R en las Ciencias Agropecuarias

Franklin Santos

2021-01-19

Contents

Bienvenido

Bienvenido a **R aplicada en las Ciencias Agropecuarias**. El contenido del libro presenta ejemplos de análisis exploratorio de datos, estadística inferencial y modelos de regresión. Generalmente, se expone las metodologías mas usuales en un reporte de trabajos de investigación y/o tesis.

Este libro es para aquellos que tienen poca o ninguna experiencia previa en programación en R o cualquier otro lenguaje de programación. Su objetivo es desarrollar los conceptos básicos y enseñarle las capacidades de R. Con la lectura y práctica, habrá adquirido una nueva habilidad valiosa para explorar conjuntos de datos y crear visualizaciones impresionantes.

0.1 Prerequisites

This is a *sample* book written in **Markdown**. You can use anything that Pandoc's Markdown supports, e.g., a math equation $a^2 + b^2 = c^2$.

The **bookdown** package can be installed from CRAN or Github:

```
install.packages("bookdown")
# or the development version
# devtools::install_github("rstudio/bookdown")
```

Remember each Rmd file contains one and only one chapter, and a chapter is defined by the first-level heading `#`.

To compile this example to PDF, you need XeLaTeX. You are recommended to install TinyTeX (which includes XeLaTeX): https://yihui.org/tinytex/.

Chapter 1

R básico

R es muy popular e increíblemente útil para las personas que trabajan como científicos de datos o en empresas. Pero también puede usar R para cosas más simples, como crear un gráfico agradable o hacer un cálculo rápido. Comenzar a utilizar R es bastante sencillo.

1.1 Introducción a R

> El lenguaje informático estadístico más potente del planeta. – Norman Nie, fundador de SPSS

R es un lenguaje de programación y un ambiente para trabajar con datos. Es muy utilizado por estadísticos y científicos de datos por su sintaxis de código expresivo, abundantes paquetes y herramientas externas y funciona en todos los sistemas operativos principales.

Es la navaja suiza para el análisis de datos y la computación estadística (¡y también puedes hacer algunos gráficos bonitos!). El lenguaje R se puede ampliar fácilmente con paquetes escritos por una gran y creciente comunidad de desarrolladores de todo el mundo. Puede encontrarlo prácticamente en cualquier lugar: lo utilizan instituciones académicas, empresas emergentes, corporaciones internacionales y muchos más.

1.1.1 ¿Por qué usar R?

R es un lenguaje popular para resolver problemas de análisis de datos y también lo usan personas que tradicionalmente no se consideran programadores. Al crear gráficos y visualizaciones con R, descubrirá que tiene muchas más posibilidades creativas que las aplicaciones gráficas, como Excel.

Estas son algunas de las **características** por las que R es más famoso:

Visualización: Crear gráficos y visualizaciones hermosos, es una de sus mayores fortalezas. El lenguaje central ya proporciona un amplio conjunto de herramientas utilizadas para trazar gráficos y para todo tipo de gráficos. El cielo es el límite.

Reproducibilidad: a diferencia del software de hoja de cálculo, el código R no está acoplado a conjuntos de datos específicos y puede reutilizarse fácilmente en diferentes proyectos, incluso cuando excede más de 1 millón de filas. Cree fácilmente informes reutilizables y genere automáticamente nuevas versiones a medida que cambien los datos.

Modelamiento avanzado: R proporciona la base de código más grande y poderosa para el análisis de datos en el mundo. La riqueza y profundidad de los modelos estadísticos disponibles no tiene paralelo y crece día a día, gracias a la gran comunidad de desarrolladores y colaboradores de paquetes de código abierto.

Automatización: el código R también se puede utilizar para automatizar informes o para realizar transformación de datos y cálculos de modelos. También se puede integrar en flujos de trabajo de producción automatizados, ambientes informáticos en la nube y sistemas de bases de datos modernos.

1.1.2 R en buena compañía

R es el estándar de facto para la computación estadística en instituciones académicas y empresas de todo el mundo. Su gran soporte para la programación alfabetizada (código que se puede combinar con texto legible por humanos) permite a los investigadores y científicos de datos crear informes listos para publicación que son fáciles de reproducir para los revisores.

El lenguaje ha tenido una amplia adopción en varias industrias; vea algunos ejemplos a continuación:

Tecnologías de la información

- Microsoft: Microsoft R Open, TrueSkill(TM), más aquí
- Google: R para investigación de mercado y análisis, prediciendo el presente con Google Trends
- Facebook: Visualizando amistades, La formación del amor, Paquete Prophet para pronósticos de series de tiempo.
- Otros (con enlaces a proyectos): AirBnB, Uber, Oracle, IBM, Twitter.

Farmacias: Merck, Genentech (Roche), Novartis, Pfizer **Periódicos**: The Economist, The New York Times, Financial Times

Finanzas

- Bancos: Bank of America, J.P.Morgan, Goldman Sachs, Credit Suisse, UBS, Deutsche Bank

- Seguros: Lloyd's, Allianz

1.2 Bloques de construcción

El lenguaje R consta de tres bloques fundamentales de construcción necesarios para realizar sus primeros pasos en el ambiente R:

- Objeto: todo lo que existe es un objeto
- Función: todo lo que sucede es una llamada a función
- Interfaz: R se basa en interfaces para muchos algoritmos

Los **objetos** más importantes en R son **vectores**. Forman la base de (casi) todas las estructuras de datos de R. La mayor parte del poder y la expresividad de R se deriva del hecho de que es un lenguaje orientado a vectores. Las **funciones** y los operadores pueden definirse fácilmente y trabajar directamente en vectores para calcular resultados.

La mayor fortaleza de R es su flexibilidad para integrar fácilmente nuevos algoritmos y construir **interfaces** a su alrededor. El sistema de paquetes R permite a los colaboradores integrar muchos modelos y bibliotecas externas de código abierto. Su repositorio principal de paquetes llamado "**CRAN**" aloja estos paquetes y permite a los usuarios instalarlos y cargarlos fácilmente en sus ambientes.

1.3 Combinar valores en un vector

R siempre crea listas de valores, incluso cuando solo hay un valor en una lista. Estas listas se denominan vectores y facilitan mucho el trabajo con datos.

- Todo es un vector
- Conozca los diferentes tipos de datos en R
- Aprenda a crear vectores
- Utilice el operador `:` para crear secuencias numéricas
- Utilice la función de concatenar `c()` para crear vectores de diferentes tipos de datos

```
1:100
c(1, 2, 3, 4)
c("abc", "def", "ghi")
c(TRUE, FALSE, TRUE)
```

1.3.1 Introducción a los vectores

Un vector es una colección de elementos del mismo tipo y la estructura de datos más básica en R. Por ejemplo, un vector podría contener los cuatro números `1`, `3`, `2` y `5`. Otro vector podría formarse con las tres cadenas de

texto `"Bienvenido"`,`"Hi"` y `"Hola"`. Estos diferentes tipos de valores (números, texto) se denominan *tipos de datos*.

Un valor único también se trata como un vector, un vector con un solo elemento. Como veremos a lo largo del curso, este concepto hace que R sea muy especial. Podemos manipular los vectores y sus valores a través de muchas operaciones proporcionadas por R.

Una ventaja clave de los vectores es que podemos aplicar una operación (por ejemplo, una multiplicación) a todos sus valores a la vez en lugar de pasar por cada elemento individualmente. Esto se llama *vectorización*.

1.3.2 Tipos de vectores

Los vectores solo pueden contener elementos del mismo *tipo de datos*. En este libro trabajaremos con los siguientes tres tipos de datos principales:

Los valores **numéricos** son números. Aunque pueden dividirse aún más en números enteros (enteros) y números con decimales (dobles), R convierte automáticamente entre estos subtipos si es necesario. Por lo tanto, colectivamente nos referiremos a ellos como valores `numéricos`.

Los valores de **caracteres** contienen contenido textual. Estos pueden ser letras, símbolos, espacios y números también. Deben estar entre comillas: comillas simples `'___'` o comillas dobles `"___"`.

Los valores **lógicos** pueden ser `TRUE` o `FALSE`. A menudo también se les conoce como valores *booleanos* o *binarios*. Debido a que un valor `lógico` solo puede ser `TRUE` o `FALSE`, se usan con mayor frecuencia para responder preguntas simples como "¿Es 1 mayor que 2?" o "¿Son más de las 3 en punto?". Este tipo de preguntas solo necesitan respuestas como "Sí" (`TRUE`) o "No" (`FALSE`). Es importante destacar que en R los valores `lógicos` distinguen entre mayúsculas y minúsculas, lo que significa que deben escribirse con mayúsculas.

1.3.3 Creando una secuencia de números

En R, incluso un solo valor se considera un vector. Crear un vector de un elemento es tan simple como escribir su valor:

`Código de entrada:`

```
4
```

`Resultado:`

```
## [1] 4
```

Para crear una secuencia de valores numéricos, podemos usar el operador `:` que toma dos números y genera un vector de todos los números enteros en ese rango:

`Código de entrada:`

```
2:11
```

Resultado:

```
##  [1]  2  3  4  5  6  7  8  9 10 11
```

El operador `:` crea un vector desde el número en el lado izquierdo hasta el número en el lado derecho. Por lo tanto, el orden de los números es importante. Si definimos el ejemplo anterior al revés, obtenemos un vector de números descendentes, en lugar de ascendentes:

Código de entrada:

```
11:2
```

Resultado:

```
##  [1] 11 10  9  8  7  6  5  4  3  2
```

El operador `:` es útil cuando necesitamos un vector de cada número entero en un rango dado. Sin embargo, si necesitamos un vector donde los números no sean lineales, necesitamos algo diferente.

1.3.4 Concatenando valores numéricos a un vector

Podemos combinar múltiples números en un solo vector usando la función concatenar `c()` que une elementos entre las llaves redondas en una cadena o mas conocido como paréntesis. Múltiples elementos deben estar separados por comas.

Para crear nuestro primer vector con siete números diferentes, podemos usar la función de concatenación `c()` de esta manera:

Código de entrada:

```
c(7, 4, 2, 5, 5, 22, 1)
```

Resultado:

```
## [1]  7  4  2  5  5 22  1
```

Tenga en cuenta que el signo "`[1]`" antes de la salida anterior es agregado por R, y siempre se agrega automáticamente al imprimir vectores. Si sus vectores se hacen más grandes, verá más de estos prefijos. Solo sepa que R solo los agrega con fines informativos, y que están allí para ayudarlo mientras codifica. No son parte del vector en sí.

Puede ver esto más claramente, cuando la salida se extiende sobre varias líneas:

Código de entrada:

```
1:60
```

Resultado:

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
## [26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
## [51] 51 52 53 54 55 56 57 58 59 60
```

1.3.5 Creando vectores de caracteres

Para crear un vector de caracteres de un elemento, todo lo que tenemos que hacer es escribir el texto. Recuerde que debemos usar comillas (`" "`) alrededor de los valores de los caracteres:

Código de entrada:

```
"golden retriever"
```

Resultado:

```
## [1] "golden retriever"
```

Para crear un vector de caracteres de múltiples elementos, podemos usar nuevamente la función concatenar `c()`. Esta vez lo usaremos con caracteres en lugar de números:

Código de entrada:

```
c("golden retriever", "labrador is a family dog", "beagle")
```

Resultado:

```
## [1] "golden retriever"         "labrador is a family dog"
## [3] "beagle"
```

1.3.6 Creando vectores lógicos

Los vectores lógicos solo pueden contener los valores `TRUE` y `FALSE`. Para crear un vector lógico con un solo valor, escriba uno de los valores válidos `TRUE` o `FALSE`. Recuerde que deben escribirse con letras mayúsculas:

Código de entrada:

```
TRUE
```

Resultado:

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

De manera similar a otros tipos de vectores, podemos usar la función concatenar `c()` para crear un vector lógico de múltiples elementos:

Código de entrada:

```
c(TRUE, FALSE, TRUE, FALSE, TRUE)
```

Resultado:

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

1.4 Crea variables a través de asignaciones

Por lo general, desea almacenar vectores y otros objetos en variables para poder trabajar con ellos más fácilmente. Las variables son como un cuadro con un nombre. A continuación, puede consultar el nombre para ver qué se almacena en el interior.

- Aprenda a crear una variable
- Usa variables para almacenar objetos y vectores
- Reutilizar objetos asignados mediante un nombre de variable

1.4.1 Asignando variables

Por lo general, queremos usar objetos como vectores más de una vez. Para evitar la molestia de volver a escribirlos y recrearlos todo el tiempo, nos gustaría guardarlos en algún lugar y reutilizarlos más tarde.

Para hacer esto, podemos asignarlos a un nombre de variable. R usa el operador de flecha especial `<-` para asignar valores a una variable. La flecha es simplemente la combinación de un carácter menor que (`<`) y un signo menos (`-`).

Veamos un ejemplo, en el que asignamos un vector numérico a una variable llamada `numbers`:

Código de entrada:

```
numbers <- c(1, 2, 3, 4, 5, 6, 7, 8, 9)
```

Ahora podemos usar el nombre de la variable a continuación para ver su contenido:

```
numbers
```

Resultado:

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

Tenga en cuenta que cuando asignamos algo a una variable que ya existe, se sobrescribe. Todos los contenidos anteriores se eliminan automáticamente:

Código de entrada:

```
numbers <- c(10, 11, 12, 13)
numbers
```

Resultado:

```
## [1] 10 11 12 13
```

Una vez que haya definido una variable, puede usarla tal como usaría el vector subyacente. En el siguiente ejemplo creamos dos vectores numéricos y los asignamos a las variables `low` y `high`. Luego, usamos estas variables y concatenamos los dos vectores en uno solo y los asignamos una variable denominada `sequence`. Finalmente llamamos a la variable `sequence` e inspeccionamos su contenido:

`Código de entrada:`

```
low <- c(1, 2, 3)
high <- c(4, 5, 6)
sequence <- c(low, high)
sequence
```

`Resultado:`

```
## [1] 1 2 3 4 5 6
```

Como puede ver, los vectores `1, 2, 3` y `4, 5, 6` almacenados en las variables `low` y `high`, se combinaron en un solo vector que ahora es el contenido de la variable `sequence`.

1.4.2 Reglas de nomenclatura

Hay algunas reglas que debemos tener en cuenta al crear variables.

Reglas en la asignación de variables

- Puede contener letras: `example`
- Puede contener números: `example1`
- Puede contener guiones bajos: `example_1`
- Puede contener puntos: `example.1`
- Puede contener punto Adelante seguido por una letra: `.task`
- **No** puede comenzar con números: `2example`
- **No** se puede comenzar con guiones bajos: `_example`
- **No** puede comenzar con un punto si va directamente seguido de un número: `.2example`

1.5 Utilice operadores básicos

R no solo es bueno para analizar y visualizar datos, sino también para resolver problemas matemáticos o comparar datos entre sí. Además, puede usarlo como una calculadora de bolsillo.

- Usa R como calculadora de bolsillo
- Usa operadores aritméticos en vectores
- Use operadores relacionales en vectores
- Usa operadores lógicos en vectores

1.5.1 Usando R como calculadora de bolsillo

```
--- + ---
--- - ---
--- / ---
--- * ---
```

R es un lenguaje de programación desarrollado principalmente para estadísticas y análisis de datos. Dentro de R puede usar operadores matemáticos tal como lo haría en una calculadora. Por ejemplo, puede sumar `+` y restar `-` números entre sí:

```
Código de entrada:
```

```
5 + 5
```

```
Resultado:
```

```
## [1] 10
```

```
7 - 3.5
```

```
## [1] 3.5
```

Del mismo modo, puede multiplicar `*` o dividir `/` números:

```
5 * 7
```

```
## [1] 35
```

```
8 / 4
```

```
## [1] 2
```

Puedes tomar el poder de un número usando el signo `^` para potenciar o elevar:

```
2^3
```

```
## [1] 8
```

De acuerdo con las reglas de las matemáticas, puede usar corchetes para especificar el orden de evaluación en tareas más complejas:

```
5 * (2 + 4 / 2)
```

```
## [1] 20
```

1.5.2 Aplicación de operadoras aritméticas en vectores

```
--- + ---
--- - ---
--- / ---
--- * ---
```

Las operaciones, como la suma, la resta, la multiplicación y la división se denominan *operaciones aritméticas*. No solo pueden operar con valores únicos sino también con vectores. Si usa operaciones aritméticas en vectores, la operación se realiza en cada número individual del primer vector y el número individual en la misma posición del segundo vector.

En el siguiente ejemplo creamos dos vectores numéricos y los asignamos a las variables `a` y `b`. Luego los agregamos juntos:

```
a <- c(1, 3, 6, 9, 12, 15)
b <- c(2, 4, 6, 8, 10, 12)
a + b
```

```
## [1]  3  7 12 17 22 27
```

Como muestra la salida, los primeros elementos de los dos vectores se agregaron juntos y dieron como resultado `1 + 2 = 3`. Los segundos elementos `3 + 4 = 7`, los terceros elementos `6 + 6 = 12` y así sucesivamente.

Podemos aplicar cualquier otra operación aritmética de manera similar:

```
a <- c(22, 10, 7, 3, 14, 4)
b <- c(4, 5, 2, 6, 14, 8)
a / b
```

```
## [1] 5.5 2.0 3.5 0.5 1.0 0.5
```

Usando el mismo principio, el primer elemento del resultado es `22 / 4 = 5.5`, el segundo es `10 / 5 = 2` y así sucesivamente.

1.5.3 Usar operadores relacionales

```
___ == ___
___ != ___
___ < ___
___ > ___
___ <= ___
___ >= ___
```

Los operadores relacionales se utilizan para comparar dos valores. La salida de estas operaciones siempre es un valor lógico `TRUE` o `FALSE`. Distinguimos seis tipos diferentes de operadores relacionales, como veremos a continuación.

Los operadores igual `==` y no igual `!=` comprueban si dos valores son iguales (o no):

```
2 == 1 + 1
```

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

```
2 != 3
```

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

Los operadores *menor que* `<` y *mayor que* `>` verifican si un valor es menor o mayor que otro:

```
2 > 4
```

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

```
2 < 4
```

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

Los operadores *menor que o igual a* `<=` y *mayor que o igual a* `>=` combinan la verificación de igualdad con la comparación menor o mayor que:

```
2 >= 2
```

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

```
2 <= 3
```

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

Todos estos operadores pueden usarse en vectores con uno o más elementos también. En ese caso, cada elemento de un vector se compara con el elemento en la misma posición en el otro vector, al igual que con los operadores matemáticos:

```
vector1 <- c(3, 5, 2, 7, 4, 2)
vector2 <- c(2, 6, 3, 3, 4, 1)
vector1 > vector2
```

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

Por lo tanto, el resultado de este ejemplo se basa en las comparaciones `3 > 2`, `5 > 6`, `2 > 3` y así sucesivamente.

1.5.4 Usando operadores lógicos

```
--- & ---
--- | ---
```

El operador *AND* `&` se usa para verificar si varias declaraciones son `TRUE` al mismo tiempo. Usando un ejemplo simple, podríamos verificar si 3 es mayor que 1 y al mismo tiempo si 4 es menor que 2:

```
3 > 1 & 4 < 2
```

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

De hecho, 3 es mayor que 1, pero 4 no es menor que 2. Dado que una de las declaraciones es `FALSE`, el resultado de esta evaluación conjunta también es `FALSE`.

El operador *OR* `|` solo verifica si alguna de las declaraciones es `TRUE`.

```
3 > 1 | 4 < 2
```

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

En una declaración *OR*, no todos los elementos tienen que ser `TRUE`. Como 3 es mayor que 1, el resultado de esta evaluación también es `TRUE`.

El operador `!` se utiliza para la negación de valores lógicos, lo que significa que convierte los valores `TRUE` en `FALSE` y los valores `FALSE` en `TRUE`. Si tenemos una declaración que resulta en un valor lógico `TRUE` o `FALSE`, podemos negar el resultado aplicando el operador `!` en él. En el siguiente ejemplo verificamos si 3 es mayor que 2 y luego negamos el resultado de esta comparación:

```
!3 > 2
```

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

Los operadores lógicos, al igual que los operadores aritméticos y relacionales, también se pueden usar en vectores más largos. En el siguiente ejemplo usamos tres vectores diferentes `a`, `b` y `c` e intentamos evaluar múltiples relaciones en combinación.

```
a <- c(1, 21, 3, 4)
b <- c(4, 2, 5, 3)
c <- c(3, 23, 5, 3)

a>b & b<c
```

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

Primero, ambas comparaciones relacionales `a > b` y `b < c` se evalúan y dan como resultado dos vectores lógicos. Por lo tanto, esencialmente comparamos los siguientes dos vectores:

```
c(FALSE, TRUE, FALSE, TRUE) & c(FALSE, TRUE, FALSE, FALSE)
```

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

El operador `&` comprueba si ambos valores en la misma posición en los vectores son `TRUE`. Si algún valor de los pares es `FALSE`, la combinación también es `FALSE`.

El operador `|` comprueba si alguno de los valores en la misma posición en los vectores es `TRUE`.

```
c(FALSE, TRUE, FALSE, TRUE) | c(FALSE, TRUE, FALSE, FALSE)
```

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

1.5.5 Usando el operador %in%

`___ %in% ___`

Un operador especial adicional, de uso frecuente, es el operador `%in%`. Comprueba si el contenido de un vector también está presente en otro.

En el siguiente ejemplo utilizamos la variable `UE` que contiene la abreviatura de todos los países de la Unión Europea. Luego, verificamos si el carácter `"AU"` está presente o no en la variable `UE`.

```
EU <- c("AU","BE","BG","CY","CZ","DE","DK","EE","ES","FI","FR","GR","HR","HU",
        "IE","IT","LT","LU","LV","MT","NL","PO","PT","RO","SE","SI","SK")

"AU" %in% EU
```

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

El siguiente ejemplo amplía la búsqueda y compara múltiples elementos con el contenido de la variable `EU`. Produce un vector lógico como resultado que contiene un valor lógico para cada elemento:

```
c("AU","HU","UK") %in% EU
```

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

Como muestra el resultado, los dos primeros elementos de caracteres `"AU"` y `"HU"` están presentes en la variable `EU`, sin embargo, el tercer elemento `"UK"` no lo está.

1.6 Llamar al código R existente a través de funciones

Cuando escribe código, las funciones son sus mejores amigas. Pueden hacer las cosas difíciles muy fáciles o proporcionar nuevas funciones de una manera agradable. A través de las funciones, obtiene acceso a todas las potentes funciones que R tiene para ofrecer.

- Funciones de llamada con nombres de funciones y corchetes
- Usar funciones matemáticas básicas en vectores
- Personalizar funciones mediante parámetros
- Cree secuencias numéricas usando `seq()`
- Crea números aleatorios usando `runif()`
- Vectores de muestra usando `sample()`

1.6.1 Introducción a las funciones.

Las funciones en cualquier lenguaje de programación se pueden describir como código predefinido y reutilizable destinado a realizar una tarea específica. Las funciones en R se pueden usar usando su nombre y corchetes justo después de eso. Dentro de los corchetes, podemos especificar parámetros para la función. Una función que ya hemos usado ampliamente es la función concatenada `c()`.

Una función simple, por ejemplo, es `abs()` que se usa para obtener el valor absoluto de un número. En el siguiente ejemplo, la función recibe `-3` como entrada y devuelve el resultado `3`:

```
abs(-3)
```

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

1.6.2 Personalización de funciones a través de parámetros

Las funciones toman parámetros que los personalizan para la tarea dada. Por ejemplo, la función `runif()` genera valores distribuidos uniformemente, lo que significa que todos los resultados tienen la misma probabilidad. Por defecto, toma los siguientes parámetros:

```
runif(n, min = 0, max = 1)
```

El primer parámetro `n` es el número de valores que queremos generar. Este es un parámetro obligatorio que debemos definir para que la función funcione.

Por otro lado, podemos ver que algunos de los parámetros tienen valores predeterminados definidos por el signo igual `=`. Esto significa que si no especificamos explícitamente estos parámetros entre paréntesis, la función tomará los predeterminados. Veamos un ejemplo:

```
runif(n = 5)
```

```
## [1] 0.2162403 0.9932776 0.8533215 0.3011281 0.1793432
```

La salida es un vector numérico de 5 números. Cada uno de ellos está entre 0 y 1, ya que no cambiamos la configuración predeterminada. Si también cambiamos los parámetros `min` y `max`, podríamos personalizar aún más la salida:

```
runif(n = 5, min = 8, max = 9)
```

```
## [1] 8.821339 8.819969 8.586649 8.061424 8.335431
```

También es posible omitir el nombre de los parámetros y simplemente escribir los valores de entrada de esta manera:

```
runif(5, 8, 9)
```

```
## [1] 8.542788 8.972193 8.146812 8.766378 8.901057
```

Sin embargo, en este caso debemos tener cuidado con el orden de las entradas, ya que cada función tiene un orden predeterminado para los parámetros. Si no nombramos explícitamente los parámetros que estamos configurando, R asumirá que los establecemos en el orden predefinido.

1.6.3 Usar la función `sample()`

La función `sample()` toma un vector y devuelve una muestra aleatoria de él. Los dos primeros de sus parámetros son:

- `x`, que define el vector
- `size`, que define la cantidad de elementos que queremos incluir en la muestra aleatoria

Utilice la función `sample()` y muestree 5 valores aleatorios de la variable `full`.

```
full <- c(7, 32, 4, 27, 48, 2, 1, 9, 27, 7)
sample(x = full, size = 5)
```

```
## [1] 48 27  7 32 27
```

1.6.4 Utilice la función `seq()`

La función `seq()` crea una secuencia de números enteros. Los primeros tres de sus parámetros son: `from`, `to` y `by`.

- `from` define el inicio de la secuencia
- `to` define el final de la secuencia
- `by` establece los pasos entre los valores individuales

Utilice la función `seq()` y cree una secuencia de números del 2 al 10, pero solo incluya cada segundo valor. Por lo tanto, la salida debe ser: `2`, `4`, `6`, `8`, `10`.

```
seq(from = 2, to = 10, by = 2)
```

```
## [1]  2  4  6  8 10
```

1.7 Utilice funciones y datos existentes a través de paquetes

Los paquetes le dan acceso a un gran conjunto de funciones y conjuntos de datos, la mayoría de los cuales son proporcionados por la generosa comunidad R. Son la salsa secreta que hace posible usar R para casi cualquier cosa que puedas imaginar. Además, muchos paquetes son de código abierto, lo que puede ser un gran recurso de aprendizaje.

- Conozca el concepto de paquetes en R
- Aprenda a llamar funciones desde paquetes

```
library(___)
data(___)
```

1.7.1 Introducción a paquetes

Los paquetes son una de las mejores cosas de R. Los paquetes son compatibles con una gran comunidad de desarrolladores y permiten que R se conecte a muchos algoritmos y bibliotecas externos diferentes, muchos de ellos incluso escritos en diferentes lenguajes de programación.

Colaboradores de todo el mundo, incluidos desarrolladores o expertos en el dominio de la física, las finanzas, las estadísticas, etc., crean una gran cantidad de contenido adicional, como funciones personalizadas para casos de uso específicos. Estas funciones, junto con la documentación, los archivos de ayuda y los conjuntos de datos se pueden recopilar en paquetes. Los paquetes pueden hacerse públicos a través de *repositorios de paquetes* para que cualquiera pueda instalarlos y utilizarlos. El repositorio de paquetes más popular es CRAN, que aloja más de 15.000 paquetes.

1.7.2 Llamar un paquete

Como demostración, utilizaremos la función `generate_primes()` del paquete `primes`. Esta función toma dos números como parámetros y genera todos los números primos dentro de su rango.

Para usar un paquete, primero tenemos que cargarlo. Esto se puede hacer aplicando la función `library()` e insertando el nombre del paquete como primer argumento de la función. Después de eso, tenemos acceso a todo el contenido del paquete y podemos usar funciones de él como de costumbre.

```
library(primes)
generate_primes(min = 500, max = 550)
```

```
## [1] 503 509 521 523 541 547
```

Compruebe si hay año bisiesto

- Cargue el paquete de `lubridate`.
- Utilice la función `leap_year` para comprobar si 2020 es un año bisiesto o no.

```
library(lubridate)
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
leap_year(2020)
```

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

El contenido de este capítulo se encuentra en la página oficial de (Quartargo, 2020b) en idioma ingles. También, existe mas material en la página oficial y es muy recomendable para introducirse en R.

1.8 Formato de introducción

You can label chapter and section titles using `{#label}` after them, e.g., we can reference Chapter 1. If you do not manually label them, there will be automatic labels anyway, e.g., Chapter **??**.

Figures and tables with captions will be placed in `figure` and `table` environments, respectively.

```
par(mar = c(4, 4, .1, .1))
plot(pressure, type = 'b', pch = 19)
```

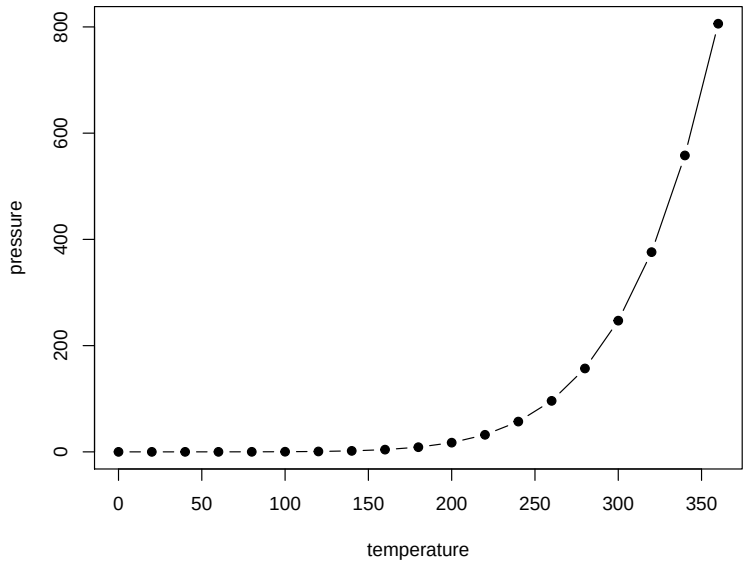


Figure 1.1: Here is a nice figure!

Reference a figure by its code chunk label with the `fig:` prefix, e.g., see Figure 1.1. Similarly, you can reference tables generated from `knitr::kable()`, e.g., see Table 1.1.

```
knitr::kable(
  head(iris, 20), caption = 'Here is a nice table!',
  booktabs = TRUE
)
```

You can write citations, too. For example, we are using the **bookdown** package

Table 1.1: Here is a nice table!

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.0	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa
5.4	3.7	1.5	0.2	setosa
4.8	3.4	1.6	0.2	setosa
4.8	3.0	1.4	0.1	setosa
4.3	3.0	1.1	0.1	setosa
5.8	4.0	1.2	0.2	setosa
5.7	4.4	1.5	0.4	setosa
5.4	3.9	1.3	0.4	setosa
5.1	3.5	1.4	0.3	setosa
5.7	3.8	1.7	0.3	setosa
5.1	3.8	1.5	0.3	setosa

(Xie, 2020) in this sample book, which was built on top of R Markdown and **knitr** (Xie, 2015).

Chapter 2

Marcos de datos y Tibbles

Cree estructuras de datos tabulares con marcos de datos y vea cómo se comparan con tibbles. Extraiga vectores de columna de marcos de datos para realizar cálculos. Obtenga información de metadatos como dimensiones. Seleccione las filas superior e inferior para obtener una descripción general rápida.

2.1 Construye un marco de datos a partir de vectores

Los datos tabulares son el formato más común utilizado por los científicos de datos. En R, las tablas se representan mediante marcos de datos. Pueden inspeccionarse imprimiéndolos en la consola.

- Comprender por qué los marcos de datos son importantes
- Interpretar la salida de la consola creada por un marco de datos
- Cree un nuevo marco de datos usando la función `data.frame()`
- Definir los vectores que se utilizarán para columnas individuales
- Especificar los nombres de las columnas del marco de datos

```
data.frame(___ = ___,
           ___ = ___,
           ...)
```

2.1.1 Introducción a los marcos de datos

En análisis y estadísticas, los datos tabulares son la estructura de datos más importante. Está presente en muchos formatos comunes como archivos de Excel, valores separados por comas (CSV) o bases de datos. R integra objetos de datos tabulares como ciudadanos de primera clase en el idioma a través de *marcos de*

datos. Los marcos de datos permiten a los usuarios leer y manipular fácilmente datos tabulares dentro del lenguaje R.

Echemos un vistazo a un objeto de marco de datos llamado `Davis`, del paquete **carData**, que incluye medidas de altura y peso para 200 hombres y mujeres:

```
tibble (Davis)
```

```
## # A tibble: 200 x 5
##    sex   weight height repwt repht
##    <fct>  <int>  <int> <int> <int>
##  1 M         77    182    77   180
##  2 F         58    161    51   159
##  3 F         53    161    54   158
##  4 M         68    177    70   175
##  5 F         59    157    59   155
##  6 M         76    170    76   165
##  7 M         76    167    77   165
##  8 M         69    186    73   180
##  9 M         71    178    71   175
## 10 M         65    171    64   170
## # ... with 190 more rows
```

De la salida impresa, podemos ver que el marco de datos abarca más de 200 **filas** y 5 **columnas**. En el ejemplo anterior, cada fila contiene datos de una persona a través de **atributos**, que corresponden a las columnas `sex`, `weight`, `height`, `repwt` (peso reportado) y `repht` (altura reportado).

Por ejemplo, la primera fila de la tabla especifica un hombre que pesa `77` kg y tiene una altura de `182` cm. Los pesos reportados están muy cerca de `77` kg y `180` cm, respectivamente.

Las filas en un marco de datos se identifican además por los *nombres de fila* a la izquierda, que son simplemente los números de fila por defecto. En el caso del conjunto de datos de `Davis` anterior, los nombres de las filas van de 1 a 200.

2.1.2 Crear marcos de datos

```
data.frame(___ = ___,
           ___ = ___,
           ...)
```

Los marcos de datos contienen datos tabulares en varias columnas o *atributos*. Cada columna está representada por un vector de diferentes *tipos de datos* como números o caracteres. La función `data.frame()` admite la construcción de objetos de marco de datos combinando diferentes vectores en una tabla. Para formar una tabla, se requiere que los vectores tengan la misma longitud. Un marco de datos también puede verse como una colección de vectores conectados entre sí para formar una tabla.

Creemos nuestro primer marco de datos con cuatro personas diferentes, incluidos sus identificadores, nombres e indicadores si son mujeres o no. Cada uno de estos atributos es creado por un vector diferente de diferentes tipos de datos (numéricos, de caracteres y lógicos). Los atributos finalmente se combinan en una tabla usando la función `data.frame()`:

```
data.frame(
  c(1, 2, 3, 4),
  c("Louisa", "Jonathan", "Luigi", "Rachel"),
  c(TRUE, FALSE, FALSE, TRUE)
)
```

```
##   c.1..2..3..4. c..Louisa....Jonathan....Luigi....Rachel..
## 1             1                                     Louisa
## 2             2                                   Jonathan
## 3             3                                      Luigi
## 4             4                                     Rachel
##   c.TRUE..FALSE..FALSE..TRUE.
## 1                        TRUE
## 2                       FALSE
## 3                       FALSE
## 4                        TRUE
```

El marco de datos resultante almacena los valores de cada vector en una columna diferente. Tiene cuatro filas y tres columnas. Sin embargo, los nombres de las columnas impresas en la primera línea parecen incluir los valores de las columnas separados por puntos, lo cual es un esquema de nombres muy extraño.

Los nombres de columna se pueden incluir en la construcción de `data.frame()` como nombres de argumentos que preceden a los valores de los vectores de columna. Para mejorar el nombre de la columna del marco de datos anterior, podemos escribir

```
data.frame(
  id = c(1, 2, 3, 4),
  name = c("Louisa", "Jonathan", "Luigi", "Rachel"),
  female = c(TRUE, FALSE, FALSE, TRUE)
)
```

```
##   id     name female
## 1  1   Louisa   TRUE
## 2  2 Jonathan  FALSE
## 3  3    Luigi  FALSE
## 4  4   Rachel   TRUE
```

El marco de datos resultante incluye los nombres de columna necesarios para ver el significado real de las diferentes columnas.

2.2 Crea y convierte tibbles

Tibbles son la reimaginación moderna de marcos de datos y comparten muchos puntos en común con sus antepasados. La diferencia más visible es cómo se imprime el contenido de tibble en la consola. Tibbles son parte del tidyverse y se utilizan por su comportamiento más consistente en comparación con los marcos de datos.

- Conozca la diferencia entre *marcos de datos* y *tibbles*
- Crear *tibbles* a partir de vectores
- Convertir *marcos de datos* en tibbles

```
tibble(___ = ___,
       ___ = ___,
       ...)
as_tibble(___)
```

2.2.1 Introducción a Tibbles

> Una reinvención moderna del marco de datos https://tibble.tidyverse.org

Tibbles son en muchos aspectos similares a los marcos de datos. De hecho, se *heredan* de los marcos de datos, lo que significa que todas las funciones y características disponibles para los marcos de datos también funcionan para tibbles. Por tanto, cuando hablamos de *marcos de datos* también nos referimos a *tibbles*.

Además de todo lo que ofrece un marco de datos, los tibbles tienen un comportamiento más consistente con una mejor usabilidad en muchos casos. Lo más importante es que cuando se imprime un objeto tibble en la consola, muestra automáticamente solo las primeras 10 filas y condensa columnas adicionales. Por el contrario, un marco de datos llena toda la pantalla de la consola con valores que pueden generar confusión. Echemos un vistazo al conjunto de datos `gapminder` del paquete **gapminder**:

```
gapminder::gapminder
```

```
## # A tibble: 1,704 x 6
##    country     continent  year lifeExp      pop gdpPercap
##    <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
##  1 Afghanistan Asia       1952    28.8  8425333      779.
##  2 Afghanistan Asia       1957    30.3  9240934      821.
##  3 Afghanistan Asia       1962    32.0 10267083      853.
##  4 Afghanistan Asia       1967    34.0 11537966      836.
##  5 Afghanistan Asia       1972    36.1 13079460      740.
##  6 Afghanistan Asia       1977    38.4 14880372      786.
##  7 Afghanistan Asia       1982    39.9 12881816      978.
##  8 Afghanistan Asia       1987    40.8 13867957      852.
```

```
##  9 Afghanistan Asia       1992    41.7 16317921      649.
## 10 Afghanistan Asia       1997    41.8 22227415      635.
## # ... with 1,694 more rows
```

Inmediatamente vemos que el conjunto de datos `gapminder` es un tibble que consta de 1,704 filas y 6 columnas en la línea superior. En la segunda línea podemos ver los nombres de las columnas y sus correspondientes *tipos de datos* directamente debajo.

Por ejemplo, la columna `country` tiene el tipo `<fct>` (que es la abreviatura de "factor"), `year` es un número entero `<int>` y la esperanza de vida `lifeExp` es un `<dbl>`, un número decimal.

2.2.2 Creando Tibbles

```
tibble(___ = ___,
       ___ = ___,
       ...)
as_tibble(___)
```

La creación de tibbles funciona exactamente igual que para los marcos de datos. Podemos usar la función `tibble()` del paquete **tibble** para crear un nuevo objeto tabular.

Por ejemplo, un tibble que contenga datos de cuatro personas diferentes y tres columnas se puede crear así:

```
library(tibble)
tibble(
  id = c(1, 2, 3, 4),
  name = c("Louisa", "Jonathan", "Luigi", "Rachel"),
  female = c(TRUE, FALSE, FALSE, TRUE)
)
```

```
## # A tibble: 4 x 3
##      id name     female
##   <dbl> <chr>    <lgl>
## 1     1 Louisa   TRUE
## 2     2 Jonathan FALSE
## 3     3 Luigi    FALSE
## 4     4 Rachel   TRUE
```

2.2.3 Conversión de marcos de datos a Tibbles

Si prefiere tibbles a marcos de datos por sus características adicionales, también se pueden convertir a partir de marcos de datos existentes con la función `as_tibble()`.

Por ejemplo, el marco de datos de `Davis` del paquete **carData** se puede convertir a un tibble así:

```
as_tibble(Davis)
```

```
## # A tibble: 200 x 5
##    sex   weight height repwt repht
##    <fct>  <int>  <int> <int> <int>
##  1 M         77    182    77   180
##  2 F         58    161    51   159
##  3 F         53    161    54   158
##  4 M         68    177    70   175
##  5 F         59    157    59   155
##  6 M         76    170    76   165
##  7 M         76    167    77   165
##  8 M         69    186    73   180
##  9 M         71    178    71   175
## 10 M         65    171    64   170
## # ... with 190 more rows
```

2.3 Extraiga o reemplace columnas en un marco de datos usando $

Las columnas de un marco de datos se pueden extraer y manipular fácilmente con el operador `$`. Incluso se pueden agregar nuevas columnas asignando un vector.

- Extraiga columnas de un marco de datos con `$`.
- Reemplazar valores de columnas existentes en un marco de datos.
- Agregue nuevas columnas a un marco de datos.

```
___$___
___$___  <- ___
```

2.3.1 Extraer columnas con $

Los marcos de datos son tablas que resultan de la combinación de vectores de columna. Los usuarios pueden interactuar con los marcos de datos a través de numerosos operadores para extraer, agregar o recombinar valores. Para extraer columnas individuales de un marco de datos, R ofrece un operador muy específico: el dólar `$`. Devuelve el vector de columna como lo indica su nombre basado en un marco de datos que precede a `$`.

Para ver el operador `$` en acción, extraigamos la población `pop` (en 1,000) de diferentes estados de los EE. UU. Según el conjunto de datos de los estados (de 1992) en el paquete **carData**:

```
carData::States$pop
```

```
##  [1]  4041   550  3665  2351 29760  3294  3287   666   607 12938  6478  1108
## [13]  1007 11431  5544  2777  2478  3685  4220  1228  4781  6016  9295  4375
## [25]  2573  5117   799  1578  1202  1109  7730  1515 17990  6629   639 10847
## [37]  3146  2842 11882  1003  3487   696  4877 16987  1723   563  6187  4867
## [49]  1793  4892   454
```

El comando extrae la columna de población como vector del marco de datos. A partir de este vector podemos calcular la `sum()` de la población total como:

```
sum(States$pop)
```

```
## [1] 248709
```

De manera similar, el salario promedio (en $1,000) de los maestros se puede calcular como la `mean()` de la columna `pay`:

```
mean(States$pay)
```

```
## [1] 30.94118
```

2.4 Determinar el tamaño de un marco de datos

El tamaño de un marco de datos, como el número de filas o columnas, a menudo es necesario y se puede determinar de varias formas.

- Obtener el número de filas de un marco de datos
- Obtener el número de columnas de un marco de datos
- Obtener dimensiones de un marco de datos

```
nrow(___)
ncol(___)
dim(___)
length(___)
```

2.4.1 Dimensiones del marco de datos

El número de filas y columnas en un marco de datos se puede adivinar a través de la salida impresa del marco de datos. Sin embargo, es mucho más fácil obtener esta información directamente a través de funciones. Además, es posible que desee utilizar esta información en algunas partes del código.

Los marcos de datos tienen dos dimensiones. El número de filas se considera la primera dimensión. Por lo general, define el número de observaciones en un conjunto de datos. Para obtener el número de filas del marco de datos de `Davis` en el conjunto de datos **carData**, use la función `nrow()`:

```
nrow(Davis)
```

```
## [1] 200
```

De manera similar, el número de columnas o *atributos* del marco de datos se puede recuperar con `ncol()`:

```
ncol(Davis)
```

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

2.4.2 Recuperar las dimensiones del marco de datos

Para recuperar el tamaño de todas las dimensiones de un marco de datos a la vez, puede usar la función `dim()`. `dim()` devuelve un vector con dos elementos, el primer elemento es el número de filas y el segundo elemento el número de columnas.

Por ejemplo, las dimensiones del conjunto de datos de `Davis` se pueden recuperar como:

```
dim(Davis)
```

```
## [1] 200   5
```

Además de los marcos de datos, `dim()` también se puede utilizar para otros objetos R multidimensionales, como matrices. Sin embargo, cuando se usa con vectores `dim` solo devuelve `NULL`:

```
dim(c(1, 3, 5, 7))
```

```
## NULL
```

En cambio, la longitud de un vector se determina mediante `length()`:

```
length(c(1, 3, 5, 7))
```

```
## [1] 4
```

En el caso de un marco de datos, `length()` devuelve su número de columnas:

```
length(Davis)
```

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

2.5 Seleccionar la primera o la última fila de un marco de datos

A menudo no necesitamos mirar todo el contenido de un marco de datos en la consola. En cambio, solo algunas partes son suficientes, como la parte superior o inferior recuperada a través de las funciones `head()` y `tail()`.

- Seleccionar la parte superior de un marco de datos
- Seleccione la parte inferior de un marco de datos
- Especifique el número de líneas a seleccionar mediante el parámetro n

```
head(___, n = ___)
tail(___, n = ___)
```

2.5.1 Seleccionar la parte superior de un marco de datos

Los marcos de datos pueden abarcar una gran cantidad de filas y columnas. Según la salida impresa en la consola, puede ser difícil obtener una impresión inicial de los datos dentro del marco de datos. Este problema no es tanto un problema para tibbles que tienen una mejor salida de consola. Además, puede ser útil recuperar fácilmente las primeras filas en un comando sin indexación ni paquetes adicionales.

El conjunto de datos `TitanicSurvival` contiene datos de 1309 pasajeros representados como filas. Una simple impresión del conjunto de datos imprimiría a todos los pasajeros, llenando toda la consola. En cambio, la función `head()` muestra solo las primeras 10 filas de un marco de datos, incluidos los nombres de sus columnas:

```
head(TitanicSurvival)
```

```
##                                survived    sex     age passengerClass
## Allen, Miss. Elisabeth Walton       yes female 29.0000            1st
## Allison, Master. Hudson Trevor      yes   male  0.9167            1st
## Allison, Miss. Helen Loraine         no female  2.0000            1st
## Allison, Mr. Hudson Joshua Crei      no   male 30.0000            1st
## Allison, Mrs. Hudson J C (Bessi      no female 25.0000            1st
## Anderson, Mr. Harry                 yes   male 48.0000            1st
```

El número de columnas se puede ajustar mediante el parámetro `n`. Para extraer solo las primeras tres filas del conjunto de datos, puede escribir:

```
head(TitanicSurvival, n = 3)
```

```
##                               survived    sex     age passengerClass
## Allen, Miss. Elisabeth Walton      yes female 29.0000            1st
## Allison, Master. Hudson Trevor     yes   male  0.9167            1st
## Allison, Miss. Helen Loraine        no female  2.0000            1st
```

2.5.2 Seleccionar la parte inferior de un marco de datos

La función `tail()` se puede utilizar para seleccionar las filas inferiores de un marco de datos. Similar a la función `head()`, también acepta un parámetro `n` para especificar el número de filas que se devolverán.

Por ejemplo, para seleccionar las últimas cinco filas del conjunto de datos `TitanicSurvival`, puede escribir:

```
tail(TitanicSurvival, n = 5)
```

```
##                            survived    sex  age passengerClass
## Zabour, Miss. Hileni             no female 14.5            3rd
## Zabour, Miss. Thamine            no female   NA            3rd
## Zakarian, Mr. Mapriededer        no   male 26.5            3rd
## Zakarian, Mr. Ortin              no   male 27.0            3rd
## Zimmerman, Mr. Leo               no   male 29.0            3rd
```

Las funciones de cabeza y cola también se pueden combinar para seleccionar un fragmento del conjunto de datos del medio. Para seleccionar las primeras cinco filas de las 500 filas inferiores, puede escribir:

```
head(tail(TitanicSurvival, n = 500), n = 5)
```

```
##                                 survived    sex age passengerClass
## Ford, Mr. Edward Watson               no   male  18            3rd
## Ford, Mr. William Neal                no   male  16            3rd
## Ford, Mrs. Edward (Margaret Ann       no female  48            3rd
## Fox, Mr. Patrick                      no   male  NA            3rd
## Franklin, Mr. Charles (Charles        no   male  NA            3rd
```

Chapter 3

Transformación de datos con dplyr

Utilice verbos `dplyr` básicos para filtrar filas, seleccionar columnas y ordenar/organizar conjuntos de datos en combinación con el operador pipe `%>%`.

3.1 Introducción a dplyr

dplyr facilita el proceso de transformación de datos al proporcionar un marco enriquecido para manipular marcos de datos. Las funciones de **dplyr** se pueden concatenar a potentes canalizaciones de transformación para *seleccionar*, *filtrar*, *ordenar*, *unir* y *agregar* datos.

- Descubra lo que hace **dplyr**
- Obtenga una descripción general de Seleccionar, Filtrar y Ordenar
- Descubra qué son las uniones, agregaciones y canalizaciones

3.1.1 ¿Qué es dplyr?

> Existe la broma de que el 80 % de la ciencia de datos está limpiando los datos y el 20 % se queja de limpiar los datos. — Anthony Goldbloom, fundador y CEO de Kaggle

Tener datos *limpios* en cualquier proyecto de Data Science es muy importante, porque los resultados solo son tan buenos como los datos correctos. La limpieza de datos también es la parte que generalmente consume la mayor parte del tiempo y causa los mayores dolores para los científicos de datos. R ya ofrece un amplio conjunto de herramientas y funciones para manipular marcos de datos.

Sin embargo, debido a su larga historia, el conjunto de herramientas base R disponible está fragmentado y es difícil de usar para nuevos usuarios.

El paquete **dplyr** facilita el proceso de transformación de datos a través de una colección consistente de funciones. Estas funciones admiten diferentes transformaciones en marcos de datos, que incluyen

- filtrar filas
- seleccionar columnas
- ordenar datos
- agregar datos

Múltiples marcos de datos también se pueden unir mediante valores de atributo comunes.

La coherencia de las funciones de **dplyr** mejora la usabilidad y permite al usuario conectar transformaciones para formar *canalizaciones de datos*. Estas canalizaciones también se pueden ver como un lenguaje de consulta de alto nivel, como p. Ej. el lenguaje SQL para consultas de bases de datos. Además, incluso es posible traducir las canalizaciones de datos creadas a otros back-end, incluidas las bases de datos.

3.1.2 Marco de funciones

Cada función de transformación de datos en **dplyr** acepta un marco de datos como su primer parámetro de entrada y devuelve el marco de datos transformado como salida. Un plano para una función **dplyr** típica se ve así:

```
transformed <- dplyr_function(my_data_frame,
                              param_one,
                              param_two,
                              ...)
```

La función `dplyr_function` se puede personalizar aún más a través de argumentos adicionales (`param_one`, `param_two`) colocados después del primer parámetro de marco de datos (`my_data_frame`).

El poder real de **dplyr** viene con el operador pipe `%>%` que permite a los usuarios concatenar funciones de **dplyr** a pipe de datos. Pipe inyecta el marco de datos resultante del cálculo anterior como primer argumento del siguiente. Una transformación de datos que consta de tres funciones se parece a:

```
dplyr_function_three(
  dplyr_function_two(
    dplyr_function_one(input_data_frame)))
```

pero con `pipe` se puede escribir de la siguiente manera:

```
input_data_frame %>%
  dplyr_function_one() %>%
```

```
  dplyr_function_two() %>%
  dplyr_function_three()
```

El orden de lectura diferente de las funciones de transformación de datos en el orden de transformación real, hace que las canalizaciones sean más fáciles de leer que las llamadas a funciones anidadas.

3.2 Seleccionar columnas de un marco de datos

Para seleccionar sólo un conjunto específico de columnas de marcos de datos interesantes, **dplyr** ofrece la función `select()` para extraer columnas por nombres, índices y rangos. Incluso puede renombrar las columnas extraídas con `select()`.

- Aprenda a usar la función `select()`
- Seleccionar las columnas de un marco de datos por nombre o índice
- Renombrar las columnas de un marco de datos

```
select(my_data_frame, column_one, column_two, ...)
select(my_data_frame, new_column_name = current_column, ...)
select(my_data_frame, column_start:column_end)
select(my_data_frame, index_one, index_two, ...)
select(my_data_frame, index_start:index_end)
```

3.2.1 Seleccionando por nombre

```
select(my_data_frame, column_one, column_two, ...)
```

En este capítulo veremos el conjunto de datos `pres_results` del paquete **politicaldata**. Contiene datos sobre las elecciones presidenciales de EE.UU. desde 1976, convertidas en un Tibble para una mejor impresión.

```
## # A tibble: 561 x 6
##     year state total_votes   dem   rep   other
##    <dbl> <chr>       <dbl> <dbl> <dbl>   <dbl>
##  1  1976 AK         123574 0.357 0.579 0.0549
##  2  1976 AL        1182850 0.557 0.426 0.0163
##  3  1976 AR         767535 0.650 0.349 0.00134
##  4  1976 AZ         742719 0.398 0.564 0.0383
##  5  1976 CA        7803770 0.480 0.497 0.0230
##  6  1976 CO        1081440 0.426 0.540 0.0336
##  7  1976 CT        1386355 0.467 0.519 0.0138
##  8  1976 DC         168830 0.816 0.165 0.0169
##  9  1976 DE         235642 0.520 0.466 0.0144
## 10  1976 FL        3150631 0.519 0.466 0.0143
## # ... with 551 more rows
```

Para este ejemplo, veremos el número total de votos en diferentes estados en diferentes elecciones. Como solo estamos interesados en la cantidad de personas que votaron, nos gustaría crear una versión personalizada del marco de datos `pres_results` que solo contenga las columnas `year`, `state` y `total_votes`. Para dicho filtrado, podemos usar la función `select()` del paquete **dplyr**.

La función `select()` toma un marco de datos como parámetro de entrada y nos permite decidir cuál de las columnas queremos mantener. El resultado de la función es un marco de datos con todas las filas, pero que contiene solo las columnas que seleccionamos explícitamente.

Podemos reducir nuestro conjunto de datos a solo `year`, `state` y `total_votes` de la siguiente manera:

```
tibble(
select(pres_results, year, state, total_votes)
)
```

```
## # A tibble: 561 x 3
##     year state total_votes
##    <dbl> <chr>       <dbl>
##  1  1976 AK         123574
##  2  1976 AL        1182850
##  3  1976 AR         767535
##  4  1976 AZ         742719
##  5  1976 CA        7803770
##  6  1976 CO        1081440
##  7  1976 CT        1386355
##  8  1976 DC         168830
##  9  1976 DE         235642
## 10  1976 FL        3150631
## # ... with 551 more rows
```

Como primer parámetro pasamos el marco de datos `pres_results`, como parámetros restantes pasamos las columnas que queremos mantener para `select()`.

Además de mantener las columnas que queremos, la función `select()` también las mantiene en el mismo orden que especificamos en los parámetros de la función.

Si cambiamos el orden de los parámetros cuando llamamos a la función, las columnas de la salida cambian en consecuencia:

```
tibble(
select(pres_results, total_votes, year, state)
)
```

```
## # A tibble: 561 x 3
##    total_votes  year state
```

```
##          <dbl> <dbl> <chr>
##  1      123574  1976 AK
##  2     1182850  1976 AL
##  3      767535  1976 AR
##  4      742719  1976 AZ
##  5     7803770  1976 CA
##  6     1081440  1976 CO
##  7     1386355  1976 CT
##  8      168830  1976 DC
##  9      235642  1976 DE
## 10     3150631  1976 FL
## # ... with 551 more rows
```

3.2.2 Renombrar nombres de columnas

```
select(my_data_frame, new_column_name = current_column, ...)
```

Además de definir las columnas que queremos conservar, también podemos cambiarles el nombre. Para hacer esto, necesitamos establecer el nuevo nombre de columna dentro de la función `select()` usando el comando

```
new_column_name = current_column
```

En el siguiente ejemplo, seleccionamos las columnas `year`, `state` y `total_votes` pero cambiamos el nombre de la columna `year` a `Election` en la salida:

```
tibble(
select(pres_results, Election = year, state, total_votes)
)
```

```
## # A tibble: 561 x 3
##    Election state total_votes
##       <dbl> <chr>       <dbl>
##  1     1976 AK         123574
##  2     1976 AL        1182850
##  3     1976 AR         767535
##  4     1976 AZ         742719
##  5     1976 CA        7803770
##  6     1976 CO        1081440
##  7     1976 CT        1386355
##  8     1976 DC         168830
##  9     1976 DE         235642
## 10     1976 FL        3150631
## # ... with 551 more rows
```

3.2.3 Selección por rango de nombre

```
select(my_data_frame, column_start:column_end)
```

Cuando usamos la función `select()` y definimos las columnas que queremos mantener, **dplyr** en realidad no usa el nombre de las columnas sino el índice de las columnas en el marco de datos. Esto significa que, cuando definimos las primeras tres columnas del marco de datos `pres_results`, `year`, `state` y `total_votes`, **dplyr** convierte estos nombres en los valores de índice `1`, `2` y `3`. Por lo tanto, también podemos usar el nombre de las columnas, aplicar el operador `:` y definir rangos de columnas, que queremos mantener:

```
tibble(
select(pres_results, year:total_votes)
)
```

```
## # A tibble: 561 x 3
##     year state total_votes
##    <dbl> <chr>       <dbl>
##  1  1976 AK         123574
##  2  1976 AL        1182850
##  3  1976 AR         767535
##  4  1976 AZ         742719
##  5  1976 CA        7803770
##  6  1976 CO        1081440
##  7  1976 CT        1386355
##  8  1976 DC         168830
##  9  1976 DE         235642
## 10  1976 FL        3150631
## # ... with 551 more rows
```

Lo que hace `year:total_votes`, se puede traducir a `1:3`, que es simplemente crear un vector de valores numéricos del 1 al 3. Luego, la función `select()` toma el marco de datos `pres_results` y genera un subconjunto del mismo, manteniendo solo Las primeras tres columnas.

3.2.4 Select() por índices

```
select(my_data_frame, index_one, index_two, ...)
select(my_data_frame, index_start:index_end)
```

La función `select()` también se puede usar con índices de columna. En lugar de usar nombres, debemos especificar las columnas que queremos seleccionar por sus índices. En comparación con otros lenguajes de programación, la indexación en R comienza con *uno* en lugar de *cero*. Para seleccionar la primera, cuarta y quinta columna del conjunto de datos `pres_results` podemos escribir

```
tibble(
select(pres_results, 1,4,5)
)
```

```
## # A tibble: 561 x 3
##     year   dem   rep
##    <dbl> <dbl> <dbl>
##  1  1976 0.357 0.579
##  2  1976 0.557 0.426
##  3  1976 0.650 0.349
##  4  1976 0.398 0.564
##  5  1976 0.480 0.497
##  6  1976 0.426 0.540
##  7  1976 0.467 0.519
##  8  1976 0.816 0.165
##  9  1976 0.520 0.466
## 10  1976 0.519 0.466
## # ... with 551 more rows
```

De manera similar a la definición de rangos de columnas usando sus nombres, podemos definir rangos (o vectores) de valores de índice en su lugar:

```
tibble(
select(pres_results, 1:3)
)
```

```
## # A tibble: 561 x 3
##     year state total_votes
##    <dbl> <chr>       <dbl>
##  1  1976 AK         123574
##  2  1976 AL        1182850
##  3  1976 AR         767535
##  4  1976 AZ         742719
##  5  1976 CA        7803770
##  6  1976 CO        1081440
##  7  1976 CT        1386355
##  8  1976 DC         168830
##  9  1976 DE         235642
## 10  1976 FL        3150631
## # ... with 551 more rows
```

3.3 Filtrar filas de marcos de datos

A menudo queremos operar sólo en un subconjunto específico de filas de un marco de datos. La función **dplyr** `filter()` proporciona una forma flexible de extraer las filas de interés en base a múltiples condiciones.

- Utilice la función `filter()` para ordenar las filas de un marco de datos que cumplan una condición específica
- Filtrar un cuadro de datos por múltiples condiciones

```
filter(my_data_frame, condition)
filter(my_data_frame, condition_one, condition_two, ...)
```

3.3.1 Función filter()

```
filter(my_data_frame, condition)
```

La función `filter()` toma un marco de datos y una o más expresiones de filtrado como parámetros de entrada. Procesa el marco de datos y mantiene solo las filas que cumplen con las expresiones de filtrado definidas. Estas expresiones pueden verse como reglas para la evaluación y el mantenimiento de filas. En la mayoría de los casos, se basan en operadores relacionales. Como ejemplo, podríamos filtrar el marco de datos `pres_results` y mantener solo las filas, donde la variable de `state` es igual a `"CA"` (California):

```
tibble(
filter(pres_results, state == "CA")
)
```

```
## # A tibble: 11 x 6
##     year state total_votes   dem   rep  other
##    <dbl> <chr>       <dbl> <dbl> <dbl>  <dbl>
##  1  1976 CA        7803770 0.480 0.497 0.0230
##  2  1980 CA        8582938 0.359 0.527 0.114
##  3  1984 CA        9505041 0.413 0.575 0.0122
##  4  1988 CA        9887065 0.476 0.511 0.0131
##  5  1992 CA       11131721 0.460 0.326 0.213
##  6  1996 CA       10019469 0.511 0.382 0.107
##  7  2000 CA       10965822 0.534 0.417 0.0490
##  8  2004 CA       12421353 0.543 0.444 0.0117
##  9  2008 CA       13561900 0.610 0.370 0.0188
## 10  2012 CA       13038547 0.602 0.371 0.0246
## 11  2016 CA       14181595 0.617 0.316 0.0581
```

En el resultado, podemos comparar los resultados de las elecciones en California para diferentes años.

Como otro ejemplo, podríamos filtrar el marco de datos `pres_results` y mantener solo aquellas filas, donde la variable `dem` (porcentaje de votos para el Partido Demócrata) es mayor que 0.85:

```
tibble(
filter(pres_results, dem > 0.85)
)
```

```
## # A tibble: 7 x 6
##    year state total_votes   dem    rep   other
##   <dbl> <chr>       <dbl> <dbl>  <dbl>   <dbl>
## 1  1984 DC         211288 0.854 0.137  0.00886
## 2  1996 DC         185726 0.852 0.0934 0.0513
## 3  2000 DC         201894 0.852 0.0895 0.0563
## 4  2004 DC         227586 0.892 0.0934 0.0125
## 5  2008 DC         265853 0.925 0.0653 0.00582
## 6  2012 DC         293764 0.909 0.0728 0.0155
## 7  2016 DC         312575 0.905 0.0407 0.0335
```

En el resultado podemos ver para cada año electoral los estados donde el Partido Demócrata obtuvo más del 85% de los votos. Según los resultados, podríamos decir que el Partido Demócrata tiene una base sólida de votantes en el Distrito de Columbia (conocido como Washington, D.C.).

3.3.2 Múltiples expresiones de filtro

```
filter(my_data_frame, condition_one, condition_two, ...)
```

La función `filter()` también puede tomar múltiples reglas de filtrado como entrada. Estos pueden verse como una combinación de reglas con el operador `&`. Para que una fila se incluya en la salida, todas las reglas de filtrado deben cumplirse. En el siguiente ejemplo, filtramos el marco de datos `pres_results` para todas las filas donde la variable `state` es igual a `"CA"` y la variable de `year` es igual a 2016:

```
tibble(
filter(pres_results, state == "CA", year==2016)
)
```

```
## # A tibble: 1 x 6
##    year state total_votes   dem   rep  other
##   <dbl> <chr>       <dbl> <dbl> <dbl>  <dbl>
## 1  2016 CA       14181595 0.617 0.316 0.0581
```

Obtenemos una sola fila como salida, que contiene los resultados de las elecciones presidenciales de 2016 en los Estados Unidos para el estado de California.

Ejercicio: Usar filter() con múltiples reglas El conjunto de datos del `gapminder` contiene datos económicos y demográficos sobre diversos países desde 1952. Filtra el tibble e inspecciona qué países tenían una esperanza de vida de más de 80 años en el año 2007. Los paquetes necesarios ya están cargados.

1. Utiliza la función `filter()` en el tibble de gapminder.
2. Filtra todas las filas en las que la variable `year` es igual a 2007 y la esperanza de vida `lifeExp` es mayor de 80!

Solución

```
tibble(
filter(gapminder, year==2007, lifeExp > 80)
)
```

```
## # A tibble: 13 x 6
##    country          continent  year lifeExp       pop gdpPercap
##    <fct>            <fct>     <int>   <dbl>     <int>     <dbl>
##  1 Australia        Oceania    2007    81.2  20434176    34435.
##  2 Canada           Americas   2007    80.7  33390141    36319.
##  3 France           Europe     2007    80.7  61083916    30470.
##  4 Hong Kong, China Asia       2007    82.2   6980412    39725.
##  5 Iceland          Europe     2007    81.8    301931    36181.
##  6 Israel           Asia       2007    80.7   6426679    25523.
##  7 Italy            Europe     2007    80.5  58147733    28570.
##  8 Japan            Asia       2007    82.6 127467972    31656.
##  9 New Zealand      Oceania    2007    80.2   4115771    25185.
## 10 Norway           Europe     2007    80.2   4627926    49357.
## 11 Spain            Europe     2007    80.9  40448191    28821.
## 12 Sweden           Europe     2007    80.9   9031088    33860.
## 13 Switzerland      Europe     2007    81.7   7554661    37506.
```

Ejercicio El conjunto de datos del `gapminder` contiene datos económicos y demográficos sobre diversos países desde 1952. Filtra el tibble de `gapminder` e inspecciona qué países tenían una población de más de 1.000.000.000 en el año 2007! Los paquetes necesarios ya están cargados.

1. Utiliza la función `filter()` en el tibble `gapminder`.
2. Filtra todas las filas donde la variable `year` es igual a 2007 y la población `pop` es mayor de 1000000000!

```
filter(gapminder, year==2007, pop > 1000000000)
```

```
## # A tibble: 2 x 6
##   country continent  year lifeExp        pop gdpPercap
##   <fct>   <fct>     <int>   <dbl>      <int>     <dbl>
## 1 China   Asia       2007    73.0 1318683096     4959.
## 2 India   Asia       2007    64.7 1110396331     2452.
```

3.4 Ordenar marcos de datos por columnas

Para seleccionar las áreas de interés en un marco de datos, a menudo es necesario ordenarlas por columnas específicas. La función **dplyr** `arrange()` permite ordenar los marcos de datos por múltiples columnas en orden ascendente y descendente.

- Utilice la función `arrange()` para ordenar los marcos de datos.
- Ordene los marcos de datos por múltiples columnas usando `arrange()`.

```
arrange(my_data_frame, column_one)
arrange(my_data_frame, column_one, column_two, ...)
```

3.4.1 La función arrange() con una sola columna

```
arrange(my_data_frame, column_one)
```

La función `arrange()` ordena las filas de un marco de datos. Toma un marco de datos o un tibble como primer parámetro y los nombres de las columnas en función de los cuales las filas deben ordenarse como parámetros adicionales. Supongamos que queremos responder la pregunta: *¿Qué estados tuvieron el mayor porcentaje de votantes republicanos en las elecciones presidenciales de 2016 en los Estados Unidos?* Para responder a esta pregunta, en el siguiente ejemplo usamos el marco de datos `pres_results_2016`, que contiene información solo para las elecciones presidenciales de EE.UU. de 2016. Nuestra función `arrange()` en el marco de datos `rep` basado en la columna (votos republicanos en porcentaje):

```
pres_results_2016 <- pres_results %>%
  filter(year==2016)
```

```
tibble(
arrange(pres_results_2016, rep)
)
```

```
## # A tibble: 51 x 6
##     year state total_votes   dem    rep  other
##    <dbl> <chr>       <dbl> <dbl>  <dbl>  <dbl>
##  1  2016 DC         312575 0.905 0.0407 0.0335
##  2  2016 HI         437664 0.610 0.294  0.0958
##  3  2016 VT         320467 0.557 0.298  0.0737
##  4  2016 CA       14181595 0.617 0.316  0.0581
##  5  2016 MA        3378821 0.591 0.323  0.0858
##  6  2016 MD        2781446 0.603 0.339  0.0415
##  7  2016 NY        7802084 0.584 0.361  0.0530
##  8  2016 WA        3317019 0.525 0.368  0.0738
##  9  2016 IL        5536424 0.558 0.388  0.0517
## 10  2016 RI         464144 0.544 0.389  0.0466
## # ... with 41 more rows
```

Como puede ver en la salida, el marco de datos se ordena en orden ascendente según la columna `rep`. Sin embargo, preferiríamos tener los resultados en orden descendente, para que podamos ver instantáneamente `state` con el mayor porcentaje de repeticiones. Para ordenar una columna en orden descendente, todo lo que tenemos que hacer es aplicar la función `desc()` en la columna dada dentro de la función `arrange()`:

```
tibble(
arrange(pres_results_2016, desc(rep))
)
```

```
## # A tibble: 51 x 6
##     year state total_votes   dem   rep  other
##    <dbl> <chr>       <dbl> <dbl> <dbl>  <dbl>
##  1  2016 WV         713051 0.265 0.686 0.0489
##  2  2016 WY         258788 0.216 0.674 0.0830
##  3  2016 OK        1452992 0.289 0.653 0.0575
##  4  2016 ND         344360 0.272 0.630 0.0971
##  5  2016 KY        1924149 0.327 0.625 0.0476
##  6  2016 AL        2123372 0.344 0.621 0.0254
##  7  2016 SD         370093 0.317 0.615 0.0673
##  8  2016 TN        2508027 0.347 0.607 0.0401
##  9  2016 AR        1130635 0.337 0.606 0.0577
## 10  2016 ID         690255 0.275 0.593 0.132
## # ... with 41 more rows
```

La organización no solo es posible en valores numéricos, sino también en valores de caracteres. En ese caso, **dplyr** ordena las filas en orden alfabético. Podemos organizar columnas de caracteres como las numéricas:

```
tibble(
arrange(pres_results_2016, state)
)
```

```
## # A tibble: 51 x 6
##     year state total_votes   dem    rep  other
##    <dbl> <chr>       <dbl> <dbl>  <dbl>  <dbl>
##  1  2016 AK         318608 0.366 0.513  0.0928
##  2  2016 AL        2123372 0.344 0.621  0.0254
##  3  2016 AR        1130635 0.337 0.606  0.0577
##  4  2016 AZ        2573165 0.451 0.487  0.0547
##  5  2016 CA       14181595 0.617 0.316  0.0581
##  6  2016 CO        2780220 0.482 0.433  0.0859
##  7  2016 CT        1644920 0.546 0.409  0.0435
##  8  2016 DC         312575 0.905 0.0407 0.0335
##  9  2016 DE         441590 0.534 0.419  0.0472
## 10  2016 FL        9420039 0.478 0.490  0.0315
## # ... with 41 more rows
```

3.4.2 La función arrange() con múltiples columnas

También podemos usar la función `arrange()` en varias columnas. En este caso, el orden de las columnas en los parámetros de la función establece una jerarquía de ordenamiento. La función comienza ordenando las filas en función de la

primera columna definida en los parámetros. En caso de que haya varias filas con el mismo valor, la función decide el orden en función de la segunda columna definida en los parámetros. Si todavía hay varias filas con los mismos valores, la función decide en función de la tercera columna definida en los parámetros (si está definida) y así sucesivamente.

En el siguiente ejemplo, utilizamos el marco de datos `pres_results_subset`, que contiene los resultados de las elecciones solo para los estados: `"TX"` (Texas), `"UT"` (Utah) y `"FL"` (Florida). Primero ordenamos el marco de datos en orden descendente según la columna de year. Luego, agregamos un segundo nivel y ordenamos el marco de datos basado en la columna `dem`:

```
pres_results_subset <- pres_results %>%
  filter(state %in% c("TX",
                      "UT",
                      "FL"))
```

```
tibble(
arrange(pres_results_subset, year, dem)
)
```

```
## # A tibble: 33 x 6
##     year state total_votes   dem   rep    other
##    <dbl> <chr>       <dbl> <dbl> <dbl>    <dbl>
##  1  1976 UT         541218 0.336 0.624 0.0392
##  2  1976 TX        4071884 0.511 0.480 0.00817
##  3  1976 FL        3150631 0.519 0.466 0.0143
##  4  1980 UT         604152 0.206 0.728 0.0665
##  5  1980 FL        3686927 0.385 0.555 0.0597
##  6  1980 TX        4541636 0.414 0.553 0.0329
##  7  1984 UT         629656 0.247 0.745 0.00823
##  8  1984 FL        4180051 0.347 0.653 0.000212
##  9  1984 TX        5397571 0.361 0.636 0.00275
## 10  1988 UT         647008 0.320 0.662 0.0173
## # ... with 23 more rows
```

Como puede ver en la salida, el marco de datos se ordena en general en función de la columna `year`. Sin embargo, cuando el valor de `year` es el mismo, el orden de las filas lo decide la columna `dem`.

3.5 Crear una tubería de transformación de datos

Todas las funciones de transformación de datos en **dplyr** pueden ser conectadas a través del operador de tubería (pipe) `%>%` para crear poderosas y a la vez expresivas tuberías de transformación de datos.

- Utilice el operador pipe `%>%` para combinar múltiples funciones **dplyr** en una tubería

```
my_data_frame %>%
  filter(___) %>%
  select(___) %>%
  arrange(___)
```

3.5.1 Usando el operador %>%

El operador pipe `%>%` es una parte especial del universo `tidyverse`. Se utiliza para combinar múltiples funciones y ejecutarlas una tras otra. En esta configuración, la entrada de cada función es la salida de la función anterior. Imagine que tenemos el marco de datos `pres_results` y queremos crear un marco de datos más pequeño y transparente para responder a la pregunta: *¿En qué estados fue el partido democrático la opción más popular en las elecciones presidenciales de 2016 en los Estados Unidos?* Para lograr esta tarea, deberíamos seguir los siguientes pasos:

1. `filter()` el marco de datos para las filas, donde la variable `year` es igual a 2016
2. `select()` las dos variables `state` y `dem`, ya que no estamos interesados en el resto de las columnas.
3. `arrange()` el marco de datos filtrado y seleccionado en función de la columna `dem` de forma descendente.

Los pasos y funciones descritos anteriormente deben ejecutarse uno tras otro, donde la entrada de cada función es la salida del paso anterior. Aplicando las cosas que aprendió hasta ahora, puede realizar esta tarea siguiendo los siguientes pasos:

```
result <- filter(pres_results, year==2016)
result <- select(result, state, dem)
result <- arrange(result, desc(dem))
tibble(result)
```

```
## # A tibble: 51 x 2
##    state   dem
##    <chr> <dbl>
##  1 DC    0.905
##  2 CA    0.617
##  3 HI    0.610
##  4 MD    0.603
##  5 MA    0.591
##  6 NY    0.584
##  7 IL    0.558
##  8 VT    0.557
##  9 NJ    0.555
```

```
## 10 CT    0.546
## # ... with 41 more rows
```

La primera función toma el marco de datos `pres_results`, lo filtra de acuerdo con la descripción de la tarea y lo asigna a la variable `result`. Luego, cada función posterior toma la variable `result` como entrada y la sobrescribe con su propia salida.

El operador `%>%` proporciona una forma práctica de combinar los pasos anteriores en aparentemente un paso. Toma un marco de datos como entrada inicial. Luego, aplica una lista de funciones y pasa la salida de cada función para la entrada de la siguiente función. La misma tarea que la anterior se puede lograr usando el operador de tubería `%>%` de esta manera:

```
tibble(
pres_results %>%
  filter(year==2016) %>%
  select(state, dem, rep) %>%
  arrange(desc(dem))
)
```

```
## # A tibble: 51 x 3
##    state   dem    rep
##    <chr> <dbl>  <dbl>
##  1 DC    0.905 0.0407
##  2 CA    0.617 0.316
##  3 HI    0.610 0.294
##  4 MD    0.603 0.339
##  5 MA    0.591 0.323
##  6 NY    0.584 0.361
##  7 IL    0.558 0.388
##  8 VT    0.557 0.298
##  9 NJ    0.555 0.414
## 10 CT    0.546 0.409
## # ... with 41 more rows
```

Podemos interpretar el código de la siguiente manera:

1. Definimos el conjunto de datos original como punto de partida.
2. El uso del operador `%>%` justo después del marco de datos le dice a **dplyr**, que viene una función, que toma el marco de datos previamente definido como entrada.
3. Usamos cada función como de costumbre, pero omite el primer parámetro. La entrada del marco de datos se proporciona automáticamente por la salida del paso anterior.
4. Mientras agreguemos el operador `%>%` después de un paso, **dplyr** esperará un paso adicional.
5. En nuestro ejemplo, la tubería se cierra con una función `arrange()`. Ob-

tiene la versión filtrada y seleccionada del marco de datos `pres_results` como entrada y la ordena en función de la columna `dem` de forma descendente. Finalmente, devuelve la salida.

Una diferencia entre los dos enfoques es que el operador `%>%` no guarda permanentemente los resultados intermedios o finales. Para guardar el marco de datos resultante, debemos asignar la salida a una variable:

```
result <- pres_results %<>%
  filter(year==2016) %>%
  select(state, dem) %>%
  arrange(desc(dem))
tibble(result)
```

```
## # A tibble: 51 x 2
##    state   dem
##    <chr> <dbl>
##  1 DC    0.905
##  2 CA    0.617
##  3 HI    0.610
##  4 MD    0.603
##  5 MA    0.591
##  6 NY    0.584
##  7 IL    0.558
##  8 VT    0.557
##  9 NJ    0.555
## 10 CT    0.546
## # ... with 41 more rows
```

3.5.2 Ejercicios:

Esperanza de vida austriaca

Utiliza el operador `%>%` en el conjunto de datos de `gapminder` y crea un simple marco de datos para responder a la siguiente pregunta: *¿Cómo ha cambiado la esperanza de vida en Austria en las últimas décadas?* Los paquetes necesarios ya están cargados.

1. Defina el marco de datos de `gapminder` como el marco de datos de base
2. Filtra sólo las filas en las que la columna `country` es igual a `Austria` mediante la tubería `gapminder` a la función `filter()`.
3. Selecciona sólo las columnas: `year` y `lifeExp` del resultado filtrado.
4. Ordena los resultados en base a la columna `year` en función de las columnas seleccionadas.

Solución:

```
gapminder %>%
  filter(country == "Austria") %>%
```

```
  select(year, lifeExp) %>%
  arrange(year)
```

```
## # A tibble: 12 x 2
##     year lifeExp
##    <int>   <dbl>
##  1  1952    66.8
##  2  1957    67.5
##  3  1962    69.5
##  4  1967    70.1
##  5  1972    70.6
##  6  1977    72.2
##  7  1982    73.2
##  8  1987    74.9
##  9  1992    76.0
## 10  1997    77.5
## 11  2002    79.0
## 12  2007    79.8
```

PIB europeo per cápita

Utiliza el operador `%>%` en el conjunto de datos de `gapminder` y crea un simple tibble para responder a la siguiente pregunta: *¿Qué país europeo tuvo el mayor PIB per cápita en 2007?* Los paquetes requeridos ya están cargados.

1. Defina el tibble de `gapminder` como la entrada
2. Filtra sólo las filas donde la columna `year` es igual a `2007`
3. Usar una segunda capa de filtro y mantener sólo las filas donde la columna del continente es igual a `Europe`
4. Seleccione sólo las columnas: `country` y `gdpPercap`
5. Ordena los resultados basados en la columna `gdpPercap` de forma descendente

```
gapminder %>%
  filter(year == 2007, continent == "Europe") %>%
  select(country, gdpPercap) %>%
  arrange(desc(gdpPercap))
```

```
## # A tibble: 30 x 2
##    country     gdpPercap
##    <fct>           <dbl>
##  1 Norway         49357.
##  2 Ireland        40676.
##  3 Switzerland    37506.
##  4 Netherlands    36798.
##  5 Iceland        36181.
##  6 Austria        36126.
```

```
##  7 Denmark        35278.
##  8 Sweden         33860.
##  9 Belgium        33693.
## 10 Finland        33207.
## # ... with 20 more rows
```

Población de las Américas

Utiliza el operador `%>%` en el conjunto de datos de `gapminder` y crea un simple tibble para responder a la siguiente pregunta: *¿Qué país del continente americano tenía la mayor población en 2007?*

1. Defina el tibble de `gapminder` como la entrada
2. Filtra sólo las filas donde la columna `year` es igual a `2007`
3. Utilice una segunda capa de filtro y mantenga sólo las filas donde la columna del `continent` es igual a `Americas`
4. Seleccione sólo las columnas: `country` y `pop`
5. Arregla los resultados basados en la columna `pop` de forma descendente

```
gapminder %>%
  filter(year == 2007, continent == "Americas") %>%
  select(country, pop) %>%
  arrange(desc(pop))
```

```
## # A tibble: 25 x 2
##    country             pop
##    <fct>             <int>
##  1 United States 301139947
##  2 Brazil        190010647
##  3 Mexico        108700891
##  4 Colombia       44227550
##  5 Argentina      40301927
##  6 Canada         33390141
##  7 Peru           28674757
##  8 Venezuela      26084662
##  9 Chile          16284741
## 10 Ecuador        13755680
## # ... with 15 more rows
```

Chapter 4

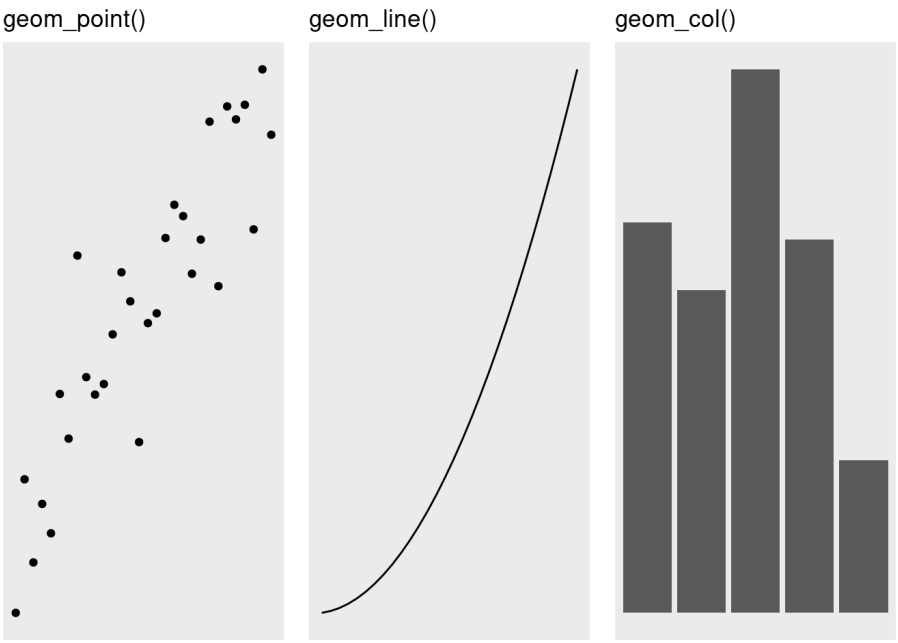
Visualización de datos con ggplot2

Comprender los tipos de gráficos y los principios básicos de la gramática de los gráficos. Cree sus primeras visualizaciones con el paquete ggplot2 que incluye diagramas de dispersión, gráficos de líneas y gráficos de barras.

4.1 Por qué es importante la visualización de datos

La visualización de datos no solo es importante para comunicar los resultados, sino también una técnica poderosa para el análisis exploratorio de datos. Cada tipo de diagrama, como diagramas de dispersión, gráficos de líneas, gráficos de barras e histogramas, tiene su propio propósito y se puede aprovechar de una manera poderosa utilizando el paquete **ggplot2**.

- Comprender los diferentes roles de la visualización de datos.
- Comprender los diferentes tipos de parcelas disponibles.
- Obtenga una descripción general del paquete **ggplot2**.



4.1.1 Introducción a la visualización de datos

Una imagen vale mas que mil palabras.

La visualización de datos es la técnica más rápida y poderosa para comprender información nueva y existente. Durante una fase de exploración inicial, los científicos de datos intentan revelar las características subyacentes de un conjunto de datos, como diferentes distribuciones, correlaciones u otros patrones visibles. Este proceso también se denomina *análisis exploratorio de datos* (EDA) y marca el punto de partida de cada proyecto de ciencia de datos.

Los gráficos producidos durante la EDA muestran al científico de datos las direcciones del viaje por delante. Los patrones revelados pueden inspirar hipótesis sobre los procesos subyacentes, las características del conjunto de datos que se extraerán o las técnicas de modelado que se probarán. Por último, pero no menos importante, las visualizaciones descubren valores atípicos y errores de datos de los que el científico de datos debe ocuparse.

El papel más importante de la visualización de datos es la comunicación de los hallazgos de la ciencia de datos a colegas y clientes a través de presentaciones, informes o paneles. El esfuerzo utilizado para la EDA y las visualizaciones es un tiempo bien invertido, ya que los resultados se pueden utilizar directamente para comunicar los hallazgos.

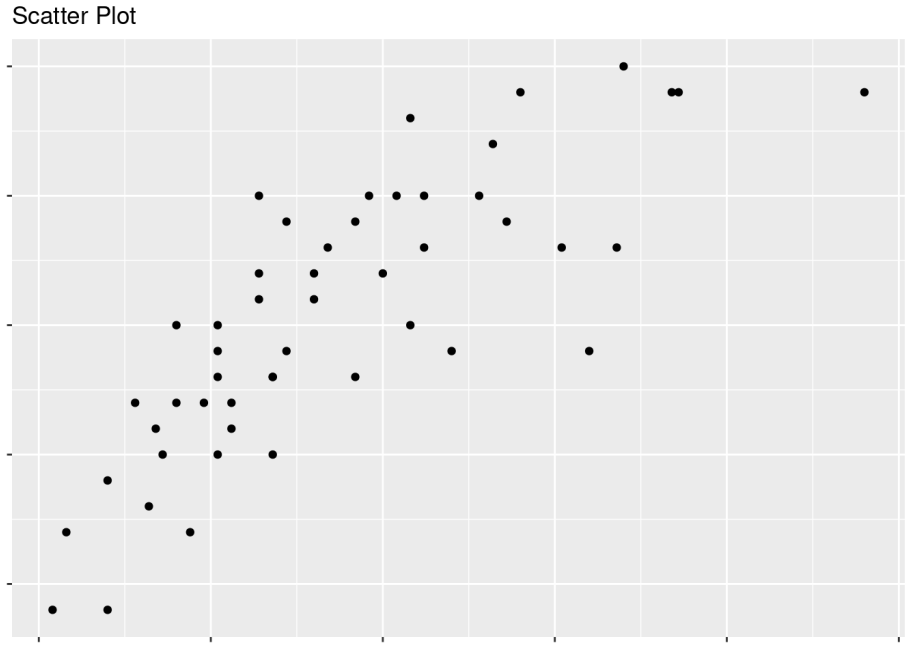
4.1.2 Tipos de gráficas disponibles

Hay muchos tipos de gráficos disponibles que ayudan a comprender las diferentes características y relaciones en el conjunto de datos.

Durante la fase de análisis de datos exploratorios, normalmente queremos detectar los patrones más obvios observando cada variable de forma aislada o detectando relaciones de variables con otras. El tipo de gráfico utilizado también está determinado por el tipo de datos de las variables de entrada, como numéricas o categóricas.

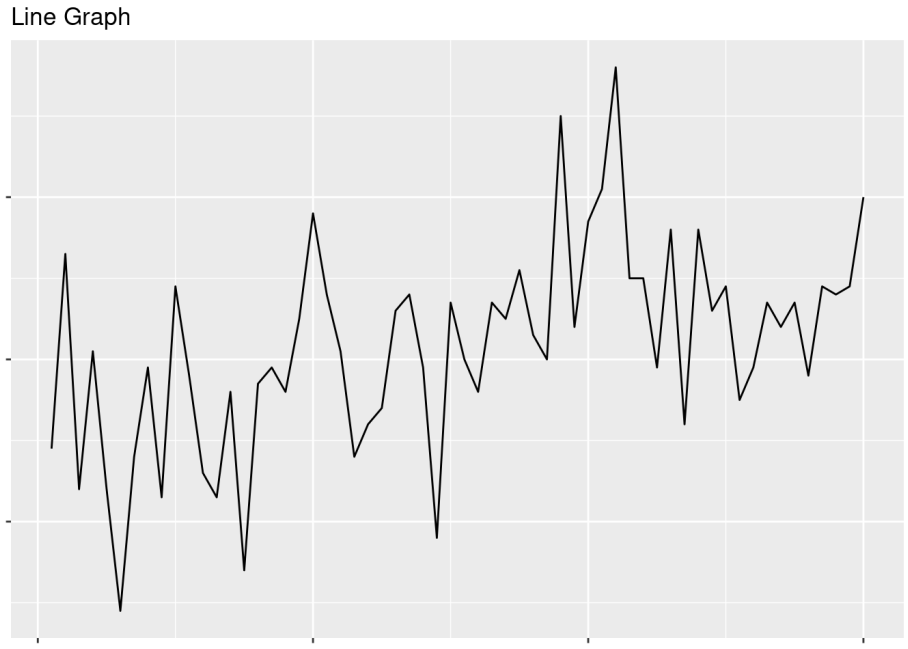
4.1.2.1 Gráfico de dispersión

Los gráficos de dispersión se utilizan para visualizar la relación entre dos variables numéricas. La posición de cada punto representa el valor de las variables en los ejes x e y.



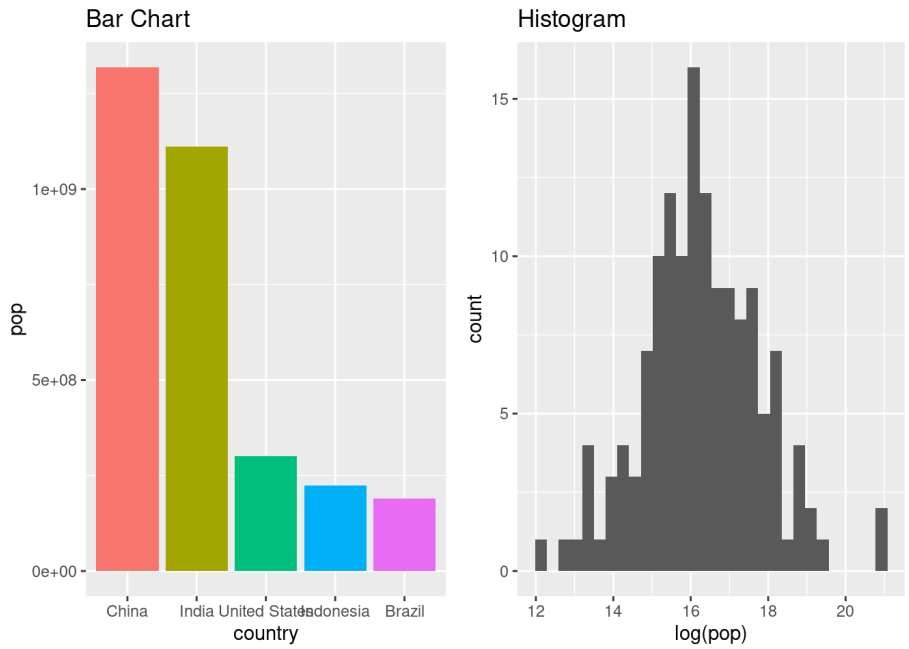
4.1.2.2 Gráficos lineales

Los gráficos de líneas se utilizan para visualizar la trayectoria de una variable numérica contra otra que están conectadas a través de líneas. Son muy adecuados si los valores solo cambian **continuamente**, como la temperatura con el tiempo.



4.1.2.3 Gráficos de barras e histogramas

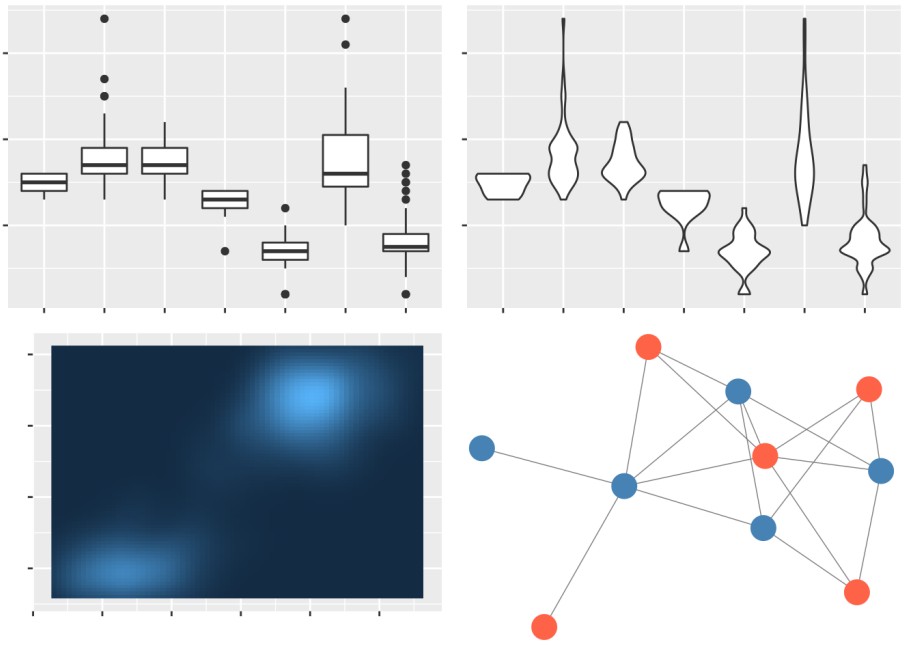
Los gráficos de barras visualizan valores `numéricos` agrupados por categorías. Cada categoría está representada por una barra con una altura definida por cada valor `numérico`. Los histogramas son gráficos de barras específicos para resumir el número de apariciones de valores numéricos en un conjunto de rangos de valores (o bins). Suelen utilizarse para determinar la *distribución* de valores numéricos.



4.1.2.4 Otros

Otros tipos de gráficos de uso frecuente en la ciencia de datos incluyen:

- **Diagramas de caja**: Muestra información de distribución de valores numéricos agrupados en categorías como cajas. Genial para comparar rápidamente múltiples distribuciones.
- **Gráficos de violín**: igual que los gráficos de caja, pero muestran distribuciones como *violines*.
- **Mapas de calor**: muestran las interacciones de las variables, generalmente correlaciones, como imágenes rasterizadas que resaltan áreas de alta interacción.
- **Gráficos de red**: muestra conexiones entre nodos



4.1.3 Presentamos: ggplot2

Debido a la importancia de la visualización para la ciencia de datos y las estadísticas, R ofrece un amplio conjunto de herramientas y paquetes. El lenguaje R básico ya proporciona un rico conjunto de funciones de trazado y tipos de trazado. Estas funciones de trazado requieren que los usuarios especifiquen cómo trazar cada elemento en el lienzo paso a paso. Por el contrario, el paquete **ggplot2** permite la especificación de trazados a través de un conjunto de *capas* de trazado. Esto requiere que el paquete averigüe los pasos necesarios para producir el gráfico.

A través del conjunto predefinido de capas geométricas, facetas y temas, **ggplot2** permite a los usuarios crear hermosos gráficos en muy poco tiempo. **ggplot2** es también la biblioteca de trazado más adoptada en la comunidad R.

4.2 Crea un diagrama de dispersión con ggplot

Realice sus primeros pasos con el paquete **ggplot2** para crear un diagrama de dispersión. Utilice la gramática de los gráficos para asignar los atributos del conjunto de datos a su gráfico y conectar diferentes capas con el operador `+`.

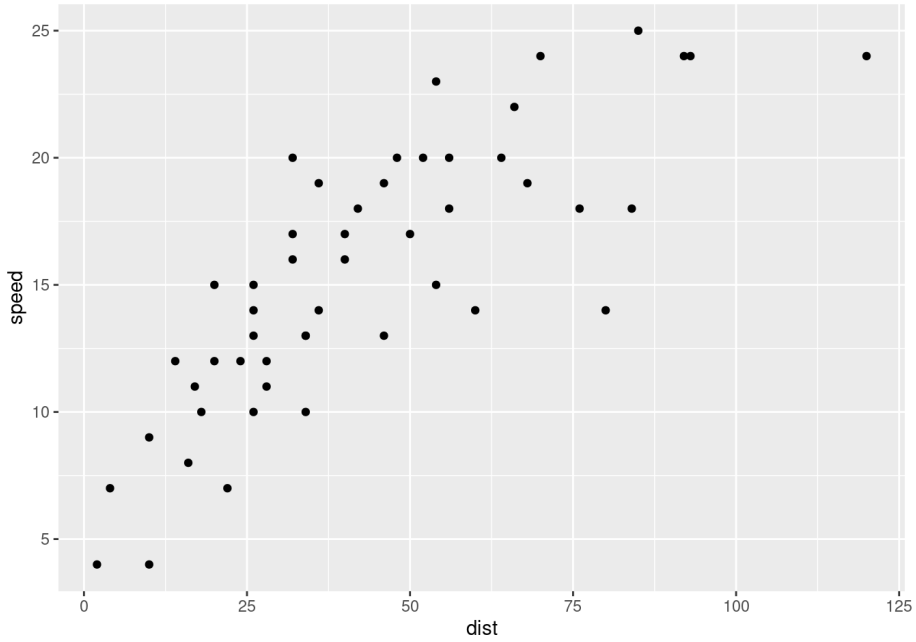
- Defina un conjunto de datos para la gráfica usando la función `ggplot()`
- Especifique una capa geométrica usando la función `geom_point()`

- Asigne los atributos del conjunto de datos a las propiedades de trazado usando el parámetro de `mapping`
- Conectar diferentes objetos `ggplot` usando el operador `+`

```
library(ggplot2)
ggplot(___) +
  geom_point(
    mapping = aes(x = ___, y = ___)
  )
```

4.2.1 Introducción a los gráficos de dispersión

Los diagramas de dispersión utilizan puntos para visualizar la relación entre dos variables numéricas. La posición de cada punto representa el valor de las variables en los ejes X e Y. Veamos un ejemplo de un diagrama de dispersión para comprender la relación entre la *speed* y la *distancia de frenado* de los automóviles:



Cada punto representa un automóvil. Cada automóvil comienza a frenar a una velocidad dada en el eje-y y recorre la distancia que se muestra en el eje-x hasta detenerse por completo. Si echamos un vistazo a todos los puntos de la trama, podemos ver claramente que los coches más rápidos necesitan más distancia para detenerse por completo.

4.2.2 Especificar un conjunto de datos

Para crear gráficos con **ggplot2**, primero debe cargar el paquete usando la `library(ggplot2)`.

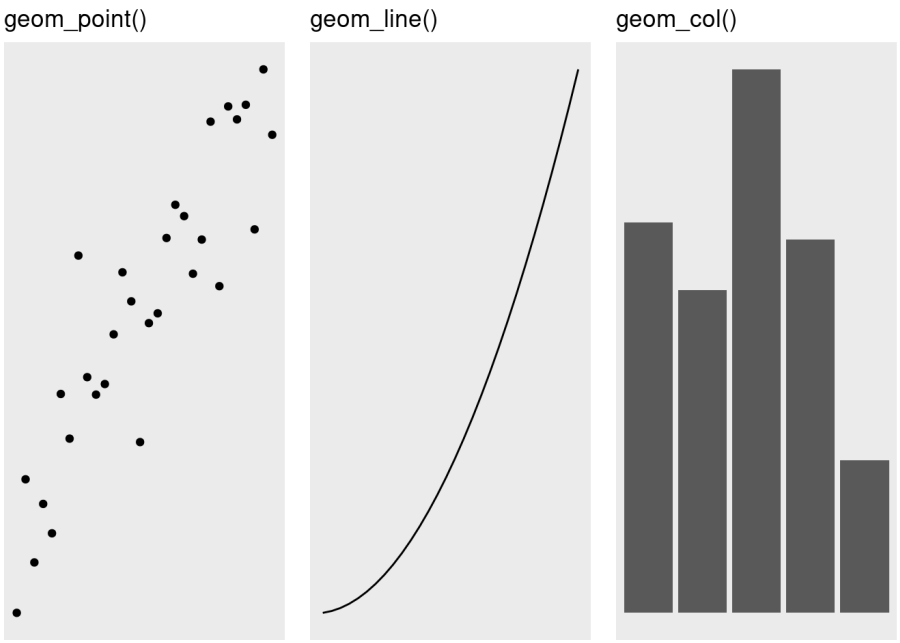
Una vez que se haya cargado el paquete, especifique el conjunto de datos que se utilizará como argumento de la función `ggplot()`. Por ejemplo, para especificar una gráfica usando el conjunto de datos de `cars`, puede usar:

```
library(ggplot2)
library(gapminder)
ggplot(cars)
```


Tenga en cuenta que este comando no traza nada más que un lienzo gris todavía. Simplemente define el conjunto de datos para el gráfico y crea una base vacía sobre la cual podemos agregar capas adicionales.

4.2.3 Especificando una capa geométrica

Podemos usar las capas geométricas de **ggplot** (o *geoms*) para definir cómo queremos visualizar nuestro conjunto de datos. Las *geoms* utilizan objetos geométricos para visualizar las variables de un conjunto de datos. Los objetos pueden tener múltiples formas como puntos, líneas y barras y se especifican mediante las funciones correspondientes `geom_point()`, `geom_line()` y `geom_col()`:



4.2.4 Creando mapeos estéticos

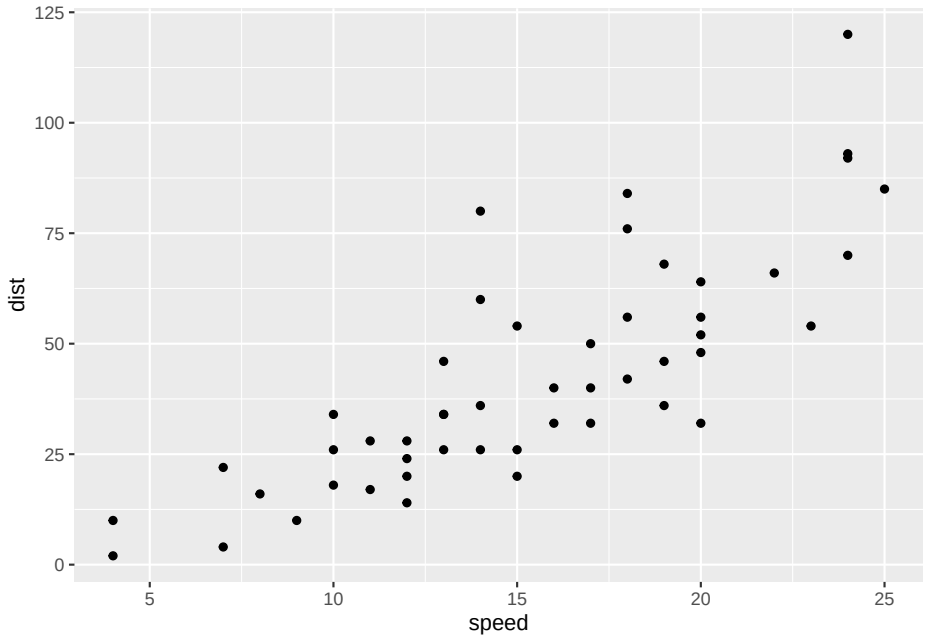
ggplot2 utiliza el concepto de *estética*, que *asigna* los atributos del conjunto de datos a las características visuales de la gráfica. Cada capa geométrica requiere un conjunto diferente de *asignaciones estéticas*, p. Ej. la función `geom_point()` usa la estética `x` e `y` para determinar las coordenadas de los ejes `x` e `y` de los puntos a graficar. La estética se asigna dentro de la función `aes()` para construir las asignaciones finales.

Para especificar una capa de puntos que traza la velocidad variable en el eje-x y la distancia `dist` en el eje-y, podemos escribir:

```
geom_point(
  mapping = aes(x=speed, y=dist)
)
```

La expresión anterior construye una capa geométrica. Sin embargo, esta capa actualmente no está vinculada a un conjunto de datos y no produce una gráfica. Para **vincular** la capa con un objeto `ggplot` que especifica el conjunto de datos de `cars`, necesitamos conectar el objeto `ggplot(cars)` con la capa `geom_point()` usando el operador `+`:

```
ggplot(cars) +
  geom_point(
    mapping = aes(x=speed, y=dist)
  )
```



A través del enlace, `ggplot()` sabe que las variables de `speed` y `dist` asignadas se toman del conjunto de datos de `cars`. `geom_point()` instruye a ggplot para trazar las variables mapeadas como puntos.

Los pasos necesarios para crear un diagrama de dispersión con `ggplot` se pueden resumir de la siguiente manera:

1. Cargue el paquete **ggplot2** usando `library(ggplot2)`.
2. Especifique el conjunto de datos que se graficará usando `ggplot()`.
3. Utilice el operador `+` para agregar capas al gráfico.
4. Agregue una capa geométrica para definir las formas que se graficaran. En caso de gráficos de dispersión, use `geom_point()`.
5. Asigne variables del conjunto de datos a propiedades de trazado a través del parámetro de `mapping` en la capa geométrica.

4.3 Especificar estética adicional para puntos

ggplot2 implementa la gramática de gráficos para mapear atributos de un conjunto de datos para trazar características a través de la estética. Este marco se puede utilizar para ajustar el `tamaño de punto`, el `color` y la transparencia `alpha` de los puntos en un diagrama de dispersión.

- Agregue dimensiones de trazado adicionales a través de la estética
- Ajuste el tamaño del punto de un gráfico de dispersión usando el parámetro `size`

- Cambiar el color del punto de un diagrama de dispersión usando el parámetro `color`
- Establecer un parámetro `alpha` para cambiar la transparencia de todos los puntos
- Diferenciar entre mapeos estéticos y parámetros constantes

```
ggplot(___) +
  geom_point(
    mapping = aes(x = ___, y = ___,
                  color = ___,
                  size  = ___),
    alpha  = ___
  )
```

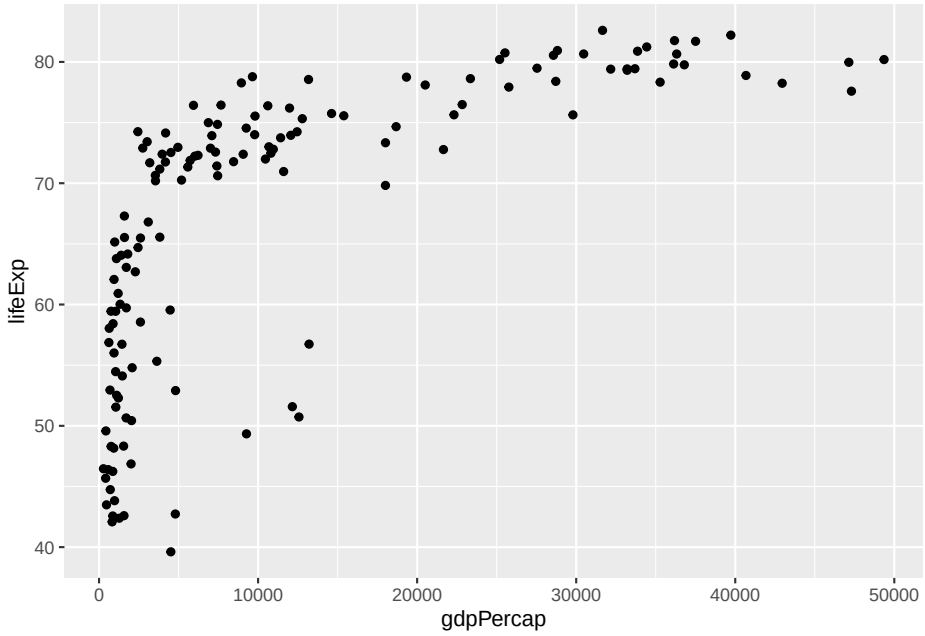
4.3.1 Añadiendo más estética a la trama

En su forma más básica, los diagramas de dispersión solo pueden visualizar conjuntos de datos en dos dimensiones a través de la estética `x` e `y` de la capa `geom_point()`. Sin embargo, la mayoría de los conjuntos de datos tienen más de dos variables y, por lo tanto, pueden requerir dimensiones de trazado adicionales. `ggplot()` hace que sea muy fácil mapear variables adicionales a diferentes estéticas de trazado como `size`, transparencia `alpha` y `color`.

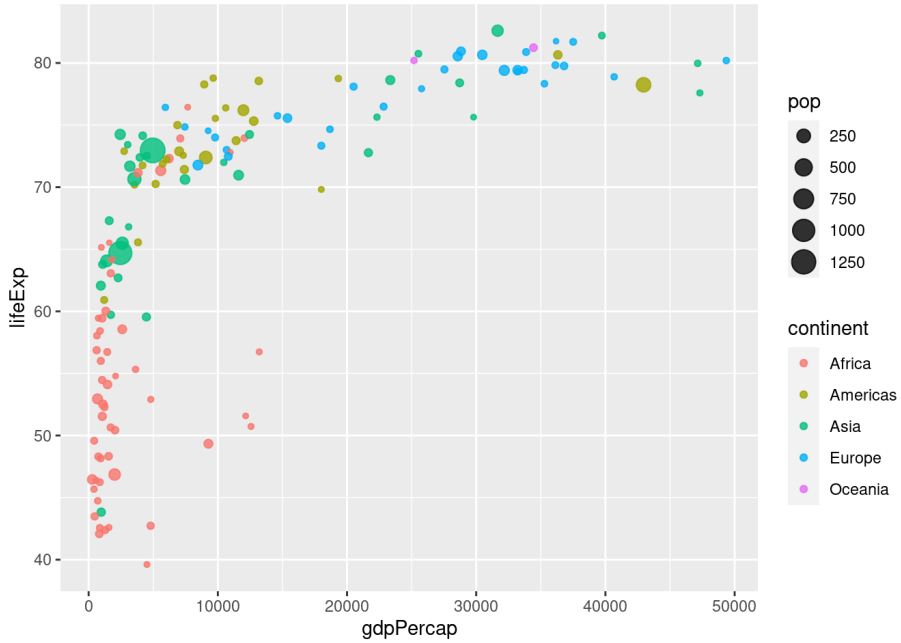
Consideremos el conjunto de datos `gapminder_2007` que contiene las variables PIB per cápita `gdpPercap` y esperanza de vida `lifeExp` para 142 países en el año 2007:

```
library(tidyverse)
library(gapminder)
## select data
gapminder_2007 <- gapminder%>%
  filter(year == 2007)
```

```
ggplot(gapminder_2007) +
  geom_point(aes(x = gdpPercap, y = lifeExp))
```



Al mapear la variable `continent` a través de la estética del `color` del punto y la población `pop` (en millones) a través del tamaño del punto `size`, obtenemos una gráfica mucho más rica que incluye 4 variables diferentes del conjunto de datos:

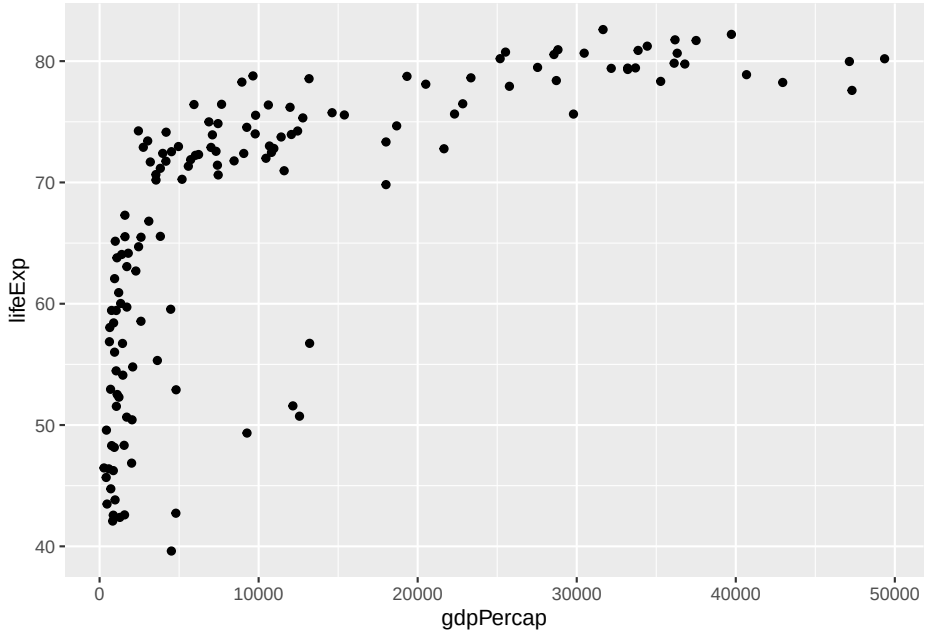


4.3.2 Ajuste del color del punto

Normalmente, el color del punto se utiliza para introducir una nueva dimensión en un diagrama de dispersión. En ggplot usamos la estética del `color` para especificar el mapeo de una variable al color de los puntos.

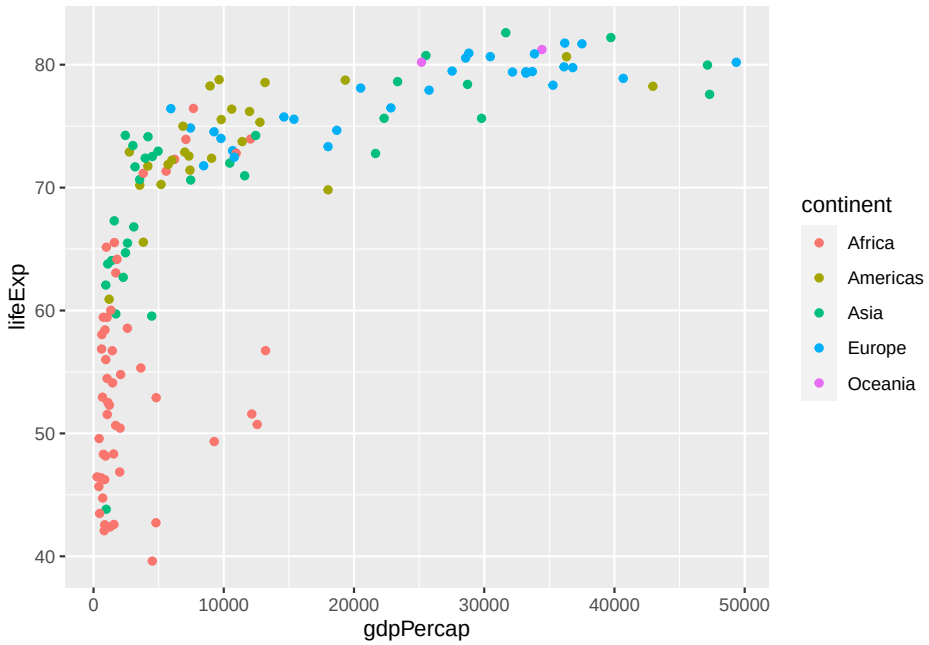
Para el conjunto de datos `gapminder_2007` podemos trazar el PIB per cápita `gdpPercap` frente a la esperanza de vida `lifeExp` de la siguiente manera:

```
ggplot(gapminder_2007) +
  geom_point(aes(x = gdpPercap, y = lifeExp))
```



Para colorear cada punto en función del `continent` de cada país podemos utilizar:

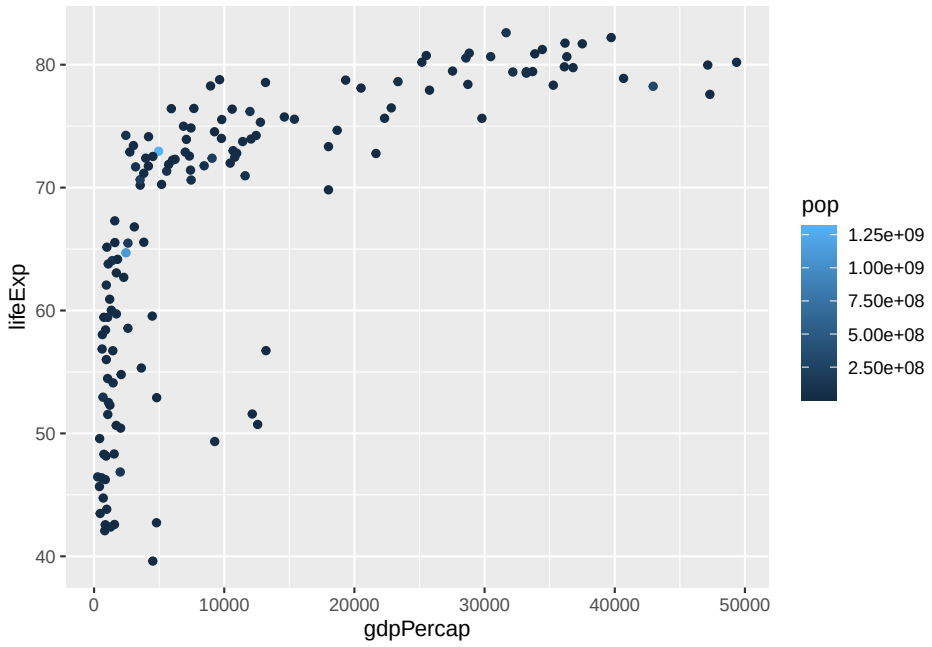
```
ggplot(gapminder_2007) +
  geom_point(aes(x = gdpPercap, y = lifeExp,
                 color = continent))
```



Vemos que en la trama resultante cada punto está coloreado de manera diferente según el `continent` de cada país. `ggplot` usa el esquema de coloración basado en el tipo de datos categóricos de la variable `continent`.

Por el contrario, veamos cómo se ve el gráfico si coloreamos los puntos por el `pop` de población de la variable `numeric`:

```
ggplot(gapminder_2007) +
  geom_point(aes(x = gdpPercap, y = lifeExp,
                 color = pop))
```

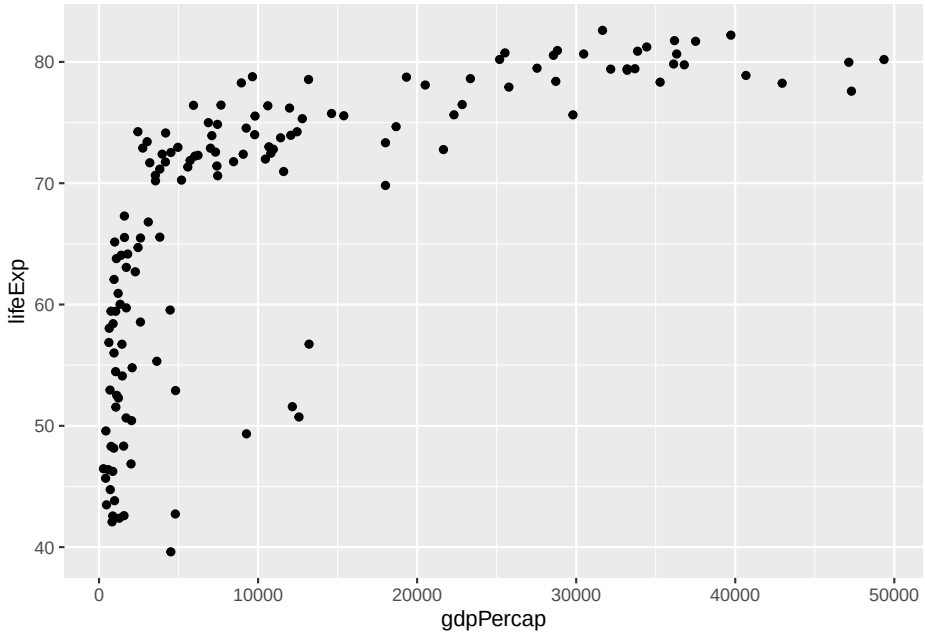


La escala cambia inmediatamente a continua como se puede ver en la leyenda y los puntos celestes son ahora los países con mayor número de población (China e India).

4.3.3 Ajustar el tamaño del punto

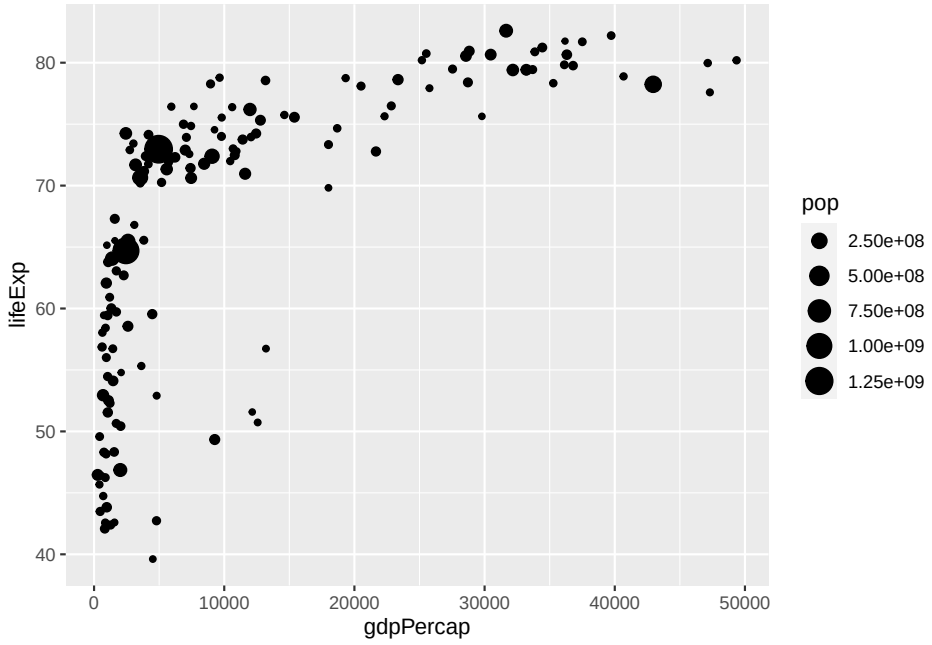
Para el conjunto de datos de `gapminder_2007` podemos trazar el PIB per cápita `gdpPercap` frente a la esperanza de vida de la siguiente manera:

```
ggplot(gapminder_2007) +
  geom_point(aes(x = gdpPercap,
                 y = lifeExp))
```



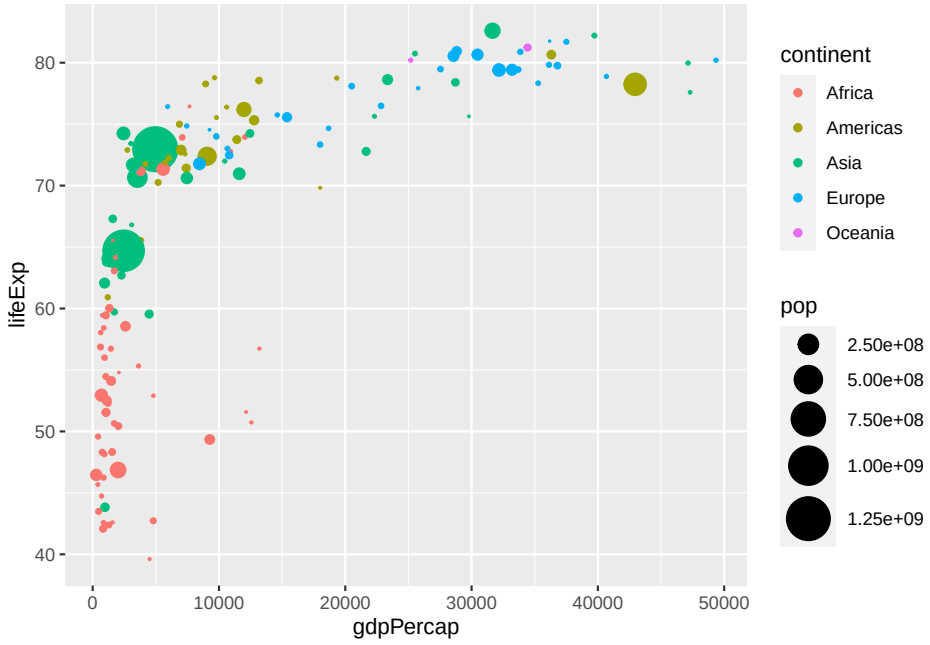
Para ajustar el tamaño de puntos en función de la población (`pop`) de cada país, podemos usar:

```
ggplot(gapminder_2007) +
  geom_point(aes(x = gdpPercap, y = lifeExp,
                 size = pop))
```



Vemos que los tamaños de puntos en el gráfico anterior no reflejan claramente las diferencias de población en cada país. Si comparamos el tamaño en puntos que representa una población de 250 millones de personas con el que muestra 750 millones, podemos ver que sus tamaños no son proporcionales. En su lugar, los tamaños en puntos están agrupados de forma predeterminada. Para reflejar las diferencias de población reales por el tamaño de puntos, podemos usar la función `scale_size_area()` en su lugar. La información de escala se puede agregar como cualquier otro objeto ggplot con el operador `+`:

```
ggplot(gapminder_2007) +
  geom_point(aes(x = gdpPercap, y = lifeExp,
                 color = continent,
                 size = pop)) +
  scale_size_area(max_size = 10)
```



Tenga en cuenta que hemos ajustado el tamaño del punto con `max_size`, lo que da como resultado tamaños de punto más grandes.

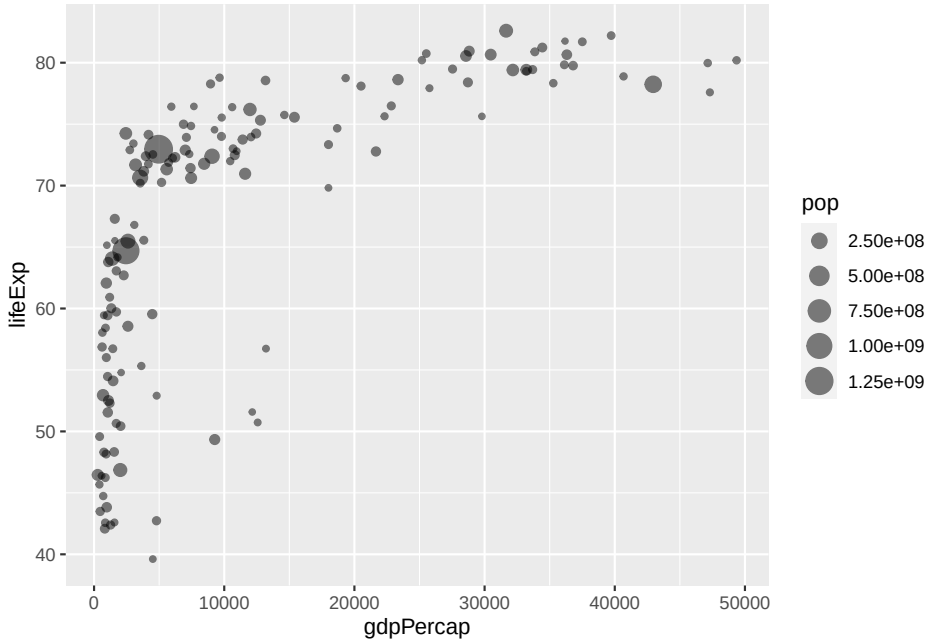
4.3.4 Establecer la estética global: transparencia

Trazar muchos puntos con coordenadas X e Y similares en un gráfico puede producir nubes de puntos densas. Muchos puntos en estas nubes están sobretrazados y el número real de observaciones en un área determinada ya no es visible. Como solución, podemos establecer la transparencia de cada punto usando el parámetro ggplot `alpha`.

Dado que **no** queremos establecer la transparencia del punto **individualmente** para cada punto sino **globalmente** para todos los puntos, no establecemos el parámetro `alpha` como un mapeo estético (dentro de `aes()`) sino fuera.

Establecemos la **opacidad** de cada punto al 50% a través del parámetro `alpha` **outside** como parámetro constante:

```
ggplot(gapminder_2007) +
  geom_point(aes(x = gdpPercap, y = lifeExp, size = pop),
             alpha = 0.5)
```



Ahora podemos ver claramente cuántos puntos se superponen entre sí y la opacidad de cada punto se establece en `0.5`.

4.4 Crea un gráfico de líneas con ggplot

Use la estética `geom_line()` para dibujar gráficos de líneas y personalizar su estilo usando el parámetro `color`. Especifique qué coordenadas usar para cada línea con el parámetro `group`.

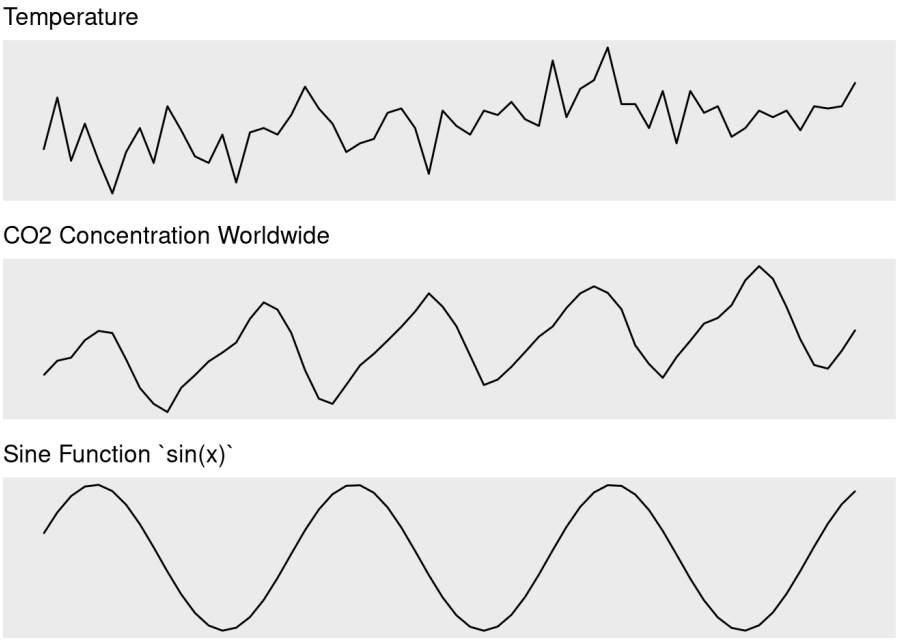
- Crea tu primer gráfico de línea usando `geom_line()`
- Defina cómo se conectan las diferentes líneas mediante el parámetro `group`
- Cambiar el color de línea de un gráfico de líneas usando el parámetro `color`

```
ggplot(___) +
  geom_line(
    mapping = aes(x = ___, y = ___,
                  group = ___,
                  color = ___)
)
```

4.4.1 Introducción a los gráficos lineales

Los gráficos de líneas se utilizan para visualizar la trayectoria de una variable numérica contra otra. A diferencia de los diagramas de dispersión, las coordenadas `x` e `y` no se visualizan a través de puntos, sino que están conectadas a

través de líneas. Los gráficos de líneas se utilizan con mayor frecuencia si una variable cambia *continuamente* contra otra variable numérica, como es el caso de la mayoría de los gráficos de series de tiempo (por ejemplo, precios, clientes, concentración de CO2, temperatura a lo largo del tiempo), funciones continuas (por ejemplo, sinusoidal (x)) u otras relaciones casi continuas (curvas de oferta/demanda del mundo real).



4.4.2 Crear un gráfico lineal simple

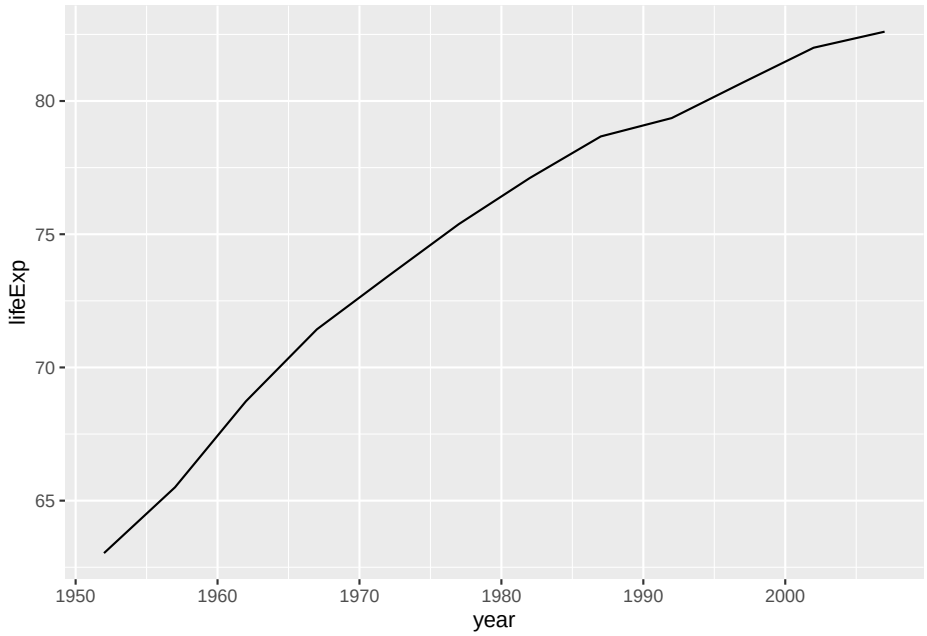
Japón se encuentra entre los países con mayor esperanza de vida. Utilizando el conjunto de datos `gapminder_japan`, determinamos cómo se ha desarrollado la esperanza de vida en Japón a lo largo del tiempo. Necesitamos que:

1. Especifique el conjunto de datos dentro de `ggplot()`
2. Definir la capa de trazado `geom_line()`
3. Asigne el `year` al eje x y la esperanza de vida `lifeExp` al eje y con la función `aes()`

Tenga en cuenta que la biblioteca **ggplot2** debe cargarse primero con `library(ggplot2)`.

```
library(ggplot2)
gapminder_japan <- gapminder %>%
  filter(country == "Japan")
## gráfica
ggplot(gapminder_japan) +
```

```
  geom_line(
    mapping = aes(x = year, y = lifeExp)
)
```



4.4.3 Agregar más líneas

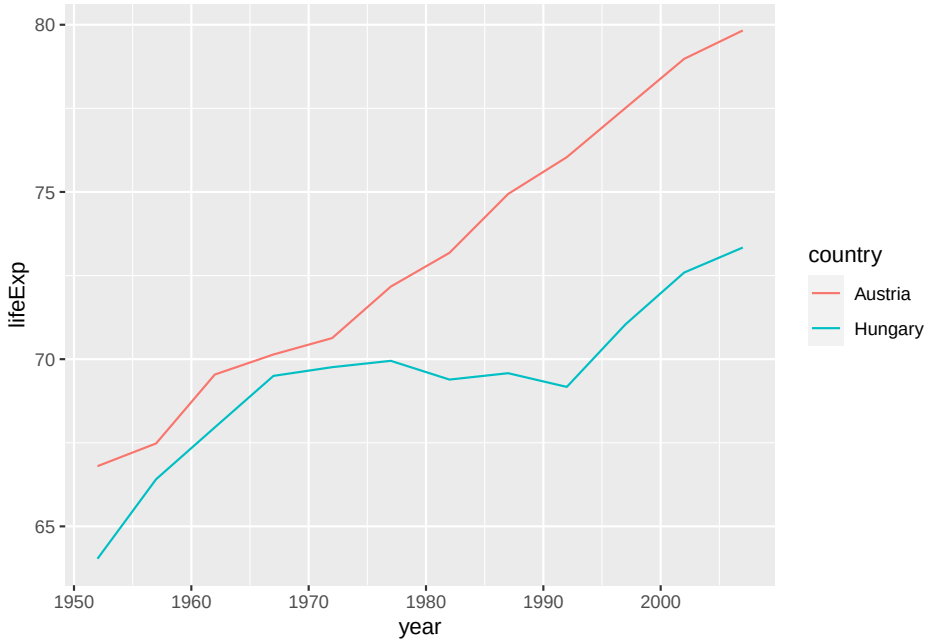
Hasta ahora solo nos hemos centrado en líneas simples, pero ¿qué pasa si tenemos varios países en el conjunto de datos y queremos diferenciarlos de alguna manera?

Los gráficos de líneas a menudo se amplían y se utilizan para comparar dos o más líneas. Los gráficos de líneas múltiples muestran las diferencias absolutas entre las observaciones, pero también cómo las trayectorias específicas se relacionan entre sí. Por ejemplo, respondamos a la pregunta: *¿Cómo ha cambiado la esperanza de vida en los países Austria y Hungría a lo largo del tiempo?*

Primero filtramos el conjunto de datos para ambos países de interés. Luego, establecemos la variable `country` como argumento de `group` para el mapeo estético. El argumento de grupo le dice a ggplot qué observaciones pertenecen juntas y deben conectarse a través de líneas. Al especificar la variable `country`, ggplot crea una línea separada para cada país. Para que las líneas sean más fáciles de distinguir, también asignamos `color` a `country` para que cada línea de país tenga un color diferente.

```
gapminder_comparison <-
  filter(gapminder, country %in% c("Austria", "Hungary"))
```

```
ggplot(data = gapminder_comparison) +
  geom_line(mapping = aes(x = year, y = lifeExp,
                          group = country,
                          color = country)
            )
```



Tenga en cuenta que ggplot también separa las líneas correctamente si solo se especifica la asignación `color` (el parámetro `group` se establece implícitamente).

4.5 Crea tu primer gráfico de barras

- Crea tu primer gráfico de barras usando `geom_col()`
- Rellenar barras con color usando la estética `fill`

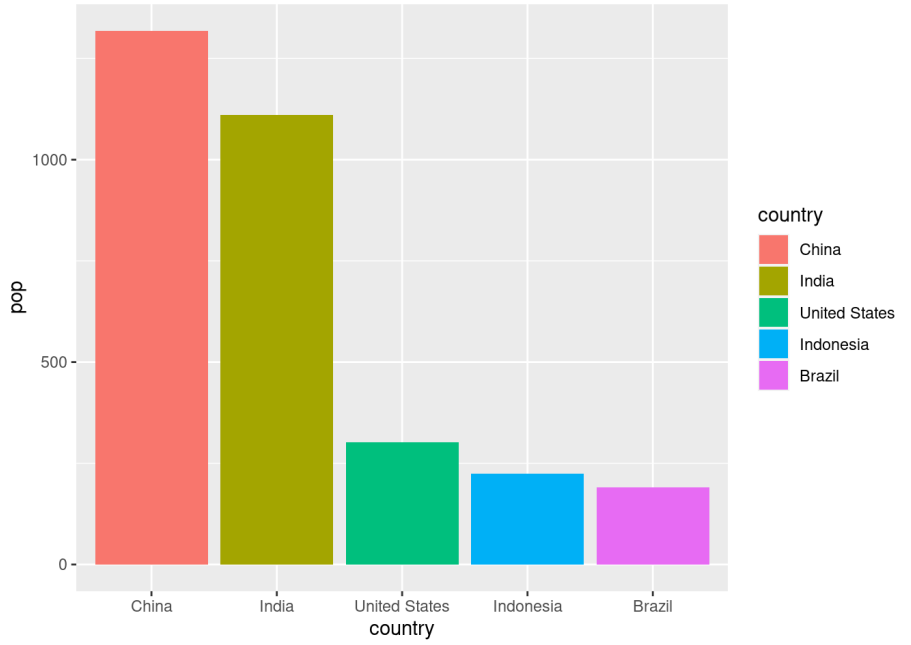
```
ggplot(___) +
  geom_col(
    mapping = aes(x = ___, y = ___,
                  fill = ___)
 )
```

4.5.1 Introducción a los gráficos de barras

Los gráficos de barras visualizan valores `numéricos` agrupados por categorías. Cada categoría está representada por una barra con una altura definida por cada valor `numérico`.

Los gráficos de barras son adecuados para comparar valores entre diferentes grupos, p. Ej. número de votos por partidos, número de personas en diferentes países o PIB per cápita en diferentes países. Los gráficos de barras son un poco espaciosos y funcionan mejor si el número de grupos a comparar es bastante pequeño.

A continuación, puede encontrar un ejemplo que muestra la cantidad de personas (en millones) en los cinco países más grandes por población en 2007:



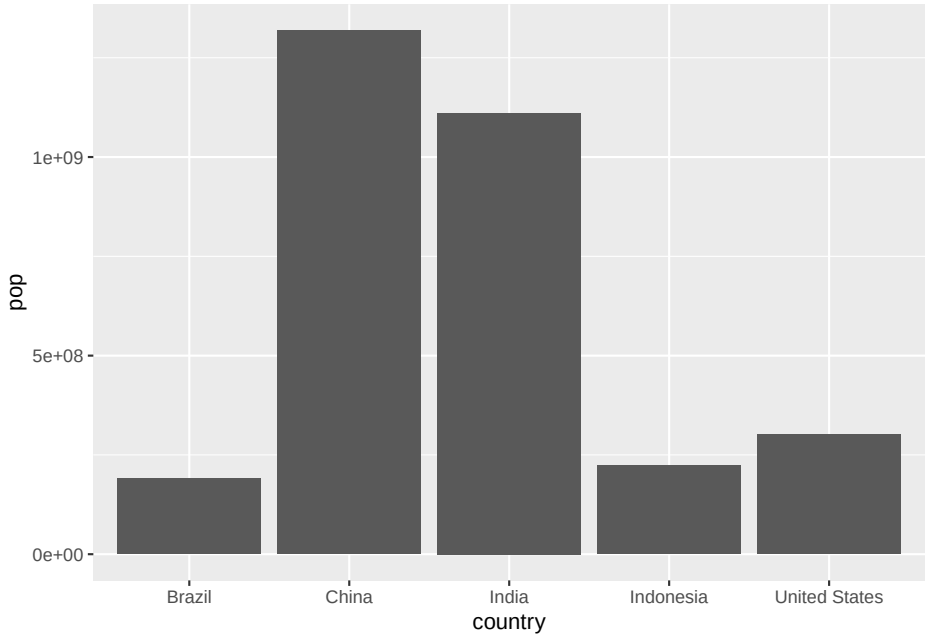
4.5.2 Crear un gráfico de barras simple

En **ggplot2**, los gráficos de barras se crean utilizando la capa geométrica `geom_col()`. La capa `geom_col()` requiere el mapeo estético `x` que define las diferentes barras a trazar. La altura de cada barra está definida por la variable especificada en el mapeo estético `y`. Ambas asignaciones, `x` e `y`, son necesarias para `geom_col()`.

Creemos nuestro primer gráfico de barras con el conjunto de datos `gapminder_top5`. Contiene datos de población (en millones) y esperanza de vida de los países más grandes por población en 2007.

```
gapminder_top5 <- gapminder %>%
  filter(year == 2007) %>%
  slice_max(pop, n=5)
```

```
ggplot(gapminder_top5) +
  geom_col(aes(x = country, y = pop))
```



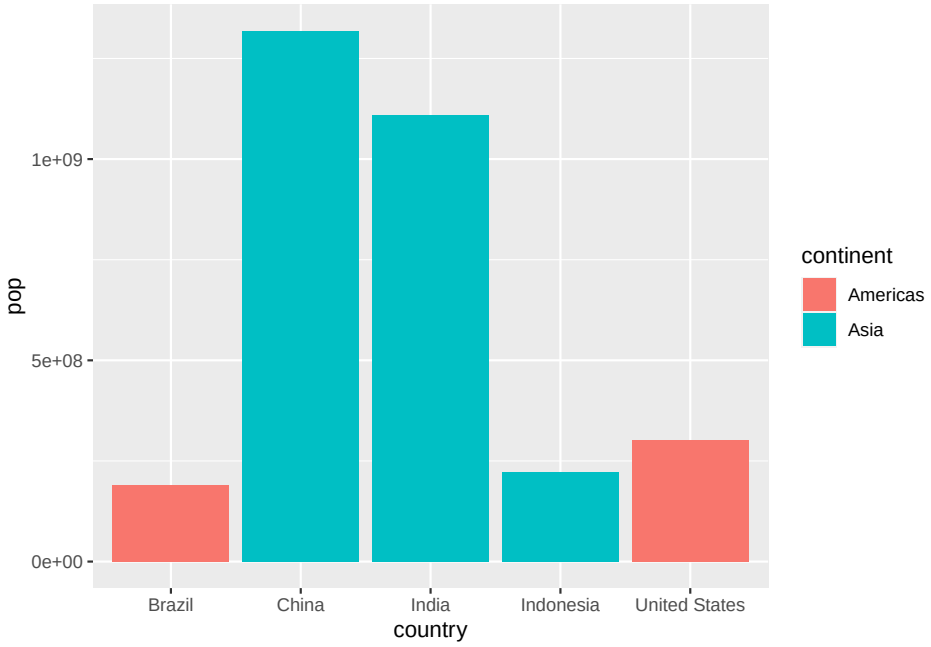
Vemos que las barras resultantes están ordenadas por los nombres de los países en orden alfabético por defecto.

4.5.3 Llenado de barras de color

Al igual que otras geoms, `geom_col()` permite a los usuarios asignar variables de conjuntos de datos adicionales al atributo de color de la barra. La estética `fill` se puede utilizar para llenar de color todas las barras. Una confusión habitual es la estética `color` que especifica el color de la *línea* del borde de cada barra en lugar del color `fill`.

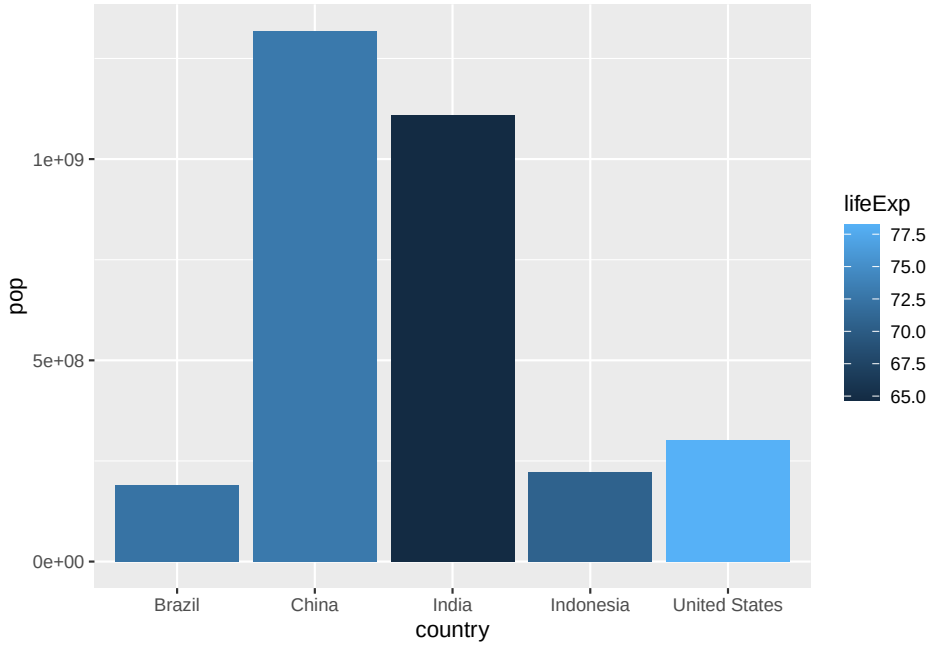
Según el conjunto de datos `gapminder_top5`, trazamos la población (en millones) de los países más grandes y usamos la variable `continent` para colorear cada barra:

```
ggplot(gapminder_top5) +
  geom_col(aes(x = country, y = pop, fill = continent))
```



Dado que la variable `continent` es una variable categórica, las barras tienen un esquema de color claro para cada continente. Veamos qué sucede si usamos una variable `numérica` como la esperanza de vida `lifeExp` en su lugar:

```
ggplot(gapminder_top5) +
  geom_col(aes(x = country, y = pop, fill = lifeExp))
```



Los colores de las barras ahora han cambiado según la leyenda **continua** de la derecha. Vemos que también se pueden utilizar variables `numéricas` para `rellenar` barras.

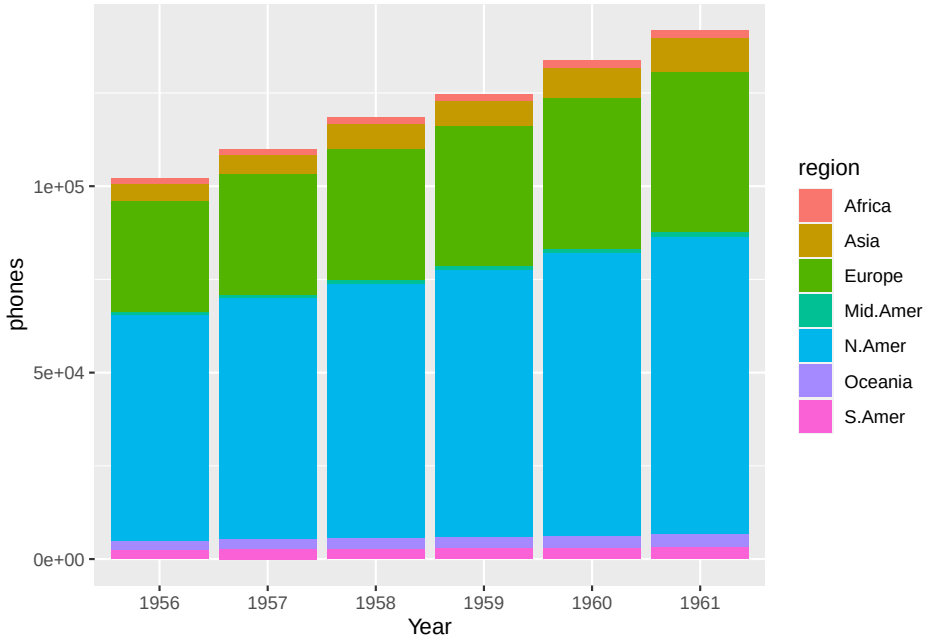
4.5.4 Gráficos de barras apiladas

En algunas circunstancias, puede resultar útil trazar múltiples variables de valores numéricos dentro de cada barra. Los ejemplos son valores numéricos que describen una entidad específica (por ejemplo, clientes) dividida entre varias categorías (segmentos de clientes) de modo que la altura de la barra representa el número total (todos los clientes).

El siguiente gráfico muestra la cantidad de teléfonos (en miles) por continente desde 1956 hasta 1961 como un gráfico de barras apiladas:

```
## Ordenando la base de dato
world_phones <- WorldPhones %>%
  as.data.frame() %>%
  rownames_to_column("Year") %>%
  pivot_longer(cols=-Year, names_to="region", values_to="phones") %>%
  filter(Year >1955)
```

```
ggplot(world_phones) +
  geom_col(aes(x = Year, y = phones,
               fill = region))
```


phones
1e+05
5e+04
0e+00
1956
1957
1958
1959
1960
1961
Year
region
Africa
Asia
Europe
Mid.Amer
N.Amer
Oceania
S.Amer

Chapter 5

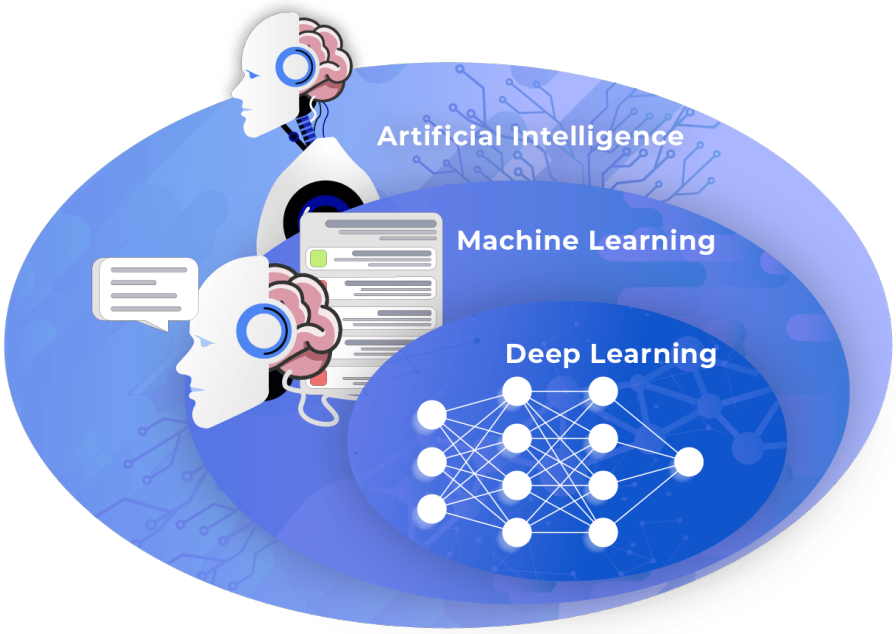
Introducción a Machine Learning

Aprenda los fundamentos del aprendizaje automático, incluidos casos de uso y diferentes técnicas de aprendizaje. Diferenciar entre regresión y clasificación.

¿Qué es Machine Learning?

En los últimos años, términos como inteligencia artificial, aprendizaje automático y aprendizaje profundo se han utilizado mucho. Tan misteriosas como suenan estas palabras, ¿cuál es su diferencia y de qué son capaces?

- Diferenciar entre inteligencia artificial, aprendizaje automático y aprendizaje profundo
- Identificar casos de uso de Machine Learning



5.1 Inteligencia artificial

La **inteligencia artificial (IA)** es la inteligencia demostrada por las máquinas, en contraste con la inteligencia natural asociada con los humanos. Por inteligencia, generalmente nos referimos a la resolución de problemas y tareas complejos, que aparentemente requieren algún tipo de habilidades cognitivas. La inteligencia artificial comenzó como una disciplina académica en la década de 1950 con el supuesto de que las computadoras pueden imitar el complejo razonamiento dentro del cerebro humano. Este campo también se conoce como *inteligencia artificial general (AGI)* y el objetivo de alcanzar una inteligencia similar a la humana, incluso hoy, parece estar fuera de alcance.

Sin embargo, surgió un subcampo llamado aprendizaje automático *machine learning (ML)* que se centra en problemas más específicos, como el reconocimiento de imágenes o la comprensión del lenguaje. Gracias a las aplicaciones de la industria, el aprendizaje automático se ha vuelto cada vez más popular y atrae a investigadores e inversores a nivel mundial.

5.2 Machine Learning (Aprendizaje automático)

El **machine learning (ML)** es un subcampo de la inteligencia artificial. Su objetivo es construir y aplicar modelos sofisticados sin la necesidad de reglas

e instrucciones codificadas. En cambio, los modelos pueden extraer reglas y patrones de los datos y aplicarlos para nuevos problemas.

La recopilación de datos, la anotación y el procesamiento previo son requisitos previos esenciales para crear modelos de aprendizaje automático. Como regla general, cuanto mayor es la complejidad de un modelo, normalmente se necesitan más datos para el entrenamiento y la validación.

Por ejemplo, podría pensar en asistentes virtuales avanzados, como Alexa (Amazon) o Siri (Apple), que aplican el aprendizaje automático para interpretar el lenguaje natural y predecir la mejor respuesta o reacción.

5.3 Aprendizaje profundo (Deep Learning)

El **aprendizaje profundo** es un área del aprendizaje automático, que cubre únicamente las redes neuronales. El nombre de red neuronal se inspiró en el hecho de que la arquitectura modela libremente el cerebro humano. Esta técnica demostró ser especialmente adecuada para ciertas tareas como el reconocimiento de imágenes, que es la piedra angular de aplicaciones como los vehículos autónomos.

Las redes neuronales profundas son actualmente el área de investigación más candente en toda la comunidad de IA. Su popularidad se basa en muchos avances en los últimos años, incluida la competencia ImageNet que clasifica imágenes en color de alta resolución en 1000 categorías diferentes y 1,2 (1,4) millones de muestras de entrenamiento. Las redes neuronales profundas también llevaron a avances en otras áreas en las que se desempeñaron comparables o incluso mejores que sus contrapartes humanas, incluido el reconocimiento de voz, la transcripción de escritura a mano (OCR), la traducción automática, la conducción autónoma, Go playing y muchos más.

5.4 Casos de uso populares

A medida que avanzaba el campo del aprendizaje automático y las computadoras se volvían cada vez más poderosas, se habilitaron nuevas soluciones que afectaron a casi todos los campos (científicos). Hoy en día, el aprendizaje automático ayuda a los científicos a realizar diagnósticos médicos, descubrir nuevos medicamentos, monitorear la superficie de la tierra y detectar incendios forestales, automatizar y optimizar procesos en la industria financiera, etc.

Aparte de estas aplicaciones de vanguardia, a menudo tendemos a pasar por alto que el aprendizaje automático también forma parte de nuestra vida diaria. En las siguientes secciones abordaremos estas aplicaciones de aprendizaje automático y nos centraremos en sistemas como:

- Sistemas de recomendación
- Los motores de búsqueda
- Máquina traductora

- Identificación de música
- Autos autónomos

5.4.1 Caso de uso: motores de búsqueda y sistemas de recomendación

Los **motores de búsqueda**, como Google, aplican el aprendizaje automático de muchas formas para brindar mejores servicios. Al escribir una consulta, por ejemplo, el aprendizaje automático proporciona sugerencias de autocompletado. Estas sugerencias se personalizan en función de los temas de tendencia actual, pero también de nuestra ubicación y búsquedas anteriores. Posteriormente, las consultas se evalúan en muchos niveles para determinar nuestras intenciones exactas y clasificar los resultados en consecuencia.

De manera similar, el aprendizaje automático determina lo que se nos recomienda en YouTube, Netflix, Amazon, etc. (un campo del aprendizaje automático, a menudo denominado **sistemas de recomendación**). Estas aplicaciones predicen nuestros intereses al analizar nuestras actividades en línea. Los artículos que hemos buscado, las películas que hemos visto, los productos que hemos comprado son todos predictores sobre nuestro comportamiento e intereses futuros.

5.4.2 Caso de uso: traducción automática

Los servicios de **traducción** como DeepL también aplican cada vez más el aprendizaje automático. La idea principal detrás de estos servicios es pasar de diccionarios simples a traductores complejos que se enfocan en el contexto e interpretan el texto como un todo. El aprendizaje automático en esta configuración se puede utilizar para crear modelos que describan cómo se expresan ciertas ideas en otros lenguajes. Esto permite una comprensión más matizada del texto escrito y proporciona traducciones más naturales. Un aspecto importante de estas soluciones es que se pueden mejorar constantemente proporcionándoles nuevos ejemplos y comentarios de los que aprender.

5.4.3 Caso de uso: identificación de música y reconocimiento de voz

El análisis de audio es un campo propio en el aprendizaje automático. Una herramienta común en este campo es transformar señales de audio en componentes de frecuencia. En el caso de los discos de música, la composición de estos componentes de frecuencia es tan única, que podemos definir las llamadas *huellas digitales* para cada canción. Esto habilita aplicaciones de **identificación de música** como Shazam, que pueden identificar y unir canciones con precisión basándose en una muestra de solo unos segundos de duración. Del mismo modo, las aplicaciones de **reconocimiento de voz** pueden identificar fácilmente las palabras habladas y convertirlas en lenguaje escrito.

5.4.4 Caso de uso: coches autónomos

Los **coches autónomos** como Google Waymo y Tesla dependen en gran medida del aprendizaje automático. Al analizar los datos provenientes de varios sensores, el aprendizaje automático controla la aceleración, frenado y dirección del automóvil. Estas instrucciones se basan no solo en las normas de tráfico y las señales de tráfico, sino que incluyen un modelo predictivo continuo para evitar posibles accidentes. Aunque los modelos finales de aprendizaje automático son capaces de evaluar los datos de entrada y tomar decisiones en fracciones de segundo, el entrenamiento de los modelos requiere inmensas cantidades de datos, poder computacional y tiempo.

5.5 Técnicas de Machine Learning

En este capítulo

- Familiarízate con diferentes enfoques de machine learning
- Diferenciar entre técnicas de aprendizaje supervisado, no supervisado y por refuerzo.
- Vea cómo se pueden aplicar las técnicas de aprendizaje para diferentes casos de uso

La idea básica del aprendizaje automático es crear modelos que se puedan utilizar para hacer predicciones y tomar decisiones. Diferenciamos entre 3 tipos de aprendizaje automático, cada uno de ellos tiene como objetivo resolver diferentes tipos de tareas: **supervisado**, **no supervisado** y **aprendizaje reforzado**.

El **aprendizaje supervisado** tiene como objetivo aprender de un conjunto de datos de ejemplo. Nuestra suposición inicial es que el valor de un resultado conocido (por ejemplo, el cliente compra un producto) está influenciado por un conjunto de entradas medibles (edad, intereses, últimos clics). Utilizando algoritmos de aprendizaje automático, intentamos detectar y modelar estas relaciones.

El **aprendizaje no supervisado** también requiere datos de entrada, pero no existe una variable de resultado predefinida. En su lugar, tratamos de detectar patrones y establecer relaciones dentro de los datos (por ejemplo, a través de agrupaciones) o reducir dimensiones (por ejemplo, con análisis de componentes principales).

El **aprendizaje reforzado** no requiere observaciones de entrada, sino un objetivo y un entorno en el que operar. Al integrar la retroalimentación continua del entorno, esperamos que el sistema cree sus propias tácticas para lograr el objetivo.

5.5.1 Aprendizaje supervisado

En el *aprendizaje supervisado*, la variable de resultado debe conocerse para el entrenamiento de modelos. Como ejemplo, podría pensar en un modelo para la predicción de precios de apartamentos. Este modelo podría ayudar a los agentes inmobiliarios a fijar el precio de los nuevos apartamentos que ingresan al mercado. Además, los valores previstos se pueden comparar con los precios del mercado para determinar las oportunidades de compra de los apartamentos más infravalorados. Las variables de entrada requeridas para este problema probablemente incluirían aspectos como:

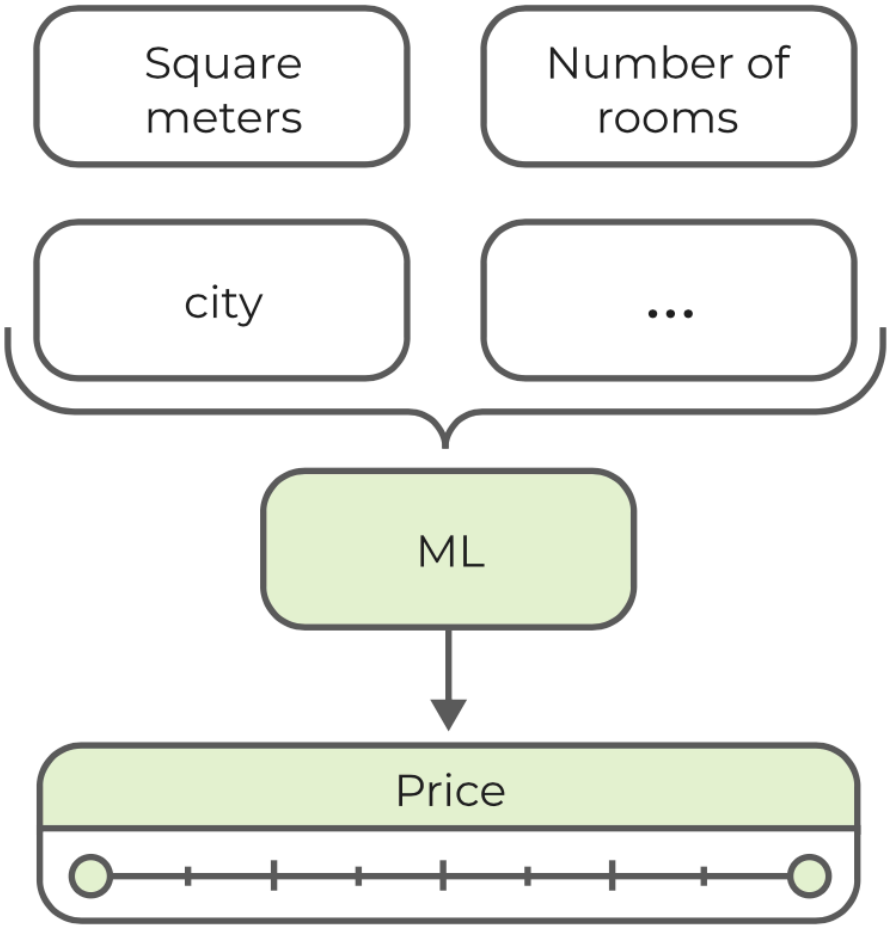
- **Ciudad**: Ubicación del apartamento por ciudad.
- **Habitaciones**: Número de habitaciones del apartamento.
- **Tamaño**: Tamaño del apartamento por metros cuadrados.

Como punto de partida, necesitamos un conjunto de datos con varios ejemplos que contengan esta información:

	City	Number of rooms	...	Square Meters	Price €
Apartment 1	Berlin	3	...	80	640.000
Apartment 2	Vienna	4	...	100	780.000
...	...	...	...	...	...
Apartment N	Zurich	1	...	40	565.000

5.5.2 Aprendizaje supervisado: objetivo

El objetivo del aprendizaje automático supervisado es tomar un conjunto de ejemplos y entrenar un modelo utilizando métodos estadísticos. Este modelo debe explicar la relación entre las variables de entrada (por ejemplo, ciudad, habitaciones, tamaño) y la variable de salida (por ejemplo, precio) con la mayor precisión posible. Una vez que se ha entrenado el modelo, se puede utilizar para predecir el resultado de nuevas combinaciones de valores de entrada invisibles.



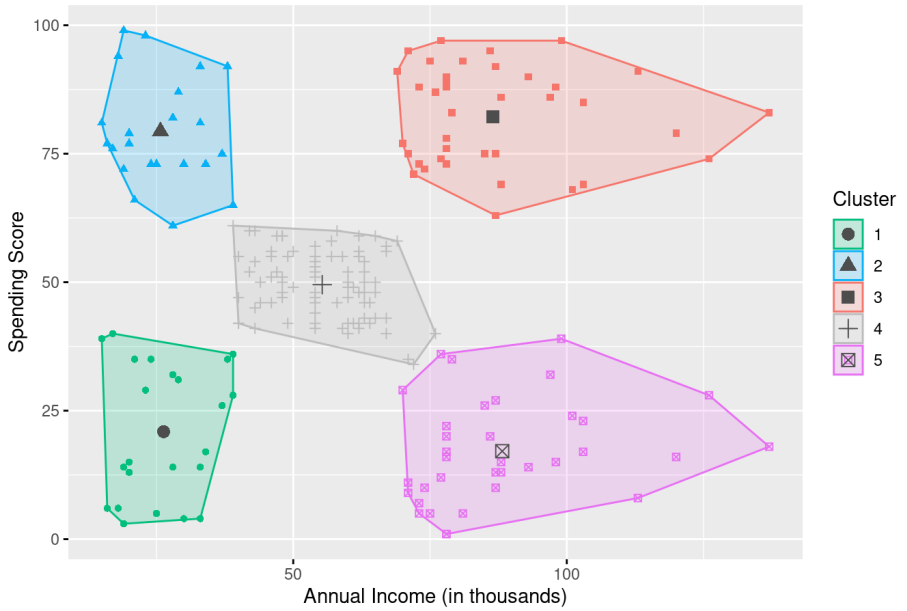
5.5.3 Aprendizaje no supervisado

Las técnicas de aprendizaje no supervisadas extraen estructuras y patrones estadísticos de conjuntos de datos. A diferencia del aprendizaje supervisado, estas técnicas no requieren una variable de resultado predefinida en la que recibir entrenamiento.

El método de aprendizaje no supervisado más importante es la **agrupación** o cluster. La agrupación intenta agrupar las observaciones en función de su similitud. Primero, la similitud entre las observaciones se calcula como una distancia entre sí. Basándonos en estos valores, podemos determinar conglomerados (grupos) en los que las observaciones están más próximas entre sí.

5.5.3.1 Aprendizaje no supervisado: ejemplo

Los métodos de agrupación en clústeres se aplican a menudo como un intento de definir grupos de clientes que son similares. La similitud puede significar intereses coincidentes, datos demográficos, ubicación geográfica, etc. Esta aplicación se denomina segmentación de mercado. El objetivo de este caso de uso es comprender mejor la base de clientes y sus necesidades. Los clústeres se pueden utilizar para proporcionar anuncios y ofertas dirigidos a los clientes de acuerdo con sus propiedades. Como ejemplo, echemos un vistazo al siguiente gráfico:



Esta gráfica compara el `Annual Income` y `Spending Score` de algunos clientes de centros comerciales. Al aplicar un algoritmo de agrupación en clústeres y definir cinco grupos, podemos separar muy bien 5 tipos diferentes de clientes. Los grupos clave en este ejemplo serían el grupo azul y rojo, que generan la mayor cantidad de ingresos. Éstos definen dos tipos de clientes. Mientras que el grupo azul contiene clientes con ingresos más bajos, el grupo rojo contiene clientes con ingresos relativamente altos. Con base en esta información, se podría adaptar la estrategia de marketing:

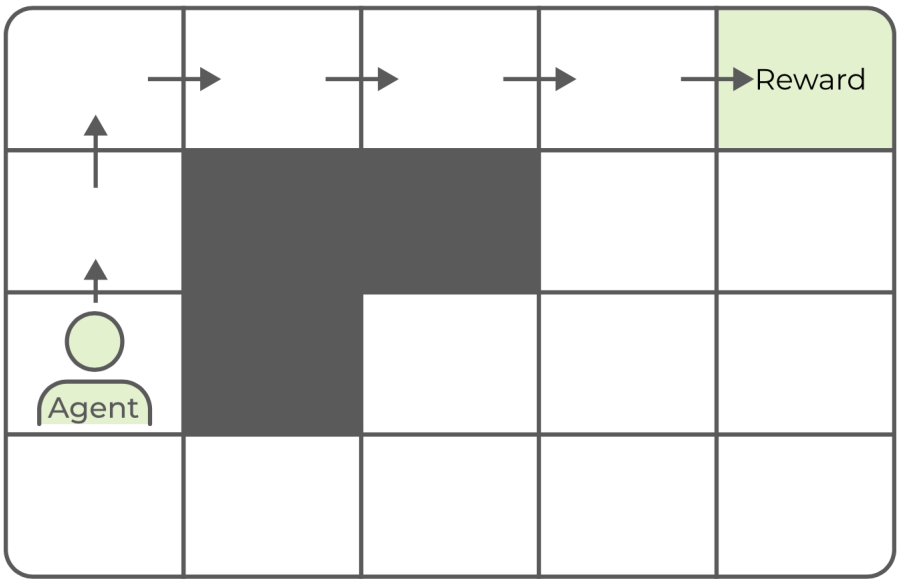
- Los clientes del clúster azul podrían ser un grupo objetivo óptimo para opciones atractivas de financiamiento y crédito.
- Los clientes del grupo rojo deben mantenerse como clientes el mayor tiempo posible. En particular, debe evitarse que cambien a competidores. Por lo tanto, podrían ser un grupo objetivo óptimo para recompensas y ofertas basadas en la lealtad.

5.5.4 Aprendizaje reforzado

En lugar de aprender de un conjunto de ejemplos, el aprendizaje por refuerzo se basa en las recompensas acumulativas que recibe una entidad virtual (a menudo denominada **agente**) al actuar en un entorno específico. El agente intenta maximizar las recompensas acumulativas tomando decisiones. Estas decisiones se basan inicialmente en prueba y error, sin embargo, el agente se ve recompensado al tomar buenas decisiones y aprende de ellas. Del mismo modo, también existen algunos costos asociados con las malas decisiones. Por lo tanto, al crear el entorno de aprendizaje, definimos las reglas y las recompensas (y los costos), pero dejamos que el agente descubra los mejores pasos y tácticas.

5.5.4.1 Aprendizaje por refuerzo: ejemplo

Un ejemplo común es un mundo de cuadrícula, en el que el agente puede moverse de un campo a otro, excepto por posibles obstáculos o paredes, y es recompensado por encontrar la meta. Inicialmente, cada paso se daría al azar, pero después de suficientes intentos, el agente siempre iría en la dirección que maximiza sus posibilidades de obtener una recompensa. Cada paso también está asociado con un costo que reduce la recompensa acumulada. Esto significa que, desde cualquier punto de partida o posición, el agente aprendería el camino más corto hacia la recompensa después de un tiempo.



5.6 Aprendizaje supervisado con regresión y clasificación

En el aprendizaje automático supervisado, tomamos un conjunto de observaciones con una variable de resultado conocida y entrenamos un modelo que describe con precisión la relación entre las variables de entrada y el resultado.

- Saber qué predictores y variables de resultado son
- Diferenciar entre regresión y clasificación

5.6.1 Aprendizaje supervisado: datos de entrada

En el aprendizaje supervisado, entrenamos modelos en un conjunto de datos para describir la relación entre un valor de interés (resultado) utilizando un conjunto de valores de entrada conocidos. Por lo tanto, nuestros datos de entrenamiento para construir el modelo deben incluir todas las entradas requeridas, así como el resultado previsto en forma tabular. La tabla consta de dos partes:

1. Los **predictores** o variables de entrada, que se utilizan para calcular la predicción (también conocida como matriz de modelo).
2. La variable de **resultado**.

P 1	P 2	P 3	P 4	Outcome
...	...	...	...	1
...	...	...	...	2
...	...	...	...	3
...	...	...	...	4

5.6.2 Predictores

Los **predictores** son un conjunto de variables de entrada (columnas) que se utilizan para explicar y predecir el resultado. A menudo se les llama variables de *entrada*, *independientes*, *explicativas* o simplemente *características*. En el caso de los precios de los apartamentos, podría pensar en la cantidad de habitaciones, metros cuadrados o el nombre de la ciudad.

	City	Number of rooms	...	Square Meters	Price €
Apartment 1	Berlin	3	...	80	640.000
Apartment 2	Vienna	4	...	100	780.000
...	...	...	...	...	...
Apartment N	Zurich	1	...	40	565.000

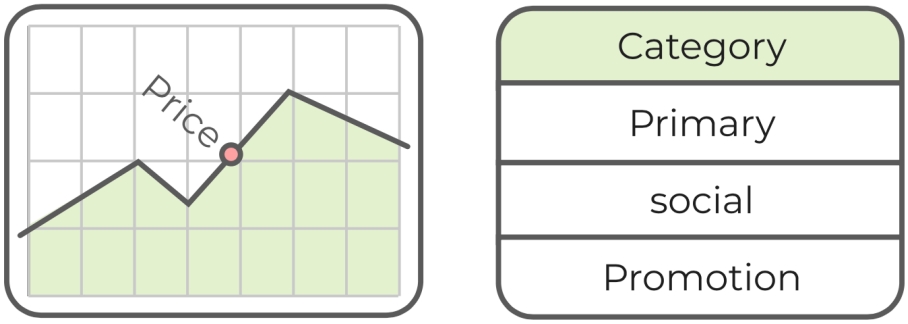
5.6.3 Variable de resultado

La variable de resultado es un solo valor/columna que queremos predecir. A menudo se la denomina *objetivo*, *respuesta*, *variable dependiente* o simplemente denominada *etiqueta*. Como ejemplo, podría pensar nuevamente en el precio de un apartamento:

	City	Number of rooms	...	Square Meters	Price €
Apartment 1	Berlin	3	...	80	640.000
Apartment 2	Vienna	4	...	100	780.000
...	...	...	...	...	...
Apartment N	Zurich	1	...	40	565.000

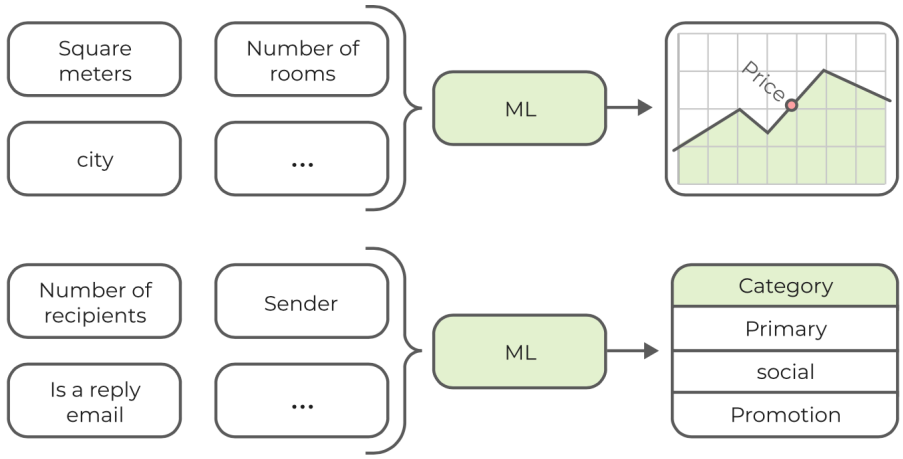
5.6.4 Regresión vs. clasificación

Dentro del dominio de aprendizaje supervisado, diferenciamos entre modelos de **regresión** y de **clasificación**. El modelo específico con el que estamos tratando depende del tipo de datos de la variable de resultado. Si la variable a predecir es continua (como `numérica`), hablamos de un modelo de regresión. Si la variable es un `factor` categórico tenemos un problema de clasificación.



5.6.4.1 Ejemplo de regresión vs. clasificación

Cuando queremos predecir el precio de un apartamento, la salida es un valor `numérico` (continuo), lo que significa que estamos tratando con un modelo de regresión. Por otro lado, cuando el cliente de correo electrónico clasifica un correo electrónico en *Primario*, *Social* o *Promociones*, el resultado es un `factor` (categórico) y necesitamos usar un modelo de clasificación.



Todo el material descrito se encuentra en idioma ingles en la página oficial de Quantargo (Quartargo, 2020a)

Bibliography

Quartargo (2020a). *Introduction to Machine Learning*.

Quartargo (2020b). *Introduction to R*.

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

Xie, Y. (2020). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.21.