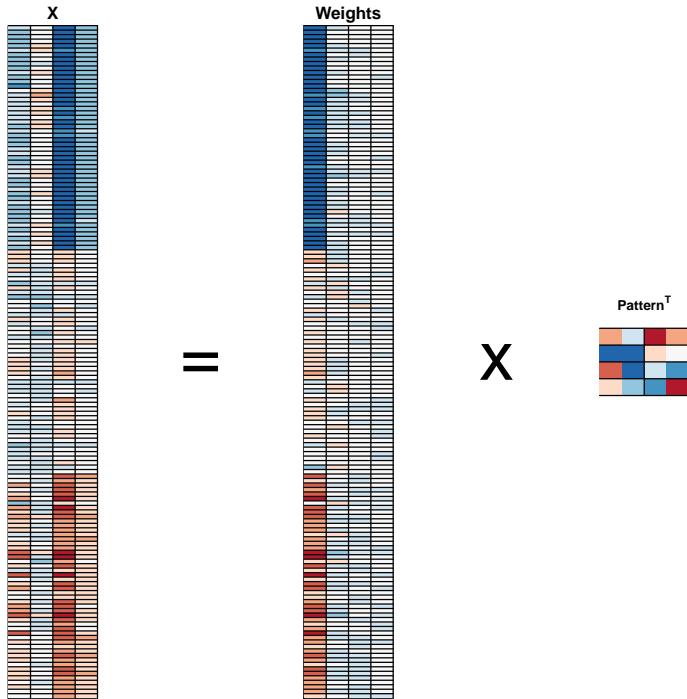
Nuestros predictores aquí tienen cuatro dimensiones, pero tres están muy correlacionadas:

```
cor(x)
#> Sepal.Length Sepal.Width Petal.Length Petal.Width
#> Sepal.Length 1.000 -0.118 0.872 0.818
#> Sepal.Width -0.118 1.000 -0.428 -0.366
#> Petal.Length 0.872 -0.428 1.000 0.963
#> Petal.Width 0.818 -0.366 0.963 1.000
```

Si aplicamos PCA, deberíamos poder aproximar esta distancia con solo dos dimensiones, comprimiendo las dimensiones altamente correlacionadas. Utilizando la función `summary`, podemos ver la variabilidad explicada por cada PC:

```
pca <- prcomp(x)
summary(pca)
#> Importance of components:
#> PC1 PC2 PC3 PC4
#> Standard deviation 2.056 0.4926 0.2797 0.15439
#> Proportion of Variance 0.925 0.0531 0.0171 0.00521
#> Cumulative Proportion 0.925 0.9777 0.9948 1.00000
```

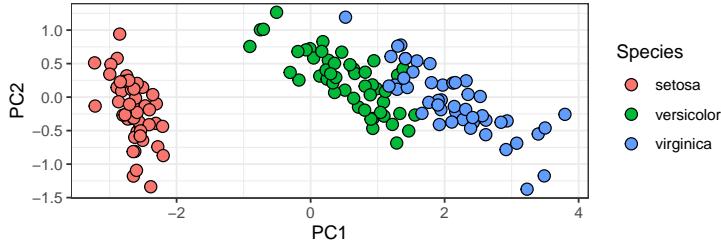
Las dos primeras dimensiones representan el 97% de la variabilidad. Por lo tanto, deberíamos poder aproximar bien la distancia con dos dimensiones. Podemos visualizar los resultados de PCA:



Y ver que el primer patrón es la longitud del sépalo, la longitud del pétalo y el ancho del pétalo (rojo) en una dirección y el ancho del sépalo (azul) en la otra. El segundo patrón es la longitud del sépalo y el ancho del pétalo en una dirección (azul) y la longitud y el ancho del pétalo en la otra (rojo). Pueden ver de los pesos que la primera PC1 controla la mayor parte de la variabilidad y separa claramente el primer tercio de las muestras (setosa) de los dos tercios (versicolor y virginica). Si miran la segunda columna de los pesos, observarán que separa un poco versicolor (rojo) de virginica (azul).

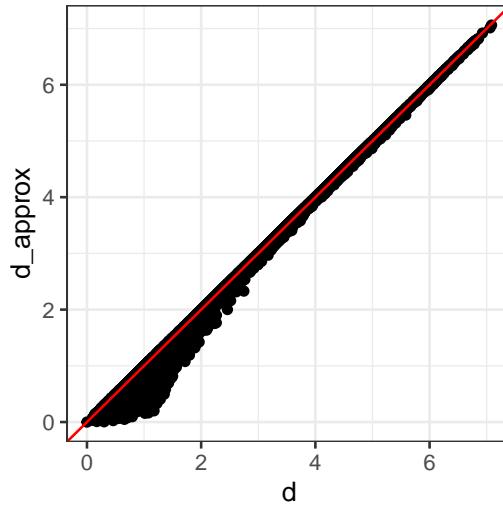
Podemos ver esto mejor al graficar los dos primeros PCs con color representando especie:

```
data.frame(pca$x[,1:2], Species=iris$Species) %>%
 ggplot(aes(PC1,PC2, fill = Species)) +
 geom_point(cex=3, pch=21) +
 coord_fixed(ratio = 1)
```



Vemos que las dos primeras dimensiones preservan la distancia:

```
d_approx <- dist(pca$x[, 1:2])
qplot(d, d_approx) + geom_abline(color="red")
```



Este ejemplo es más realista que el primer ejemplo artificial que utilizamos, ya que mostramos cómo podemos visualizar los datos usando dos dimensiones cuando los datos eran de cuatro dimensiones.

### 33.5.6 Ejemplo de MNIST

El ejemplo de dígitos escritos tiene 784 atributos. ¿Hay espacio para la reducción de datos?  
¿Podemos crear algoritmos sencillos de *machine learning* con menos atributos?

Carguemos los datos:

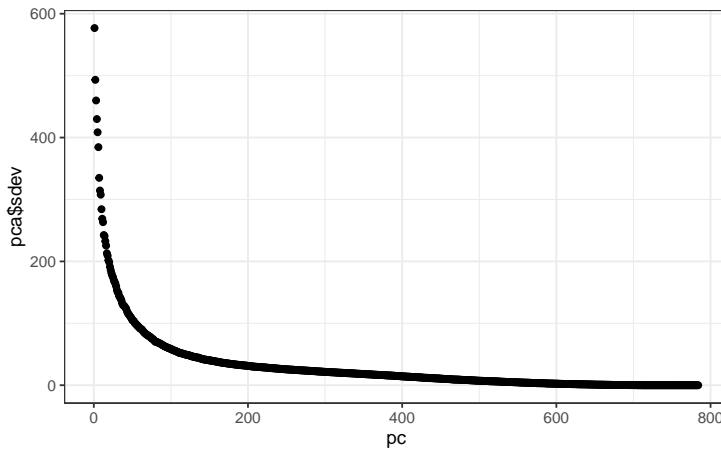
```
library(dslabs)
if(!exists("mnist")) mnist <- read_mnist()
```

Debido a que los píxeles son tan pequeños, esperamos que los píxeles cercanos entre sí en la cuadrícula estén correlacionados, lo que significa que la reducción de dimensión debería ser posible.

Probemos PCA y exploremos la variación de los PCs. Esto tomará unos segundos ya que es una matriz bastante grande.

```
col_means <- colMeans(mnist$test$images)
pca <- prcomp(mnist$train$images)

pc <- 1:ncol(mnist$test$images)
qplot(pc, pca$sdev)
```

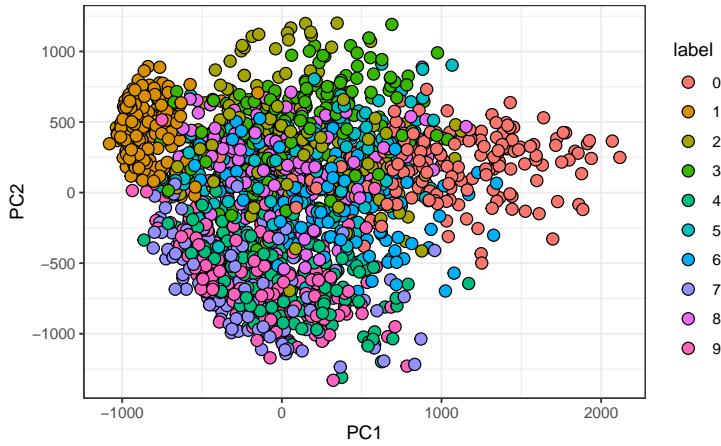


Podemos ver que los primeros PCs ya explican un gran porcentaje de la variabilidad:

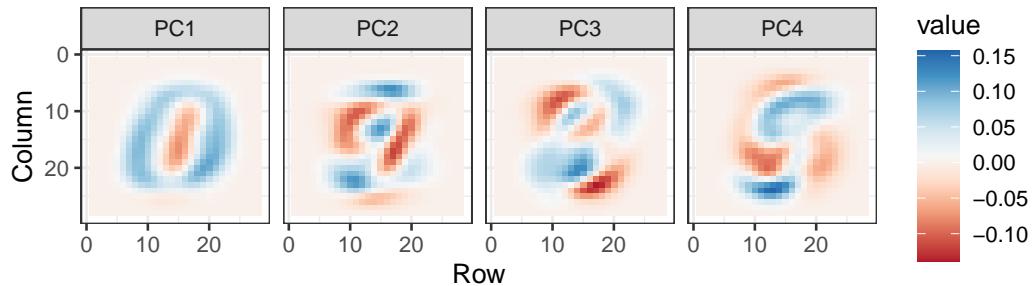
```
summary(pca)$importance[,1:5]
#> PC1 PC2 PC3 PC4 PC5
#> Standard deviation 576.823 493.238 459.8993 429.8562 408.5668
#> Proportion of Variance 0.097 0.071 0.0617 0.0539 0.0487
#> Cumulative Proportion 0.097 0.168 0.2297 0.2836 0.3323
```

Y con solo mirar las dos primeras PCs vemos información sobre la clase. Aquí hay una muestra aleatoria de 2,000 dígitos:

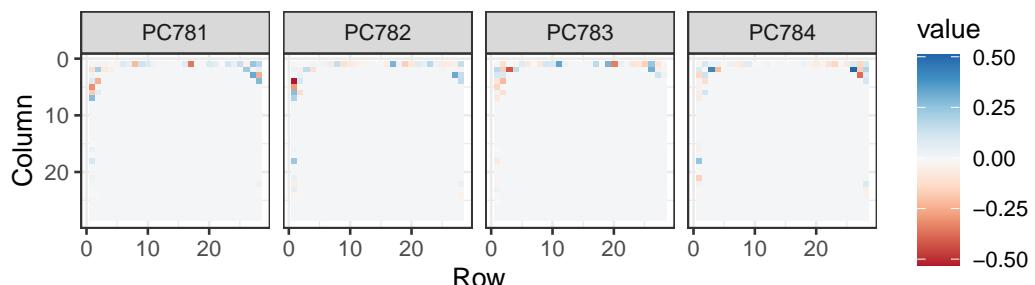
```
data.frame(PC1 = pca$x[,1], PC2 = pca$x[,2],
 label=factor(mnist$train$label)) %>%
sample_n(2000) %>%
ggplot(aes(PC1, PC2, fill=label)) +
geom_point(cex=3, pch=21)
```



También podemos ver las combinaciones lineales en la cuadrícula para tener una idea de lo que se está ponderando:



Los PCs de menor varianza parecen estar relacionadas a la variabilidad irrelevante en las esquinas:



Ahora apliquemos la transformación que aprendimos con los datos de entrenamiento a los datos de evaluación, reduzcamos la dimensión y ejecutemos `knn3` en solo un pequeño número de dimensiones.

Intentamos 36 dimensiones ya que esto explica aproximadamente el 80% de los datos. Primero, ajuste el modelo:

```
library(caret)
k <- 36
x_train <- pca$x[,1:k]
y <- factor(mnist$train$labels)
fit <- knn3(x_train, y)
```

Ahora, transformen el set de evaluación:

```
x_test <- sweep(mnist$test$images, 2, col_means) %*% pca$rotation
x_test <- x_test[,1:k]
```

Y estamos listos para predecir y evaluar los resultados:

```
y_hat <- predict(fit, x_test, type = "class")
confusionMatrix(y_hat, factor(mnist$test$labels))$overall["Accuracy"]
#> Accuracy
#> 0.975
```

Con solo 36 dimensiones, obtenemos una exactitud muy superior a 0.95.

## 33.6 Ejercicios

- Queremos explorar los predictores `tissue_gene_expression` graficándolos.

```
data("tissue_gene_expression")
dim(tissue_gene_expression$x)
```

Procuramos tener una idea de qué observaciones están cercas entre sí, pero como los predictores son de 500 dimensiones, es difícil graficar. Grafique los dos primeros componentes principales con color representando el tipo de tejido.

- Los predictores de cada observación se miden en el mismo dispositivo de medición (un microarreglo de expresión génica) después de un procedimiento experimental. Se utiliza un dispositivo y procedimiento diferente para cada observación. Esto puede introducir sesgos que afecten a todos los predictores de cada observación de la misma manera. Para explorar el efecto de este posible sesgo, para cada observación, calcule el promedio de todos los predictores y luego grafique esto contra el primer PC con el color que representa el tejido. Indique la correlación.

- Vemos una asociación con el primer PC y los promedios de observación. Vuelva a hacer el PCA pero solo después de quitar el centro.
- Para los primeros 10 PCs, haga un diagrama de caja que muestre los valores para cada tejido.
- Grafique el porcentaje de varianza explicado por el número de PC. Sugerencia: use la función `summary`.

### 33.7 Sistemas de recomendación

Los sistemas de recomendación utilizan clasificaciones que los *consumidores* le han dado a *artículos* para hacer recomendaciones específicas. Las compañías que venden muchos productos a muchos clientes y permiten que estos clientes califiquen sus productos, como Amazon, pueden recopilar sets de datos masivos que se pueden utilizar para predecir qué calificación le otorgará un usuario en particular a un artículo específico. Artículos para los cuales se predice una calificación alta para un usuario particular, se recomiendan a ese usuario.

Netflix utiliza un sistema de recomendación para predecir cuántas *estrellas* le dará un usuario a una película específica. Una estrella sugiere que no es una buena película, mientras que cinco estrellas sugieren que es una película excelente. Aquí, ofrecemos los conceptos básicos de cómo se hacen estas recomendaciones, motivados por algunos de los enfoques adoptados por los ganadores del *Netflix challenge*.

En octubre de 2006, Netflix le dio un reto a la comunidad de ciencia de datos: mejoren nuestro algoritmo de recomendación por un 10% y ganen un millón de dólares. En septiembre de 2009, los ganadores se anunciaron<sup>1</sup>. Pueden leer un buen resumen de cómo se creó el algoritmo ganador aquí: <http://blog.echen.me/2011/10/24/winning-the-netflix-prize-a-summary/> y una explicación más detallada aquí: [http://www.netflixprize.com/assets/GrandPrize2009\\_BPC\\_BellKor.pdf](http://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf). Ahora le mostraremos algunas de las estrategias de análisis de datos utilizadas por el equipo ganador.

#### 33.7.1 Datos de MovieLens

Los datos de Netflix no están disponibles públicamente, pero el laboratorio de investigación GroupLens<sup>2</sup> generó su propia base de datos con más de 20 millones de calificaciones para más de 27,000 películas por más de 138,000 usuarios. Ponemos a disposición un pequeño subconjunto de estos datos a través del paquete **dslabs**:

```
library(tidyverse)
library(dslabs)
data("movielens")
```

Podemos ver que esta tabla está en formato *tidy* con miles de filas:

```
movielens %>% as_tibble()
#> # A tibble: 100,004 x 7
#> movieId title year genres userId rating timestamp
#> <int> <chr> <int> <fct> <int> <dbl> <int>
#> 1 31 Dangerous Minds 1995 Drama 1 2.5 1.26e9
#> 2 1029 Dumbo 1941 Animation/Chi~ 1 3 1.26e9
#> 3 1061 Sleepers 1996 Thriller 1 3 1.26e9
#> 4 1129 Escape from New ~ 1981 Action/Advent~ 1 2 1.26e9
#> 5 1172 Cinema Paradiso ~ 1989 Drama 1 4 1.26e9
#> # ... with 99,999 more rows
```

<sup>1</sup><http://bits.blogs.nytimes.com/2009/09/21/netflix-awards-1-million-prize-and-starts-a-new-contest/>  
<sup>2</sup><https://grouplens.org/>

userId	Pulp Fiction	Shawshank Redemption	Forrest Gump	Silence of the Lambs
13	3.5	4.5	5.0	NA
15	5.0	2.0	1.0	5.0
16	NA	4.0	NA	NA
17	5.0	5.0	2.5	4.5
19	5.0	4.0	5.0	3.0
20	0.5	4.5	2.0	0.5

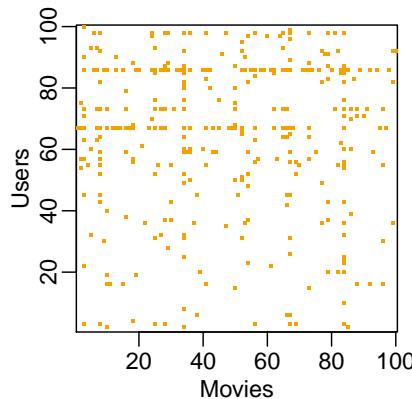
Cada fila representa una calificación dada por un usuario a una película.

Podemos ver la cantidad de usuarios únicos que dan calificaciones y cuántas películas únicas fueron calificadas:

```
movielens %>%
 summarize(n_users = n_distinct(userId) ,
 n_movies = n_distinct(movieId))
#> n_users n_movies
#> 1 671 9066
```

Si multiplicamos esos dos números, obtenemos un número mayor de 5 millones. Sin embargo, nuestra tabla de datos tiene aproximadamente 100,000 filas. Esto implica que no todos los usuarios calificaron todas las películas. Por lo tanto, podemos pensar en estos datos como una matriz muy grande, con usuarios en las filas y películas en las columnas, con muchas celdas vacías. La función `pivot_longer` nos permite convertirla a este formato, pero si lo intentamos para toda la matriz, colgaremos a R. Mostremos la matriz para seis usuarios y cuatro películas.

Pueden pensar en la tarea de un sistema de recomendación como completar los NAs en la tabla de arriba. Para ver cuán dispersa es la matriz, aquí tenemos la matriz para una muestra aleatoria de 100 películas y 100 usuarios con amarillo indicando una combinación de usuario/película para la que tenemos una calificación.

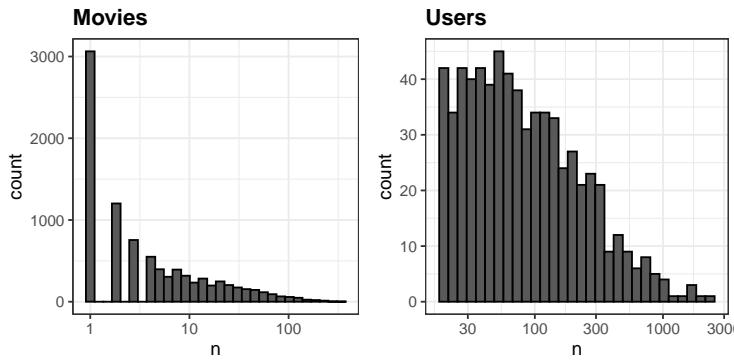


Este reto de *machine learning* es más complicado de lo que hemos estudiado hasta ahora porque cada resultado  $Y$  tiene un set diferente de predictores. Para ver esto, tengan en cuenta que si estamos prediciendo la calificación de la película  $i$  por usuario  $u$ , en principio, todas las otras clasificaciones relacionadas con la película  $i$  y por el usuario  $u$  pueden usarse

como predictores, pero diferentes usuarios califican diferentes películas y diferentes números de películas. Además, podemos usar información de otras películas que hemos determinado que son parecidas a la película  $i$  o de usuarios que se consideran similares al usuario  $u$ . Básicamente, toda la matriz se puede utilizar como predictores para cada celda.

Veamos algunas de las propiedades generales de los datos para entender mejor los retos.

Lo primero que notamos es que algunas películas se evalúan más que otras. A continuación se muestra la distribución. Esto no debería sorprendernos dado que hay películas de gran éxito vistas por millones y películas artísticas e independientes vistas por pocos. Nuestra segunda observación es que algunos usuarios son más activos que otros en la calificación de películas:



### 33.7.2 Sistemas de recomendación como un desafío de *machine learning*

Para ver cómo esto se puede considerar *machine learning*, noten que necesitamos construir un algoritmo con los datos que hemos recopilado que luego se aplicarán fuera de nuestro control, a medida que los usuarios busquen recomendaciones de películas. Así que creamos un set de evaluación para evaluar la exactitud de los modelos que implementamos.

```
library(caret)
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,]
```

Para asegurarnos de que no incluimos usuarios y películas en el set de evaluación que no aparecen en el set de entrenamiento, eliminamos estas entradas usando la función `semi_join`:

```
test_set <- test_set %>%
 semi_join(train_set, by = "movieId") %>%
 semi_join(train_set, by = "userId")
```

### 33.7.3 Función de pérdida

El *Netflix challenge* usó la pérdida de error típica: decidieron un ganador basado en la *desviación cuadrática media* (RMSE por sus siglas en inglés) en un set de evaluación. Defin-

imos  $y_{u,i}$  como la calificación de la película  $i$  por usuario  $u$  y denotamos nuestra predicción con  $\hat{y}_{u,i}$ . El RMSE se define entonces como:

$$\text{RMSE} = \sqrt{\frac{1}{N} \sum_{u,i} (\hat{y}_{u,i} - y_{u,i})^2}$$

con  $N$  siendo el número de combinaciones de usuario/película y la suma que ocurre en todas estas combinaciones.

Recuerden que podemos interpretar el RMSE de manera similar a una desviación estándar: es el error típico que cometemos al predecir una calificación de película. Si este número es mayor que 1, significa que nuestro error típico es mayor que una estrella, lo cual no es bueno.

Escribamos una función que calcule el RMSE para vectores de clasificaciones y sus predictores correspondientes:

```
RMSE <- function(true_ratings, predicted_ratings){
 sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

### 33.7.4 Un primer modelo

Comencemos construyendo el sistema de recomendación más sencillo posible: predecimos la misma calificación para todas las películas, independientemente del usuario. ¿Qué número debería ser esta predicción? Podemos usar un enfoque basado en modelos para responder a esto. Un modelo que supone la misma calificación para todas las películas y usuarios con todas las diferencias explicadas por la variación aleatoria se vería así:

$$Y_{u,i} = \mu + \varepsilon_{u,i}$$

con  $\varepsilon_{i,u}$  errores independientes muestreados de la misma distribución centrada en 0 y  $\mu$  la calificación “verdadera” para todas las películas. Sabemos que el estimador que minimiza el RMSE es el estimador de mínimos cuadrados de  $\mu$  y, en este caso, es el promedio de todas las calificaciones:

```
mu_hat <- mean(train_set$rating)
mu_hat
#> [1] 3.54
```

Si predecimos todas las calificaciones desconocidas con  $\hat{\mu}$ , obtenemos el siguiente RMSE:

```
naive_rmse <- RMSE(test_set$rating, mu_hat)
naive_rmse
#> [1] 1.05
```

Tengan en cuenta que si usan cualquier otro número, obtendrán un RMSE más alto. Por ejemplo:

```
predictions <- rep(3, nrow(test_set))
RMSE(test_set$rating, predictions)
#> [1] 1.19
```

Al observar la distribución de calificaciones, podemos visualizar que esta es la desviación estándar de esa distribución. Obtenemos un RMSE de aproximadamente 1. Para ganar el gran premio de \$1,000,000, un equipo participante tuvo que obtener un RMSE de aproximadamente 0.857. ¡Definitivamente podemos mejorar!

A medida que avanzamos, compararemos diferentes enfoques. Comencemos creando una tabla de resultados con este enfoque simplista:

```
rmse_results <- tibble(method = "Just the average", RMSE = naive_rmse)
```

### 33.7.5 Modelando los efectos de películas

Sabemos por experiencia que algunas películas generalmente tienen una calificación más alta que otras. Esta intuición, que las diferentes películas se clasifican de manera diferente, la confirma los datos. Podemos expandir nuestro modelo anterior agregando el término  $b_i$  para representar la clasificación promedio de la película  $i$ :

$$Y_{u,i} = \mu + b_i + \varepsilon_{u,i}$$

Los libros de texto de estadísticas se refieren a  $b$ s como efectos. Sin embargo, en los artículos sobre el *Netflix challenge*, se refieren a ellos como “sesgo” (o *bias* en inglés; por lo tanto, la notación  $b$ ).

De nuevo podemos usar mínimos cuadrados para estimar  $b_i$  de la siguiente manera:

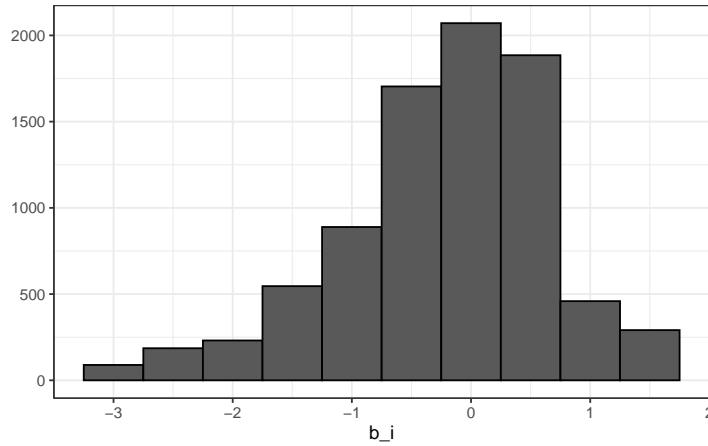
```
fit <- lm(rating ~ as.factor(movieId), data = movielens)
```

Como hay miles de  $b_i$ , a medida que cada película obtiene una, la función `lm()` será muy lenta aquí. Por eso, no recomendamos ejecutar el código anterior. Pero en esta situación particular, sabemos que el estimador de los mínimos cuadrados  $\hat{b}_i$  es solo el promedio de  $Y_{u,i} - \hat{\mu}$  para cada película  $i$ . Entonces podemos calcularlos de esta manera (dejaremos de usar la notación `hat` en el código para representar los estimadores en el futuro):

```
mu <- mean(train_set$rating)
movie_avgs <- train_set %>%
 group_by(movieId) %>%
 summarize(b_i = mean(rating - mu))
```

Podemos ver que estos estimadores varían sustancialmente:

```
qplot(b_i, data = movie_avgs, bins = 10, color = I("black"))
```



Recuerden  $\hat{\mu} = 3.5$ , entonces una  $b_i = 1.5$  implica una calificación perfecta de cinco estrellas.

Veamos cuánto mejora nuestra predicción cuando usamos  $\hat{y}_{u,i} = \hat{\mu} + \hat{b}_i$ :

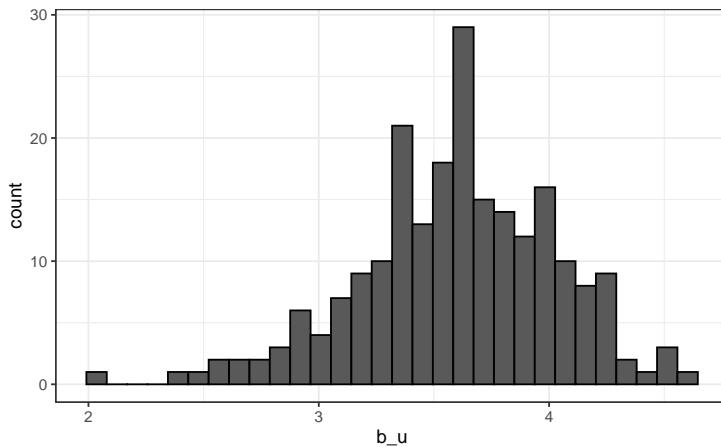
```
predicted_ratings <- mu + test_set %>%
 left_join(movie_avgs, by='movieId') %>%
 pull(b_i)
RMSE(predicted_ratings, test_set$rating)
#> [1] 0.989
```

Ya observamos una mejora. ¿Pero podemos mejorar más?

### 33.7.6 Efectos de usuario

Calculemos la calificación promedio para el usuario  $u$  para aquellos que han calificado 100 o más películas:

```
train_set %>%
 group_by(userId) %>%
 filter(n()>=100) %>%
 summarize(b_u = mean(rating)) %>%
 ggplot(aes(b_u)) +
 geom_histogram(bins = 30, color = "black")
```



Noten que también existe una variabilidad sustancial entre los usuarios: algunos usuarios son muy exigentes y otros adoran cada película. Esto implica que una mejora adicional de nuestro modelo puede ser:

$$Y_{u,i} = \mu + b_i + b_u + \varepsilon_{u,i}$$

dónde  $b_u$  es un efecto específico de cada usuario. Ahora, si un usuario exigente ( $b_u$  negativo) califica una película excelente ( $b_i$  positiva), los efectos se contrarrestan y podemos predecir correctamente que este usuario le dio a esta gran película un 3 en lugar de un 5.

Para ajustar este modelo, podríamos nuevamente usar `lm` así:

```
lm(rating ~ as.factor(movieId) + as.factor(userId))
```

pero, por las razones descritas anteriormente, no lo haremos. En cambio, calcularemos una aproximación calculando  $\hat{\mu}$  y  $\hat{b}_i$  y estimando  $b_u$  como el promedio de  $y_{u,i} - \hat{\mu} - \hat{b}_i$ :

```
user_avgs <- train_set %>%
 left_join(movie_avgs, by='movieId') %>%
 group_by(userId) %>%
 summarize(b_u = mean(rating - mu - b_i))
```

Ahora podemos construir predictores y ver cuánto mejora el RMSE:

```
predicted_ratings <- test_set %>%
 left_join(movie_avgs, by='movieId') %>%
 left_join(user_avgs, by='userId') %>%
 mutate(pred = mu + b_i + b_u) %>%
 pull(pred)
RMSE(predicted_ratings, test_set$rating)
#> [1] 0.905
```

### 33.8 Ejercicios

1. Cargue los datos `movielens`.

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

Calcule el número de calificaciones para cada película y luego compárelo con el año en que salió la película. Transforme los datos usando la raíz cuadrada en los recuentos.

2. Vemos que, en promedio, las películas que salieron después de 1993 obtienen más calificaciones. También vemos que con las películas más nuevas, a partir de 1993, el número de calificaciones disminuye con el año: entre más reciente sea una película, menos tiempo han tenido los usuarios para calificarla.

Entre las películas que salieron en 1993 o más tarde, ¿cuáles son las 25 películas con más calificaciones por año? Además, indique la calificación promedio.

3. De la tabla construida en el ejemplo anterior, vemos que las películas mejor calificadas tienden a tener calificaciones superiores al promedio. Esto no es sorprendente: más personas ven películas populares. Para confirmar esto, estratifique las películas posteriores a 1993 por calificaciones por año y calcule sus calificaciones promedio. Haga un gráfico de la calificación promedio versus calificaciones por año y muestre un estimador de la tendencia.

4. En el ejercicio anterior, vemos que entre más se califica una película, mayor es la calificación. Suponga que está haciendo un análisis predictivo en el que necesita completar las calificaciones faltantes con algún valor. ¿Cuál de las siguientes estrategias usaría?

- Completar los valores faltantes con la calificación promedio de todas las películas.
- Completar los valores faltantes con 0.
- Completar el valor con un valor más bajo que el promedio ya que la falta de calificación se asocia con calificaciones más bajas. Pruebe diferentes valores y evalúe la predicción en un set de evaluación.
- Ninguna de las anteriores.

5. El set de datos `movielens` también incluye un sello de tiempo. Esta variable representa el tiempo y los datos en los que se le dio la calificación. Las unidades son segundos desde el 1 de enero de 1970. Cree una nueva columna `date` con la fecha. Sugerencia: use la función `as_datetime` en el paquete `lubridate`.

6. Calcule la calificación promedio de cada semana y calcule este promedio para cada día. Sugerencia: use la función `round_date` antes de `group_by`.

7. El gráfico muestra alguna evidencia de un efecto temporero. Si definimos  $d_{u,i}$  como el día que el usuario  $u$  hizo su calificación de la película  $i$ , ¿cuál de los siguientes modelos es el más apropiado?

- $Y_{u,i} = \mu + b_i + b_u + d_{u,i} + \varepsilon_{u,i}$ .
- $Y_{u,i} = \mu + b_i + b_u + d_{u,i}\beta + \varepsilon_{u,i}$ .
- $Y_{u,i} = \mu + b_i + b_u + d_{u,i}\beta_i + \varepsilon_{u,i}$ .
- $Y_{u,i} = \mu + b_i + b_u + f(d_{u,i}) + \varepsilon_{u,i}$ , con  $f$  una función suave de  $d_{u,i}$ .

8. Los datos `movielens` también tienen un columna `genres`. Esta columna incluye todos los géneros que aplican a la película. Algunas películas pertenecen a varios géneros. Defina una categoría como cualquier combinación que aparezca en esta columna. Mantenga solo categorías con más de 1,000 calificaciones. Luego, calcule el promedio y error estándar para cada categoría. Grafique estos usando diagramas de barras de error.

9. El gráfico muestra evidencia convincente de un efecto de género. Si definimos  $g_{u,i}$  como el género para la calificación del usuario  $u$  de la película  $i$ , ¿cuál de los siguientes modelos es el más apropiado?

- a.  $Y_{u,i} = \mu + b_i + b_u + d_{u,i} + \varepsilon_{u,i}$ .
- b.  $Y_{u,i} = \mu + b_i + b_u + d_{u,i}\beta + \varepsilon_{u,i}$ .
- c.  $Y_{u,i} = \mu + b_i + b_u + \sum_{k=1}^K x_{u,i}\beta_k + \varepsilon_{u,i}$ , con  $x_{u,i}^k = 1$  si  $g_{u,i}$  es genero  $k$ .
- d.  $Y_{u,i} = \mu + b_i + b_u + f(d_{u,i}) + \varepsilon_{u,i}$ , con  $f$  una función suave de  $d_{u,i}$ .

## 33.9 Regularización

### 33.9.1 Motivación

A pesar de la gran variación de película a película, nuestra mejora en RMSE fue solo como 5%. Exploraremos dónde cometimos errores en nuestro primer modelo, usando solo efectos de película  $b_i$ . Aquí están los 10 errores más grandes:

```
test_set %>%
 left_join(movie_avgs, by='movieId') %>%
 mutate(residual = rating - (mu + b_i)) %>%
 arrange(desc(abs(residual))) %>%
 slice(1:10) %>%
 pull(title)
#> [1] "Kingdom, The (Riget)" "Heaven Knows, Mr. Allison"
#> [3] "American Pimp" "Chinatown"
#> [5] "American Beauty" "Apocalypse Now"
#> [7] "Taxi Driver" "Wallace & Gromit: A Close Shave"
#> [9] "Down in the Delta" "Stalag 17"
```

Todas estas parecen ser películas desconocidas. Para muchas de ellas, predecimos calificaciones altas. Echemos un vistazo a las 10 peores y 10 mejores películas basadas en  $\hat{b}_i$ . Primero, vamos a crear una base de datos que conecta `movieId` al título de la película:

```
movie_titles <- movielens %>%
 select(movieId, title) %>%
 distinct()
```

Aquí están las 10 mejores películas según nuestro estimador:

```
movie_avgs %>% left_join(movie_titles, by="movieId") %>%
 arrange(desc(b_i)) %>%
 slice(1:10) %>%
 pull(title)
#> [1] "When Night Is Falling"
#> [2] "Lamerica"
#> [3] "Mute Witness"
#> [4] "Picture Bride (Bijo photo)"
#> [5] "Red Firecracker, Green Firecracker (Pao Da Shuang Deng)"
#> [6] "Paris, France"
#> [7] "Faces"
#> [8] "Maya Lin: A Strong Clear Vision"
#> [9] "Heavy"
#> [10] "Gate of Heavenly Peace, The"
```

Y aquí están las 10 peores:

```
movie_avgs %>% left_join(movie_titles, by="movieId") %>%
 arrange(b_i) %>%
 slice(1:10) %>%
 pull(title)
#> [1] "Children of the Corn IV: The Gathering"
#> [2] "Barney's Great Adventure"
#> [3] "Merry War, A"
#> [4] "Whiteboyz"
#> [5] "Catfish in Black Bean Sauce"
#> [6] "Killer Shrews, The"
#> [7] "Horrors of Spider Island (Ein Toter Hing im Netz)"
#> [8] "Monkeybone"
#> [9] "Arthur 2: On the Rocks"
#> [10] "Red Heat"
```

Todas parecen ser bastante desconocidas. Veamos con qué frecuencia se califican.

```
train_set %>% count(movieId) %>%
 left_join(movie_avgs, by="movieId") %>%
 left_join(movie_titles, by="movieId") %>%
 arrange(desc(b_i)) %>%
 slice(1:10) %>%
 pull(n)
#> [1] 1 1 1 1 3 1 1 2 1 1

train_set %>% count(movieId) %>%
 left_join(movie_avgs) %>%
 left_join(movie_titles, by="movieId") %>%
 arrange(b_i) %>%
 slice(1:10) %>%
 pull(n)
#> Joining, by = "movieId"
#> [1] 1 1 1 1 1 1 1 1 1 1
```

Las supuestas películas “mejores” y “peores” fueron calificadas por muy pocos usuarios, en la mayoría de los casos por solo 1. Estas películas son en su mayoría desconocidas. Esto se debe a que con solo unos pocos usuarios, tenemos más incertidumbre. Por lo tanto, mayores estimadores de  $b_i$ , negativo o positivo, son más probables.

Estos son estimadores ruidosos en los que no debemos confiar, especialmente cuando se trata de predicciones. Grandes errores pueden aumentar nuestro RMSE, por lo que preferimos ser conservadores cuando no estamos seguros.

En secciones anteriores, calculamos el error estándar y construimos intervalos de confianza para tomar en cuenta los diferentes niveles de incertidumbre. Sin embargo, al hacer predicciones, necesitamos un número, una predicción, no un intervalo. Para esto, presentamos el concepto de regularización.

La regularización nos permite penalizar estimadores más grandes que se forman utilizando pequeños tamaños de muestra. Tienen puntos en común con el enfoque bayesiano que redujo las predicciones descritas en la Sección 16.4.

### 33.9.2 Mínimos cuadrados penalizados

La idea general detrás de la regularización es restringir la variabilidad total de los tamaños del efecto. ¿Por qué esto ayuda? Consideren un caso en el que tenemos película  $i = 1$  con 100 clasificaciones de usuarios y 4 películas  $i = 2, 3, 4, 5$  con solo una calificación de usuario. Tenemos la intención de ajustar el modelo:

$$Y_{u,i} = \mu + b_i + \varepsilon_{u,i}$$

Supongan que sabemos que la calificación promedio es, digamos,  $\mu = 3$ . Si usamos mínimos cuadrados, el estimador para el primer efecto de película  $b_1$  es el promedio de las 100 calificaciones de los usuarios,  $1/100 \sum_{i=1}^{100} (Y_{i,1} - \mu)$ , que esperamos sea bastante preciso. Sin embargo, el estimador para las películas 2, 3, 4 y 5 será simplemente la desviación observada de la calificación promedio  $\hat{b}_i = Y_{u,i} - \hat{\mu}$ . Pero esto es un estimador basado en un solo número, por lo cual no será preciso. Tengan en cuenta que estos estimadores hacen el error  $Y_{u,i} - \mu + \hat{b}_i$  igual a 0 para  $i = 2, 3, 4, 5$ , pero este es un caso de sobreentrenamiento. De hecho, ignorando al único usuario y adivinando que las películas 2, 3, 4 y 5 son solo películas promedio ( $b_i = 0$ ) podría ofrecer una mejor predicción. La idea general de la regresión penalizada es controlar la variabilidad total de los efectos de la película:  $\sum_{i=1}^5 b_i^2$ . Específicamente, en lugar de minimizar la ecuación de mínimos cuadrados, minimizamos una ecuación que añade una penalización:

$$\sum_{u,i} (y_{u,i} - \mu - b_i)^2 + \lambda \sum_i b_i^2$$

El primer término es solo la suma de errores cuadrados y el segundo es una penalización que aumenta cuando muchos  $b_i$  son grandes. Usando cálculo, podemos mostrar que los valores de  $b_i$  que minimizan esta ecuación son:

$$\hat{b}_i(\lambda) = \frac{1}{\lambda + n_i} \sum_{u=1}^{n_i} (Y_{u,i} - \hat{\mu})$$

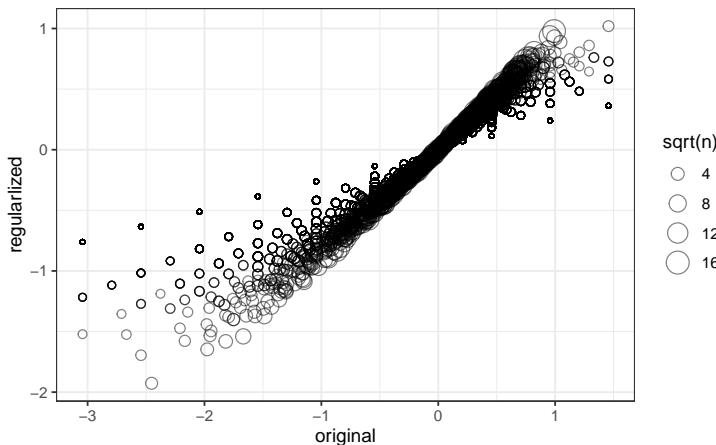
dónde  $n_i$  es la cantidad de clasificaciones hechas para la película  $i$ . Este enfoque tendrá el efecto deseado: cuando nuestro tamaño de muestra  $n_i$  es muy grande, un caso que nos dará un estimador estable, entonces la penalización  $\lambda$  es efectivamente ignorada ya que  $n_i + \lambda \approx n_i$ . Sin embargo, cuando el  $n_i$  es pequeño, entonces el estimador  $\hat{b}_i(\lambda)$  se encoge hacia 0. Entre más grande  $\lambda$ , más nos encogemos.

Calculemos estos estimadores regularizados de  $b_i$  utilizando  $\lambda = 3$ . Más adelante, veremos por qué elegimos 3.

```
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 se reducen los estimadores, hagamos un gráfico de los estimadores regularizados versus los estimadores de mínimos cuadrados.

```
tibble(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)
```



Ahora, echemos un vistazo a las 10 mejores películas según los estimadores penalizados  $\hat{b}_i(\lambda)$ :

```
train_set %>%
 count(movieId) %>%
 left_join(movie_reg_avgs, by = "movieId") %>%
 left_join(movie_titles, by = "movieId") %>%
 arrange(desc(b_i)) %>%
 slice(1:10) %>%
 pull(title)
#> [1] "Paris Is Burning" "Shawshank Redemption, The"
```

```
#> [3] "Godfather, The" "African Queen, The"
#> [5] "Band of Brothers" "Paperman"
#> [7] "On the Waterfront" "All About Eve"
#> [9] "Usual Suspects, The" "Ikiru"
```

¡Esto tiene mucho más sentido! Estas películas son más populares y tienen más calificaciones. Aquí están las 10 peores películas:

```
train_set %>%
 count(movieId) %>%
 left_join(movie_reg_avgs, by = "movieId") %>%
 left_join(movie_titles, by="movieId") %>%
 arrange(b_i) %>%
 select(title, b_i, n) %>%
 slice(1:10) %>%
 pull(title)
#> [1] "Battlefield Earth"
#> [2] "Joe's Apartment"
#> [3] "Super Mario Bros."
#> [4] "Speed 2: Cruise Control"
#> [5] "Dungeons & Dragons"
#> [6] "Batman & Robin"
#> [7] "Police Academy 6: City Under Siege"
#> [8] "Cats & Dogs"
#> [9] "Disaster Movie"
#> [10] "Mighty Morphin Power Rangers: The Movie"
```

¿Mejoramos nuestros resultados?

```
predicted_ratings <- test_set %>%
 left_join(movie_reg_avgs, by = "movieId") %>%
 mutate(pred = mu + b_i) %>%
 pull(pred)
RMSE(predicted_ratings, test_set$rating)
#> [1] 0.97
```

```
#> # A tibble: 4 x 2
#> method RMSE
#> <chr> <dbl>
#> 1 Just the average 1.05
#> 2 Movie Effect Model 0.989
#> 3 Movie + User Effects Model 0.905
#> 4 Regularized Movie Effect Model 0.970
```

Los estimadores penalizados ofrecen una gran mejora sobre los estimadores de mínimos cuadrados.

### 33.9.3 Cómo elegir los términos de penalización

Noten que  $\lambda$  es un parámetro de ajuste. Podemos usar validación cruzada para elegirlo.

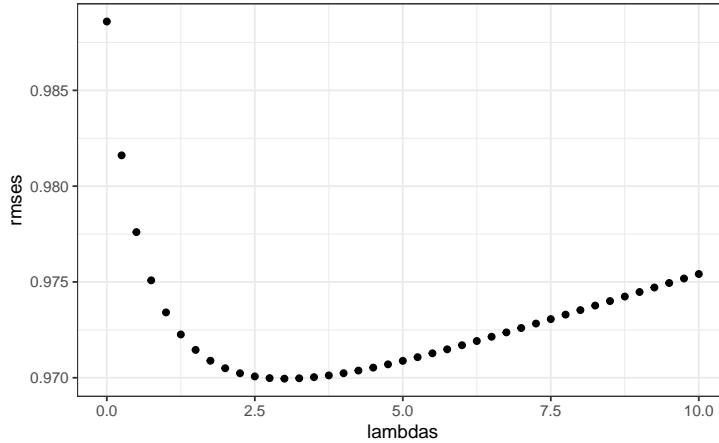
```

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) %>%
 pull(pred)
 return(RMSE(predicted_ratings, test_set$rating))
})
qplot(lambdas, rmses)
lambdas[which.min(rmses)]
#> [1] 3

```



Sin embargo, si bien mostramos esto como una ilustración, en la práctica deberíamos usar validación cruzada completa solo en el set de entrenamiento, sin usar el set de evaluación hasta la evaluación final. El set de evaluación nunca debe utilizarse para el ajustamiento.

También podemos utilizar la regularización para estimar los efectos del usuario. Estamos minimizando:

$$\sum_{u,i} (y_{u,i} - \mu - b_i - b_u)^2 + \lambda \left( \sum_i b_i^2 + \sum_u b_u^2 \right)$$

Los estimadores que minimizan esto se pueden encontrar de manera similar a lo que hicimos anteriormente. Aquí usamos validación cruzada para elegir un  $\lambda$ :

```

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))

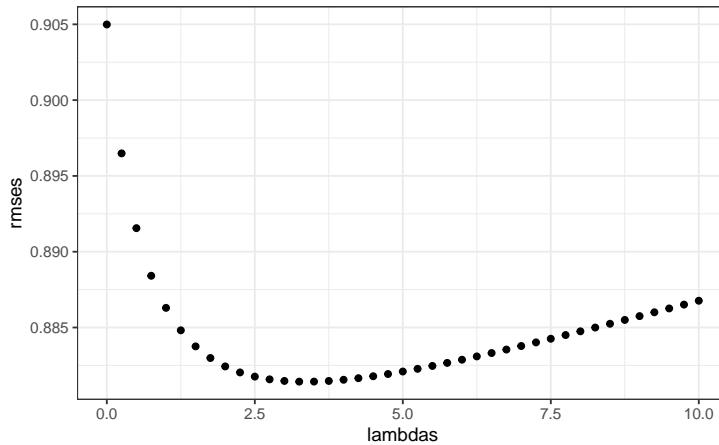
 b_u <- train_set %>%
 left_join(b_i, by="movieId") %>%
 group_by(userId) %>%
 summarize(b_u = sum(rating - b_i - mu)/(n()+1))

 predicted_ratings <-
 test_set %>%
 left_join(b_i, by = "movieId") %>%
 left_join(b_u, by = "userId") %>%
 mutate(pred = mu + b_i + b_u) %>%
 pull(pred)

 return(RMSE(predicted_ratings, test_set$rating))
})

qplot(lambdas, rmses)

```



Para el modelo completo, el  $\lambda$  óptimo es:

```

lambda <- lambdas[which.min(rmses)]
lambda
#> [1] 3.25

```

method	RMSE
Just the average	1.053
Movie Effect Model	0.989
Movie + User Effects Model	0.905
Regularized Movie Effect Model	0.970
Regularized Movie + User Effect Model	0.881

### 33.10 Ejercicios

Un experto en educación aboga por escuelas más pequeñas. El experto basa esta recomendación en el hecho de que entre las mejores escuelas, muchas son escuelas pequeñas. Simulemos un set de datos para 100 escuelas. Primero, simulemos el número de estudiantes en cada escuela.

```
set.seed(1986)
n <- round(2^rnorm(1000, 8, 1))
```

Ahora asignemos una calidad *verdadera* para cada escuela completamente independiente del tamaño. Este es el parámetro que queremos estimar.

```
mu <- round(80 + 2 * rt(1000, 5))
range(mu)
schools <- data.frame(id = paste("PS", 1:100),
 size = n,
 quality = mu,
 rank = rank(-mu))
```

Podemos ver que las 10 mejores escuelas son:

```
schools %>% top_n(10, quality) %>% arrange(desc(quality))
```

Ahora hagamos que los estudiantes en la escuela tomen un examen. Existe una variabilidad aleatoria en la toma de exámenes, por lo que simularemos las puntuaciones de los exámenes distribuidos normalmente con el promedio determinado por la calidad de la escuela y las desviaciones estándar de 30 puntos porcentuales:

```
scores <- sapply(1:nrow(schools), function(i){
 scores <- rnorm(schools$size[i], schools$quality[i], 30)
 scores
})
schools <- schools %>% mutate(score = sapply(scores, mean))
```

1. ¿Cuáles son las mejores escuelas según la puntuación promedio? Muestre solo la identificación, el tamaño y la puntuación promedio.
2. Compare el tamaño medio de la escuela con el tamaño medio de las 10 mejores escuelas según la puntuación.

3. Según esta prueba, parece que las escuelas pequeñas son mejores que las grandes. Cinco de las 10 mejores escuelas tienen 100 estudiantes o menos. ¿Pero cómo puede ser ésto? Construimos la simulación para que la calidad y el tamaño sean independientes. Repita el ejercicio para las peores 10 escuelas.

4. ¡Lo mismo es cierto para las peores escuelas! También son pequeñas. Grafique la puntuación promedio versus el tamaño de la escuela para ver qué está pasando. Destaque las 10 mejores escuelas según la calidad *verdadera*. Use la transformación de escala logarítmica para el tamaño.

5. Podemos ver que el error estándar de la puntuación tiene una mayor variabilidad cuando la escuela es más pequeña. Esta es una realidad estadística básica que aprendimos en las secciones de probabilidad e inferencia. De hecho, noten que 4 de las 10 mejores escuelas se encuentran en las 10 mejores escuelas según la puntuación del examen.

Usemos la regularización para elegir las mejores escuelas. Recuerde que la regularización encoge las desviaciones del promedio hacia 0. Entonces para aplicar la regularización aquí, primero debemos definir el promedio general para todas las escuelas:

```
overall <- mean(sapply(scores, mean))
```

y luego definir, para cada escuela, cómo se desvía de ese promedio. Escriba un código que calcule la puntuación por encima del promedio de cada escuela pero dividiéndolo por  $n + \lambda$  en lugar de  $n$ , con  $n$  el tamaño de la escuela y  $\lambda$  un parámetro de regularización. Intente con  $\lambda = 3$ .

6. Noten que esto mejora un poco las cosas. El número de escuelas pequeñas que no figuran entre las mejores ahora es 4. ¿Existe un  $\lambda$  mejor? Encuentre el  $\lambda$  que minimiza el  $\text{RMSE} = \frac{1}{100} \sum_{i=1}^{100} (\text{quality} - \text{estimate})^2$ .

7. Clasifique las escuelas según el promedio obtenido con los mejores  $\alpha$ . Tengan en cuenta que ninguna escuela pequeña se incluye incorrectamente.

8. Un error común al usar la regularización es reducir los valores hacia 0 que no están centrados alrededor de 0. Por ejemplo, si no restamos el promedio general antes de reducir, obtenemos un resultado muy similar. Confirme esto volviendo a ejecutar el código del ejercicio 6, pero sin eliminar la media general.

### 33.11 Factorización de matrices

La factorización de matrices es un concepto ampliamente utilizado en *machine learning*. Está muy relacionado con el análisis de factores, la descomposición de valores singulares (*singular value decomposition* o SVD por sus siglas en inglés) y el análisis de componentes principales (PCA). Aquí describimos el concepto en el contexto de los sistemas de recomendación de películas.

Hemos descrito cómo el modelo:

$$Y_{u,i} = \mu + b_i + b_u + \varepsilon_{u,i}$$

explica las diferencias de película a película a través de la  $b_i$  y las diferencias de usuario a usuario a través de la  $b_u$ . Pero este modelo omite una fuente importante de variación relacionada con el hecho de que los grupos de películas tienen patrones de calificación similares y los grupos de usuarios también tienen patrones de calificación similares. Descubriremos estos patrones estudiando los residuos:

$$r_{u,i} = y_{u,i} - \hat{b}_i - \hat{b}_u$$

Para ver esto, convertiremos los datos en una matriz para que cada usuario obtenga una fila, cada película obtenga una columna y  $y_{u,i}$  sea la entrada en fila  $u$  y columna  $i$ . Con fines ilustrativos, solo consideraremos un pequeño subconjunto de películas con muchas calificaciones y usuarios que han calificado muchas películas. También incluimos *Scent of a Woman* (`movieId == 3252`) porque la usamos para un ejemplo específico:

```
train_small <- movieLens %>%
 group_by(movieId) %>%
 filter(n() >= 50 | movieId == 3252) %>% ungroup() %>%
 group_by(userId) %>%
 filter(n() >= 50) %>% ungroup()

y <- train_small %>%
 select(userId, movieId, rating) %>%
 pivot_wider(names_from = "movieId", values_from = "rating") %>%
 as.matrix()
```

Agregamos nombres de fila y columna:

```
rownames(y) <- y[,1]
y <- y[,-1]

movie_titles <- movieLens %>%
 select(movieId, title) %>%
 distinct()

colnames(y) <- with(movie_titles, title[match(colnames(y), movieId)])
```

y los convertimos en residuos eliminando los efectos de columna y fila:

```
y <- sweep(y, 2, colMeans(y, na.rm=TRUE))
y <- sweep(y, 1, rowMeans(y, na.rm=TRUE))
```

Si el modelo anterior explica todas las señales, y los  $\varepsilon$  son solo ruido, entonces los residuos para diferentes películas deben ser independientes entre sí. Pero no lo son. Aquí hay unos ejemplos:

```
m_1 <- "Godfather, The"
m_2 <- "Godfather: Part II, The"
p1 <- qplot(y[,m_1], y[,m_2], xlab = m_1, ylab = m_2)
```

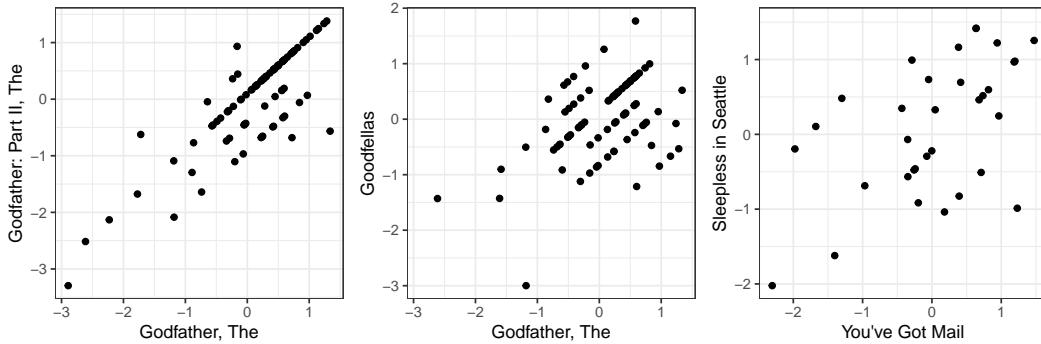
```

m_1 <- "Godfather, The"
m_3 <- "Goodfellas"
p2 <- qplot(y[,m_1], y[,m_3], xlab = m_1, ylab = m_3)

m_4 <- "You've Got Mail"
m_5 <- "Sleepless in Seattle"
p3 <- qplot(y[,m_4], y[,m_5], xlab = m_4, ylab = m_5)

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

```



Estos gráficos muestran que a los usuarios que les gustó *The Godfather* más de lo que el modelo espera de ellos, según la película y los efectos del usuario, también les gustó *The Godfather II* más de lo esperado. Se observa una relación similar al comparar *The Godfather* y *Goodfellas*. Aunque no es tan fuerte, todavía hay correlación. También vemos correlaciones entre *You've Got Mail* y *Sleepless in Seattle*.

Al observar la correlación entre películas, podemos ver un patrón (cambiamos los nombres de las columnas para ahorrar espacio de impresión):

```

x <- y[, c(m_1, m_2, m_3, m_4, m_5)]
short_names <- c("Godfather", "Godfather2", "Goodfellas",
 "You've Got", "Sleepless")
colnames(x) <- short_names
cor(x, use="pairwise.complete")
#> Godfather Godfather2 Goodfellas You've Got Sleepless
#> Godfather 1.000 0.829 0.444 -0.440 -0.378
#> Godfather2 0.829 1.000 0.521 -0.331 -0.358
#> Goodfellas 0.444 0.521 1.000 -0.481 -0.402
#> You've Got -0.440 -0.331 -0.481 1.000 0.533
#> Sleepless -0.378 -0.358 -0.402 0.533 1.000

```

Parece que hay personas a las que les gustan las comedias románticas más de lo esperado, mientras que hay otras personas a las que les gustan las películas de gángsters más de lo esperado.

Estos resultados nos dicen que hay estructura en los datos. Pero, ¿cómo podemos modelar esto?

### 33.11.1 Análisis de factores

Aquí hay una ilustración, usando una simulación, de cómo podemos usar un poco de estructura para predecir el  $r_{u,i}$ . Supongan que nuestros residuos  $\mathbf{r}$  se ven así:

```
round(r, 1)
#> Godfather Godfather2 Goodfellas You've Got Sleepless
#> 1 2.0 2.3 2.2 -1.8 -1.9
#> 2 2.0 1.7 2.0 -1.9 -1.7
#> 3 1.9 2.4 2.1 -2.3 -2.0
#> 4 -0.3 0.3 0.3 -0.4 -0.3
#> 5 -0.3 -0.4 0.3 0.2 0.3
#> 6 -0.1 0.1 0.2 -0.3 0.2
#> 7 -0.1 0.0 -0.2 -0.2 0.3
#> 8 0.2 0.2 0.1 0.0 0.4
#> 9 -1.7 -2.1 -1.8 2.0 2.4
#> 10 -2.3 -1.8 -1.7 1.8 1.7
#> 11 -1.7 -2.0 -2.1 1.9 2.3
#> 12 -1.8 -1.7 -2.1 2.3 2.0
```

Parece que hay un patrón aquí. De hecho, podemos ver patrones de correlación muy fuertes:

```
cor(r)
#> Godfather Godfather2 Goodfellas You've Got Sleepless
#> Godfather 1.000 0.980 0.978 -0.974 -0.966
#> Godfather2 0.980 1.000 0.983 -0.987 -0.992
#> Goodfellas 0.978 0.983 1.000 -0.986 -0.989
#> You've Got -0.974 -0.987 -0.986 1.000 0.986
#> Sleepless -0.966 -0.992 -0.989 0.986 1.000
```

Podemos crear vectores  $\mathbf{q}$  y  $\mathbf{p}$ , que pueden explicar gran parte de la estructura que vemos. Los  $\mathbf{q}$  se verían así:

```
t(q)
#> Godfather Godfather2 Goodfellas You've Got Sleepless
#> [1,] 1 1 1 -1 -1
```

y reduce las películas a dos grupos: gángster (codificado con 1) y romance (codificado con -1). También podemos reducir los usuarios a tres grupos:

```
t(p)
#> 1 2 3 4 5 6 7 8 9 10 11 12
#> [1,] 2 2 2 0 0 0 0 0 -2 -2 -2 -2
```

los que les gustan las películas de gángsters y no les gustan las películas románticas (codificadas como 2), los que les gustan las películas románticas y no les gustan las películas de gángsters (codificadas como -2), y los que no les importa (codificadas como 0). El punto principal aquí es que casi podemos reconstruir  $\mathbf{r}$ , que tiene 60 valores, con un par de vectores que totalizan 17 valores. Noten que  $\mathbf{p}$  y  $\mathbf{q}$  son equivalentes a los patrones y pesos que describimos en la Sección 33.5.4.

Si  $r$  contiene los residuos para usuarios  $u = 1, \dots, 12$  para películas  $i = 1, \dots, 5$ , podemos escribir la siguiente fórmula matemática para nuestros residuos  $r_{u,i}$ .

$$r_{u,i} \approx p_u q_i$$

Esto implica que podemos explicar más variabilidad modificando nuestro modelo anterior para recomendaciones de películas a:

$$Y_{u,i} = \mu + b_i + b_u + p_u q_i + \varepsilon_{u,i}$$

Sin embargo, motivamos la necesidad del término  $p_u q_i$  con una simulación sencilla. La estructura que se encuentra en los datos suele ser más compleja. Por ejemplo, en esta primera simulación supusimos que solo había un factor  $p_u$  que determinaba a cuál de las dos películas de géneros  $u$  pertenece. Pero la estructura en nuestros datos de películas parece ser mucho más complicada que las películas de gángsters versus las románticas. Podemos tener muchos otros factores. Aquí presentamos una simulación un poco más compleja. Ahora agregamos una sexta película.

```
round(r, 1)
#> Godfather Godfather2 Goodfellas You've Got Sleepless Scent
#> 1 0.5 0.6 1.6 -0.5 -0.5 -1.6
#> 2 1.5 1.4 0.5 -1.5 -1.4 -0.4
#> 3 1.5 1.6 0.5 -1.6 -1.5 -0.5
#> 4 -0.1 0.1 0.1 -0.1 -0.1 0.1
#> 5 -0.1 -0.1 0.1 0.0 0.1 -0.1
#> 6 0.5 0.5 -0.4 -0.6 -0.5 0.5
#> 7 0.5 0.5 -0.5 -0.6 -0.4 0.4
#> 8 0.5 0.6 -0.5 -0.5 -0.4 0.4
#> 9 -0.9 -1.0 -0.9 1.0 1.1 0.9
#> 10 -1.6 -1.4 -0.4 1.5 1.4 0.5
#> 11 -1.4 -1.5 -0.5 1.5 1.6 0.6
#> 12 -1.4 -1.4 -0.5 1.6 1.5 0.6
```

Al explorar la estructura de correlación de este nuevo set de datos:

```
colnames(r)[4:6] <- c("YGM", "SS", "SW")
cor(r)
#> Godfather Godfather2 Goodfellas YGM SS SW
#> Godfather 1.000 0.997 0.562 -0.997 -0.996 -0.571
#> Godfather2 0.997 1.000 0.577 -0.998 -0.999 -0.583
#> Goodfellas 0.562 0.577 1.000 -0.552 -0.583 -0.994
#> YGM -0.997 -0.998 -0.552 1.000 0.998 0.558
#> SS -0.996 -0.999 -0.583 0.998 1.000 0.588
#> SW -0.571 -0.583 -0.994 0.558 0.588 1.000
```

notamos que quizás necesitamos un segundo factor para tomar en cuenta el hecho de que a algunos usuarios les gusta Al Pacino, mientras que a otros no les gusta o no les importa. Observen que la estructura general de la correlación obtenida de los datos simulados no está tan lejos de la correlación real:

```

six_movies <- c(m_1, m_2, m_3, m_4, m_5, m_6)
x <- y[, six_movies]
colnames(x) <- colnames(r)
cor(x, use="pairwise.complete")
#> Godfather Godfather2 Goodfellas YGM SS SW
#> Godfather 1.0000 0.829 0.444 -0.440 -0.378 0.0589
#> Godfather2 0.8285 1.000 0.521 -0.331 -0.358 0.1186
#> Goodfellas 0.4441 0.521 1.000 -0.481 -0.402 -0.1230
#> YGM -0.4397 -0.331 -0.481 1.000 0.533 -0.1699
#> SS -0.3781 -0.358 -0.402 0.533 1.000 -0.1822
#> SW 0.0589 0.119 -0.123 -0.170 -0.182 1.0000

```

Para explicar esta estructura más complicada, necesitamos dos factores. Por ejemplo, algo como lo siguiente:

```

t(q)
#> Godfather Godfather2 Goodfellas You've Got Sleepless Scent
#> [1,] 1 1 1 -1 -1 -1
#> [2,] 1 1 -1 -1 -1 1

```

con el primer factor (la primera fila) utilizado para codificar las películas de gángster versus las películas románticas y un segundo factor (la segunda fila) para explicar los grupos de películas con Al Pacino versus los grupos sin Al Pacino. También necesitaremos dos sets de coeficientes para explicar la variabilidad introducida por los tipos de grupos  $3 \times 3$ :

```

t(p)
#> 1 2 3 4 5 6 7 8 9 10 11 12
#> [1,] 1.0 1.0 1.0 0 0 0.0 0.0 0.0 -1 -1.0 -1.0 -1.0
#> [2,] -0.5 0.5 0.5 0 0 0.5 0.5 0.5 0 -0.5 -0.5 -0.5

```

El modelo con dos factores tiene 36 parámetros que se pueden usar para explicar gran parte de la variabilidad en las 72 clasificaciones:

$$Y_{u,i} = \mu + b_i + b_u + p_{u,1}q_{1,i} + p_{u,2}q_{2,i} + \varepsilon_{u,i}$$

Tengan en cuenta que en una aplicación de datos real, necesitamos ajustar este modelo a los datos. Para explicar la correlación compleja que observamos en datos reales, generalmente permitimos que las entradas de  $p$  y  $q$  sean valores continuos, en lugar de discretos como las que usamos en la simulación. Por ejemplo, en lugar de dividir las películas en gángster o romance, definimos un continuo. Además, recuerden que este no es un modelo lineal y para ajustarlo necesitamos usar un algoritmo diferente a ese usado por `lm` para encontrar los parámetros que minimizan los mínimos cuadrados. Los algoritmos ganadores del *Netflix challenge* ajustaron un modelo similar al anterior y utilizaron la regularización para penalizar por grandes valores de  $p$  y  $q$ , en lugar de usar mínimos cuadrados. Implementar este enfoque está más allá del alcance de este libro.

### 33.11.2 Conexión a SVD y PCA

La descomposición:

$$r_{u,i} \approx p_{u,1}q_{1,i} + p_{u,2}q_{2,i}$$

está muy relacionada a SVD y PCA. SVD y PCA son conceptos complicados, pero una forma de entenderlos es que SVD es un algoritmo que encuentra los vectores  $p$  y  $q$  que nos permiten reescribir la matriz  $r$  con  $m$  filas y  $n$  columnas como:

$$r_{u,i} = p_{u,1}q_{1,i} + p_{u,2}q_{2,i} + \cdots + p_{u,n}q_{n,i}$$

con la variabilidad de cada término disminuyendo y con las  $ps$  no correlacionadas. El algoritmo también calcula esta variabilidad para que podamos saber cuánta de la variabilidad total de las matrices se explica a medida que vayamos agregando nuevos términos. Esto nos deja ver que, con solo unos pocos términos, podemos explicar la mayor parte de la variabilidad.

Veamos un ejemplo con los datos de la película. Para calcular la descomposición, haremos que los residuos con NAs sean iguales a 0:

```
y[is.na(y)] <- 0
pca <- prcomp(y)
```

Los vectores  $q$  se denominan componentes principales y se almacenan en esta matriz:

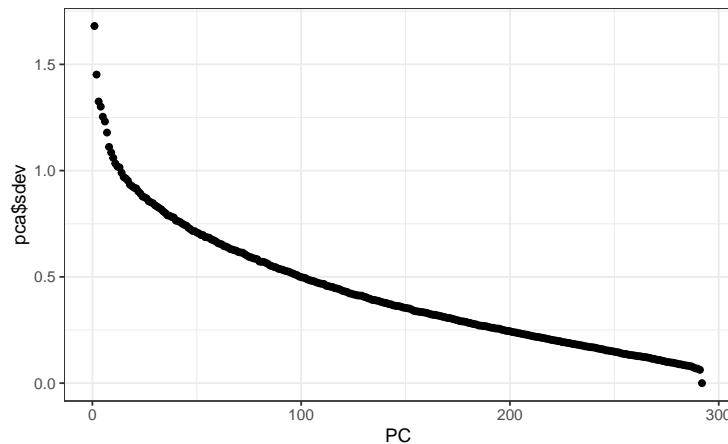
```
dim(pca$rotation)
#> [1] 454 292
```

Mientras que la  $p$ , o los efectos del usuario, están aquí:

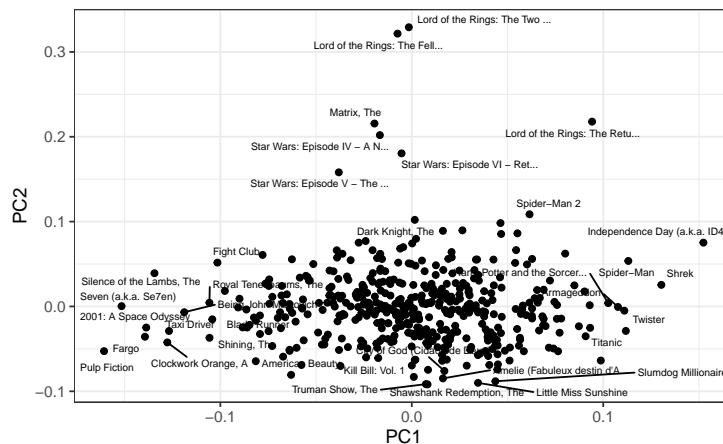
```
dim(pca$x)
#> [1] 292 292
```

Podemos ver la variabilidad de cada uno de los vectores:

```
qplot(1:nrow(x), pca$sdev, xlab = "PC")
```



También notamos que los dos primeros componentes principales están relacionados a la estructura en las opiniones sobre películas:



Con solo mirar los 10 primeros en cada dirección, vemos un patrón significativo. El primer PC muestra la diferencia entre las películas aclamadas por la crítica en un lado:

```
#> [1] "Pulp Fiction" "Seven (a.k.a. Se7en)"
#> [3] "Fargo" "2001: A Space Odyssey"
#> [5] "Silence of the Lambs, The" "Clockwork Orange, A"
#> [7] "Taxi Driver" "Being John Malkovich"
#> [9] "Royal Tenenbaums, The" "Shining, The"
```

y los éxitos de taquilla de Hollywood en otro:

```
#> [1] "Independence Day (a.k.a. ID4)" "Shrek"
#> [3] "Spider-Man" "Titanic"
#> [5] "Twister" "Armageddon"
#> [7] "Harry Potter and the Sorcer..." "Forrest Gump"
#> [9] "Lord of the Rings: The Retu..." "Enemy of the State"
```

Mientras que el segundo PC parece ir de películas artísticas e independientes:

```
#> [1] "Shawshank Redemption, The" "Truman Show, The"
#> [3] "Little Miss Sunshine" "Slumdog Millionaire"
#> [5] "Amelie (Fabuleux destin d'A...)" "Kill Bill: Vol. 1"
#> [7] "American Beauty" "City of God (Cidade de Deus)"
#> [9] "Mars Attacks!" "Beautiful Mind, A"
```

hacia favoritas de los *nerds*:

```
#> [1] "Lord of the Rings: The Two ..." "Lord of the Rings: The Fell..."
#> [3] "Lord of the Rings: The Retu..." "Matrix, The"
#> [5] "Star Wars: Episode IV - A N...." "Star Wars: Episode VI - Ret..."
#> [7] "Star Wars: Episode V - The ..." "Spider-Man 2"
#> [9] "Dark Knight, The" "Speed"
```

Ajustar un modelo que incorpora estos estimadores es complicado. Para aquellos interesados en implementar un enfoque que incorpore estas ideas, recomendamos el paquete **recomenderlab**. Los detalles están más allá del alcance de este libro.

### 33.12 Ejercicios

En este set de ejercicios, trataremos un tema útil para comprender la factorización de matrices: la *descomposición de valores singulares* (*singular value decomposition* o SVD por sus siglas en inglés). SVD es un resultado matemático que se usa ampliamente en *machine learning*, tanto en la práctica como para comprender las propiedades matemáticas de algunos algoritmos. Este es un tema bastante avanzado y para completar este set de ejercicios tendrán que estar familiarizados con conceptos de álgebra lineal, como la multiplicación de matrices, las matrices ortogonales y las matrices diagonales.

El SVD nos dice que podemos *descomponer* un  $N \times p$  matriz  $Y$  con  $p < N$  como:

$$Y = UDV^\top$$

con  $U$  y  $V$  *ortogonal* de dimensiones  $N \times p$  y  $p \times p$ , respectivamente, y  $D$  un  $p \times p$  matriz *diagonal* con los valores de la diagonal decreciendo:

$$d_{1,1} \geq d_{2,2} \geq \dots d_{p,p}.$$

En este ejercicio, veremos una de las formas en que esta descomposición puede ser útil. Para hacer esto, construiremos un set de datos que representa las calificaciones de 100 estudiantes en 24 materias diferentes. El promedio general se ha eliminado, por lo que estos datos representan el punto porcentual que cada estudiante recibió por encima o por debajo de la puntuación promedio de la prueba. Entonces un 0 representa una calificación promedio (C), un 25 es una calificación alta (A +) y un -25 representa una calificación baja (F). Puede simular los datos de esta manera:

```
set.seed(1987)
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="_"))
```

Nuestro objetivo es describir el desempeño de los estudiantes de la manera más sucinta posible. Por ejemplo, queremos saber si los resultados de estas pruebas son solo números independientes aleatorios. ¿Todos los estudiantes son igual de buenos? ¿Ser bueno en un

tema implica ser bueno en otro? ¿Cómo ayuda el SVD con todo esto? Iremos paso a paso para mostrar que con solo tres pares relativamente pequeños de vectores podemos explicar gran parte de la variabilidad en este  $100 \times 24$  set de datos.

Puede visualizar las 24 puntuaciones de las pruebas para los 100 estudiantes al graficar una imagen:

```
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)
```

1. ¿Cómo describiría los datos basados en esta figura?

- a. Las puntuaciones de las pruebas son independientes entre sí.
- b. Los estudiantes que evalúan bien están en la parte superior de la imagen y parece que hay tres agrupaciones por materia.
- c. Los estudiantes que son buenos en matemáticas no son buenos en ciencias.
- d. Los estudiantes que son buenos en matemáticas no son buenos en humanidades.

2. Puede examinar la correlación entre las puntuaciones de la prueba directamente de esta manera:

```
my_image(cor(y), zlim = c(-1,1))
range(cor(y))
axis(side = 2, 1:ncol(y), rev(colnames(y)), las = 2)
```

¿Cuál de las siguientes opciones describe mejor lo que ve?

- a. Las puntuaciones de las pruebas son independientes.
- b. Las matemáticas y las ciencias están altamente correlacionadas, pero las humanidades no.
- c. Existe una alta correlación entre las pruebas en la misma materia pero no hay correlación entre las materias.
- d. Hay una correlación entre todas las pruebas, pero es mayor si las pruebas son de ciencias y matemáticas e incluso mayor dentro de cada materia.

3. Recuerde que la ortogonalidad significa que  $U^\top U$  y  $V^\top V$  son iguales a la matriz de identidad. Esto implica que también podemos reescribir la descomposición como:

$$YV = UD \text{ or } U^\top Y = DV^\top$$

Podemos pensar en  $YV$  y  $U^\top V$  como dos transformaciones de  $Y$  que preservan la variabilidad total de  $Y$  ya que  $U$  y  $V$  son ortogonales.

Use la función `svd` para calcular el SVD de  $y$ . Esta función devolverá  $U$ ,  $V$  y las entradas diagonales de  $D$ .

```
s <- svd(y)
names(s)
```

Puede verificar que el SVD funciona al escribir:

```
y_svd <- s$u %*% diag(s$d) %*% t(s$v)
max(abs(y - y_svd))
```

Calcule la suma de cuadrados de las columnas de  $Y$  y guárdelas en `ss_y`. Luego calcule la suma de cuadrados de columnas del transformado  $YV$  y guárdelas en `ss_yv`. Confirme que `sum(ss_y)` es igual a `sum(ss_yv)`.

4. Vemos que se conserva la suma total de cuadrados. Esto es porque  $V$  es ortogonal. Ahora para comenzar a entender cómo  $YV$  es útil, grafique `ss_y` contra el número de columna y luego haga lo mismo para `ss_yv`. ¿Qué observa?

5. Vemos que la variabilidad de las columnas de  $YV$  está disminuyendo. Además, vemos que, en relación con los tres primeros, la variabilidad de las columnas más allá del tercero es casi 0. Ahora observe que no tuvimos que calcular `ss_yv` porque ya tenemos la respuesta ¿Cómo? Recuerde que  $YV = UD$  y como  $U$  es ortogonal, sabemos que la suma de cuadrados de las columnas de  $UD$  son las entradas diagonales de  $D$  al cuadrado. Confirme esto graficando la raíz cuadrada de `ss_yv` versus las entradas diagonales de  $D$ .

6. De lo anterior, sabemos que la suma de cuadrados de las columnas de  $Y$  (la suma total de cuadrados) se añade a la suma de  $s\$d^2$  y que la transformación  $YV$  nos da columnas con sumas de cuadrados iguales a  $s\$d^2$ . Ahora calcule qué porcentaje de la variabilidad total se explica solo por las tres primeras columnas de  $YV$ .

7. Vemos que casi el 99% de la variabilidad se explica por las primeras tres columnas de  $YV = UD$ . Entonces esto implica que deberíamos poder explicar gran parte de la variabilidad y estructura que encontramos al explorar los datos con unas pocas columnas. Antes de continuar, vamos a mostrar un truco computacional útil para evitar crear la matriz `diag(s$d)`. Para motivar esto, notamos que si escribimos  $U$  en sus columnas  $[U_1, U_2, \dots, U_p]$ , entonces  $UD$  es igual a:

$$UD = [U_1 d_{1,1}, U_2 d_{2,2}, \dots, U_p d_{p,p}]$$

Utilice la función `sweep` para calcular  $UD$  sin construir `diag(s$d)` y sin usar multiplicación de matrices.

8. Sabemos que  $U_1 d_{1,1}$ , la primera columna de  $UD$ , tiene la mayor variabilidad de todas las columnas de  $UD$ . Anteriormente vimos una imagen de  $Y$ :

```
my_image(y)
```

en la que podemos ver que la variabilidad de estudiante a estudiante es bastante grande y que parece que los estudiantes que son buenos en una materia son buenos en todas. Esto implica que el promedio (en todas las materias) de cada alumno debe explicar en gran medida la

variabilidad. Calcule la puntuación promedio de cada estudiante y grafíquelo contra  $U_1 d_{1,1}$ . Describa lo que encuentra.

9. Notamos que los signos en SVD son arbitrarios porque:

$$UDV^\top = (-U)D(-V)^\top$$

Con esto en mente, vemos que la primera columna de  $UD$  es casi idéntica a la puntuación promedio de cada estudiante, excepto por el signo.

Esto implica que multiplicar  $Y$  por la primera columna de  $V$  debe realizar una operación similar a tomar el promedio. Haga un gráfico de imagen de  $V$  y describa la primera columna en relación con las otras y cómo se relaciona esto con tomar un promedio.

10. Ya vimos que podemos reescribir  $UD$  como:

$$U_1 d_{1,1} + U_2 d_{2,2} + \cdots + U_p d_{p,p}$$

con  $U_j$  la columna  $j$  de  $U$ . Esto implica que podemos reescribir todo el SVD como:

$$Y = U_1 d_{1,1} V_1^\top + U_2 d_{2,2} V_2^\top + \cdots + U_p d_{p,p} V_p^\top$$

con  $V_j$  la columna  $j$  de  $V$ . Grafique  $U_1$ , luego grafique  $V_1^\top$  usando el mismo rango para los límites del eje y, entonces haga una imagen de  $U_1 d_{1,1} V_1^\top$  y compárela con la imagen de  $Y$ . Sugerencia: use la función `my_image` definida anteriormente y el argumento `drop=FALSE` para asegurar que los subconjuntos de matrices son matrices.

11. Vemos que con solo un vector de longitud 100, un escalar y un vector de longitud 24, nos acercamos a reconstruir la matriz  $100 \times 24$  original. Esta es nuestra primera factorización de matrices:

$$Y \approx d_{1,1} U_1 V_1^\top$$

Sabemos que explica  $s \$d[1]^2 / sum(s\$ d^2) * 100$  por ciento de la variabilidad total. Nuestra aproximación solo explica la observación de que los buenos estudiantes tienden a ser buenos en todas las materias. Pero otro aspecto de los datos originales que nuestra aproximación no explica es la mayor similitud que observamos dentro de las materias. Podemos ver esto calculando la diferencia entre nuestra aproximación y los datos originales y luego calculando las correlaciones. Pueden ver esto ejecutando este código:

```
resid <- y - with(s,(u[,1, drop=FALSE]*d[1]) %*% t(v[,1, drop=FALSE]))
my_image(cor(resid), zlim = c(-1,1))
axis(side = 2, 1:ncol(y), rev(colnames(y)), las = 2)
```

Ahora que hemos eliminado el efecto general del estudiante, el gráfico de correlación revela que todavía no hemos explicado la correlación interna de la materia ni el hecho de que las puntuaciones en matemáticas y ciencias son más parecidas entre sí que a las puntuaciones en las artes. Así que exploremos la segunda columna del SVD. Repita el ejercicio anterior pero para la segunda columna: grafique  $U_2$ , entonces grafique  $V_2^\top$  usando el mismo rango para los límites del eje y, finalmente haga una imagen de  $U_2 d_{2,2} V_2^\top$  y compárela con la imagen de `resid`.

12. La segunda columna se relaciona claramente con la diferencia de habilidad del estudiante en matemáticas/ciencias versus las artes. Podemos ver esto más claramente en el gráfico de `s$v[,2]`. Sumar las matrices resultantes usando estas dos columnas ayudará con nuestra aproximación:

$$Y \approx d_{1,1}U_1V_1^\top + d_{2,2}U_2V_2^\top$$

Sabemos que explicará:

```
sum(s$d[1:2]^2)/sum(s$d^2) * 100
```

porcentaje de la variabilidad total. Podemos calcular nuevos residuos así:

```
resid <- y - with(s,sweep(u[,1:2], 2, d[1:2], FUN="*") %*% t(v[,1:2]))
my_image(cor(resid), zlim = c(-1,1))
axis(side = 2, 1:ncol(y), rev(colnames(y)), las = 2)
```

y ver que la estructura que queda es impulsada por las diferencias entre matemáticas y ciencias. Confirme esto graficando  $U_3$ , luego grafique  $V_3^\top$  usando el mismo rango para los límites del eje y, luego haga una imagen de  $U_3d_{3,3}V_3^\top$  y compárela con la imagen de `resid`.

13. La tercera columna se relaciona claramente con la diferencia de habilidad del estudiante en matemáticas y ciencias. Podemos ver esto más claramente en el gráfico de `s$v[,3]`. Agregar la matriz que obtenemos con estas dos columnas ayudará con nuestra aproximación:

$$Y \approx d_{1,1}U_1V_1^\top + d_{2,2}U_2V_2^\top + d_{3,3}U_3V_3^\top$$

Sabemos que explicará:

```
sum(s$d[1:3]^2)/sum(s$d^2) * 100
```

porcentaje de la variabilidad total. Podemos calcular nuevos residuos como este:

```
resid <- y - with(s,sweep(u[,1:3], 2, d[1:3], FUN="*") %*% t(v[,1:3]))
my_image(cor(resid), zlim = c(-1,1))
axis(side = 2, 1:ncol(y), rev(colnames(y)), las = 2)
```

Ya no vemos estructura en los residuos: parecen ser independientes entre sí. Esto implica que podemos describir los datos con el siguiente modelo:

$$Y = d_{1,1}U_1V_1^\top + d_{2,2}U_2V_2^\top + d_{3,3}U_3V_3^\top + \varepsilon$$

con  $\varepsilon$  una matriz de errores independientes idénticamente distribuidos. Este modelo es útil porque resumimos  $100 \times 24$  observaciones con  $3 \times (100 + 24 + 1) = 375$  números. Además, los tres componentes del modelo tienen interpretaciones útiles: 1) la capacidad general de un estudiante, 2) la diferencia en la habilidad entre las matemáticas/ciencias y las artes, y 3) las diferencias restantes entre las tres materias. Los tamaños  $d_{1,1}$ ,  $d_{2,2}$  y  $d_{3,3}$  nos dicen la variabilidad explicada por cada componente. Finalmente, tengan en cuenta que los componentes  $d_{j,j}U_jV_j^\top$  son equivalentes al componente principal j.

Termine el ejercicio graficando una imagen de  $Y$ , una imagen de  $d_{1,1}U_1V_1^\top + d_{2,2}U_2V_2^\top + d_{3,3}U_3V_3^\top$  y una imagen de los residuos, todos con el mismo `zlim`.

14. **Avanzado:** El set de datos `movielens` incluido en el paquete `dslabs` es un pequeño subconjunto de un set de datos más grande con millones de clasificaciones. Puede encontrar el set de datos más reciente aquí: <https://grouplens.org/datasets/movielens/20m/>. Cree su propio sistema de recomendaciones utilizando todas las herramientas que le hemos mostrado.



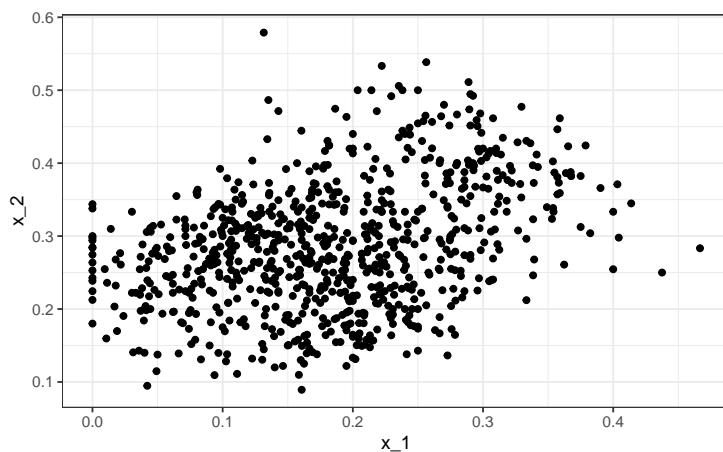
# 34

## Agrupación

Los algoritmos que hemos descrito hasta ahora son ejemplos de un enfoque general denominado *machine learning supervisado*. El nombre proviene del hecho de que usamos los resultados en un set de entrenamiento para supervisar la creación de nuestro algoritmo de predicción. Hay otro subconjunto de *machine learning* denominado *no supervisado*. En este subconjunto, no necesariamente conocemos los resultados y, en cambio, estamos interesados en descubrir grupos. Estos algoritmos también se denominan algoritmos de *agrupamiento* (*clustering* en inglés) ya que los predictores se utilizan para definir *grupos* (*clusters* en inglés).

En los dos ejemplos que hemos utilizado en esta parte del libro, la agrupación no sería muy útil. En el primer ejemplo, si simplemente se nos dan las alturas, no podremos descubrir dos grupos, hombres y mujeres, porque la intersección es grande. En el segundo ejemplo, al graficar los predictores, podemos ver que descubrir los dos dígitos, 2 y 7, será retante:

```
library(tidyverse)
library(dslabs)
data("mnist_27")
mnist_27$train %>% qplot(x_1, x_2, data = .)
```



Sin embargo, hay aplicaciones en las que el aprendizaje no supervisado puede ser una técnica poderosa, en particular como una herramienta exploratoria.

Un primer paso en cualquier algoritmo de agrupamiento es definir una distancia entre observaciones o grupos de observaciones. Luego, decidimos cómo unir las observaciones en grupos. Hay muchos algoritmos para hacer esto. Aquí presentamos dos como ejemplos: jerárquico y *k-means*.

Construiremos un ejemplo sencillo basado en clasificaciones de películas. Aquí construimos rápidamente una matriz  $\mathbf{x}$  que tiene calificaciones para las 50 películas con más calificaciones.

```

data("movielens")
top <- movielens %>%
 group_by(movieId) %>%
 summarize(n=n(), title = first(title)) %>%
 top_n(50, n) %>%
 pull(movieId)

x <- movielens %>%
 filter(movieId %in% top) %>%
 group_by(userId) %>%
 filter(n() >= 25) %>%
 ungroup() %>%
 select(title, userId, rating) %>%
 spread(userId, rating)

row_names <- str_remove(x$title, ": Episode") %>% str_trunc(20)
x <- x[,-1] %>% as.matrix()
x <- sweep(x, 2, colMeans(x, na.rm = TRUE))
x <- sweep(x, 1, rowMeans(x, na.rm = TRUE))
rownames(x) <- row_names

```

Queremos utilizar estos datos para averiguar si hay grupos de películas basados en las calificaciones de 139 calificadores de películas. Un primer paso es encontrar la distancia entre cada par de películas usando la función `dist`:

```
d <- dist(x)
```

---

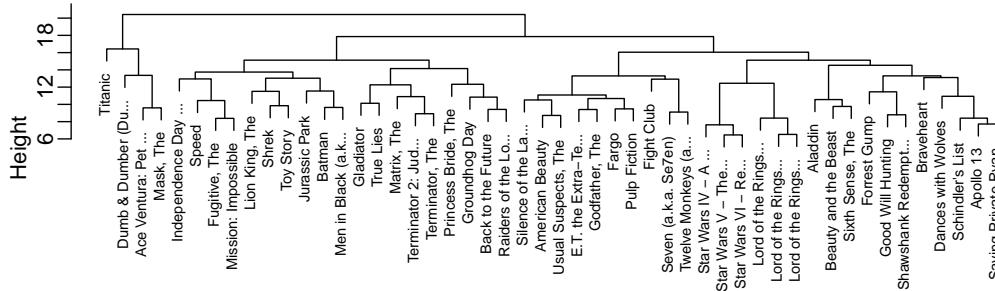
### 34.1 Agrupación jerárquica

Con la distancia calculada entre cada par de películas, necesitamos un algoritmo para definir grupos a partir de estas. La agrupación jerárquica comienza definiendo cada observación como un grupo separado. Entonces, los dos grupos más cercanos se unen en un grupo de forma iterativa hasta que solo haya un grupo que incluye todas las observaciones. La función `hclust` implementa este algoritmo y toma una distancia como entrada.

```
h <- hclust(d)
```

Podemos ver los grupos resultantes usando un *dendrograma*.

```
plot(h, cex = 0.65, main = "", xlab = "")
```



Para interpretar este gráfico, primero, determinamos la distancia entre dos películas encontrando la primera posición, de arriba a abajo, donde las películas se dividen en dos grupos diferentes. La altura de esta ubicación es la distancia entre estos dos grupos. Entonces, la distancia entre las películas de *Star Wars* es de 8 o menos, mientras que la distancia entre *Raiders of the Lost Ark* y *Silence of the Lambs* es de aproximadamente 17.

Para generar grupos reales, podemos hacer una de dos cosas: 1) decidir la distancia mínima necesaria para que las observaciones estén en el mismo grupo o 2) decidir la cantidad de grupos que desean y luego encontrar la distancia mínima que lo logra. La función `cutree` se puede aplicar al resultado de `hclust` para realizar cualquiera de estas dos operaciones y generar grupos.

```
groups <- cutree(h, k = 10)
```

Noten que la agrupación provee algunas ideas sobre los tipos de películas. El grupo 4 parece ser éxitos de taquilla:

```
names(groups)[groups==4]
#> [1] "Apollo 13" "Braveheart" "Dances with Wolves"
#> [4] "Forrest Gump" "Good Will Hunting" "Saving Private Ryan"
#> [7] "Schindler's List" "Shawshank Redempt..."
```

Y el grupo 9 parece ser películas *nerd*:

```
names(groups)[groups==9]
#> [1] "Lord of the Rings..." "Lord of the Rings..." "Lord of the Rings..."
#> [4] "Star Wars IV - A ..." "Star Wars V - The..." "Star Wars VI - Re..."
```

Podemos cambiar el tamaño del grupo haciendo `k` más grande o `h` más pequeño. También podemos explorar los datos para ver si hay grupos de evaluadores de películas.

```
h_2 <- dist(t(x)) %>% hclust()
```

## 34.2 k-means

Para usar el algoritmo de agrupamiento *k-means*, tenemos que predefinir  $k$ , el número de grupos que queremos definir. El algoritmo *k-means* es iterativo. El primer paso es definir  $k$  centros. Luego, cada observación se asigna al grupo con el centro más cercano a esa observación. En un segundo paso, los centros se redefinen utilizando la observación en cada grupo: los medios de columna se utilizan para definir un *centroide*. Repetimos estos dos pasos hasta que los centros converjan.

La función `kmeans` incluida en base R no funciona con NAs. Con fines ilustrativos, reemplazaremos las NAs con 0s. En general, la decisión de cómo completar los datos que faltan, o si uno debería hacerlo, debe hacerse con cuidado.

```
x_0 <- x
x_0[is.na(x_0)] <- 0
k <- kmeans(x_0, centers = 10)
```

Las asignaciones de grupos están en el componente `cluster`:

```
groups <- k$cluster
```

Recuerden que debido a que el primer centro se elige al azar, los grupos finales son aleatorios. Imponemos cierta estabilidad al repetir la función entera varias veces y tomar el promedio de los resultados. El número de valores iniciales aleatorios para utilizar se puede asignar a través del argumento `nstart`.

```
k <- kmeans(x_0, centers = 10, nstart = 25)
```

---

## 34.3 Mapas de calor

Una poderosa herramienta de visualización para descubrir grupos o patrones en sus datos es el *mapa de calor* (*heatmap* en inglés). La idea es sencilla: graficar una imagen de su matriz de datos con colores utilizados como señal visual y con tanto las columnas como las filas ordenadas según los resultados de un algoritmo de agrupamiento. Demostraremos esto con el set de datos `tissue_gene_expression`. Escalaremos las filas de la matriz de expresión génica.

El primer paso es calcular:

```
data("tissue_gene_expression")
x <- sweep(tissue_gene_expression$x, 2, colMeans(tissue_gene_expression$x))
h_1 <- hclust(dist(x))
h_2 <- hclust(dist(t(x)))
```

Ahora podemos usar los resultados de esta agrupación para ordenar las filas y columnas.

```
image(x[h_1$order, h_2$order])
```

Pero hay una función, `heatmap`, que lo hace por nosotros:

```
heatmap(x, col = RColorBrewer::brewer.pal(11, "Spectral"))
```

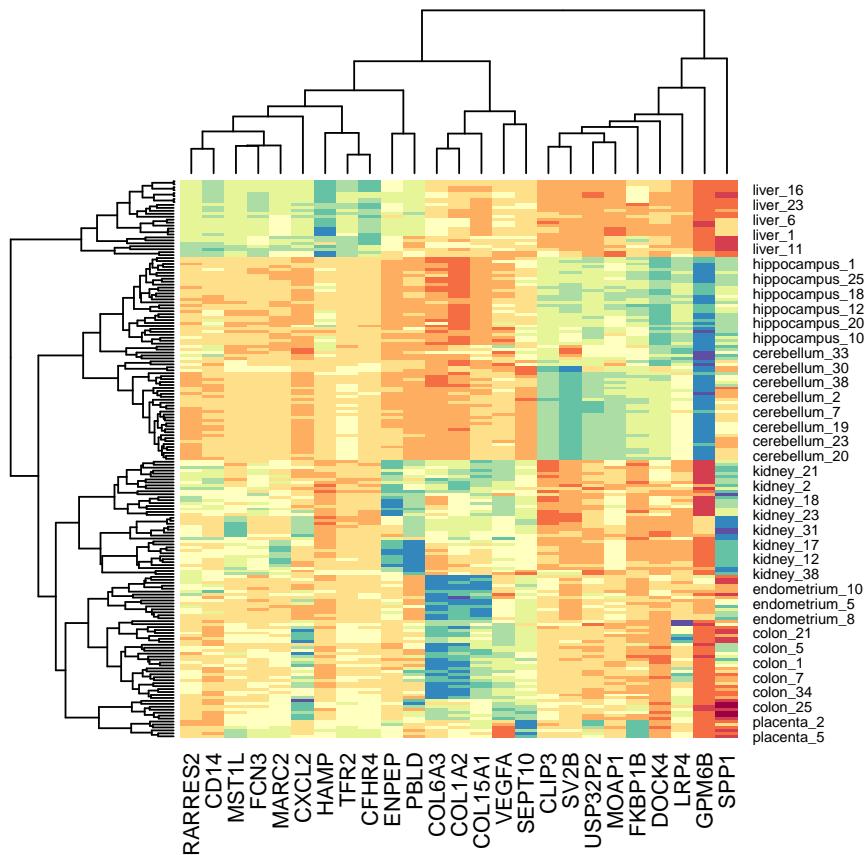
No mostramos los resultados de la función `heatmap` porque hay demasiados atributos para que el gráfico sea útil. Por lo tanto, filtramos algunas columnas y rehacemos los gráficos.

---

## 34.4 Filtrando atributos

Si la información sobre los grupos se incluye en unos pocos atributos, incluir todos los atributos puede agregar suficiente ruido como para que detectar grupos sea retante. Un enfoque sencillo para tratar de eliminar atributos sin información es incluir solo aquellos con alta varianza. En el ejemplo de la película, un usuario con baja variación en sus calificaciones no es realmente informativo: todas las películas le parecen iguales. Aquí hay un ejemplo de cómo podemos incluir solo los atributos con alta varianza.

```
library(matrixStats)
sds <- colSds(x, na.rm = TRUE)
o <- order(sds, decreasing = TRUE)[1:25]
heatmap(x[,o], col = RColorBrewer::brewer.pal(11, "Spectral"))
```



### 34.5 Ejercicios

1. Cargue el set de datos `tissue_gene_expression`. Reste las medias de cada fila y calcule la distancia entre cada observación. Guarde el resultado en `d`.
2. Haga un gráfico de agrupamiento jerárquico y agregue los tipos de tejido como etiquetas.
3. Ejecute una agrupación *k-means* en los datos con  $K = 7$ . Haga una tabla que compara los grupos identificados con los tipos de tejidos correctos. Ejecute el algoritmo varias veces para ver cómo cambia la respuesta.
4. Seleccione los 50 genes más variables. Asegúrese de que las observaciones aparezcan en las columnas y que los predictores estén centrados. Agregue una barra de colores para mostrar los diferentes tipos de tejidos. Sugerencia: use el argumento `ColSideColors` para asignar colores. Además, use `col = RColorBrewer::brewer.pal(11, "RdBu")` para un mejor uso de los colores.

## Part VI

# Herramientas de productividad



# 35

## Introducción a las herramientas de productividad

En términos generales, no recomendamos utilizar enfoques de apuntar y hacer clic para el análisis de datos. En cambio, recomendamos lenguajes de script, como R, ya que son más flexibles y facilitan enormemente la reproducibilidad. Del mismo modo, recomendamos no utilizar enfoques de apuntar y hacer clic para organizar archivos y preparar documentos. En este capítulo, demostraremos enfoques alternativos. Específicamente, aprenderemos a usar herramientas disponibles de forma gratuita que, aunque al principio parezcan complicadas y no intuitivas, eventualmente los convertirán en científicos de datos mucho más eficientes y productivos.

Tres principios generales que motivan lo que aprendemos aquí son: 1) ser sistemáticos al organizar sus sistemas de archivos, 2) automatizar cuando posible y 3) minimizar el uso del mouse. A medida que se vuelvan más competentes en la codificación, encontrarán que: 1) querrán minimizar el tiempo que pasan recordando lo que nombraron un archivo o dónde lo colocaron, 2) si se encuentran repitiendo la misma tarea una y otra vez, probablemente hay una forma de automatizar y 3) cada vez que sus dedos abandonan el teclado, pierden productividad.

Un proyecto de análisis de datos no siempre es un set de datos y un *script*. Un desafío típico de análisis de datos puede involucrar varias partes, cada una con varios archivos de datos, incluyendo los archivos que contienen los *scripts* que usamos para analizar los datos. Mantener todo esto organizado puede ser retante. Aprenderemos a usar el *Unix shell* como herramienta para administrar archivos y directorios en sus sistemas informáticos. El uso de Unix les permitirá usar el teclado, en lugar del mouse, al crear carpetas (*folders* en inglés), moverse de un directorio a otro, además de mover, eliminar o cambiar el nombre de archivos. También ofrecemos sugerencias específicas sobre cómo mantener el sistema de archivos organizado.

El proceso de análisis de datos también es iterativo y adaptativo. Como resultado, estamos constantemente editando nuestros *scripts* e informes. En este capítulo, les presentamos el sistema de control de versiones *Git*, que es una herramienta poderosa para darle seguimiento a estos cambios. También les presentamos GitHub<sup>1</sup>, un servicio que les permite alojar y compartir su código. Les demostraremos cómo pueden utilizar este servicio para facilitar las colaboraciones. Recuerden que otro beneficio positivo de usar GitHub es que pueden mostrar fácilmente su trabajo a posibles empleadores.

Finalmente, aprendemos a escribir informes en R Markdown, lo que les permite incorporar texto y código en un solo documento. Vamos a demostrar cómo, utilizando el paquete *knitr*, podemos escribir informes reproducibles y estéticamente agradables ejecutando el análisis y generando el informe simultáneamente.

Arreglaremos todo esto utilizando el entorno de escritorio integrado RStudio<sup>2</sup>. A lo largo del capítulo vamos a construir un ejemplo usando los asesinatos con armas de fuego en

<sup>1</sup><http://github.com>

<sup>2</sup><https://www.rstudio.com/>

EE.UU. El proyecto final, que incluye varios archivos y carpetas, se puede ver aquí: <https://github.com/rairizarry/murders>. Tengan en cuenta que uno de los archivos de ese proyecto es el informe final: <https://github.com/rairizarry/murders/blob/master/report.md>.

# 36

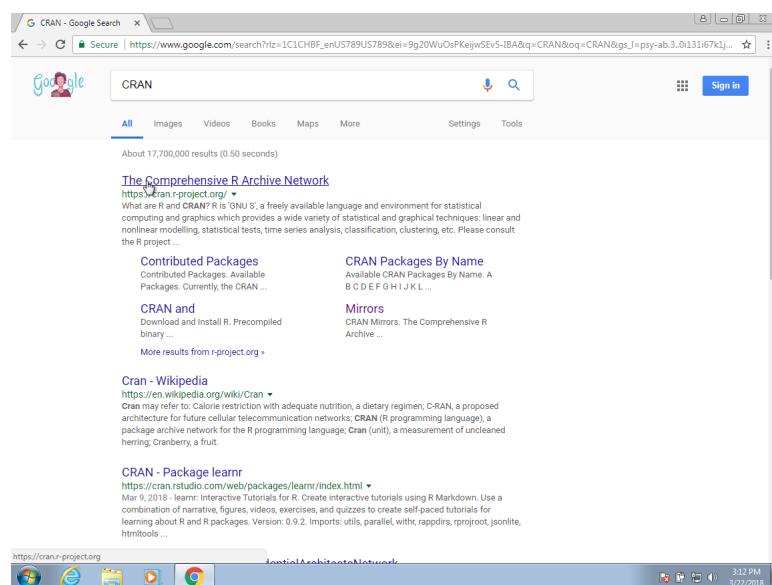
## Instalación de R y RStudio

Las instrucciones a continuación incluyen capturas de pantalla del proceso de instalación. En este libro mostramos ejemplos usando el navegador Chrome (pero pueden usar otros navegadores) que pueden descargar e instalar libremente aquí: <https://www.google.com/chrome/>.

### 36.1 Instalando R

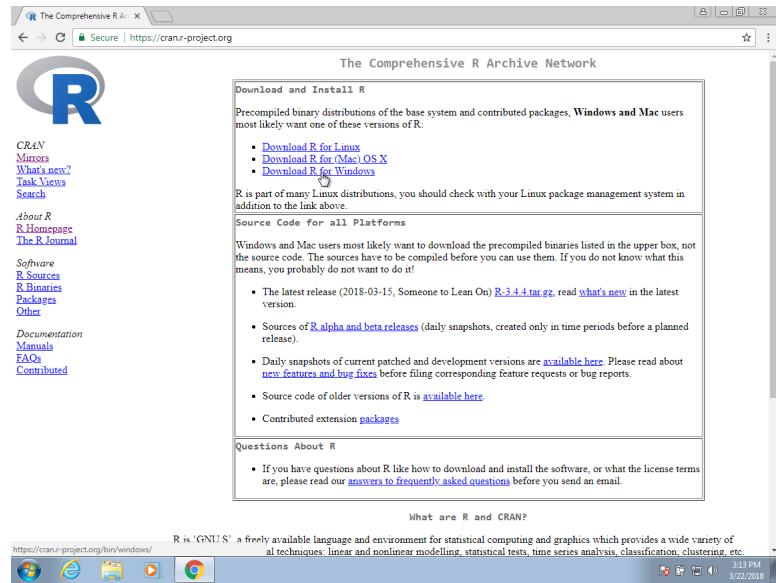
RStudio es un entorno de desarrollo integrado (IDE por sus siglas en inglés), pero no es R, ni incluye R cuando se descarga e instala. Por lo tanto, para usar RStudio, primero necesitamos instalar R.

1. Pueden descargar R desde la Red Integral de Archivo R (CRAN)<sup>1</sup>. Busquen CRAN en su navegador:



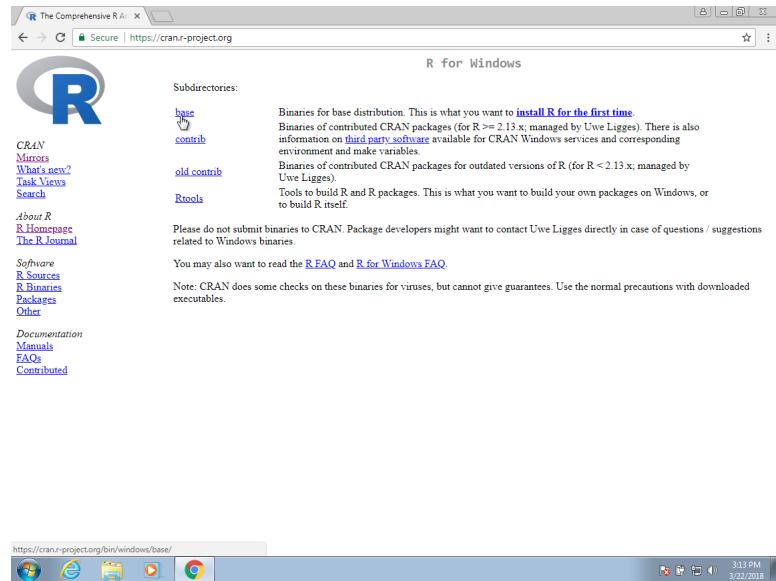
2. Una vez en la página de CRAN, seleccionen la versión para su sistema operativo: Linux, Mac OS X o Windows.

<sup>1</sup><https://cran.r-project.org/>

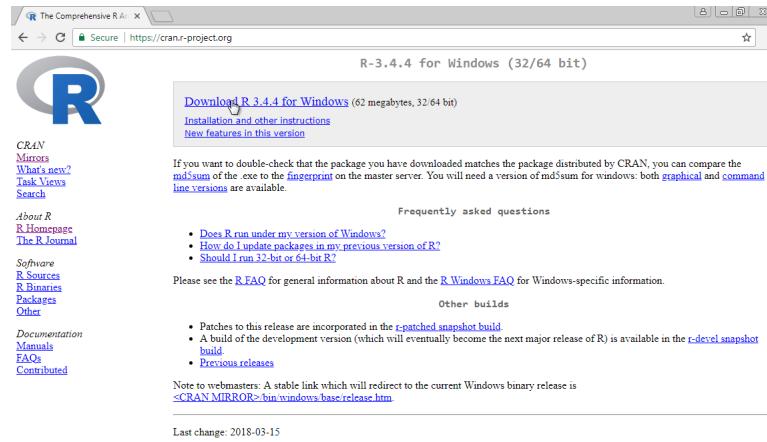


Aquí mostramos capturas de pantalla para Windows, pero el proceso es similar para las otras plataformas. Cuando difieren, también mostraremos capturas de pantalla para Mac OS X.

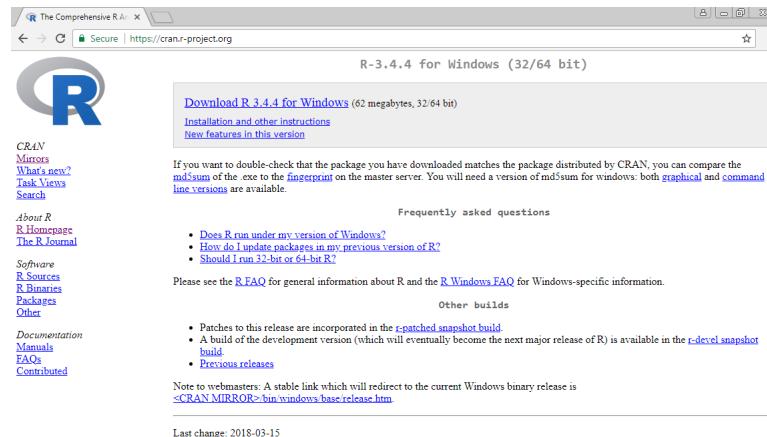
- Una vez en la página de descarga de CRAN, tendrán varias opciones. Quieren instalar el subdirectorio *base*. Esto instala los paquetes básicos que necesitarán para comenzar. Más adelante, aprenderemos cómo instalar otros paquetes necesarios desde R, en lugar de hacerlo desde esta página web.



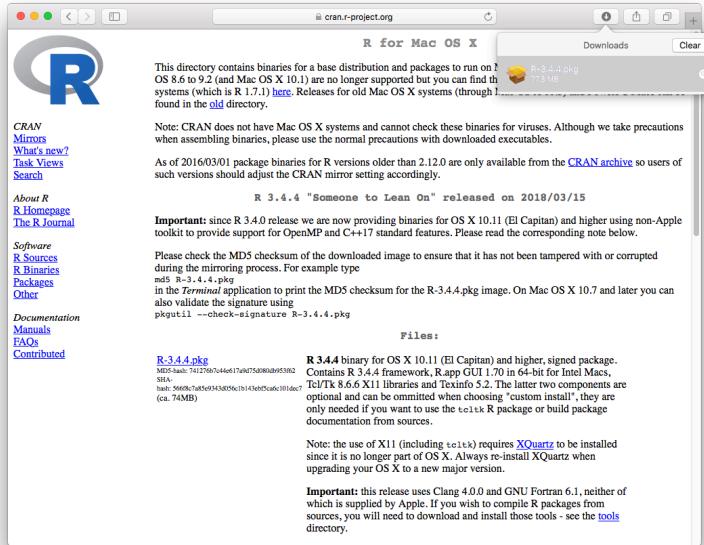
- Hagan clic en el enlace de la última versión para iniciar la descarga.



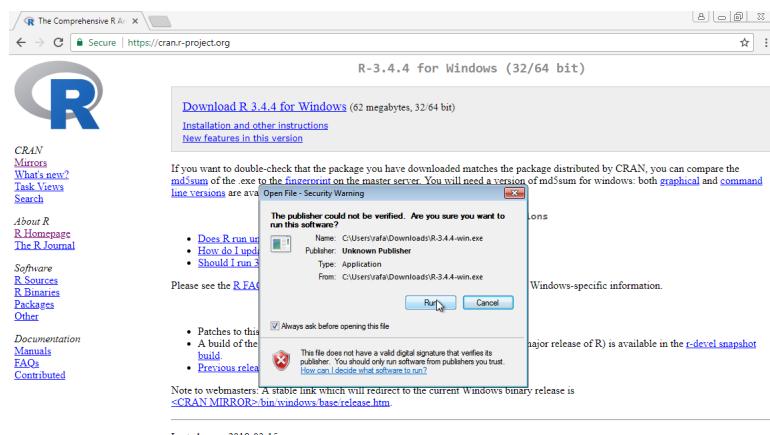
- Si están utilizando Chrome, en la parte inferior de su navegador deberían ver una pestaña que les muestra el progreso de la descarga. Una vez que se descargue el archivo instalador, pueden hacer clic en esa pestaña para comenzar el proceso de instalación. Otros navegadores pueden ser diferentes, por lo que tendrán que encontrar dónde almacenan los archivos descargados y hacer clic en ellos para comenzar el proceso.



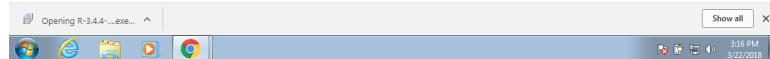
Si usan Safari en la Mac, pueden acceder el archivo descargado a través del botón *Downloads*.



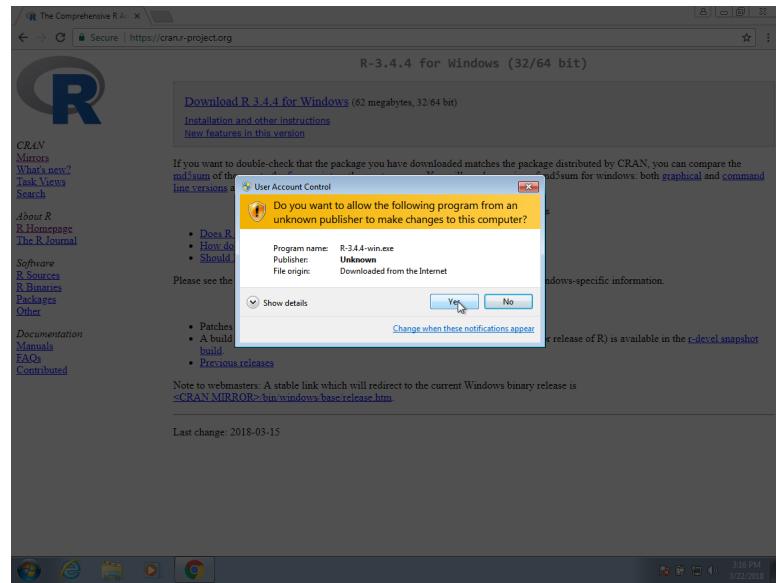
6. Ahora pueden hacer clic en diferentes opciones para finalizar la instalación. Le recomendamos que seleccionen todas las opciones predeterminadas.



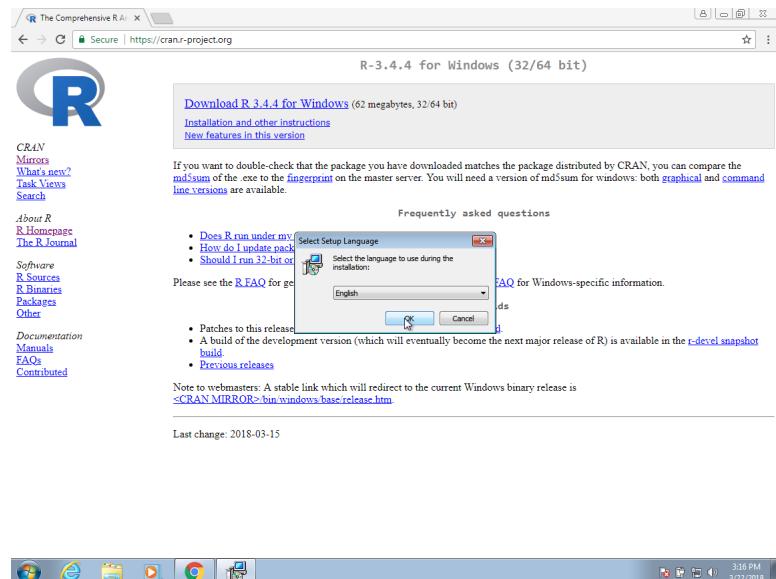
Last change: 2018-03-15



Seleccionen el valor predeterminado incluso cuando reciban una advertencia ominosa.

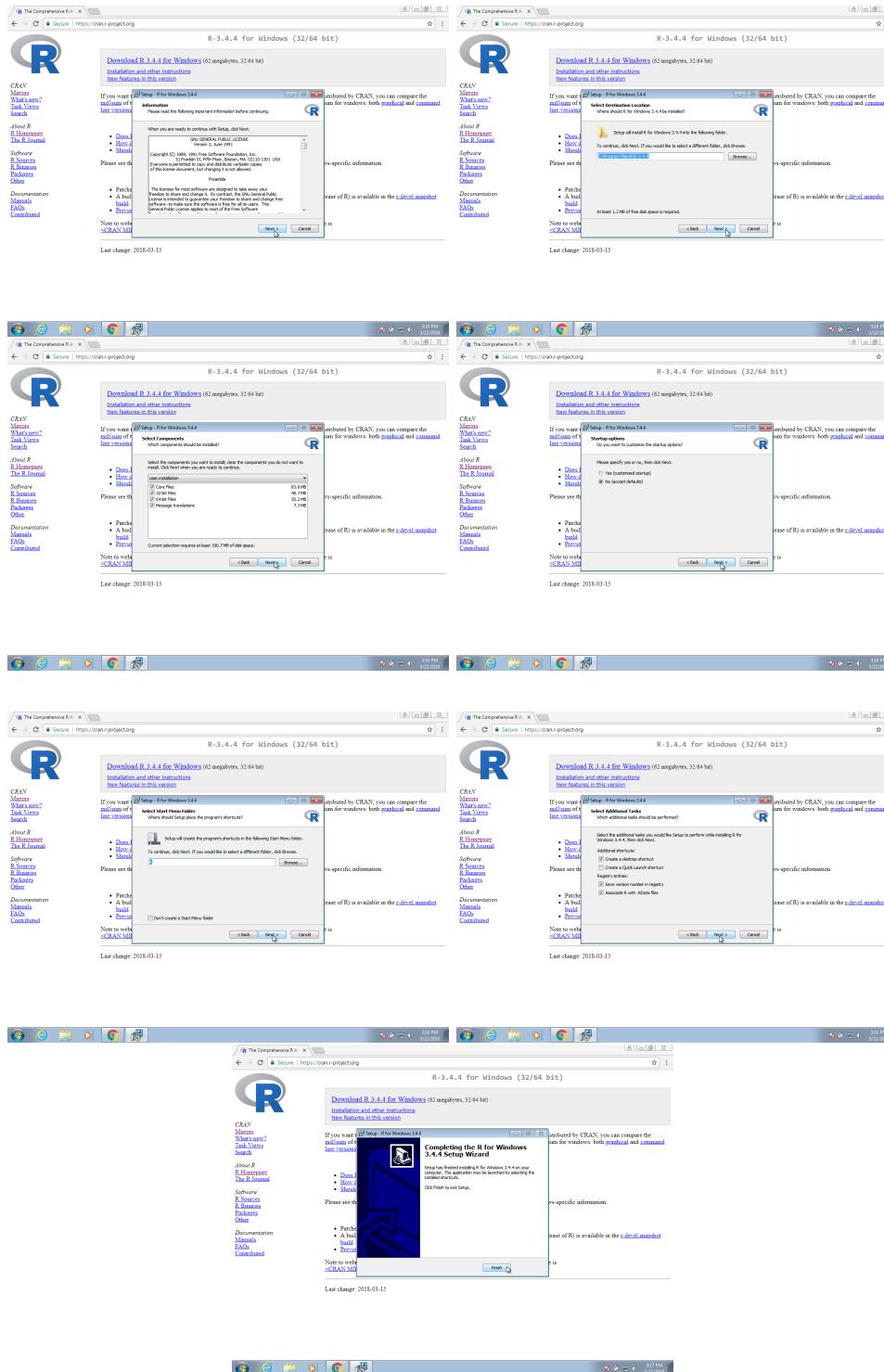


Al seleccionar el idioma, tengan en cuenta que será más fácil seguir este libro si seleccionan inglés.

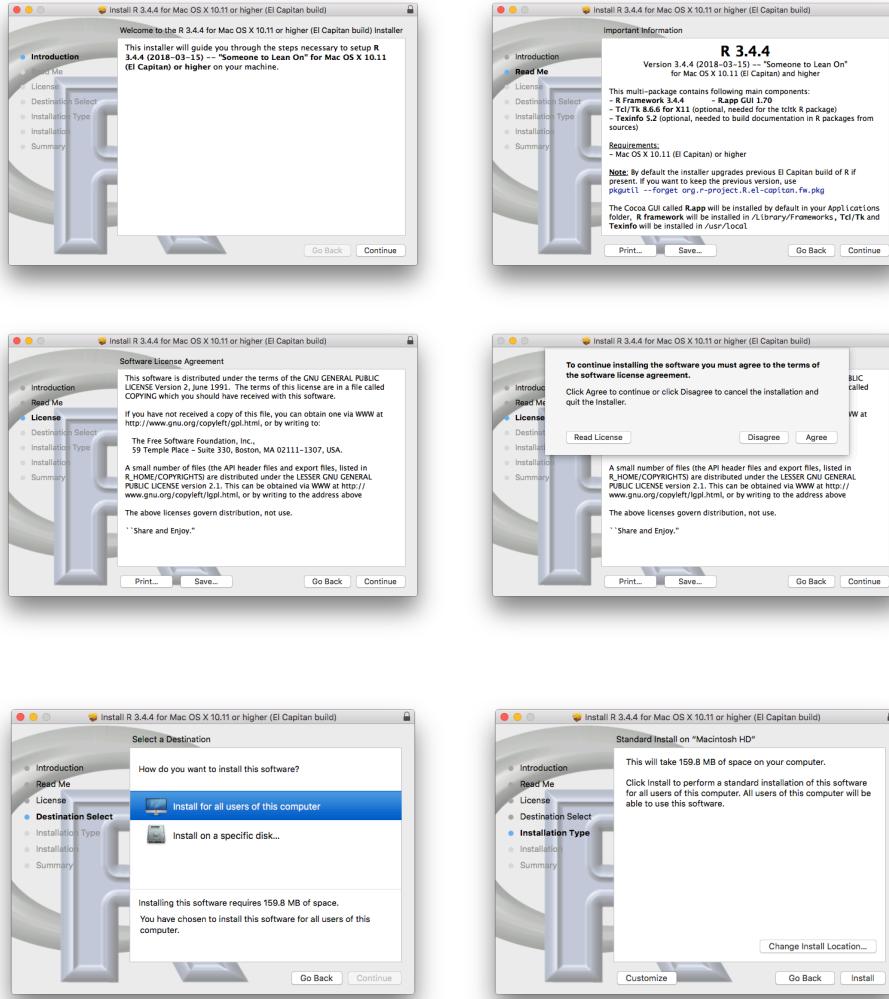


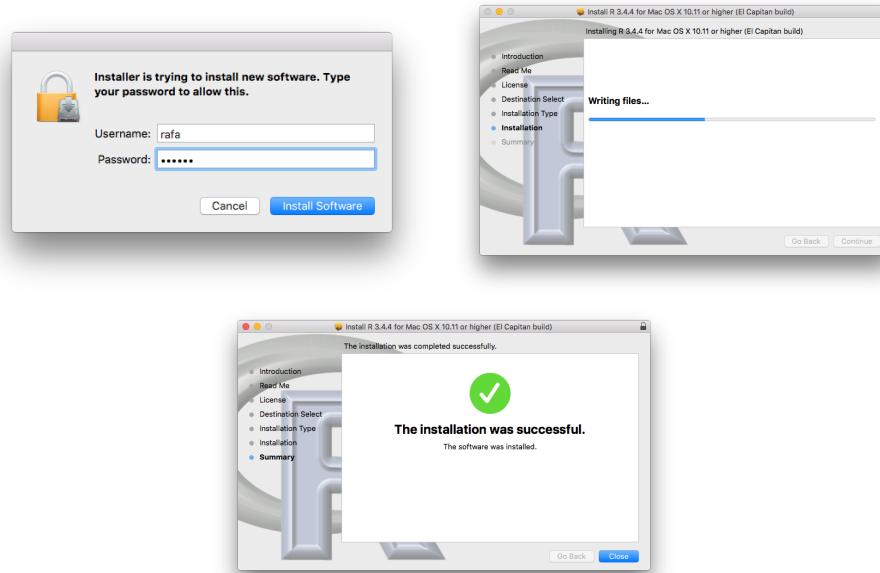
Continúen seleccionando todos los valores predeterminados:

## 36 Instalación de R y RStudio



En una Mac se ve diferente, pero aún así deben aceptar los valores predeterminados:





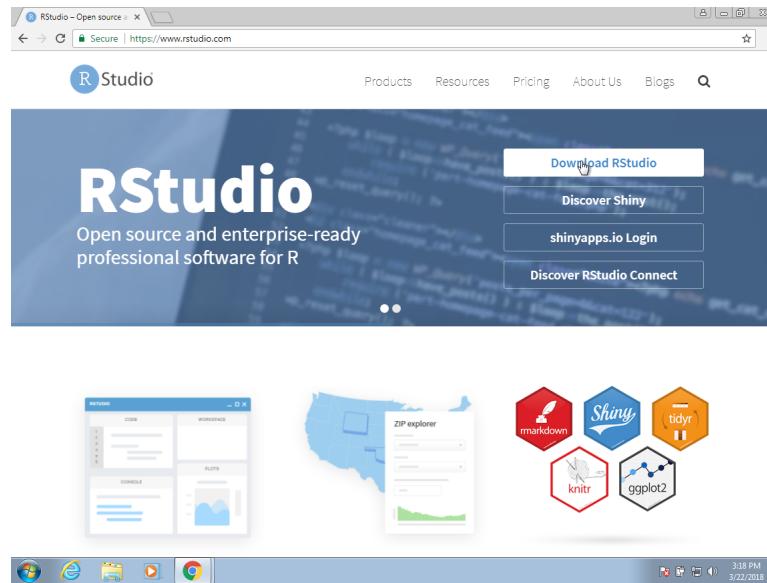
¡Felicitaciones! Han instalado R.

## 36.2 Instalación de RStudio

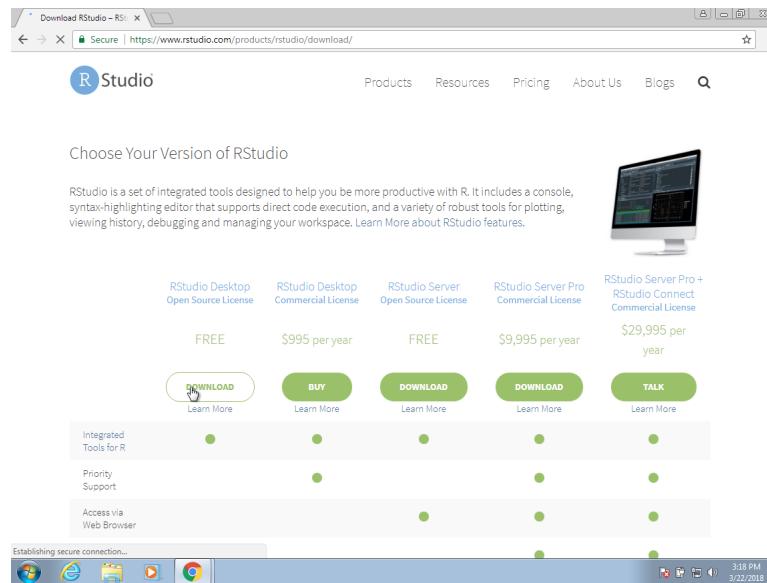
- Pueden comenzar buscando RStudio en su navegador:

The screenshot shows a Google search results page for the query 'rstudio'. The top result is the official RStudio website (<https://www.rstudio.com/>). Other results include links to RStudio Desktop, Server, and Connect, as well as news articles and tweets about RStudio. The search results are displayed in a standard browser interface with a sidebar for filters like 'All', 'Images', 'Videos', etc.

2. Deben encontrar el sitio web de RStudio como se muestra arriba. Una vez allí, hagan clic en *Download RStudio*.



3. Esto les dará varias opciones. Para lo que hacemos en este libro, es más que suficiente usar la versión gratuita *RStudio Desktop*:



4. Una vez que seleccionen esta opción, el programa los llevará a una página con las opciones del sistema operativo. Hagan clic en el enlace que muestra su sistema operativo.

**Installers**

- RStudio 1.1.442 - Windows Vista 7.196
- RStudio 1.1.442 - Mac OS X 10.6+ (64-bit)
- RStudio 1.1.442 - Ubuntu 12.04.12.10 (Debian 8) (32-bit)
- RStudio 1.1.442 - Ubuntu 12.04.12.10 (Debian 8) (64-bit)
- RStudio 1.1.442 - Ubuntu 16.04+ (Debian 9+) (32-bit)
- RStudio 1.1.442 - Ubuntu 16.04+ (Debian 9+) (64-bit)
- RStudio 1.1.442 - Fedora 19+ (Red Hat 7+ / openSUSE 13.1+ ) (32-bit)
- RStudio 1.1.442 - Fedora 19+ (Red Hat 7+ / openSUSE 13.1+ ) (64-bit)

**Zip/Tarballs**

- RStudio 1.1.442 - Windows Vista 7.196
- RStudio 1.1.442 - Ubuntu 12.04.15.10 (Debian 8) (32-bit)
- RStudio 1.1.442 - Ubuntu 12.04.15.10 (Debian 8) (64-bit)
- RStudio 1.1.442 - Fedora 19+ (Red Hat 7+ / openSUSE 13.1+ ) (32-bit)
- RStudio 1.1.442 - Fedora 19+ (Red Hat 7+ / openSUSE 13.1+ ) (64-bit)

**Source Code**

A tarball containing source code for RStudio v1.1.442 can be downloaded from here



- Una vez que hayan descargado el archivo de instalación, hagan clic en el archivo descargado para iniciar el proceso de instalación:

**Installers**

- RStudio 1.1.442 - Windows Vista 7.19
- RStudio 1.1.442 - Mac OS X 10.6+ (64-bit)
- RStudio 1.1.442 - Ubuntu 12.04.15.10 (Debian 8) (32-bit)
- RStudio 1.1.442 - Ubuntu 12.04.15.10 (Debian 8) (64-bit)
- RStudio 1.1.442 - Ubuntu 16.04+ (Debian 9+) (32-bit)
- RStudio 1.1.442 - Ubuntu 16.04+ (Debian 9+) (64-bit)
- RStudio 1.1.442 - Fedora 19+ (Red Hat 7+ / openSUSE 13.1+ ) (32-bit)
- RStudio 1.1.442 - Fedora 19+ (Red Hat 7+ / openSUSE 13.1+ ) (64-bit)

**Zip/Tarballs**

- RStudio 1.1.442 - Windows Vista 7.19
- RStudio 1.1.442 - Ubuntu 12.04.15.10 (Debian 8) (32-bit)
- RStudio 1.1.442 - Ubuntu 12.04.15.10 (Debian 8) (64-bit)
- RStudio 1.1.442 - Fedora 19+ (Red Hat 7+ / openSUSE 13.1+ ) (32-bit)
- RStudio 1.1.442 - Fedora 19+ (Red Hat 7+ / openSUSE 13.1+ ) (64-bit)

**Source Code**

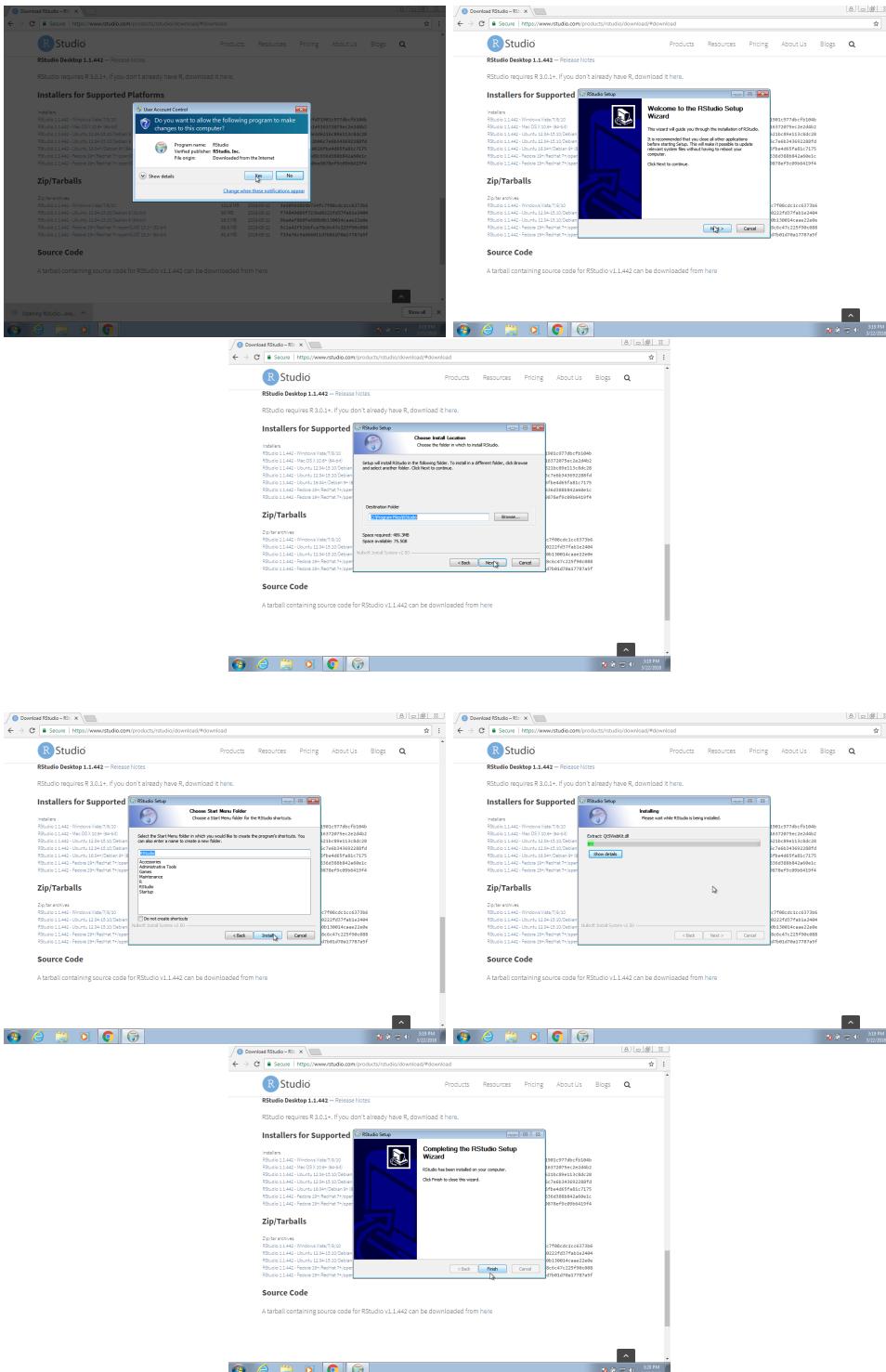
A tarball containing source code for RStudio v1.1.442 can be downloaded from here



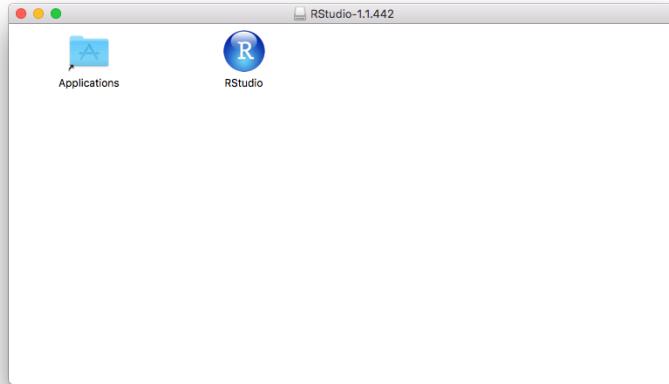
- Recomendamos hacer clic en *yes* en todos los valores predeterminados.

## 36.2 Instalación de RStudio

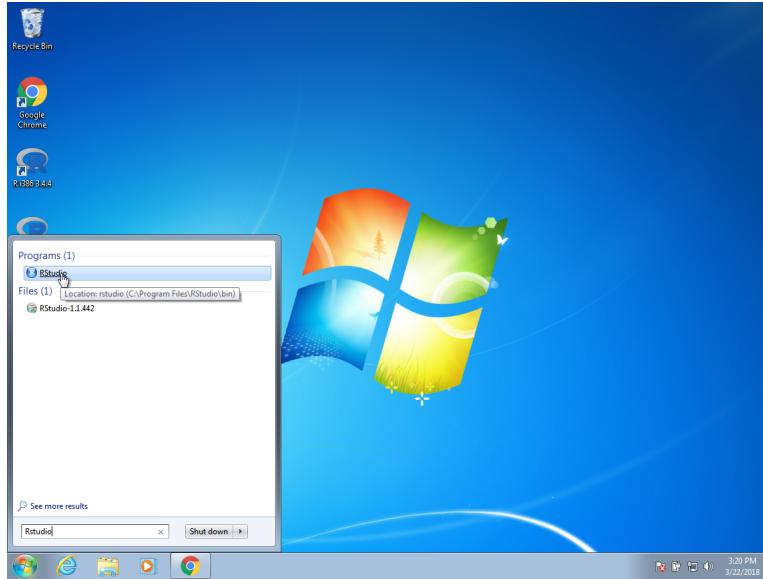
723



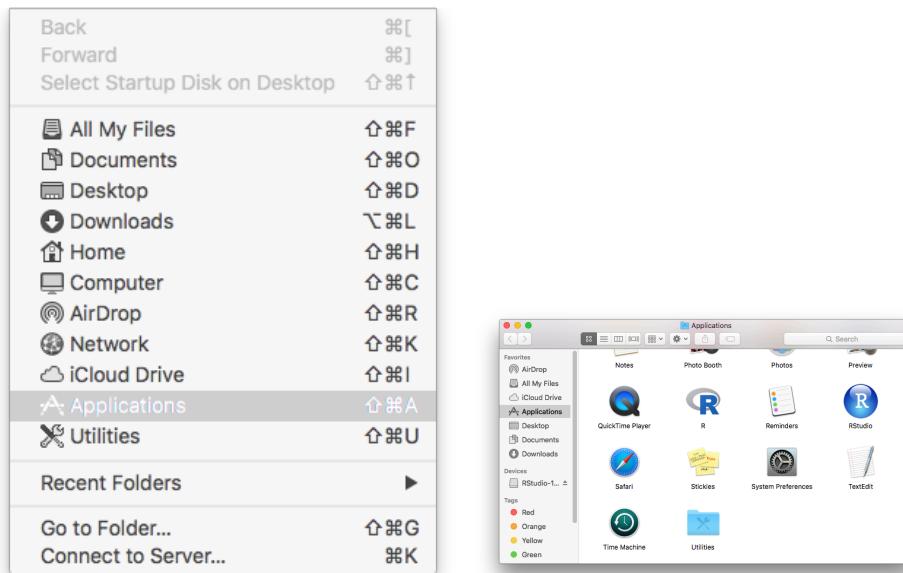
En la Mac, hay menos clics. Básicamente, arrastran y sueltan el ícono RStudio en el ícono de la carpeta *Applications* como ven aquí:



¡Felicitaciones! Han instalado RStudio. Ahora pueden comenzar como lo hacen en cualquier otro programa en su computadora. En Windows, pueden abrir RStudio desde el menú *Start*. Si RStudio no aparece, pueden buscarlo:



En la Mac, estará en la carpeta *Applications*:



**Consejo profesional para la Mac:** Para evitar usar el mouse para abrir RStudio, presionen comando+barra espaciadora para abrir *Spotlight Search* y escriban RStudio en esa barra de búsqueda. Luego presionen *enter*.



# 37

---

## Accediendo al terminal e instalando Git

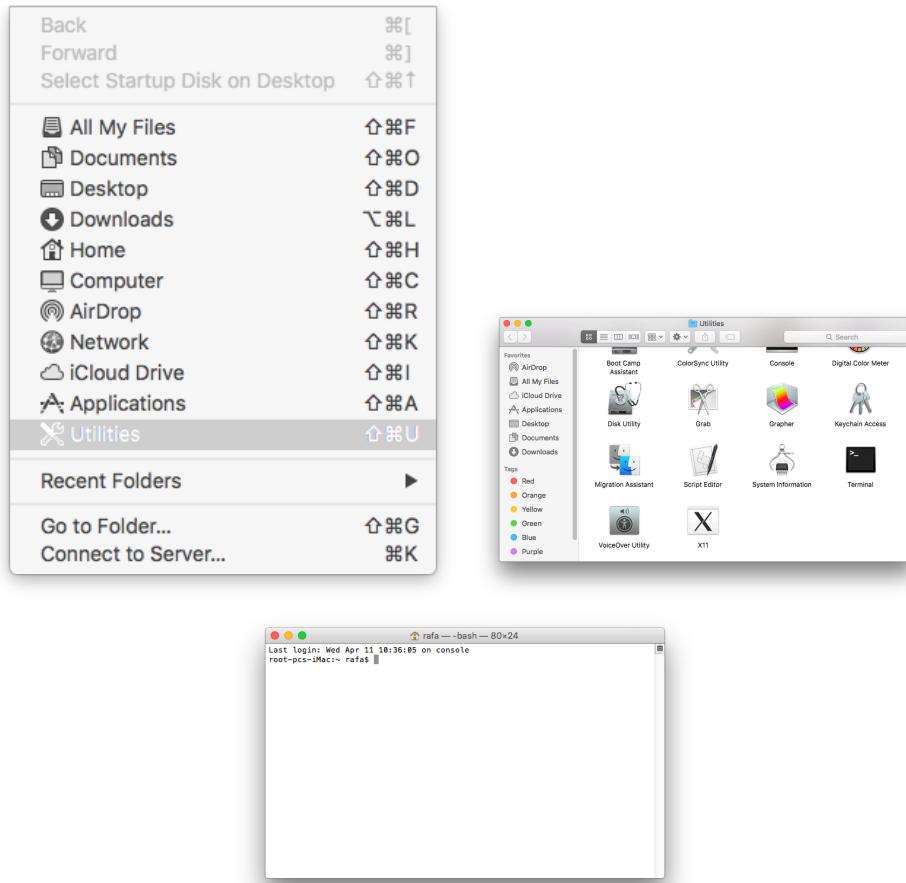
---

Antes de comenzar, debemos asegurarnos de que tengan acceso a un *terminal* y que Git esté instalado. El terminal está integrado en los sistemas Mac y Linux, pero los usuarios de Windows deberán instalar un *emulador*. Hay muchas opciones de emulador disponibles, pero aquí mostramos cómo instalar Git Bash porque se puede hacer como parte de la instalación de Windows Git. Debido a las diferencias entre Mac y Windows, las secciones de este capítulo se dividen tomando esto en cuenta.

---

### 37.1 Accediendo al terminal en una Mac

En el Capítulo 38, describimos cómo el terminal es nuestra ventana al mundo de Unix. En una Mac, pueden acceder a un terminal abriendo la aplicación en la carpeta *Utilities*:



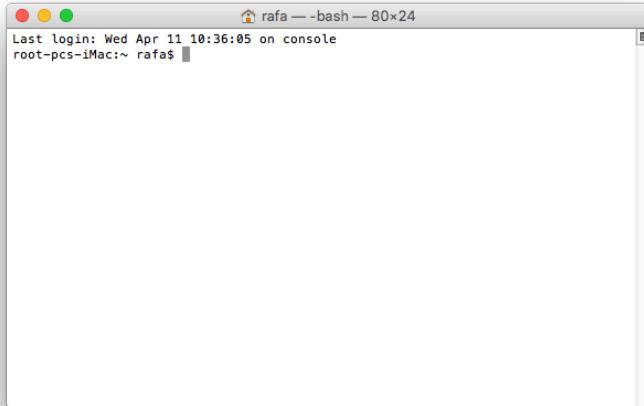
También pueden usar la aplicación *Spotlight* en la Mac presionando comando-barra espaciadora y entonces escribiendo *Terminal*.

Otra forma de acceder al terminal es desde RStudio. En el panel *Console*, deberían ver una pestaña *Terminal*. Si hacen clic en esa pestaña, abrirá una ventana de terminal.

## 37.2 Instalando Git en la Mac

**Advertencia:** Las instrucciones en esta subsección no son para usuarios de Windows.

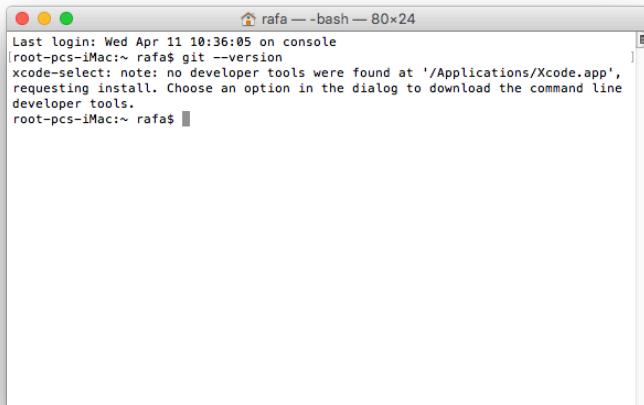
1. Comiencen abriendo un terminal como se describe en la sección anterior.
2. Una vez que inicien el terminal, verán una consola como esta:



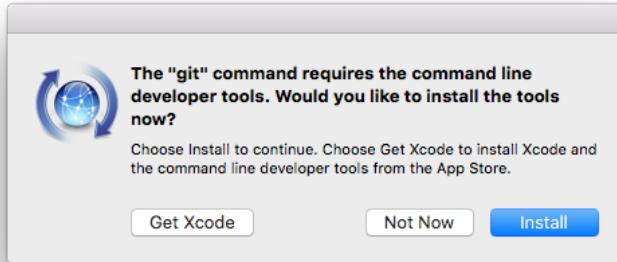
3. Es posible que ya hayan instalado Git. Una manera de verificarlo es preguntando por la versión escribiendo:

```
git --version
```

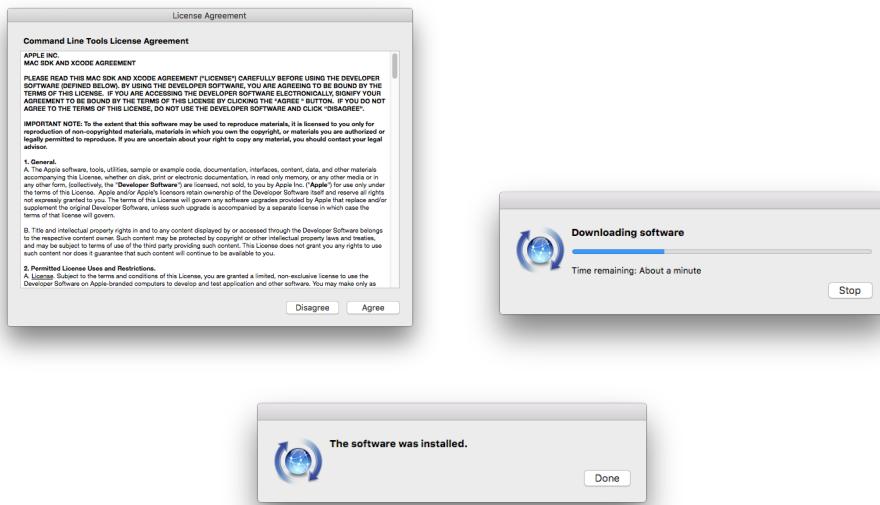
Si les devuelve un número de versión, Git ya está instalado. Si no, recibirán el siguiente mensaje:



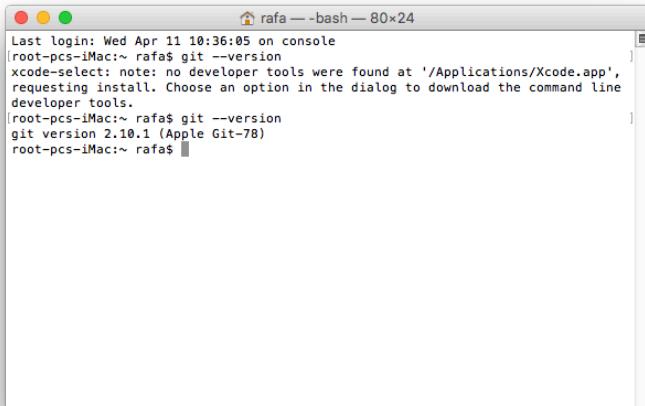
y se les preguntará si quieren instalarlo. Deben hacer clic en *Install*:



4. Esto los guiará por el proceso de instalación:



5. Una vez instalado, pueden verificar la versión nuevamente y deberían ver algo como esto:



```
Last login: Wed Apr 11 10:36:05 on console
[root-pcs-iMac:~ rafa$ git --version
xcode-select: note: no developer tools were found at '/Applications/Xcode.app',
requesting install. Choose an option in the dialog to download the command line
developer tools.
[root-pcs-iMac:~ rafa$ git --version
git version 2.10.1 (Apple Git-78)
root-pcs-iMac:~ rafa$]
```

Felicidades. Han instalado Git en sus Macs.

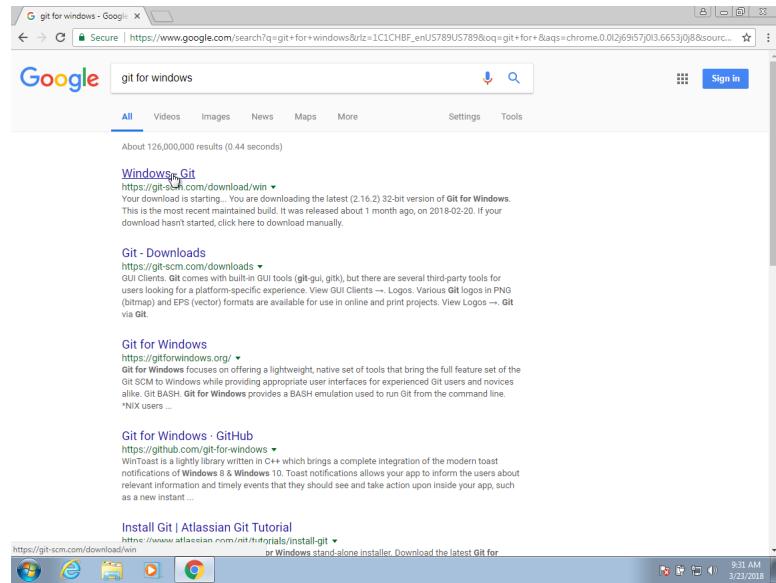
---

### 37.3 Instalación de Git y Git Bash en Windows

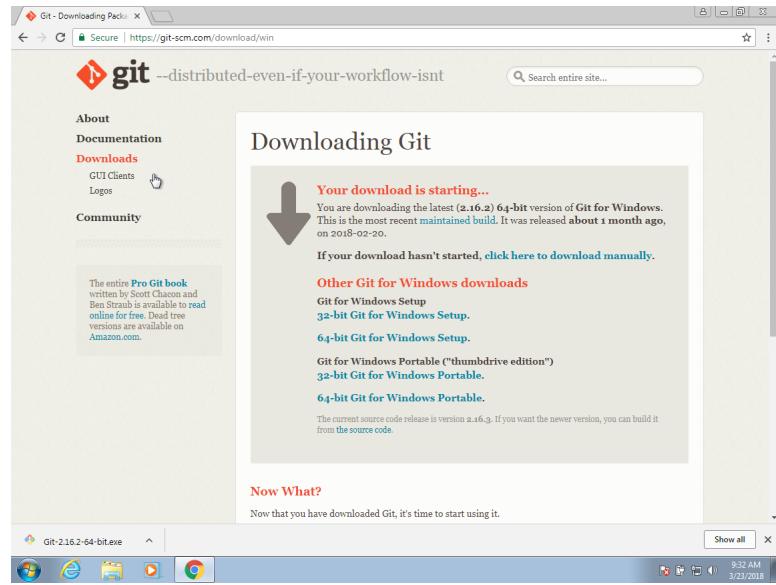
**Advertencia: Las instrucciones en esta subsección no son para usuarios de Mac.**

Hay varios programas de software que les permitirán ejecutar comandos de Unix en Windows. Usaremos Git Bash ya que interactúa con RStudio y se instala automáticamente cuando instalamos Git for Windows.

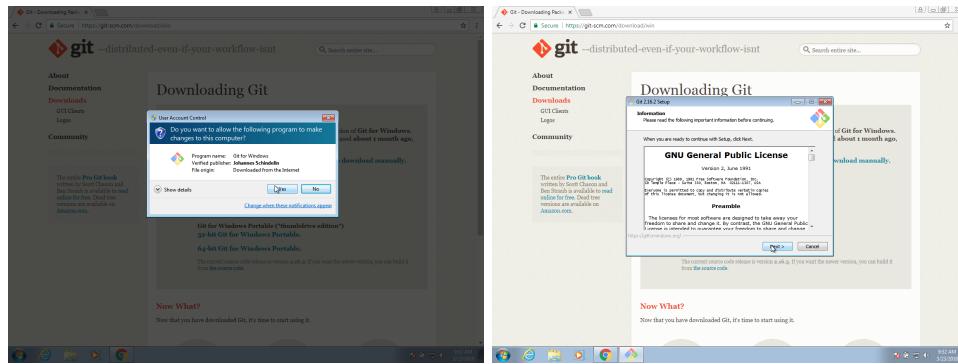
1. Comiencen buscando *Git for Windows* en su navegador y haciendo clic en el enlace de [git-scm.com](http://git-scm.com).



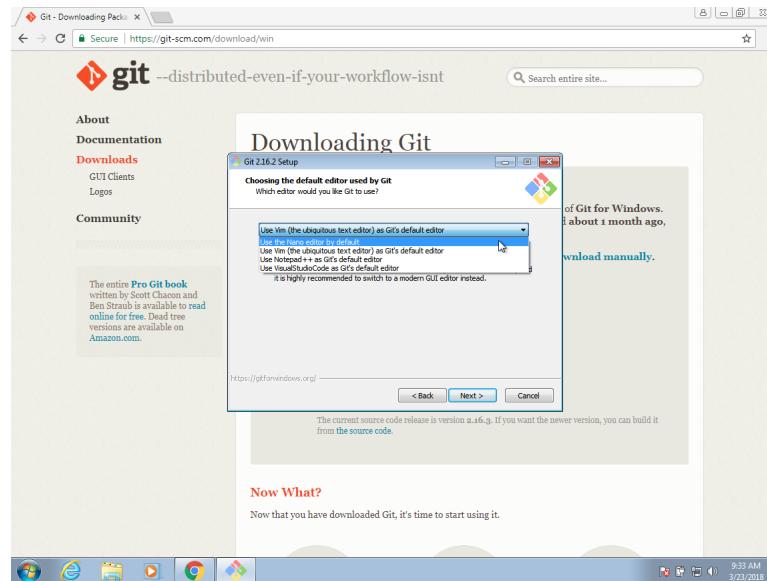
2. Esto los llevará a la página *Download Git* donde pueden descargar el *maintained build* mas reciente:



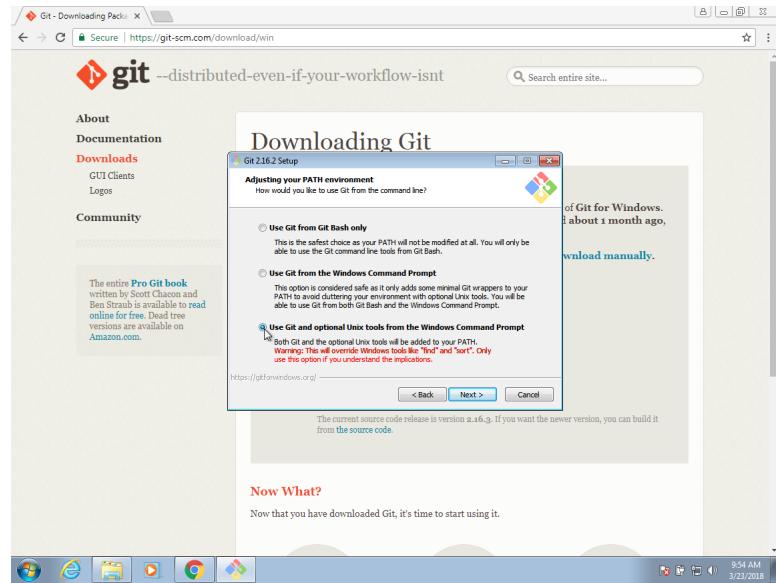
3. Entonces pueden aceptar ejecutar el instalador y aceptar la licencia:



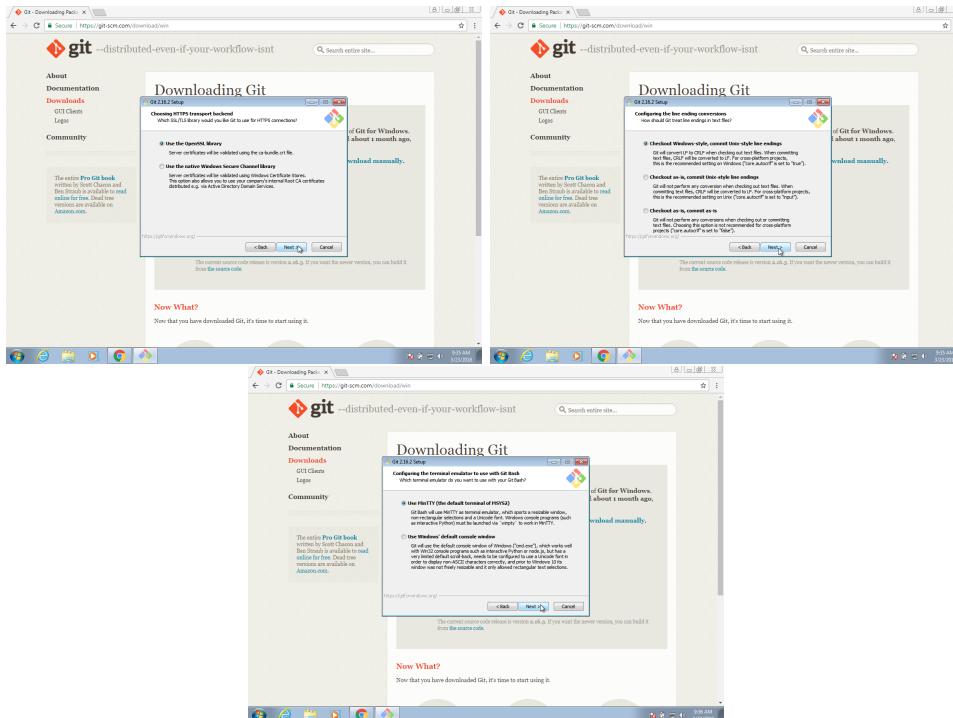
4. En uno de los pasos de la instalación, se les pedirá que elijan el editor predeterminado para Git. A menos que ya sean usuarios de *vi* o *vim*, no les recomendamos que seleccionen *vim*, que puede ser el predeterminado. Si no reconocen un editor con el que están familiarizados entre las opciones, les recomendamos que seleccionen *nano* como su editor predeterminado para Git, ya que es el más fácil de aprender:

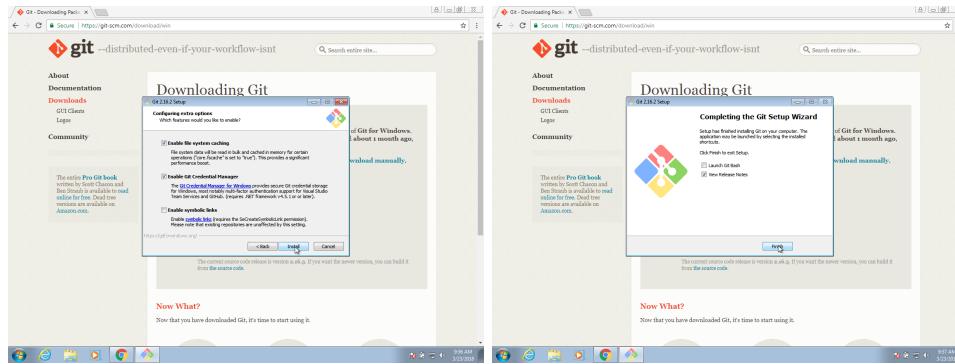


5. La siguiente decisión de instalación es una **muy importante**. Este proceso de instalación instala Git Bash. Recomendamos que seleccionen *Git and optional Unix tools from the Windows Command* ya que esto les permitirá aprender Unix desde RStudio. Sin embargo, si hacen esto, **algunos comandos que se ejecutan en su línea de comandos de Windows dejarán de funcionar**. Si no usan su línea de comandos de Windows, entonces esto no será un problema. Además, la mayoría, si no todas, de estas líneas de comando de Windows tienen un equivalente de Unix que podrán usar.



6. Ahora pueden continuar seleccionando las opciones predeterminadas.



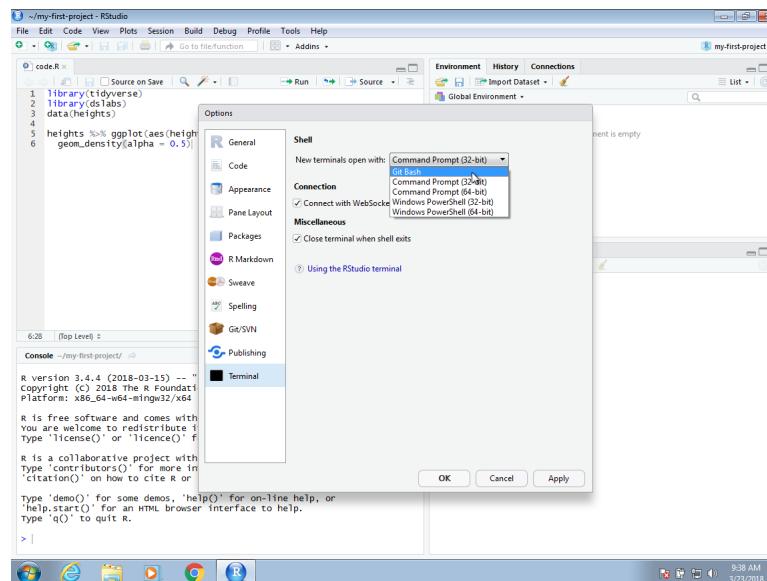


Ya han instalado Git en Windows.

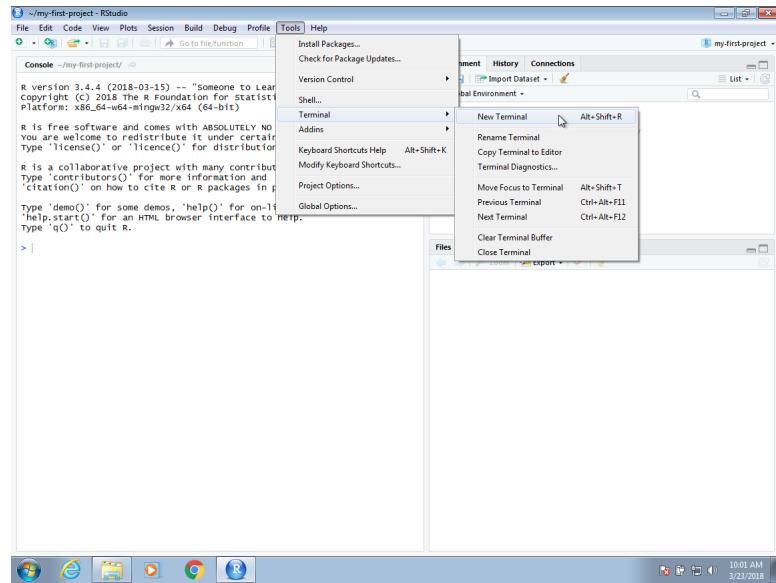
## 37.4 Accediendo el terminal en Windows

Ahora que Git Bash está instalado, podemos acceder al terminal a través de RStudio o abriendo Git Bash directamente.

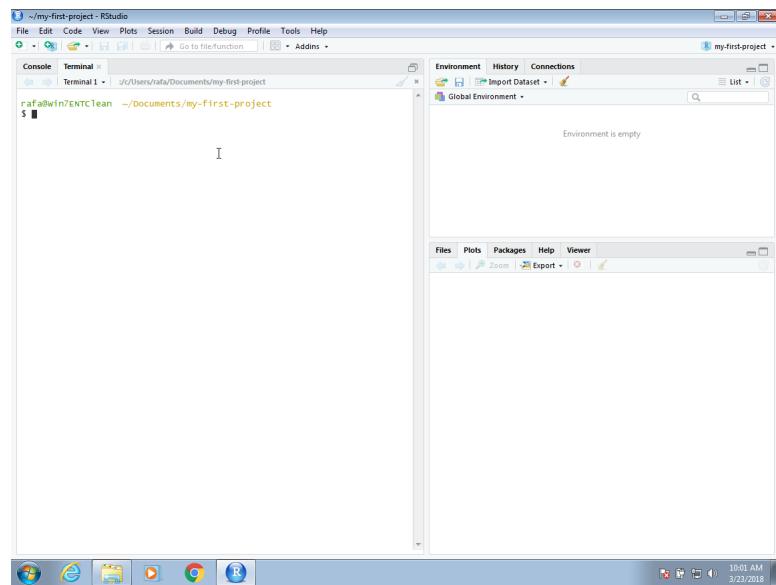
Para acceder al terminal a través de RStudio, necesitamos cambiar una preferencia para que Git Bash se convierta en el *shell* predeterminado de Unix en RStudio. En RStudio, vayan a *Preferences* (en el menú de *File*), luego seleccionen *Terminal* y entonces seleccionen *Git Bash*:



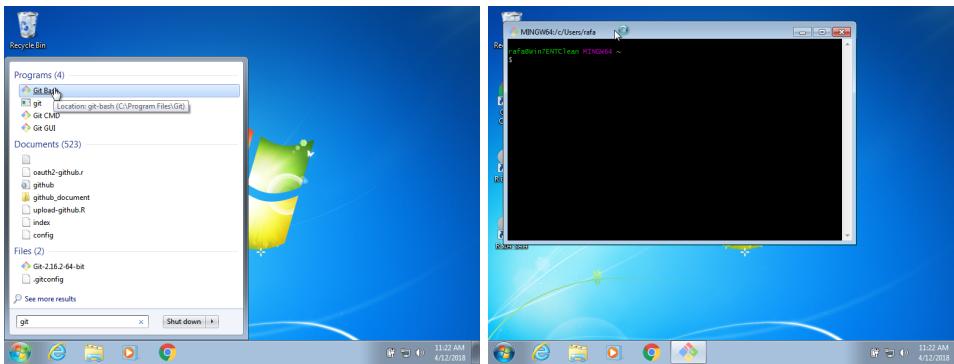
Para comprobar que están utilizando Git Bash en RStudio, pueden abrir *New Terminal* en RStudio:



Debería verse más o menos así:



A menudo queremos acceder al terminal, pero no necesitamos RStudio. Pueden hacer esto ejecutando el programa Git Bash directamente:





## 38

---

### Organizando con Unix

---

Unix es el sistema operativo de elección en el campo de ciencia de datos. Le presentaremos la forma de pensar de Unix utilizando un ejemplo: cómo mantener organizado un proyecto de análisis de datos. Aprenderemos algunos de los comandos más utilizados a lo largo del camino. Sin embargo, no entraremos en detalles aquí. Les recomendamos que aprendan más, especialmente cuando se encuentran usando demasiado el mouse o realizando una tarea repetitiva con frecuencia. En esos casos, probablemente hay una forma más eficiente de hacerlo en Unix. Aquí tenemos algunos cursos básicos para comenzar:

- <https://www.codecademy.com/learn/learn-the-command-line>
- <https://www.edx.org/course/introduction-linux-linuxfoundationx-lfs101x-1>
- <https://www.coursera.org/learn/unix>

También hay muchos libros de referencia<sup>1</sup>. Por ejemplo, *Bite Size Linux*<sup>2</sup> y *Bite Size Command Line*<sup>3</sup> son particularmente claros, concisos y completos.

Cuando busquen recursos de Unix, recuerden que otros términos utilizados para describir lo que aprenderemos aquí son *Linux*, el *shell* y la *línea de comando*. Básicamente, lo que estamos aprendiendo es una serie de comandos y una forma de pensar que facilita la organización de los archivos sin usar el mouse.

Como motivación, vamos a comenzar construyendo un directorio utilizando herramientas de Unix y RStudio.

---

#### 38.1 Convención de nomenclatura

Antes de comenzar a organizar proyectos con Unix, deben elegir una convención que usarán para sistemáticamente nombrar sus archivos y directorios. Esto les ayudará a encontrar archivos y saber qué hay en ellos.

En general, quieren nombrar sus archivos de una manera que se relaciona con sus contenidos y que especifica cómo se relacionan con otros archivos. El *Smithsonian Data Management Best Practices*<sup>4</sup> ofrece “cinco preceptos para el nombramiento y la organización de archivos”. Estos son:

- Tener un nombre distintivo, legible por humanos que indique el contenido.

<sup>1</sup><https://www.quora.com/Which-are-the-best-Unix-Linux-reference-books>

<sup>2</sup><https://gumroad.com/l/bite-size-linux>

<sup>3</sup><https://jvns.ca/blog/2018/08/05/new-zine--bite-size-command-line/>

<sup>4</sup><https://library.si.edu/sites/default/files/tutorial/pdf/filenamingorganizing20180227.pdf>

- Seguir un patrón consistente que sea conveniente para la automatización.
- Organizar los archivos en directorios (cuando sea necesario) que siguen un patrón consistente.
- Evitar la repetición de elementos semánticos en los nombres de los archivos y directorios.
- Tener una extensión de archivo que coincida con el formato del archivo (¡sin cambiar las extensiones!)

Para recomendaciones específicas, pueden consultar *The Tidyverse Style Guide*<sup>5</sup>.

## 38.2 El terminal

En lugar de hacer clic, arrastrar y soltar para organizar nuestros archivos y carpetas, escribiremos comandos de Unix en el terminal. La forma en que hacemos esto es similar a cómo escribimos comandos en la consola R, pero en lugar de generar gráficos y resúmenes estadísticos, organizaremos archivos en nuestro sistema.

Necesitarán acceso a un terminal<sup>6</sup>. Una vez que tengan un terminal abierto, pueden comenzar a escribir comandos. Deberían ver un cursor intermitente en el lugar donde se muestra lo que escriben. Esta posición se llama la *línea de comando* (*command line* en inglés). Una vez que escriban algo y presionen *enter* en Windows o *return* en la Mac, Unix intentará ejecutar este comando. Si quieren intentar un ejemplo, escriban lo siguiente en su línea de comando:

```
echo "hello world"
```

El comando `echo` es parecido a `cat` en R. Al ejecutar esta línea deberían ver `hello world` y entonces vuelven a la línea de comando.

Tengan en cuenta que no pueden usar el mouse para moverse en el terminal. Tienen que usar el teclado. Para volver a un comando que escribieron anteriormente, pueden usar la flecha hacia arriba.

Noten que anteriormente incluimos un fragmento de código que muestra los comandos de Unix de la misma manera que anteriormente mostramos los comandos R. Nos aseguraremos de distinguir cuándo el comando está destinado a R y cuándo está destinado a Unix.

## 38.3 El sistema de archivos

Nos referimos a todos los archivos, carpetas y programas en su computadora como el *sistema de archivos* (*filesystem* en inglés). Recuerden que las carpetas y los programas también son archivos, pero este es un tecnicismo en que rara vez pensamos e ignoraremos en este libro. Nos enfocaremos en los archivos y las carpetas por el momento y discutiremos los programas, o *executables*, en una sección posterior.

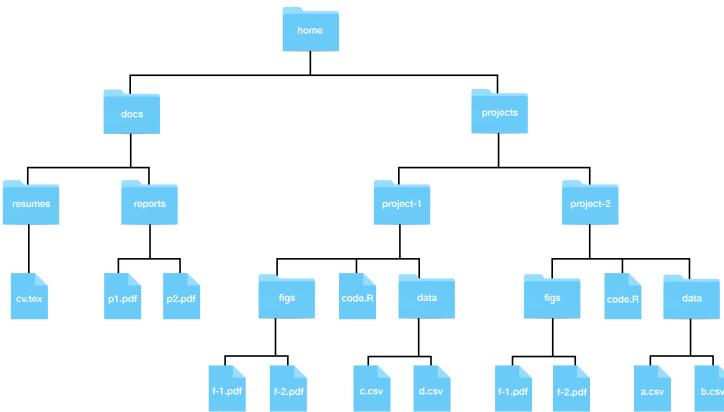
<sup>5</sup><https://style.tidyverse.org/>

<sup>6</sup><https://rafalab.github.io/dsbook/accessing-the-terminal-and-installing-git.html>

### 38.3.1 Directorios y subdirectorios

El primer concepto que necesitan entender para convertirse en un usuario de Unix es cómo está organizado su sistema de archivos. Deberían considerarlo como una serie de carpetas anidadas, cada una con archivos, carpetas y ejecutables.

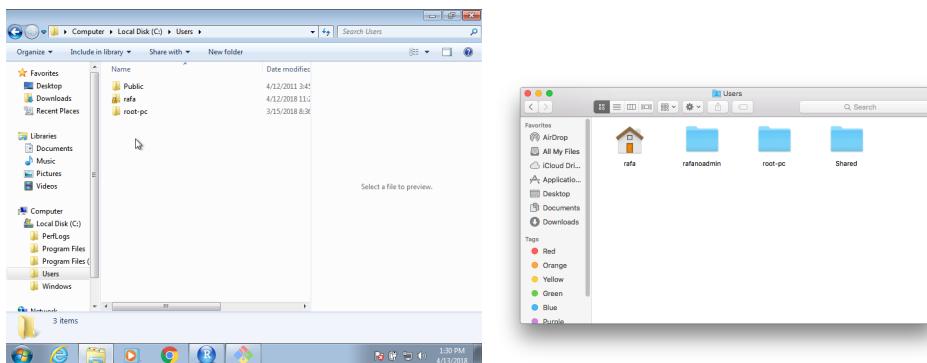
Aquí hay una representación visual de la estructura que estamos describiendo:



En Unix, nos referimos a las carpetas como *directorios*. Los directorios que están dentro de otros directorios a menudo se denominan *subdirectorios*. Entonces, por ejemplo, en la figura anterior, el directorio *docs* tiene dos subdirectorios: *reports* y *resumes*, y *docs* es un subdirectorio de *home*.

### 38.3.2 El directorio *home*

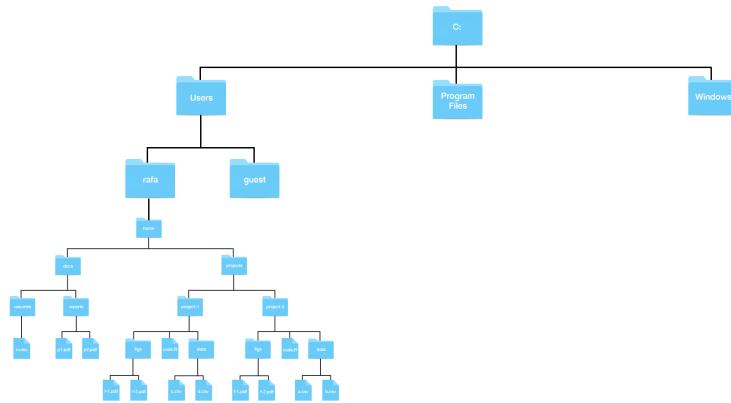
El directorio *home* es donde se guardan todas sus cosas, a diferencia de los archivos del sistema que vienen con sus computadoras, que se guardan en otro lugar. En la figura anterior, el directorio denominado “*home*” representa su directorio *home*, pero raras veces se llama así. En sus sistemas, el nombre de su directorio *home* probablemente será el mismo que su nombre de usuario en ese sistema. A continuación vemos un ejemplo en Windows y Mac que muestra un directorio *home*, en este caso, llamado *rafa*:



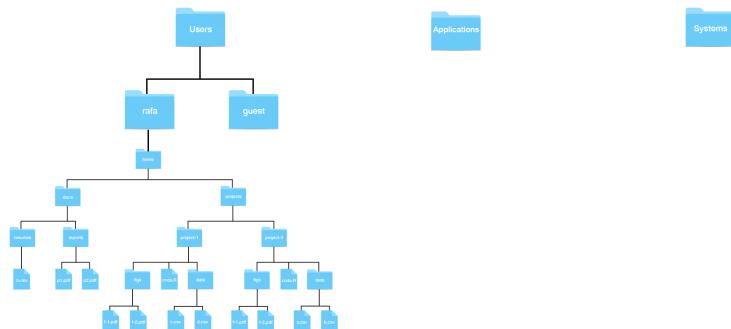
Ahora, miren a la figura anterior que muestra un sistema de archivos. Supongan que están

utilizando un sistema de apuntar y hacer clic y desea eliminar el archivo *cv.tex*. Imaginen que en su pantalla pueden ver el directorio *home*. Para borrar este archivo, deben hacer doble clic en el directorio *home*, luego *docs*, entonces *resumes* y luego arrastrar *cv.tex* a la basura. Aquí están experimentando la naturaleza jerárquica del sistema: *cv.tex* es un archivo dentro del directorio *resumes*, que es un subdirectorio dentro del directorio *docs*, que es un subdirectorio del directorio *home*.

Ahora supongan que no pueden ver su directorio *home* en su pantalla. De alguna manera, deberían hacer que aparezca en su pantalla. Una forma de hacerlo es navegar desde lo que se llama el *directorio raíz* (*root directory* en inglés) hasta el directorio *home*. Cualquier sistema de archivos tendrá lo que se llama un directorio raíz, que es el directorio que contiene todos los directorios. El directorio *home*, que se muestra en la figura anterior, generalmente estará a dos o más niveles del directorio raíz. En Windows, tendrán una estructura como esta:



mientras en la Mac, será así:



**Nota para usuarios de Windows:** La instalación típica de R hará que su directorio *Documents* sea su directorio *home* en R. Esto probablemente será diferente de su directorio *home* en Git Bash. En general, cuando discutimos directorios *home*, nos referimos al directorio *home* de Unix que para Windows, en este libro, es el directorio de Git Bash Unix.

### 38.3.3 Directorio de trabajo

El concepto de una *ubicación actual* (*current location* en inglés) es parte de la experiencia de apuntar y hacer clic: en cualquier momento estamos *en una carpeta* y vemos el con-

tenido de esa carpeta. A medida que busquen un archivo, como lo hicimos anteriormente, experimentarán el concepto de una ubicación actual: una vez que hagan doble clic en un directorio, cambiarán de ubicación y estarán *en esa carpeta*, a diferencia de la carpeta en la que estaban antes.

En Unix, no tenemos las mismas señales visuales, pero el concepto de una *ubicación actual* es indispensable. Nos referimos a esto como el *directorio de trabajo* (*working directory* en inglés). Cada ventana del terminal que tienen abierta tiene un directorio de trabajo asociado.

¿Cómo sabemos cuál es nuestro directorio de trabajo? Para responder a esto, aprendemos nuestro primer comando de Unix: `pwd`, que significa *imprimir el directorio de trabajo* (*print working directory* en inglés). Este comando devuelve el directorio de trabajo.

Abran un terminal y escriban:

```
pwd
```

No mostramos el resultado de ejecutar este comando porque será muy diferente en sus sistemas en comparación con otros. Si abren un terminal y escriben `pwd` como su primer comando, deberían ver algo como `/Users/yourusername` en la Mac o algo como `/c/Users/yourusername` en Windows. La cadena de caracteres devuelta al llamar `pwd` representa su directorio de trabajo. Cuando abrimos un terminal por primera vez, empezamos en nuestro directorio *home*, por lo que en este caso el directorio de trabajo es el directorio *home*.

Tengan en cuenta que las barras diagonales / en las cadenas de arriba separan los directorios. Entonces, por ejemplo, la ubicación `/c/Users/rafa` implica que nuestro directorio de trabajo se llama `rafa` y es un subdirectorío de `Users`, que es un subdirectorío de `c`, que es un subdirectorío del directorio raíz. Por lo tanto, el directorio raíz está representado solo por una barra diagonal: /.

### 38.3.4 Rutas

Nos referimos a la cadena devuelta por `pwd` como la *ruta completa* (*full path* en inglés) del directorio de trabajo. El nombre proviene del hecho de que esta cadena explica la *ruta* que deben seguir para llegar al directorio en cuestión desde el directorio raíz. Cada directorio tiene una ruta completa. Más adelante, aprenderemos sobre *rutas relativas*, que nos dicen cómo llegar a un directorio desde el directorio de trabajo.

En Unix, usamos la abreviatura ~ para representar su directorio *home*. Entonces, por ejemplo, si `docs` es un directorio en su directorio *home*, la ruta completa para `docs` puede escribirse así `~/docs`.

La mayoría de los terminales mostrarán la ruta de su directorio de trabajo directamente en la línea de comando. Si están utilizando la configuración predeterminada y abren un terminal en la Mac, verán que justo en la línea de comando tienen algo como `computername:~ username` con ~ representando su directorio de trabajo, que en este ejemplo es el directorio *home* ~. Lo mismo es cierto para el terminal Git Bash, donde verán algo como `username@computername MINGW64 ~`, con el directorio de trabajo al final. Cuando cambiamos los directorios, veremos este cambio tanto en Macs como en Windows.

## 38.4 Comandos de Unix

Ahora aprenderemos una serie de comandos de Unix que nos permitirán preparar un directorio para un proyecto de ciencia de datos. También ofrecemos ejemplos de comandos que, si escriben en su terminal, devolverán un error. Esto es porque suponemos que tenemos el sistema de archivos mostrado en el diagrama anterior. Su sistema de archivos es diferente. En la siguiente sección, le ofreceremos ejemplos que pueden escribir.

### 38.4.1 `ls`: Listado de contenido del directorio

En un sistema de apuntar y hacer clic, sabemos lo que hay en un directorio porque lo vemos. En el terminal, no vemos los íconos. En cambio, usamos el comando `ls` para enumerar el contenido del directorio.

Para ver el contenido de su directorio personal, abran un terminal y escriban:

```
ls
```

Veremos más ejemplos pronto.

### 38.4.2 `mkdir` y `rmdir`: crear y eliminar un directorio

Cuando nos estamos preparando para un proyecto de ciencia de datos, necesitaremos crear directorios. En Unix, podemos hacer esto con el comando `mkdir`, que significa *crear directorio* (*make directory* en inglés).

Debido a que pronto estarán trabajando en varios proyectos, les recomendamos crear un directorio llamado *proyectos* en su directorio *home*.

Pueden intentar este ejemplo en su sistema. Abran un terminal y escriban:

```
mkdir projects
```

Si hacen esto correctamente, no pasará nada: no tener noticias es buena noticia. Si el directorio ya existe, recibirán un mensaje de error y el directorio existente permanecerá intacto.

Para confirmar que crearon este directorio, pueden verlo usando:

```
ls
```

Verán una lista de cualquier directorio o archivo, incluyendo los que acaban de crear.

Con fines ilustrativos, vamos a crear algunos directorios más. Pueden enumerar más de un nombre de directorio así:

```
mkdir docs teaching
```

Pueden verificar si se crearon los tres directorios:

```
ls
```

Si cometieron un error y necesitan eliminar un directorio, pueden usar el comando `rmdir` (*remove directory* en inglés):

```
mkdir junk
rmdir junk
```

Esto eliminará el directorio siempre y cuando esté vacío. Si no está vacío, recibirán un mensaje de error y el directorio permanecerá intacto. Para eliminar directorios que no están vacíos, más tarde aprenderemos sobre el comando `rm`.

### 38.4.3 cd: navegando por el sistema de archivos cambiando directorios

A continuación, queremos crear directorios dentro de los directorios que ya hemos creado. También queremos evitar apuntar y hacer clic en nuestro sistema de archivos. Explicamos cómo hacer esto en Unix, usando la línea de comando.

Supongan que abrimos un terminal y nuestro directorio de trabajo es nuestro directorio `home`. Queremos cambiar nuestro directorio de trabajo a `projects`. Hacemos esto usando el comando `cd`, que significa *cambiar directorio* (*change directory* en inglés):

```
cd projects
```

Para verificar que el directorio de trabajo cambió, podemos usar un comando que aprendimos previamente para ver nuestra ubicación:

```
pwd
```

Nuestro directorio de trabajo ahora debería ser `~/projects`. Recuerden que en su computadora el directorio `home` ~ se verá algo como: `/c/Users/yourusername`).

**Consejo profesional importante:** En Unix, pueden completar automáticamente presionando `tab`. Esto significa que podemos escribir `cd d`, entonces presionar `tab`. Unix se completará automáticamente si `docs` es el único directorio/archivo que comienza con `d` o les mostrará las opciones. ¡Inténtelo! Usar Unix sin poder autocompletar sería insoportable.

Cuando usamos `cd`, podemos escribir una ruta completa, que comenzará con `/` o `~`, o una ruta relativa. En el ejemplo anterior, en el que escribimos `cd projects`, usamos una ruta relativa. Si la ruta que escriben no comienza con `/` o `~`, Unix supondrá que están escribiendo una ruta relativa y buscará el directorio en su directorio de trabajo actual. Entonces algo como esto les dará un error:

```
cd Users
```

porque no hay directorio `Users` en su directorio de trabajo.

Ahora supongan que queremos volver al directorio en el que `projects` es un subdirectorío, denominado el *directorio padre* (*parent directory* en inglés). Podríamos usar la ruta completa del directorio padre, pero Unix proporciona un acceso directo para esto: el directorio padre del directorio de trabajo se representa con dos puntos: `..`, así que para retroceder simplemente escribimos:

```
cd ..
```

Ahora deberían haber vuelto a su directorio *home*, que pueden confirmar usando `pwd`.

Como podemos usar rutas completas con `cd`, el siguiente comando:

```
cd ~
```

siempre nos regresará al directorio *home*, sin importar dónde estemos en el sistema de archivos.

El directorio de trabajo también tiene una abreviatura, que es un solo `..`, así que si escriben:

```
cd ..
```

no se moverán. Aunque este uso particular de `..` no es útil, la abreviatura a veces lo es. Las razones no son relevantes para esta sección, pero aún deben tomar en cuenta este hecho.

En resumen, hemos aprendido que al usar `cd` una de tres cosas pasan: nos quedamos en la misma ubicación, nos movemos a un nuevo directorio usando el nombre del directorio deseado o volvemos al directorio padre usando `..`.

Al escribir nombres de directorios, podemos concatenar directorios con barras diagonales. Entonces, si queremos un comando que nos lleve al directorio `projects` sin importar dónde nos encontremos en el sistema de archivos, podemos escribir:

```
cd ~/projects
```

que es equivalente a escribir toda la ruta. Por ejemplo, en Windows escribiríamos algo como:

```
cd/c/Users/yourusername/projects
```

Los dos últimos comandos son equivalentes y en ambos casos estamos escribiendo la ruta completa.

Al escribir la ruta del directorio que queremos, ya sea completa o relativa, podemos concatenar directorios con barras diagonales. Ya vimos que podemos pasar al directorio `projects` independientemente de dónde estamos escribiendo la ruta completa de esta manera:

```
cd ~/projects
```

También podemos concatenar nombres de directorio para rutas relativas. Por ejemplo, si queremos volver al directorio padre del directorio padre del directorio de trabajo, podemos escribir:

```
cd ../../
```

Aquí ofrecemos un par de consejos finales relacionados con el comando `cd`. Primero, pueden volver al directorio que acaban de dejar escribiendo:

```
cd -
```

Esto puede ser útil si escriben una ruta muy larga y luego se dan cuenta de que quieren volver a donde estaban, y ese también tiene una ruta muy larga.

En segundo lugar, si solo escriben:

```
cd
```

regresarán a su directorio *home*.

## 38.5 Algunos ejemplos

Exploraremos algunos ejemplos de como usar `cd`. Para ayudarles visualizar, mostraremos la representación gráfica de nuestro sistema de archivos verticalmente:



Supongan que nuestro directorio de trabajo es `~/projects` y queremos mover `figs` a `project-1`.

En este caso, es conveniente usar rutas relativas:

```
cd project-1/figs
```

Ahora supongan que nuestro directorio de trabajo es `~/projects` y queremos mover `reports` a `docs`, ¿cómo podemos hacer esto?

Una forma es usar rutas relativas:

```
cd ../docs/reports
```

Otra es usar la ruta completa:

```
cd ~/docs/reports
```

Si están intentando esto en su sistema, recuerden aprovecharse de que Unix autocompleta.

Examinemos un ejemplo más. Supongan que estamos en `~/projects/project-1/figs` y queremos cambiar a `~/projects/project-2`. De nuevo, hay dos formas.

Con rutas relativas:

```
cd ../../project-2
```

y con rutas completas:

```
cd ~/projects/project-2
```

---

## 38.6 Más comandos de Unix

### 38.6.1 mv: mover archivos

En un sistema de apuntar y hacer clic, movemos los archivos de un directorio a otro arrastrando y soltando. En Unix, usamos el comando `mv`.

**Advertencia:** `mv` no les preguntará “¿estás seguro?” si su cambio resulta en sobrescribir un archivo.

Ahora que saben cómo usar rutas completas y relativas, usar `mv` es relativamente sencillo. La forma general es:

```
mv path-to-file path-to-destination-directory
```

Por ejemplo, si queremos mover el archivo `cv.tex` desde `resumes` a `reports`, podrían usar las rutas completas de esta manera:

```
mv ~/docs/resumes/cv.tex ~/docs/reports/
```

También pueden usar rutas relativas y hacer lo siguiente:

```
cd ~/docs/resumes
mv cv.tex ../reports/
```

o esto:

```
cd ~/docs/reports/
mv ../../resumes/cv.tex ./
```

Observen que en el último usamos el acceso directo al directorio de trabajo . para dar una ruta relativa como el directorio de destino.

También podemos usar `mv` para cambiar el nombre de un archivo. Para hacer esto, en lugar de que el segundo argumento sea el directorio de destino, también incluye un nombre de archivo. Entonces, por ejemplo, para cambiar el nombre de `cv.tex` a `resume.tex`, simplemente escribimos:

```
cd ~/docs/resumes
mv cv.tex resume.tex
```

También podemos combinar el movimiento y un cambio de nombre. Por ejemplo:

```
cd ~/docs/resumes
mv cv.tex ../reports/resume.tex
```

Y podemos mover directorios completos. Para mover el directorio `resumes` a `reports`, hacemos lo siguiente:

```
mv ~/docs/resumes ~/docs/reports/
```

Es importante agregar el último / para que quede claro que no desean cambiar el nombre del directorio `resumes` a `reports`, sino más bien moverlo al directorio `reports`.

### 38.6.2 cp: copiando archivos

El comando `cp` se comporta de manera similar a `mv` excepto que en lugar de mover, copiamos el archivo, que significa que el archivo original permanece intacto.

Entonces, en todos los ejemplos `mv` anteriores, pueden cambiar `mv` a `cp` y copiarán en lugar de mover con una excepción: no podemos copiar directorios completos sin aprender sobre argumentos, que haremos más adelante.

### 38.6.3 rm: eliminar archivos

En los sistemas de apuntar y hacer clic, eliminamos los archivos arrastrándolos y soltándolos en la basura o haciendo un clic especial con el mouse. En Unix, usamos el comando `rm`.

**Advertencia:** A diferencia de echar archivos a la basura, `rm` es permanente ¡Tengan cuidado!

La forma general en que funciona es la siguiente:

```
rm filename
```

De hecho, pueden enumerar archivos así:

```
rm filename-1 filename-2 filename-3
```

Pueden usar rutas completas o relativas. Para eliminar directorios, tendrán que aprender sobre argumentos, que haremos más adelante.

### 38.6.4 less: mirando un archivo

A menudo, desearán ver rápidamente el contenido de un archivo. Si tal archivo es un archivo de texto, la forma más rápida de hacerlo es mediante el comando `less`. Para ver el archivo `cv.tex`, deben hacer lo siguiente:

```
cd ~/docs/resumes
less cv.tex
```

Para escapar del visualizador, escriban `q`. Si los archivos son largos, pueden usar las teclas de flecha para moverse hacia arriba y hacia abajo. Hay muchos otros comandos de teclado que pueden usar dentro de `less` para, por ejemplo, buscar o saltar páginas. Aprenderán más sobre esto en una sección posterior. Si se preguntan por qué se llama el comando `less`, es porque el original fue llamado `more`, como en “show me more of this file” o “muéstrame más de este archivo”. La segunda versión se llamó `less` por el dicho “less is more” o “menos es más”.

---

## 38.7 Preparación para un proyecto de ciencia de datos

Ahora estamos listos para preparar un directorio para un proyecto. Utilizaremos el proyecto de asesinatos de Estados Unidos<sup>7</sup> como ejemplo.

Deberían comenzar creando un directorio donde guardarán todos sus proyectos. Recomendamos un directorio llamado `projects` en su directorio `home`. Para hacer esto, escriban:

---

<sup>7</sup><https://github.com/rairizarry/murders>

```
cd ~
mkdir projects
```

Nuestro proyecto se relaciona con asesinatos por armas de fuego, por lo que llamaremos al directorio de nuestro proyecto: `murders`. Será un subdirectorio en nuestro directorio de proyecto. En el directorio `murders`, crearemos dos subdirectorios para contener los datos sin procesar y los datos intermedios. Llamaremos a estos `data` y `rda`, respectivamente.

Abran un terminal y asegúrense de estar en el directorio `home`:

```
cd ~
```

Ahora ejecuten los siguientes comandos para crear la estructura de directorio que queremos. Al final, usamos `ls` y `pwd` para confirmar que hemos generado el directorio correcto en el directorio de trabajo correcto:

```
cd projects
mkdir murders
cd murders
mkdir data rdas
ls
pwd
```

Tengan en cuenta que la ruta completa de nuestro set de datos `murders` es `~/projects/murders`.

Entonces, si abrimos un nuevo terminal y queremos navegar en ese directorio, escribimos:

```
cd projects/murders
```

En la Sección 40.3, describiremos cómo podemos usar RStudio para organizar un proyecto de análisis de datos, una vez que se hayan creado este directorio.

---

## 38.8 Unix avanzado

La mayoría de las implementaciones de Unix incluyen una gran cantidad de herramientas y utilidades eficaces. Acabamos de aprender los conceptos básicos. Recomendamos que utilicen Unix como su herramienta principal de administración de archivos. Tomará tiempo sentirse cómodo con él, pero durante este tiempo, aprenderán mucho buscando soluciones en el Internet. En esta sección, superficialmente cubrimos temas ligeramente más avanzados. El objetivo principal de la sección es informarles sobre lo que está disponible en lugar de explicar todo en detalle.

### 38.8.1 Argumentos

La mayoría de los comandos de Unix se pueden ejecutar con argumentos. Los argumentos generalmente se definen usando un guión - o dos guiones -- (según el comando) seguido de

una letra o una palabra. Un ejemplo de un argumento es una **-r** detrás **rm**. La **r** significa recursivo y el resultado es que los archivos y directorios se eliminan recursivamente. O sea, si escriben:

```
rm -r directory-name
```

se eliminarán todos los archivos, subdirectorios, archivos en subdirectorios, subdirectorios en subdirectorios, etc. Esto es equivalente a echar una carpeta en la basura, excepto que no pueden recuperarla. Una vez que la eliminan, se eliminará para siempre. Frecuentemente, cuando eliminan directorios, encontrarán archivos que están protegidos. En tales casos, pueden usar el argumento **-f** que significa **force**.

También pueden combinar argumentos. Por ejemplo, para eliminar un directorio independientemente de los archivos protegidos, escribirían:

```
rm -rf directory-name
```

Recuerden que una vez que eliminan no hay marcha atrás. Por lo tanto, deben usar este comando con mucho cuidado.

Un comando que a menudo se llama con argumentos es **ls**. Aquí hay un ejemplo:

```
ls -a
```

La **a** representa a “todos” (*all* en inglés). Este argumento hace que **ls** les muestre todos los archivos en el directorio, incluyendo los archivos ocultos. En Unix, todos los archivos que comienzan con un **.** están escondidos. Muchas aplicaciones crean directorios ocultos para almacenar información importante sin interferir con su trabajo. Un ejemplo es **git** (que discutimos en detalle más adelante). Una vez que inicializan un directorio como un directorio **git** con **git init**, se crea un directorio oculto llamado **.git**. Otro archivo oculto es el archivo **.gitignore**.

Otro ejemplo de usar un argumento es:

```
ls -l
```

La **l** significa “largo” y el resultado es que se muestra más información sobre los archivos.

A menudo es útil ver los archivos en orden cronológico. Para eso usamos:

```
ls -t
```

y para invertir (*reverse* en inglés) el orden de cómo se muestran los archivos, pueden usar:

```
ls -r
```

Podemos combinar todos estos argumentos para mostrar más información para todos los archivos en orden cronológico inverso:

```
ls -lart
```

Cada comando tiene un conjunto diferente de argumentos. En la siguiente sección, aprendemos cómo averiguar qué hacen cada uno.

### 38.8.2 Obtener ayuda

Como habrán notado, Unix usa una versión extrema de abreviaturas. Esto lo hace muy eficiente, pero también difícil de adivinar cómo llamar a los comandos. Para compensar por esta debilidad, Unix incluye archivos de ayuda completos o *man pages* (“man” es la abreviatura de manual). En la mayoría de los sistemas, pueden escribir `man`, seguido por el nombre del comando para obtener ayuda. Entonces para `ls`, escribiríamos:

```
man ls
```

Este comando no está disponible en algunas de las implementaciones compactas de Unix, como Git Bash. Una forma alternativa de obtener ayuda que funciona en Git Bash es escribir el comando seguido de `--help`. Entonces para `ls`, sería lo siguiente:

```
ls --help
```

### 38.8.3 El *pipe*

Las páginas de ayuda suelen ser largas y si escriben los comandos que hemos discutido, enseña todo el documento. Sería útil si pudiéramos guardar el manual en un archivo y luego usar `less` para verlo. El *pipe*, escrito así `|`, hace algo parecido. Transmite los resultados de un comando al comando después de *pipe*. Esto es similar al *pipe* `%>%` que usamos en R. Para obtener más ayuda, podemos escribir:

```
man ls | less
```

o en Git Bash:

```
ls --help | less
```

Esto también es útil cuando se enumeran archivos con muchos archivos. Podemos escribir:

```
ls -lart | less
```

### 38.8.4 Comodines

Algunos de los aspectos más poderosos de Unix son los *comodines* (*wild cards* en inglés). Supongan que queremos eliminar todos los archivos HTML temporeros producidos durante la resolución de problemas para un proyecto. Imagínense que hay docenas de archivos. Tomaría muchísimo tiempo eliminarlos uno por uno. En Unix, podemos escribir una expresión que

significa todos los archivos que terminan en `.html`. Para hacer esto, escribimos el comodín `*`. Como se discutió en la parte de *wrangling* de datos de este libro, este carácter significa cualquier número de cualquier combinación de caracteres. Específicamente, para enumerar todos los archivos HTML, escribimos:

```
ls *.html
```

Para eliminar todos los archivos HTML en un directorio, escribimos:

```
rm *.html
```

El otro comodín útil es el símbolo `?`. Esto significa cualquier carácter individual. Entonces, si todos los archivos que queremos borrar tienen la forma `file-001.html` con los números que van del 1 al 999, podemos escribir:

```
rm file-???.html
```

Esto solo eliminará archivos con ese formato.

Además, podemos combinar comodines. Por ejemplo, para eliminar todos los archivos con el nombre `file-001` independientemente del sufijo, podemos escribir:

```
rm file-001.*
```

**Advertencia: combinando `rm` con el comodín `*` puede ser peligroso. Hay combinaciones de estos comandos que borrarán todo sus sistemas de archivos sin preguntar “¿está seguro?”. Asegúrense de entender cómo funciona antes de usar este comodín con el comando `rm`.**

### 38.8.5 Variables de entorno

Unix tiene configuraciones que afectan el *entorno* (*environment* en inglés) de sus líneas de comando. Estas se llaman variables de entorno. El directorio *home* es uno de ellos. De hecho, podemos cambiar algunos de estos. En Unix, las variables se distinguen de otras entidades agregando un `$` en le comienzo. El directorio *home* se guarda en `$ HOME`.

Anteriormente vimos que `echo` es el comando Unix para imprimir. Entonces, podemos ver nuestro directorio *home* al escribir:

```
echo $HOME
```

Pueden verlos todos al escribir:

```
env
```

Pueden cambiar algunas de estas variables del entorno. Pero sus nombres varían según los diferentes *shells*. Describimos los *shells* en la siguiente sección.

### 38.8.6 Shells

Mucho lo que usamos en este capítulo es parte de lo que se llama el *Unix shell*. Hay varios *shells* diferentes, aunque estas diferencias son casi imperceptibles. A la vez, son importantes, aunque no los cubrimos aquí. Pueden ver qué *shell* están utilizando escribiendo:

```
echo $SHELL
```

El más común es `bash`.

Una vez que sepan el *shell* que están usando, pueden cambiar las variables de entorno. En Bash Shell, lo hacemos usando `export variable value`. Para cambiar la ruta, que se describirá con más detalle pronto, escriban: (**Esto es solo un ejemplo. ¡Asegúrense de no ejecutar este comando!**)

```
export PATH =/usr/bin/
```

Hay un programa que se ejecuta cuando usan el terminal donde pueden editar variables para que cambien cada vez que usen el terminal. Esto cambia en diferentes implementaciones, pero si usan bash, pueden crear un archivo llamado `.bashrc`, `.bash_profile`, `.bash_login` o `.profile`. Es posible que ya tengan uno.

### 38.8.7 Ejecutables

En Unix, todos los programas son archivos. Se llaman *ejecutables* (*executables* en inglés). Entonces `ls`, `mv` y `git` son todos archivos. ¿Pero dónde están estos archivos de programas? Pueden averiguarlo usando el comando `which`:

```
which git
#> /usr/bin/git
```

Ese directorio probablemente está lleno de archivos de programas. El directorio `/usr/bin` usualmente contiene muchos archivos de programa. Si escriben:

```
ls/usr/bin
```

en su terminal, verán varios archivos ejecutables.

Hay otros directorios que generalmente contienen archivos de programas. El directorio *Application* en la Mac o *Program Files* en Windows son ejemplos.

Cuando escriben `ls`, Unix sabe que debe correr un programa que es un ejecutable y que se almacena en algún otro directorio. Entonces, ¿cómo sabe Unix dónde encontrarlo? Esta información se incluye en la variable de entorno `$PATH`. Si escriben:

```
echo $PATH
```

verán una lista de directorios separados por `:`. El directorio `/usr/bin` es probablemente uno de los primeros en la lista.

Unix busca archivos de programas en esos directorios en ese orden. Aunque no lo discutimos aquí, pueden crear ejecutables ustedes mismos. Sin embargo, si lo colocan en su directorio de trabajo y este directorio no está en la ruta, no pueden ejecutarlo simplemente escribiendo el comando. Se evita esto escribiendo la ruta completa. Entonces, si sus comandos se llama `my-ls`, pueden escribir:

```
./my-ls
```

Una vez que hayan dominado los conceptos básicos de Unix, consideren aprender a escribir sus propios ejecutables, ya que pueden ayudar a reducir el trabajo repetitivo.

### 38.8.8 Permisos y tipos de archivo

Si escriben:

```
ls -l
```

Al principio, verán una serie de símbolos como este `-rw-r--r--`. Esta cadena indica el tipo de archivo: archivo normal `-`, directorio `d` o ejecutable `x`. Esta cadena también indica el permiso del archivo: ¿Se puede leer? ¿Se puede cambiar? ¿Es ejecutable? ¿Otros usuarios del sistema pueden leer el archivo? ¿Otros usuarios pueden editar el archivo? ¿Otros usuarios pueden ejecutar si el archivo es ejecutable? Esto es más avanzado que lo que cubrimos aquí, pero pueden aprender mucho más en un libro de referencia de Unix.

### 38.8.9 Comandos que deben aprender

Hay muchos comandos que no discutimos en detalle en este libro, pero queremos darles un poco de información sobre ellos y lo que hacen. Son:

- `open/start` - En la Mac, `open filename` intenta averiguar la aplicación correcta del nombre de archivo y abrirlo con esa aplicación. Este es un comando muy útil. En Git Bash, pueden tratar `start filename`. Intenten abrir un archivo R o Rmd con `open` o `start`: deberían abrirlo con RStudio.
- `nano`: Un editor de texto básico.
- `ln`: Crea un enlace simbólico. No recomendamos su uso, pero deben estar familiarizados con él.
- `tar`: Archiva archivos y subdirectorios de un directorio en un solo archivo.
- `ssh`: Se conecta a otra computadora.
- `grep`: Busca patrones en un archivo.
- `awk/sed`: Estos son dos comandos muy útiles que les permite encontrar cadenas específicas en archivos y cambiarlas.

### 38.8.10 Manipulación de archivos en R

También podemos realizar el manejo de archivos desde R. Las funciones claves para aprender se pueden ver mirando el archivo de ayuda para `?files`. Otra función útil es `unlink`.

Aunque generalmente no lo recomendamos, tengan en cuenta que pueden ejecutar comandos de Unix en R usando `system`.



# 39

---

## *Git y GitHub*

---

Aquí proveemos algunos detalles sobre Git y GitHub. Sin embargo, solo estamos rascando la superficie. Para obtener más información sobre este tema, recomendamos los siguientes recursos:

- Codecademy: <https://www.codecademy.com/learn/learn-git>
  - Guías de GitHub: <https://guides.github.com/activities/hello-world/>
  - Try Git tutorial: <https://try.github.io/levels/1/challenges/1>
  - Happy Git y GitHub para el useR: <http://happygitwithr.com>
- 

### 39.1 ¿Por qué usar Git y GitHub?

Hay tres razones principales para usar Git y GitHub.

1. Compartir: Incluso si no aprovechamos la funcionalidad avanzada y útiles de control de versiones, aún podemos usar Git y GitHub para compartir nuestro código.
2. Colaboración: Una vez que configuren un repositorio central, pueden hacer que varias personas realicen cambios en el código y mantengan las versiones sincronizadas. GitHub ofrece un servicio gratuito para repositorios centralizados. GitHub también tiene una herramienta, llamada *pull request*, que cualquier persona puede usar para sugerir cambios a su código. Pueden aceptar o rechazar fácilmente la recomendación.
3. Control de versiones: Las capacidades de control de versiones de Git nos permite darle seguimiento a los cambios que realizamos en nuestro código. Además, podemos volver a las versiones anteriores de los archivos. Git también nos permite crear *branches* en las que podemos probar ideas, luego decidir si fusionamos (*merge* en inglés) la nueva rama con la original.

Aquí nos enfocamos en los aspectos de uso compartido de Git y GitHub y remitimos al lector a los enlaces anteriores para obtener más información sobre esta herramienta útil.

## 39.2 Cuentas GitHub

Después de instalar git<sup>1</sup>, el primer paso es obtener una cuenta de GitHub. Las cuentas básicas de GitHub son gratuitas. Para hacer esto, vayan a [GitHub.com](https://GitHub.com) donde verán un cuadro en el que pueden registrarse.

Quieren escoger un nombre con cuidado. Debe ser breve, fácil de recordar y deletrear, relacionado de alguna manera con su nombre y, por supuesto, profesional. Este último es importante ya que podrían estar enviando enlaces a sus cuentas de GitHub a posibles empleadores. En el siguiente ejemplo, sacrifico la facilidad de deletreo para incorporar mi nombre. Sus iniciales y apellidos suelen ser una buena opción. Si tienen un nombre muy común, es posible que tengan que tomar eso en cuenta. Una solución sencilla sería agregar números o deletrear parte de su nombre.

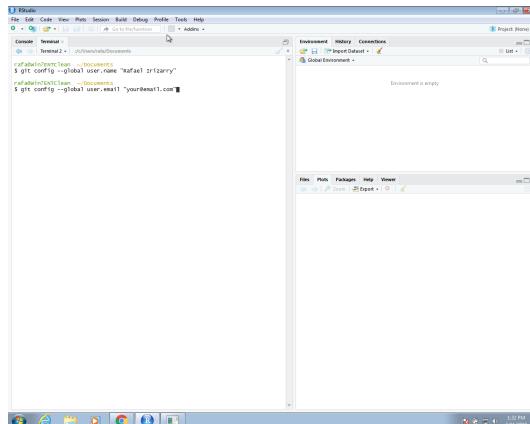
La cuenta que uso para mi investigación, *rafalab*, es la misma que uso para mi página web<sup>2</sup> y Twitter<sup>3</sup>, que lo hace fácil de recordar para los que siguen mi trabajo.

Una vez que tengan una cuenta de GitHub, estarán listos para conectar Git y RStudio a esta cuenta.

Un primer paso es dejar que Git sepa quiénes somos. Esto facilitará la conexión con GitHub. Comenzamos abriendo una ventana de terminal en RStudio (recuerden que pueden obtener una a través de *Tools* en la barra de menú). Ahora usamos el comando `git config` para decirle a Git quiénes somos. Escribiremos los siguientes dos comandos en nuestra ventana de terminal:

```
git config --global user.name "Your Name"
git config --global user.mail "your@email.com"
```

Deben usar la cuenta de correo electrónico que utilizaron para abrir su cuenta de GitHub. La sesión de RStudio debería verse así:

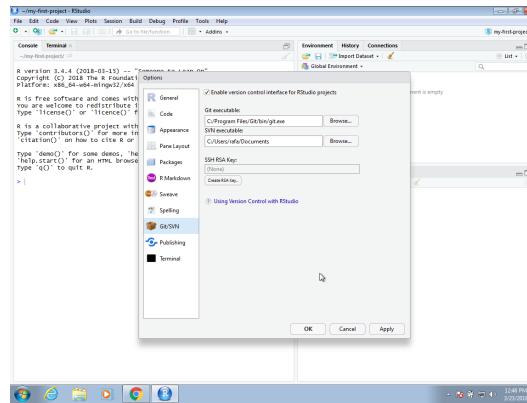


<sup>1</sup><https://rafalab.github.io/dsbook/accessing-the-terminal-and-installing-git.html>

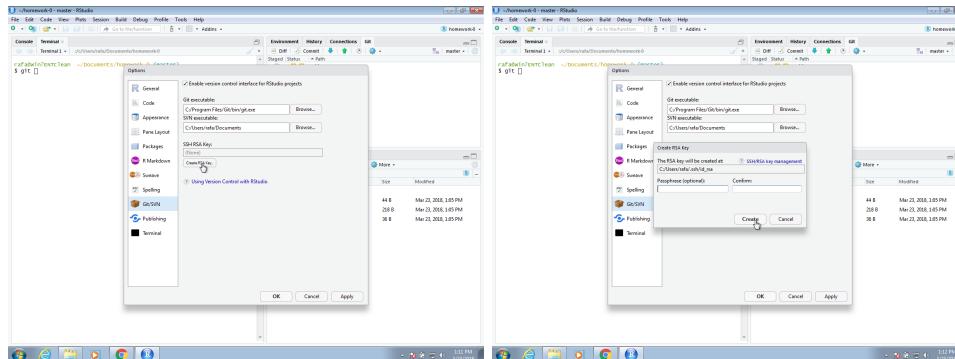
<sup>2</sup><http://rafalab.org>

<sup>3</sup><http://twitter.com/rafalab>

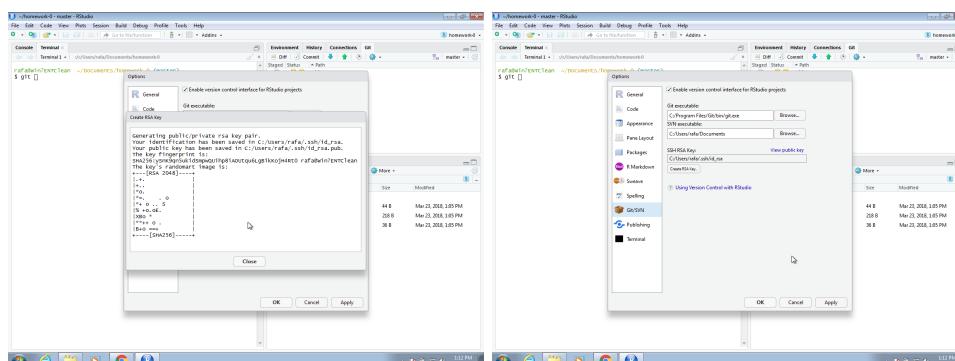
Empiecen yendo a *Global Options*, seleccionando *Git/SVN* y luego ingresando una ruta para el ejecutable de Git que acabamos de instalar.



En la instalación predeterminada de Windows, la ruta será *C:/Program File/Git/bin/git.exe*, pero deben encontrarla explorando su sistema, ya que esto puede cambiar de un sistema a otro. Ahora, para evitar ingresar nuestra contraseña de GitHub cada vez que intentemos acceder a nuestro repositorio, crearemos lo que se llama una *SSH RSA Key*. RStudio puede hacer esto por nosotros automáticamente si hacemos clic en el botón *Create RSA Key*:



Pueden seguir las instrucciones predeterminadas como se muestra a continuación:



Git, RStudio y GitHub ahora deben poder conectarse y estamos listos para crear un primer repositorio de código de GitHub.

### 39.3 Repositorios de GitHub

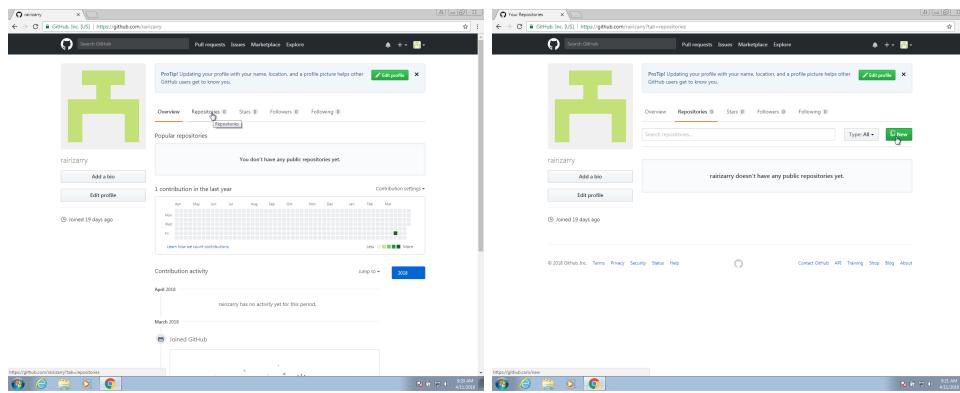
Ya pueden crear un repositorio o “repo” de GitHub. La idea general es tener al menos dos copias de su código: una copia en sus computadoras y otra en GitHub. Si añaden colaboradores a este proyecto, cada uno tendrá una copia en su computadora. La copia de GitHub generalmente se considera la copia *main* (antes llamada *master*) con la que se sincroniza cada colaborador. Git les ayudará a mantener sincronizadas todas las copias diferentes.

Como se mencionó anteriormente, una de las ventajas de mantener el código en un repositorio de GitHub es que pueden compartirlo fácilmente con posibles empleadores interesados en ver ejemplos de su trabajo. Debido a que muchas compañías de ciencia de datos usan sistemas de control de versiones, como Git, para colaborar en proyectos, también pueden estar impresionadas de que ya conocen al menos los conceptos básicos.

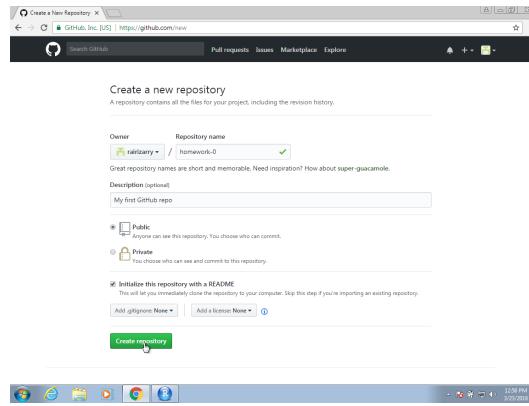
El primer paso para crear un repositorio para su código es inicializar en GitHub. Como ya crearon una cuenta, tendrán una página en GitHub con el URL <http://github.com/username>.

Para crear un repositorio, primero inicien una sesión en su cuenta haciendo clic en el botón *Sign In* en <https://github.com>. Es posible que ya hayan iniciado una sesión, en cuyo caso el botón *Sign In* no aparecerá. Si inician sesión, deben ingresar su nombre de usuario y contraseña. Le recomendamos que configuren su navegador para recordar esto y evitar escribirlo cada vez.

Una vez en sus cuentas, pueden hacer clic en *Repositories* y luego hacer clic en *New* para crear un nuevo repositorio:

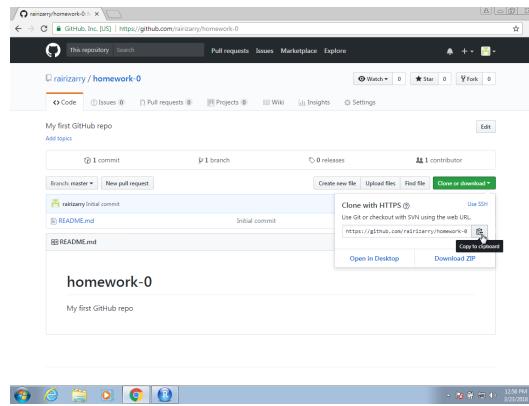


Entonces querrán elegir un buen nombre descriptivo para el proyecto. En el futuro, es posible que tengan docenas de repositorios, así que tomen esto en cuenta al elegir un nombre. Aquí usaremos `homework-0`. Le recomendamos que hagan público el repositorio. Si prefieren mantenerlo en privado, tendrán que pagar un cargo mensual.



Ahora tienen su primer repositorio en GitHub. El siguiente paso será *clonarlo* (*clone it* en inglés) en su computadora y comenzar a editar y sincronizar usando Git.

Para hacer esto, es conveniente copiar el enlace proporcionado por GitHub específicamente para conectarse a este repositorio, usando Git como se muestra a continuación. Más tarde, necesitaremos copiar y pegar esto, así que asegúrense de recordar este paso.



## 39.4 Descripción general de Git

Las principales acciones en Git son:

1. **pull:** Jalar/tirar cambios desde el repositorio remoto, en este caso el repositorio de GitHub.
2. **add:** Añadir archivos o, como decimos en la jerga de Git, *stage* los archivos.
3. **commit:** Asignar cambios al repositorio local.
4. **push:** Empujar cambios al repositorio *remote*, en nuestro caso el repositorio GitHub.

Para permitir efectivamente el control de versiones y la colaboración en Git, los archivos se mueven a través de cuatro áreas diferentes:



Pero, ¿cómo comienza todo? Hay dos formas: podemos clonar un repositorio existente o inicializar uno. Exploraremos la clonación primero.

#### 39.4.1 Clonar

Vamos a *clonar* un *Upstream Repository* existente. Pueden verlo en GitHub aquí: <https://github.com/rairizarry/murders>. Al visitar esta página, pueden ver varios archivos y directorios. Estos son el *Upstream Repository*. Al hacer clic en el botón verde de clonar, podemos copiar el URL del repositorio: <https://github.com/rairizarry/murders.git>.

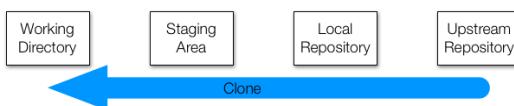
Pero, ¿qué significa *clonar*? En lugar de descargar todos estos archivos a su computadora, vamos a copiar toda la estructura de Git, que significa que agregaremos los archivos y directorios a cada una de las tres etapas locales: *Working Directory*, *Staging Area* y *Local Repository*. Cuando clonian, estos tres son exactamente iguales en el comienzo.

Pueden ver rápidamente un ejemplo de esto haciendo lo siguiente. Abran una terminal y escriban:

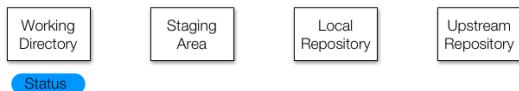
```
pwd
mkdir git-example
cd git-example
git clone https://github.com/rairizarry/murders.git
cd murders
#> /Users/rafa/myDocuments/teaching/data-science/dslibro
#> Cloning into 'murders'...
```

Ahora han clonado un repositorio de GitHub y tienen un directorio de trabajo de Git, con todos los archivos, en su sistema.

```
ls
#> README.txt
#> analysis.R
#> data
#> download-data.R
#> murders.Rproj
#> rdas
#> report.Rmd
#> report.md
#> report_files
#> wrangle-data.R
```



El *Working Directory* es el mismo que el directorio de trabajo de Unix. Cuando editan archivos usando un editor como RStudio, cambian los archivos en esta área y solo en esta área. Git puede decirles cómo se relacionan estos archivos con las versiones de los archivos en otras áreas con el comando `git status`:



Si verifican el estatus ahora, verán que nada ha cambiado y recibirán el siguiente mensaje:

```

git status
#> On branch master
#> Your branch is up to date with 'origin/master'.
#>
#> nothing to commit, working tree clean

```

Ahora vamos a hacer cambios en estos archivos. Eventualmente, queremos que estas nuevas versiones de los archivos sean rastreadas y sincronizadas con el *Upstream Repository*. Pero no queremos darle seguimiento a cada cambio pequeño: no queremos sincronizar hasta que estemos seguros de que estas versiones son lo suficientemente finales como para compartirlas. Por esta razón, el sistema de control de versiones no guarda las ediciones en el *Staging Area*.

Para demostrar, agregamos un archivo al *Staging Area* con el comando `git add`. A continuación creamos un archivo usando el comando de Unix `echo` solo como ejemplo (normalmente usarían RStudio):

```
echo "test" >> new-file.txt
```

También estamos agregando un archivo temporero del cual que no queremos llevar cuenta:

```
echo "temporary" >> tmp.txt
```

Ahora podemos organizar el archivo que finalmente queremos agregar a nuestro repositorio:

```
git add new-file.txt
```

Observen lo que dice el estatus ahora:

```

git status
#> On branch master
#> Your branch is up to date with 'origin/master'.
#>
#> Changes to be committed:
#> (use "git restore --staged <file>..." to unstage)
#> new file: new-file.txt
#>
#> Untracked files:
#> (use "git add <file>..." to include in what will be committed)
#> tmp.txt

```

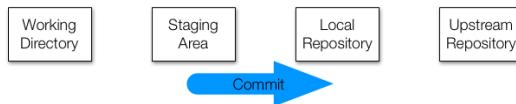


Como `new-file.txt` está *staged*, la versión actual del archivo se agregará al repositorio local la próxima vez que decidimos *commit*, que hacemos de la siguiente manera:

```
git commit -m "adding a new file"
#> [master 5f51e8a] adding a new file
#> 1 file changed, 1 insertion(+)
#> create mode 100644 new-file.txt
```

Ahora hemos cambiado el repositorio local, que pueden confirmar usando:

```
git status
```



Sin embargo, si volvemos a editar ese archivo, solo cambia en el directorio de trabajo. Para agregar al repositorio local, necesitamos añadirlo y *commit* los cambios que se agregan al repositorio local:

```
echo "adding a line" >> new-file.txt
git add new-file.txt
git commit -m "adding a new line to new-file"
#> [master 6b5314c] adding a new line to new-file
#> 1 file changed, 1 insertion(+)
```

Tengan en cuenta que este paso a menudo es innecesario en nuestros usos de Git. Podemos omitir la parte de preparación si agregamos el nombre del archivo al comando *commit* de esta manera:

```
echo "adding a second line" >> new-file.txt
git commit -m "minor change to new-file" new-file.txt
#> [master d7da6d9] minor change to new-file
#> 1 file changed, 1 insertion(+)
```

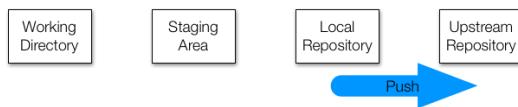
Podemos darle seguimiento a todos los cambios que hemos realizado con:

```
git log new-file.txt
#> commit d7da6d9d4ea1c3f27c2287fc8974e775b7f2bfca
#> Author: Rafael A. Irizarry <rairizarry@gmail.com>
#> Date: Mon May 24 08:18:45 2021 -0400
#>
#> minor change to new-file
```

```
#>
#> commit 6b5314c6005d1bd14415d7bfd38d1e1abb0635b
#> Author: Rafael A. Irizarry <rairizarrry@gmail.com>
#> Date: Mon May 24 08:18:44 2021 -0400
#>
#> adding a new line to new-file
#>
#> commit 5f51e8a15fa862b019a3692c1cb7a06e4210174e
#> Author: Rafael A. Irizarry <rairizarrry@gmail.com>
#> Date: Mon May 24 08:18:44 2021 -0400
#>
#> adding a new file
```

Para mantener todo sincronizado, el paso final es impulsar los cambios al *Upstream Repository*. Esto se hace con el comando `git push` así:

```
git push
```



Sin embargo, en este ejemplo particular, no podrán hacerlo porque no tienen permiso para editar el *Upstream Repository*. Si este fuera su repositorio, podrían.

Si este es un proyecto colaborativo, el *Upstream Repository* puede cambiar y volverse diferente a nuestra versión. Para actualizar nuestro repositorio local para que sea como el *Upstream Repository*, usamos el comando `fetch`:

```
git fetch
```



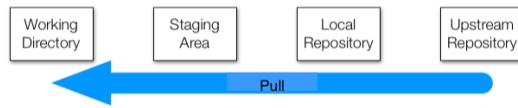
Y entonces para hacer estos cambios al *Staging Area* y *Working Directory*, utilizamos el comando:

```
git merge
```



No obstante, a menudo solo queremos cambiar ambos con un solo comando. Para esto, utilizamos:

```
git pull
```



Aprenderemos en la Sección 39.6 cómo RStudio tiene botones para hacer todo esto. Los detalles ofrecidos aquí les ayuda a entender lo que sucede en el trasfondo.

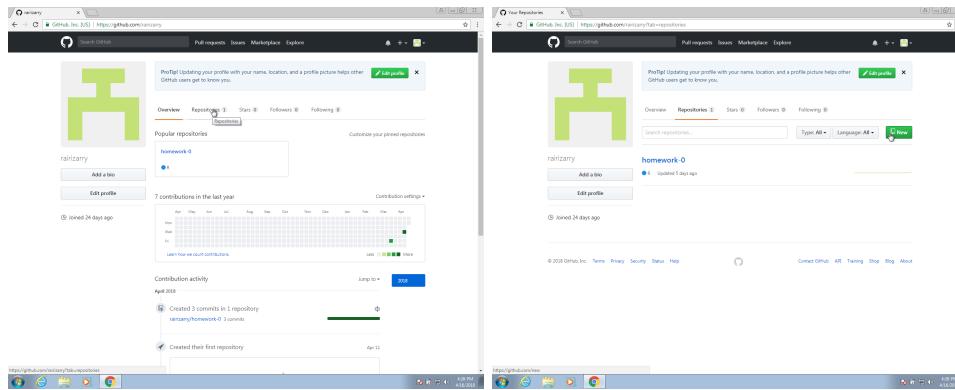
### 39.5 Inicializando un directorio Git

Ahora exploremos la segunda manera en que podemos comenzar: inicializando un directorio en nuestra propia computadora en lugar de clonar.

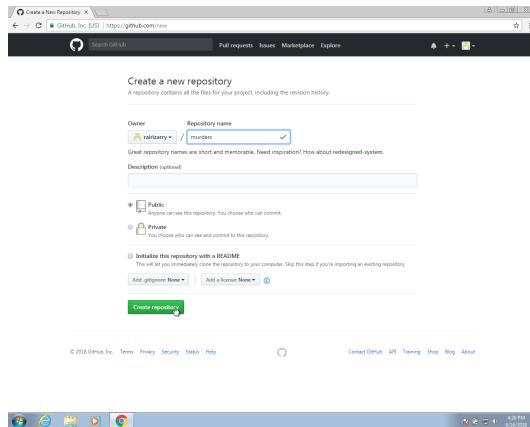
Supongan que ya tenemos un directorio local con archivos y queremos convertir este directorio en un repositorio colaborativo de GitHub. La forma más eficiente de lograr esto es *inicializando* el directorio local.

Para demostrar cómo hacer esto, inicializaremos el directorio de asesinatos con armas de fuego que creamos en la Sección 38.7. Tengan en cuenta que ya creamos un directorio con varios subdirectorios en nuestra computadora, pero aún no tenemos un repositorio local de Git o un *Upstream Repository* de GitHub.

Comenzamos creando un nuevo repositorio en nuestra página de GitHub. Hacemos clic en el botón *New*:



Lo llamamos **murders** aquí para que coincida con el nombre del directorio en nuestro sistema local. Pero si están haciendo esto para otro proyecto, elijan un nombre apropiado.



Luego obtenemos una serie de instrucciones sobre cómo comenzar, pero en su lugar podemos usar lo que hemos aprendido. Lo principal que necesitamos de esta página es copiar el URL del repositorio, en este caso: <https://github.com/rairizarry/murders.git>.

En este momento, podemos abrir un terminal y `cd` a nuestro directorio de proyectos locales. En nuestro ejemplo, sería:

```
cd ~/projects/murders
```

Entonces, *inicializamos* el directorio. Esto convierte el directorio en un directorio Git y Git comienza a llevar cuenta:

```
git init
```

Todos los archivos ahora están **solo** en nuestro directorio de trabajo; no hay archivos en nuestro repositorio local o en GitHub.

El siguiente paso es conectar el repositorio local con el repositorio de GitHub. En un ejemplo anterior, hicimos que RStudio hiciera esto por nosotros. Ahora tenemos que hacerlo nosotros mismos. Podemos agregar cualquiera de los archivos y hacer *commit*:

```
git add README.txt
git commit -m "First commit. Adding README.txt file just to get started"
```

Ahora tenemos un archivo en nuestro repositorio local y podemos conectarlo al *Upstream Repository*, que tiene URL: <https://github.com/rairizarry/murders.git>.

Para hacer esto, usamos el comando `git remote add`.

```
git remote add origin `https://github.com/rairizarry/murders.git`
```

Ahora podemos usar `git push` ya que hay una conexión a un *Upstream Repository*:

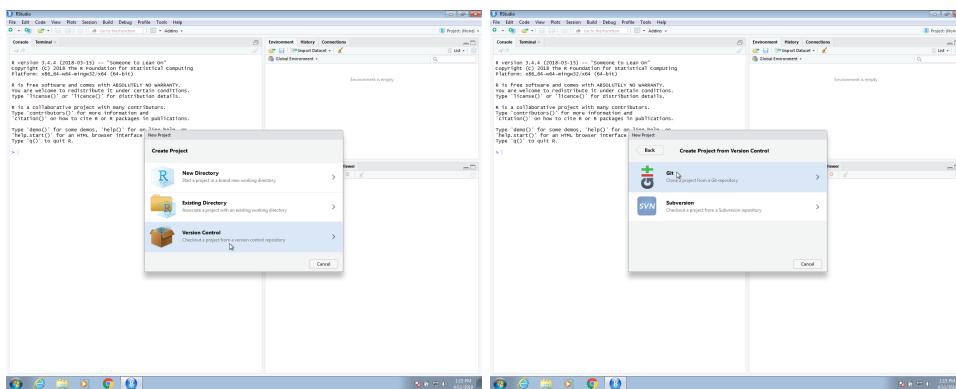
```
git push
```

En la Sección 40.3, continuamos trabajando con este ejemplo, mientras demostramos cómo podemos usar RStudio para trabajar con Git y mantener un proyecto sincronizado en GitHub.

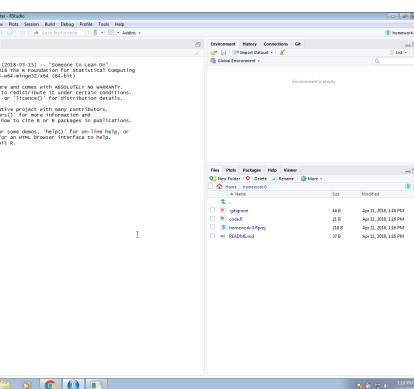
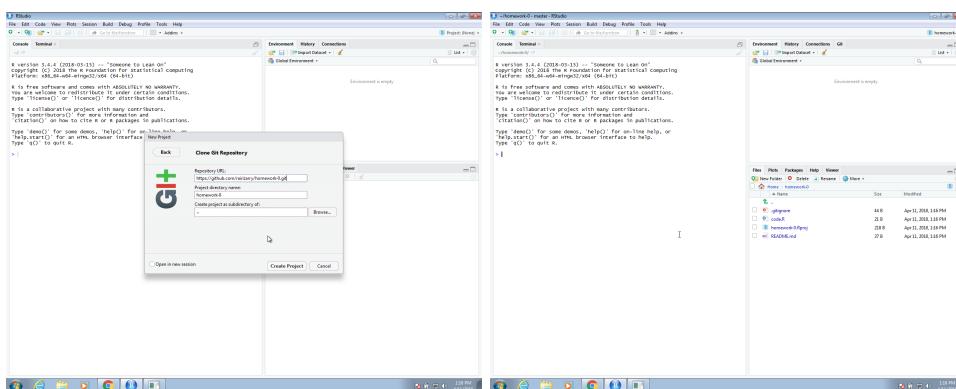
## 39.6 Usando Git y GitHub en RStudio

Si bien la línea de comandos Git es una herramienta eficaz y flexible, puede ser algo desalentador cuando estamos comenzando. RStudio proporciona una interfaz gráfica que facilita el uso de Git en el contexto de un proyecto de análisis de datos. Describimos cómo usar este atributo de RStudio para hacer esto aquí.

Ahora estamos listos para comenzar un proyecto de RStudio que usa control de versiones y almacena el código en un repositorio de GitHub. Para hacer esto, comenzamos un proyecto pero, en lugar de *New Directory*, seleccionaremos *Version Control* y luego seleccionaremos *Git* como nuestro sistema de control de versiones:

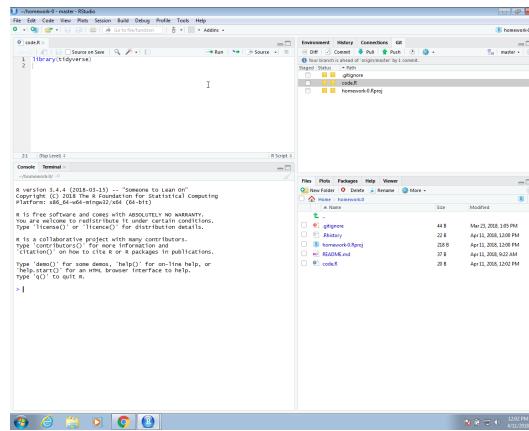


El URL del repositorio es el enlace que usaron para clonar. En la Sección 39.3, usamos <https://github.com/username/homework-0.git> como ejemplo. En el nombre del directorio del proyecto, deben poner el nombre de la carpeta que se generó, que en nuestro ejemplo será el nombre del repositorio `homework-0`. Esto creará una carpeta llamada `homework-0` en su sistema local. Una vez que hagan esto, se crea el proyecto y está al tanto de la conexión a un repositorio de GitHub. Verán en la esquina superior derecha el nombre y el tipo de proyecto, así como una nueva pestaña en el panel superior derecho titulada *Git*.



Si seleccionan esta pestaña, les mostrará los archivos de sus proyectos con algunos iconos que le brindan información sobre estos archivos y su relación con el repositorio. En el ejemplo a

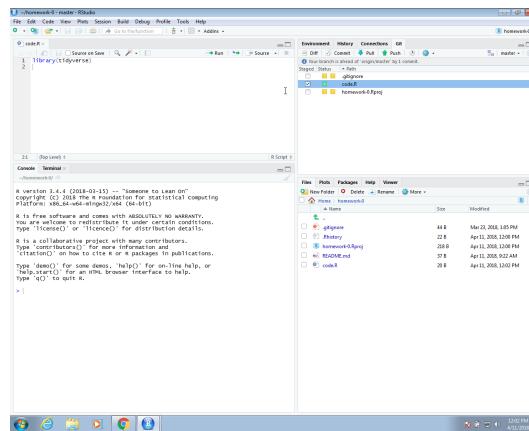
continuación, ya agregamos un archivo a la carpeta, llamado *code.R*, que pueden ver en el panel para editar.



Ahora debemos prestar atención al panel Git. Es importante saber que **sus archivos locales y el repositorio de GitHub no se sincronizarán automáticamente**. Como se describe en la Sección 39.4, deben sincronizar usando git push cuando estén listos. A continuación, les mostramos que pueden hacerlo a través de RStudio en lugar del terminal.

Antes de comenzar a trabajar en un proyecto colaborativo, generalmente lo primero que hacemos es *pull* los cambios desde el repositorio remoto, en nuestro caso el de GitHub. Sin embargo, para el ejemplo que se muestra aquí, dado que estamos comenzando con un repositorio vacío y somos los únicos que hacemos cambios, no tenemos que comenzar con *pull*.

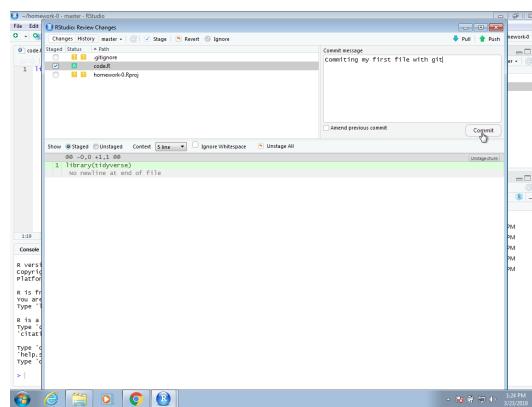
En RStudio, el status del archivo en relación con los repositorios remotos y locales se representa en los símbolos de status con colores. Un cuadrado amarillo significa que Git no sabe nada sobre este archivo. Para sincronizar con el repositorio de GitHub, necesitamos *add* el archivo, luego *commit* el cambio a nuestro repositorio de Git local y entonces *push* el cambio al repositorio de GitHub. En este momento, el archivo está en nuestra computadora. Para añadir el archivo usando RStudio, hacemos clic en la caja *Stage*. Verán que el ícono de status ahora cambia a una A verde.



Nota: solo estamos añadiendo el archivo *code.R*. No necesariamente tenemos que añadir todos

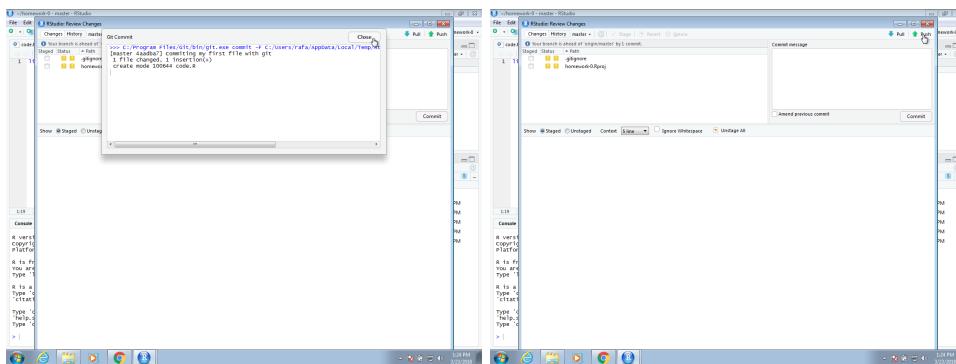
los archivos de nuestro repositorio local al repositorio de GitHub, solo los que queremos darle seguimiento o los que queremos compartir. Si nuestro trabajo está produciendo archivos de cierto tipo a los que no queremos darle seguimiento, podemos agregar el sufijo que define estos archivos al archivo `.gitignore`. Aquí pueden ver más detalles sobre el uso de `.gitignore`: <https://git-scm.com/docs/gitignore>. Estos archivos dejarán de aparecer en su panel RStudio Git. Para el ejemplo que se muestra aquí, solo agregaremos `code.R`. Pero, en general, para un proyecto de RStudio, recomendamos agregar los archivos `.gitignore` y `.Rproj`.

Ahora estamos listos para enviar el archivo a nuestro repositorio local. En RStudio, podemos usar el botón *Commit*. Esto abrirá una nueva ventana de diálogo. Con Git, cada vez que hacemos *commit* a un cambio, debemos ingresar un comentario que describe los cambios.

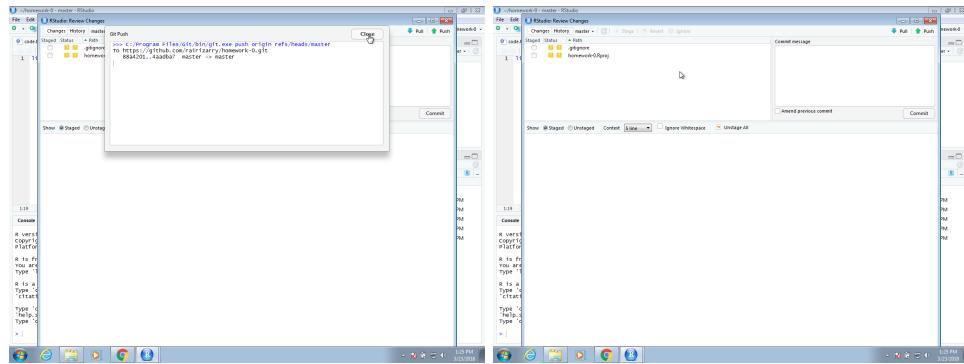


En este caso, simplemente escribimos que estamos agregando un nuevo script. En este ventana de diálogo, RStudio también le ofrece un resumen de lo que está cambiando al repositorio de GitHub. Aquí, como es un archivo nuevo, todo el archivo se resalta en verde, que resalta los cambios.

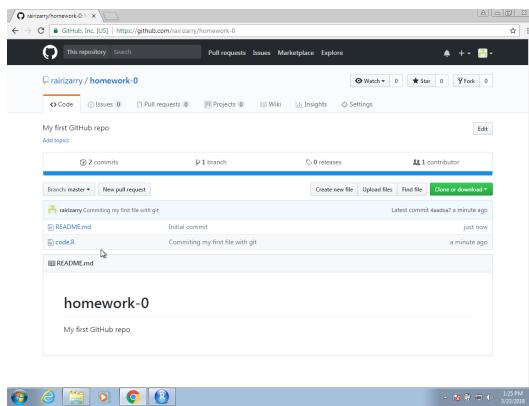
Una vez que presionen el botón *Commit*, deben ver un mensaje de Git con un resumen de los cambios que se confirmaron. Ahora estamos listos para empujar estos cambios al repositorio de GitHub. Podemos hacer esto haciendo clic en el botón *Push* en la esquina superior derecha:



Ahora vemos un mensaje de Git que nos dice que el *push* ha sido exitoso. En la ventana emergente ya no vemos el archivo `code.R`. Esto se debe a que no se han realizado nuevos cambios desde la última vez que hicimos *push*. Podemos salir de esta ventana emergente ahora y continuar trabajando en nuestro código.



Si ahora visitamos nuestro repositorio en la web, veremos que coincide con nuestra copia local.



¡Felicitaciones, han compartido código con éxito en un repositorio de GitHub!



# 40

---

## *Proyectos reproducibles con RStudio y R Markdown*

---

El producto final de un proyecto de análisis de datos frecuentemente es un informe. Muchas publicaciones científicas pueden considerarse como un informe final de un análisis de datos. Lo mismo es cierto para los artículos de noticias basados en datos, un informe de análisis para su empresa o notas para una clase sobre cómo analizar datos. Los informes suelen estar en papel o en un PDF que incluyen una descripción textual de los resultados junto con algunos gráficos y tablas resultantes del análisis.

Imaginen que después de finalizar el análisis y el informe, les informan que les dieron el set de datos incorrecto, les enviarán uno nuevo y les piden que ejecuten el mismo análisis con este nuevo set de datos. ¿O qué pasa si se dan cuenta de que cometieron un error y necesitan volver a examinar el código, corregir el error y volver a ejecutar el análisis? ¿O supongan que alguien a quien están entrenando quiere ver el código y poder reproducir los resultados para conocer su enfoque?

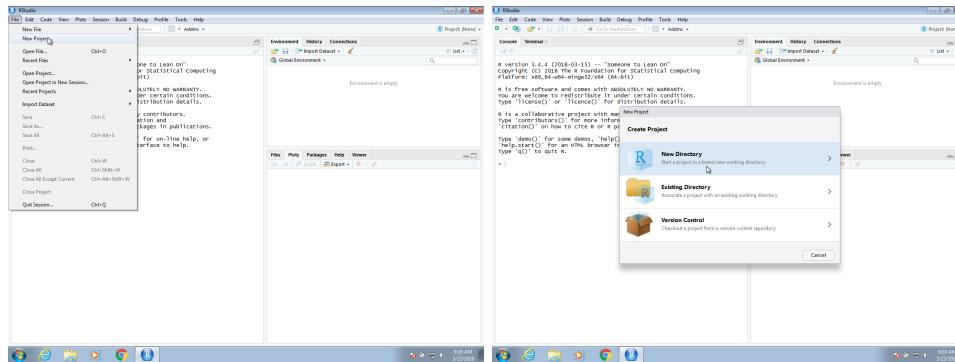
Situaciones como las que acabamos de describir son bastante comunes para los científicos de datos. Aquí, describimos cómo pueden mantener sus proyectos de ciencia de datos organizados con RStudio para que el proceso de volver a ejecutar un análisis sea sencillo. Luego, demostraremos cómo generar informes reproducibles con R Markdown y el paquete **knitr** de una manera que ayudará enormemente a recrear informes con esfuerzo mínimo. Esto es posible debido a que los documentos de R Markdown permiten combinar códigos y descripciones textuales en el mismo documento, y las figuras y tablas producidas por el código se agregan automáticamente al documento.

---

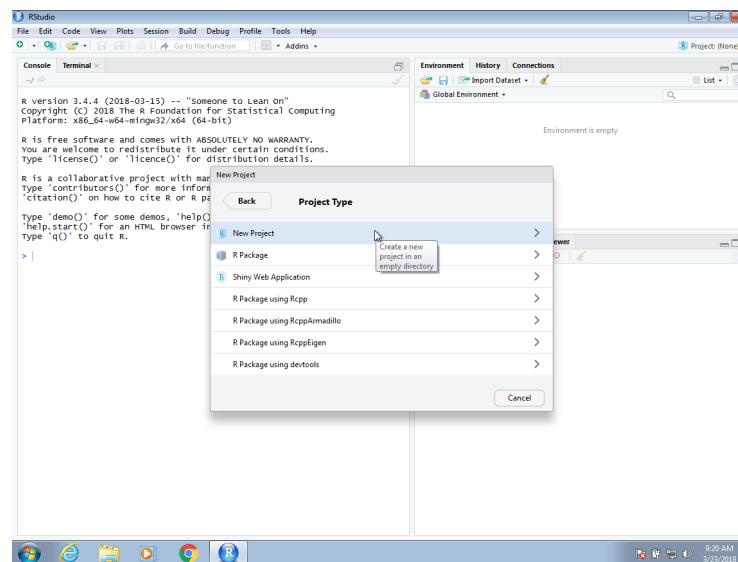
### 40.1 Proyectos de RStudio

RStudio ofrece una manera de mantener todos los componentes de un proyecto de análisis de datos organizados en una carpeta y mantener un registro de la información sobre este proyecto, como el estatus Git de los archivos, en un archivo. En la Sección 39.6, demostramos cómo RStudio facilita el uso de Git y GitHub a través de proyectos de RStudio. En esta sección, demostramos cómo comenzar un nuevo proyecto y ofrecemos algunas recomendaciones sobre cómo mantenerlo organizado. Los proyectos de RStudio también les permiten tener abiertas varias sesiones de RStudio y mantener un registro de cuál es cuál.

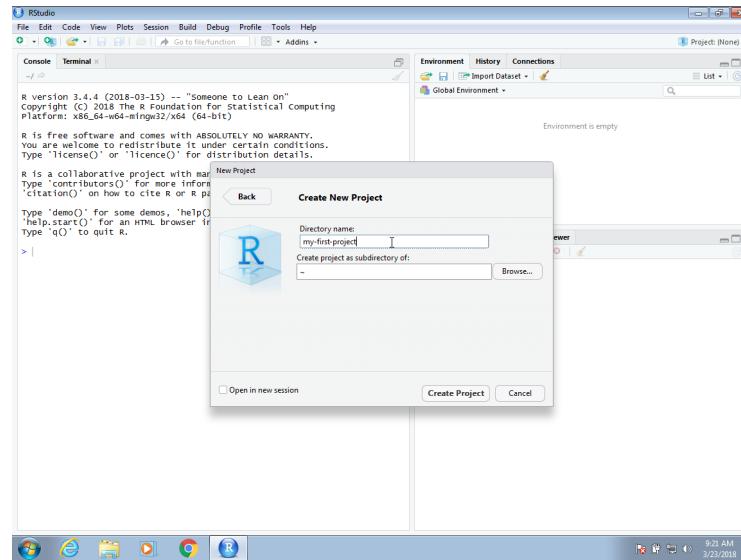
Para comenzar un proyecto, hagan clic en *File* y luego en *New Project*. Muchas veces ya hemos creado una carpeta para guardar el trabajo, como lo hicimos en la Sección 38.7, en cual caso seleccionamos *Existing Directory*. Aquí les mostramos un ejemplo en el que aún no hemos creado una carpeta y seleccionamos la opción *New Directory*.



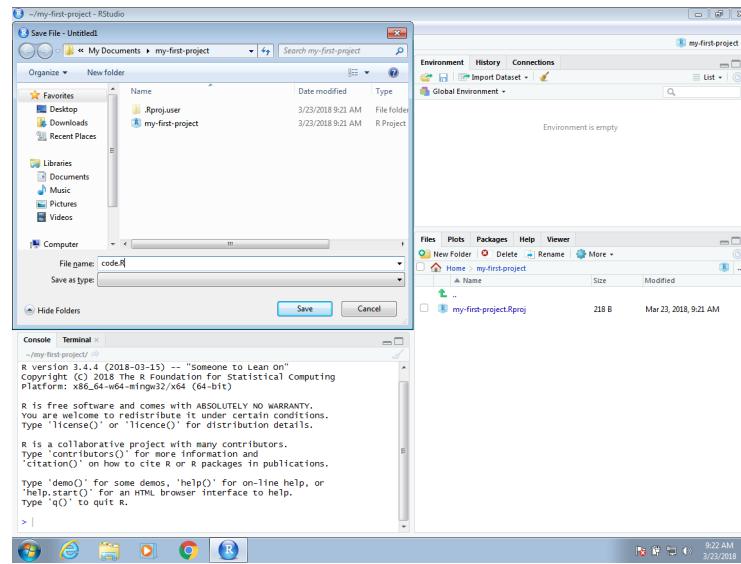
Entonces para un proyecto de análisis de datos, generalmente seleccionan la opción *New Project*:



Ahora tendrán que decidir la ubicación de la carpeta que se asociará con su proyecto, así como el nombre de la carpeta. Al igual que con los nombres de archivo, cuando eligen un nombre de carpeta, aasegúrense de que sea un nombre significativo que los ayude a recordar de qué se trata el proyecto. Además, al igual que con los archivos, recomendamos usar letras minúsculas, sin espacios y con guiones para separar las palabras. Llamaremos a la carpeta para este proyecto *my-first-project*. Esto generará un archivo *Rproj* llamado *my-first-project.Rproj* en la carpeta asociada con el proyecto. Veremos cómo esto es útil más abajo.



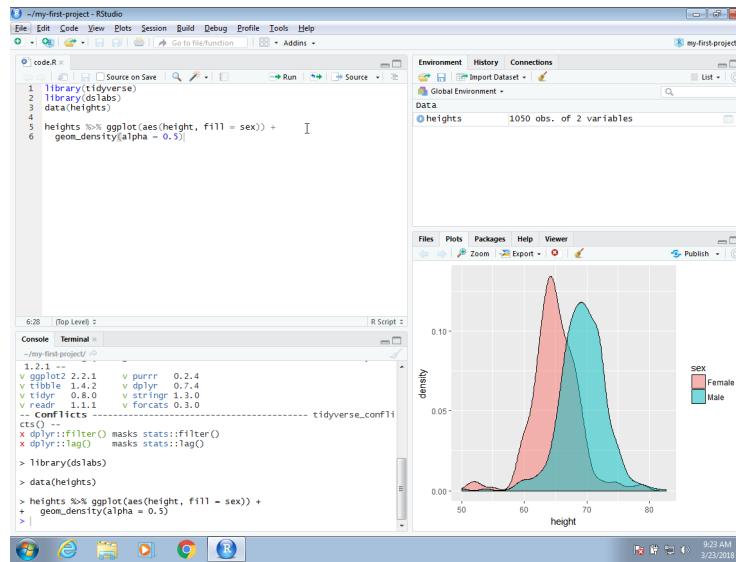
Les dará opciones sobre dónde debe estar esta carpeta en su sistema de archivos. En este ejemplo, lo colocaremos en nuestra carpeta de inicio, aunque esto generalmente no es una buena práctica. Como describimos en la Sección 38.7 en el capítulo de Unix, quieren organizar su sistema de archivos siguiendo un enfoque jerárquico y con una carpeta llamada *proyectos* donde guardarán una carpeta para cada proyecto.



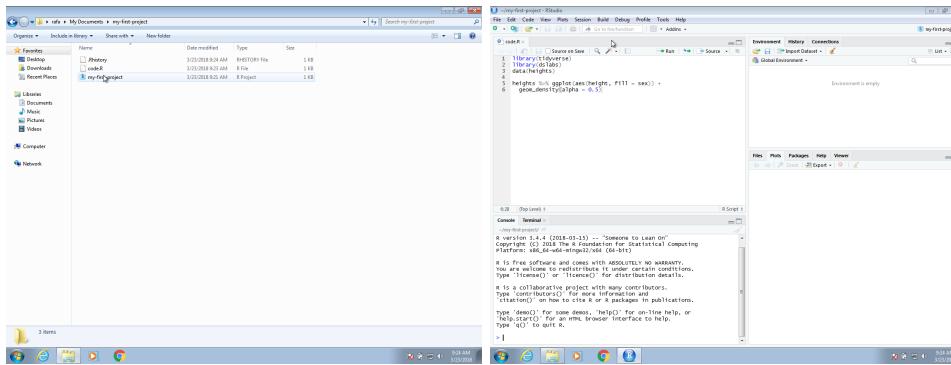
Cuando comiencen a usar RStudio con un proyecto, verán el nombre del proyecto en la esquina superior izquierda. Esto les recordará a qué proyecto pertenece esta sesión de RStudio en particular. Cuando abran una sesión de RStudio sin ningún proyecto, dirá *Project:(None)*.

Al trabajar en un proyecto, todos los archivos se guardarán y se buscarán en la carpeta asociada con el proyecto. A continuación, mostramos un ejemplo de un *script* que escribimos y guardamos con el nombre *code.R*. Como usamos un nombre significativo para el proyecto, podemos ser un poco menos informativos cuando nombramos los archivos. Aunque no lo

hacemos aquí, pueden tener varios *scripts* abiertos a la vez. Simplemente necesitan hacer clic en *File*, luego en *New File* y entonces elegir el tipo de archivo que desean editar.



Una de las principales ventajas de usar *Projects* es que después de cerrar RStudio, si deseamos continuar donde pausamos, simplemente hacemos doble clic o abrimos el archivo guardado cuando creamos el proyecto de RStudio. En este caso, el archivo se llama *my-first-project.Rproj*. Si lo abrimos, RStudio se iniciará y abrirá los *scripts* que estabamos editando.



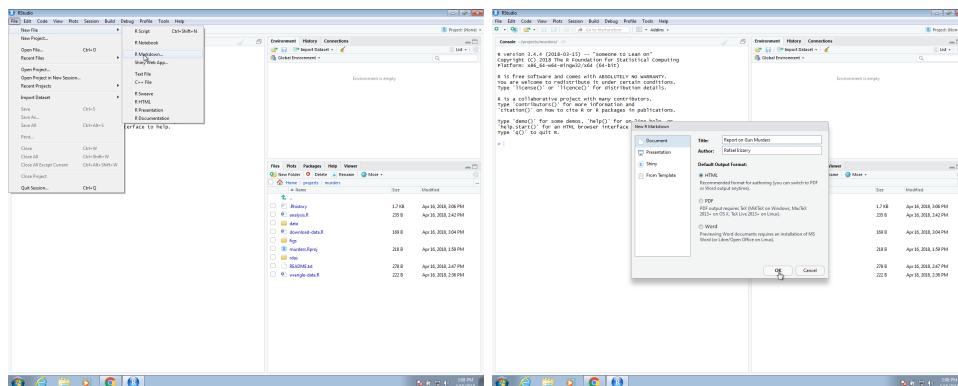
Otra ventaja es que si hacen clic en dos o más archivos diferentes de Rproj, iniciará nuevas sesiones de RStudio y R para cada uno.

## 40.2 R Markdown

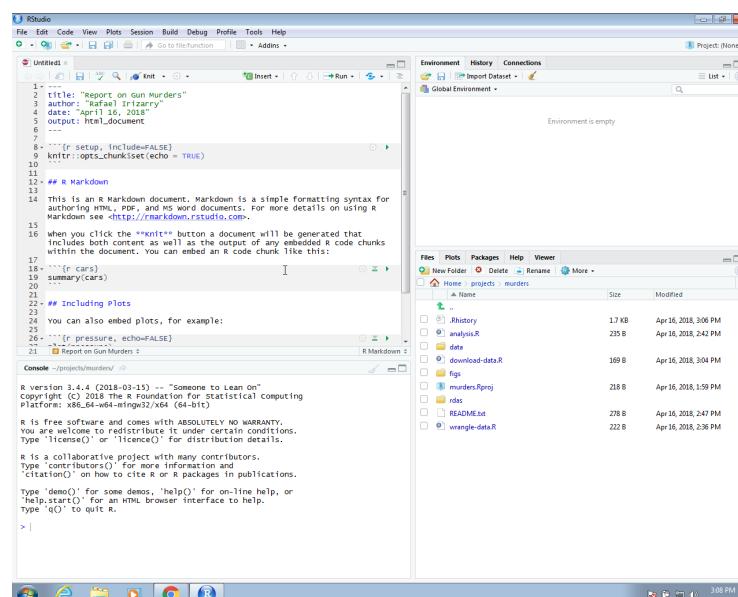
R Markdown es un formato para documentos de *programación literaria* (*literate programming* en inglés). Se basa en *markdown*, un lenguaje de *markup* que se usa ampliamente para

generar páginas HTML<sup>1</sup>. La programación literaria teje instrucciones, documentación y comentarios detallados entre el código ejecutable de la máquina, produciendo un documento que describe el programa que es mejor para la comprensión humana (Knuth 1984). A diferencia de un procesador de textos, como Microsoft Word, donde lo que ven es lo que obtienen, con R Markdown, necesitan *compilar* el documento en el informe final. El documento de R Markdown se ve diferente al producto final. De primer instancia, esto puede parecer una desventaja. Sin embargo, no lo es ya que en vez de producir gráficos e insertarlos uno por uno en el documento de procesamiento de texto, los gráficos se agregan automáticamente.

En RStudio, puede iniciar un documento de R Markdown haciendo clic en *File*, *New File* y entonces *R Markdown*. Luego se les pedirá que ingresen un título y un autor para su documento. Vamos a preparar un informe sobre asesinatos con armas de fuego, por lo que le daremos un nombre apropiado. También pueden escoger el formato del informe final: HTML, PDF o Microsoft Word. Más adelante, podemos cambiar esto fácilmente, pero aquí seleccionamos HTML ya que es el formato preferido para propósitos de depuración:



Esto generará un archivo de plantilla:



<sup>1</sup> <https://www.markdowntutorial.com/>

Como convención, usamos el sufijo `Rmd` para estos archivos.

Una vez que tengan más experiencia con R Markdown, podrán hacer esto sin la plantilla y simplemente comenzarán desde una plantilla en blanco.

En la plantilla, verán varias cosas para considerar.

### 40.2.1 El encabezado

En la parte superior ven:

```

```

```
title: "Report on Gun Murders"
author: "Rafael Irizarry"
date: "April 16, 2018"
output: html_document

```

Todo lo que está entre `---` es el encabezado. No necesitamos un encabezado, pero a menudo es útil. Pueden definir muchas otras cosas en el encabezado además de lo que se incluye en la plantilla. No discutimos esos aquí, pero hay mucha información disponible en línea. El único parámetro que destacaremos es `output`. Al cambiar esto a, por ejemplo, `pdf_document`, podemos controlar el tipo de `output` que se produce cuando compilamos.

### 40.2.2 Fragmentos de código R

En varios lugares del documento, vemos algo como lo siguiente:

```
```{r}
summary(pressure)
```
```

Estos son fragmentos de código. Cuando compilan el documento, el código R dentro del fragmento, en este caso `summary(pressure)`, será evaluado y el resultado incluido en esa posición en el documento final.

Para añadir sus propios fragmentos de R, pueden escribir los caracteres de arriba rápidamente con la combinación de teclas opción-I en Mac y Ctrl-Alt-I en Windows.

Esto aplica también a los gráficos; el gráfico se colocará en esa posición. Podemos escribir algo como esto:

```
```{r}
plot(pressure)
```
```

Por defecto, el código también aparecerá. Para evitar esto, pueden usar el argumento `echo=FALSE`. Por ejemplo:

```
```{r echo=FALSE}
summary(pressure)
```
```

Recomendamos acostumbrarse a añadir una etiqueta a los fragmentos de código R. Esto será muy útil al depurar, entre otras situaciones. Para hacer esto, agreguen una palabra descriptiva como esta:

```
```{r pressure-summary}
summary(pressure)
```
```

### 40.2.3 Opciones globales

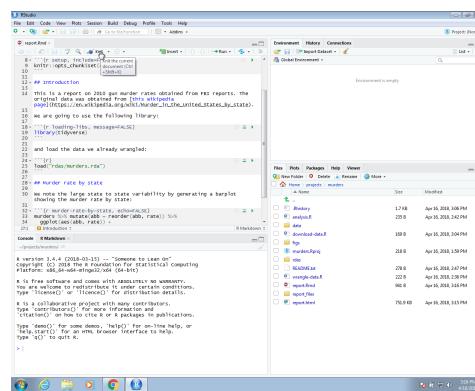
Uno de los fragmentos de R contiene una llamada que parece complicada:

```
```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```
```

No cubriremos esto aquí, pero a medida que tengan más experiencia con R Markdown, aprenderán las ventajas de establecer opciones globales para el proceso de compilación.

### 40.2.4 knitR

Usamos el paquete **knitR** para compilar documentos de R Markdown. La función específica utilizada para compilar es **knit**, que toma un nombre de archivo como entrada. RStudio provee un botón que facilita la compilación del documento. En la siguiente captura de pantalla, hemos editado el documento para que se produzca un informe sobre asesinatos con armas de fuego. Pueden ver el archivo aquí: <https://raw.githubusercontent.com/rairizarry/murders/master/report.Rmd>. Ahora pueden hacer clic en el botón *Knit*:



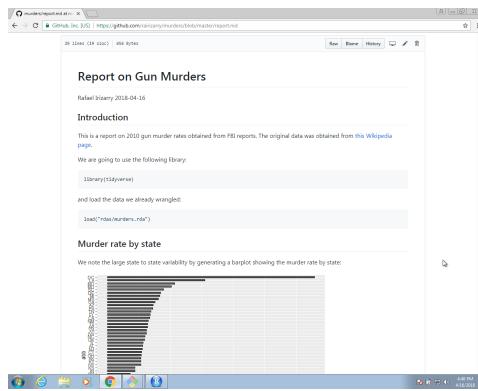
La primera vez que hacen clic en el botón *Knit*, puede aparecer un cuadro de diálogo pidiéndoles que instalen los paquetes que necesitan.

Una vez que hayan instalado los paquetes, al hacer clic en *Knit* se compilará su archivo de R Markdown y emergerá el documento resultante.

Esto produce un documento HTML que pueden ver en su directorio de trabajo. Para verlo, abran un terminal y enumeren los archivos. Pueden abrir el archivo en un navegador y usarlo para presentar su análisis. Además, pueden producir un documento PDF o de Microsoft cambiando:

```
output: html_document a output: pdf_document o output: word_document.
```

También podemos producir documentos que se procesan en GitHub usando `output: github_document`. Esto producirá un archivo de Markdown, con sufijo `md`, que se puede visualizar como una página de web en GitHub. Como hemos subido estos archivos a GitHub, pueden hacer clic en el archivo `md` y verán el informe:



Esta es una manera conveniente de compartir sus informes.

#### 40.2.5 Más sobre R Markdown

Hay mucho más que pueden hacer con R Markdown. Le recomendamos que continúen aprendiendo a medida que adquieran más experiencia escribiendo informes en R. Hay muchos recursos gratuitos en el Internet que incluyen:

- Tutorial de RStudio: <https://rmarkdown.rstudio.com>
- La hoja de referencia: [https://github.com/rstudio/cheatsheets/raw/master/translations/spanish/rmarkdown-2.0\\_Spanish.pdf](https://github.com/rstudio/cheatsheets/raw/master/translations/spanish/rmarkdown-2.0_Spanish.pdf)
- El libro de knitr: <https://yihui.name/knitr/>

---

### 40.3 Organizando un proyecto de ciencia de datos

En esta sección, juntamos todo para crear el proyecto de asesinatos de EE.UU. y compartirlo en GitHub.

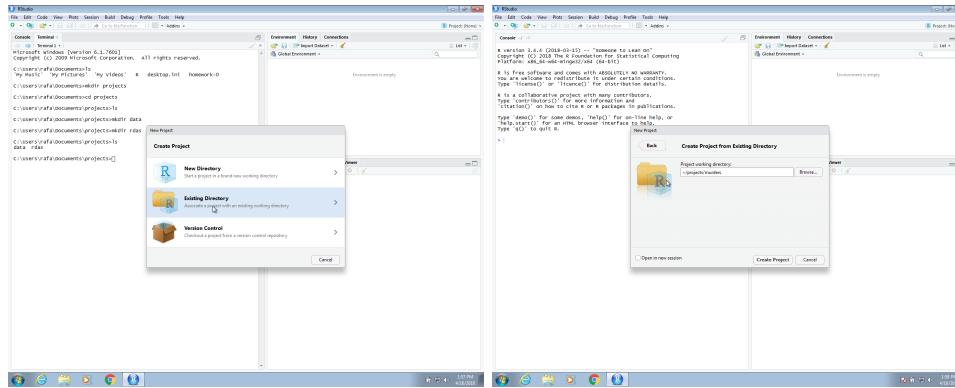
### 40.3.1 Crear directorios en Unix

En la Sección 38.7, demostramos cómo usar Unix para prepararnos para un proyecto de ciencia de datos usando un ejemplo. Aquí continuamos con este ejemplo y mostramos cómo usar RStudio. En la Sección 38.7, creamos los siguientes directorios usando Unix:

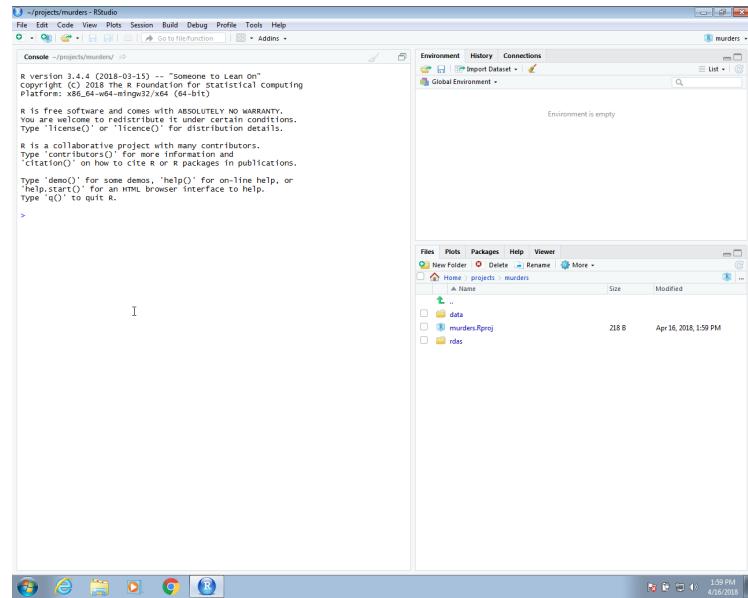
```
cd ~
cd projects
mkdir murders
cd murders
mkdir data rdas
```

### 40.3.2 Crear un proyecto en RStudio

En la siguiente sección, crearemos un proyecto en RStudio. Primero vamos a *File* y luego a *New Project...* y, al ver las opciones, elegimos *Existing Directory*. Entonces escribimos la ruta completa del directorio **murders** creado anteriormente.



Una vez que hagan esto, verán los directorios **rdas** y **data** que crearon en la pestaña *Files* de RStudio.



Tengan en cuenta que cuando estamos en este proyecto, nuestro directorio de trabajo predeterminado será `~/projects/murders`. Pueden confirmar esto escribiendo `getwd()` en su sesión de R. Esto es importante porque nos ayudará a organizar el código cuando necesitemos escribir rutas de archivos. Consejo profesional: siempre usen rutas relativas en el código para proyectos de ciencia de datos. Estas deben ser relativas al directorio de trabajo que ha sido predeterminado. El problema con el uso de rutas completas es que es poco probable que sus códigos funcionen en sistemas de archivos distintos a los suyos, ya que las estructuras de directorio serán diferentes. Esto incluye el uso del directorio `home ~` como parte de su ruta.

### 40.3.3 Editar algunos scripts de R

Ahora escribamos un *script* que descargue un archivo en el directorio de datos. Llamaremos a este archivo `download-data.R`.

El contenido de este archivo será:

```
url <- "https://raw.githubusercontent.com/rafalab/dslabs/master/inst/
extdata/murders.csv"
dest_file <- "data/murders.csv"
download.file(url, destfile = dest_file)
```

Recuerden que estamos utilizando la ruta relativa `data/murders.csv`.

Ejecuten este código en R y verán que se agrega un archivo al directorio `data`.

Ahora estamos listos para escribir un *script* para leer estos datos y preparar una tabla que podamos usar para el análisis. Llamen al archivo `wrangle-data.R`. El contenido de este archivo será:

```
library(tidyverse)
murders <- read_csv("data/murders.csv")
```

```
murders <-murders %>% mutate(region = factor(region),
 rate = total/ population * 10^5)
save(murders, file = "rdas/murders.rda")
```

Una vez más, tengan en cuenta que utilizamos rutas relativas exclusivamente.

En este archivo, presentamos un comando R que no hemos visto antes: **save**. El comando **save** guarda objetos en lo que se llama un *archivo rda*: *rda* es la abreviatura de datos de R. Recomendamos usar el sufijo **.rda** en archivos que guardan objetos R. Verán que **.RData** también se usa.

Si ejecutan el código anterior, el objeto de datos procesados se guardará en un archivo en el directorio **rda**. Aunque no es el caso aquí, este enfoque a menudo es práctico porque generar el objeto de datos que usamos para los análisis y gráficos finales puede ser un proceso complejo que requiere mucho tiempo. Entonces ejecutamos este proceso una vez y guardamos el archivo. Pero aún queremos poder generar el análisis completo a partir de los datos sin procesar.

Ahora estamos listos para escribir el archivo de análisis. Vamos a llamarlo **analysis.R**. El contenido debe ser el siguiente:

```
library(tidyverse)
load("rdas/murders.rda")

murders %>% mutate(abb = reorder(abb, rate)) %>%
 ggplot(aes(abb, rate)) +
 geom_bar(width = 0.5, stat = "identity", color = "black") +
 coord_flip()
```

Si ejecutan este análisis, verán que genera un gráfico.

#### 40.3.4 Crear más directorios usando Unix

Ahora supongan que queremos guardar el gráfico generado para un informe o presentación. Podemos hacer esto con el comando **ggsave** de **ggplot**. ¿Pero dónde ponemos el gráfico? Deberíamos organizarnos sistemáticamente para que podamos guardar los gráficos en un directorio llamado **figs**. Comiencen creando un directorio escribiendo lo siguiente en el terminal:

```
mkdir figs
```

y luego pueden agregar la línea:

```
ggsave("figs/barplot.png")
```

a su *script* de R. Si ejecutan el *script* ahora, se guardará en un archivo **png** en el directorio **figs**. Si queremos copiar ese archivo a otro directorio donde estamos desarrollando una presentación, podemos evitar usar el mouse usando el comando **cp** en nuestro terminal.

### 40.3.5 Agregar un archivo README

Ahora tienen un análisis autónomo en un directorio. Una recomendación final es crear un archivo `README.txt` que describe lo que cada uno de estos archivos hace para el beneficio de otros que lean su código, incluyendo ustedes en el futuro. Esto no sería un *script*, sino solo algunas notas. Una de las opciones proporcionadas al abrir un nuevo archivo en RStudio es un archivo de texto. Puede guardar algo como esto en el archivo de texto:

```
We analyze US gun murder data collected by the FBI.
```

```
download-data.R - Downloads csv file to data directory
```

```
wrangle-data.R - Creates a derived dataset and saves as R object in rdas
directory
```

```
analysis.R - A plot is generated and saved in the figs directory.
```

### 40.3.6 Inicializando un directorio Git

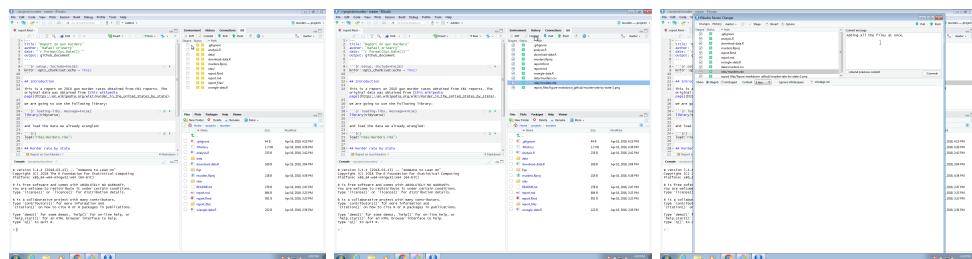
En la Sección 39.5, demostramos cómo inicializar un directorio Git y conectarlo al *Upstream Repository* en GitHub, que ya creamos en esa sección.

Podemos hacer esto en el terminal de Unix:

```
cd ~/projects/murders
git init
git add README.txt
git commit -m "First commit. Adding README.txt file just to get started"
git remote add origin `https://github.com/rairizarry/murders.git`
git push
```

### 40.3.7 Add, commit y push archivos con RStudio

Podemos continuar *adding* y *committing* cada archivo, pero podría ser más fácil usar RStudio. Para hacer esto, inicien el proyecto abriendo el archivo Rproj. Deben aparecer los iconos git y pueden *add*, *commit* y *push* con estos.



Ahora podemos ir a GitHub y confirmar que nuestros archivos están allí. Pueden ver una

versión de este proyecto, organizada con directorios de Unix, en GitHub<sup>2</sup>. Pueden descargar una copia a su computadora utilizando el comando `git clone` en su terminal. Este comando creará un directorio llamado `murders` en su directorio de trabajo, así que tengan cuidado desde dónde lo llama.

```
git clone https://github.com/rairizarry/murders.git
```

---

<sup>2</sup><https://github.com/rairizarry/murders>