

Introducción al análisis de datos con R, 3^a versión

Curso dirigido a personas de las Ciencias Sociales

Bastían Olea Herrera

2026-06-22

Inicio

Bastían Olea Herrera

bastianolea.rbind.io

Encuentra todo sobre el curso [en este post](#) de mi blog.



Curso



Código



Grabaciones



Diapositivas



Contacto

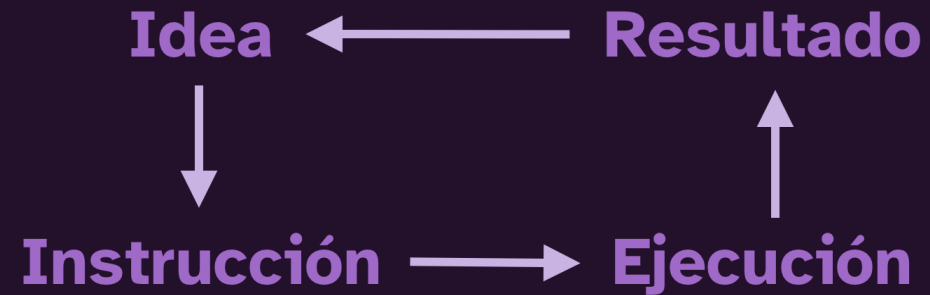
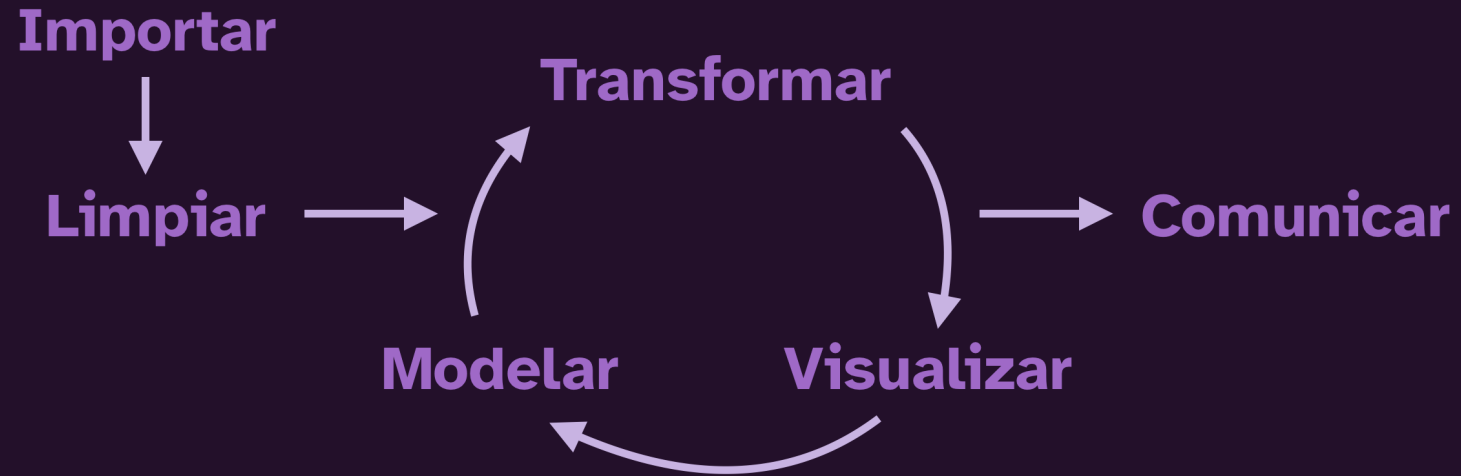
Puedes encontrar más contenidos para [aprender R en este sitio](#), y más cosas de R [en mi sitio web](#).

¿Por qué programar para analizar datos?

Existen muchas diferencias entre *hacer* algo y **programarlo**:

- ▶ Aprender a programar es un **cambio de perspectiva**: pasamos de *hacer* cosas a *planificar* cosas 🖐️
- ▶ Desde la mera **Acción** 🔧 a la **Instrucción** 🧠 y luego al **Proceso** ⚙️
- ▶ Trabajamos **desde arriba**, viendo todos sus pasos y componentes 🧑💻
- ▶ Pasos independientes, reutilizables, mejorables, intercambiables ⚙️
- ▶ El trabajo que hagas puede **servirte** a futuro, a ti y a otrxs ♻️
- ▶ Resultados **replicables** 🔄 que dan paso a la **automatización** 🏭

Flujo de trabajo



¿Qué es R?

R es un lenguaje de programación enfocado al **análisis de datos**, las **estadísticas** y la **visualización de datos**.



↓ Descargar R

¿Para qué sirve?

- ▶ Para procesar, limpiar, y analizar datos
- ▶ Crear gráficos, tablas, mapas, y otros resultados visuales
- ▶ Repetir, mejorar, adaptar y reutilizar lo que ya has hecho ♻️
- ▶ Automatizar tareas repetitivas 🤖
- ▶ Expandir tus resultados con aplicaciones, sitios web, APIs, y más 🚀

¿Por qué usar R?

Gratuito

R es un lenguaje de programación abierto y gratuito, mantenido por su propia comunidad de usuari@s. Al ser *software libre*, nunca tendrás que pagar nada!

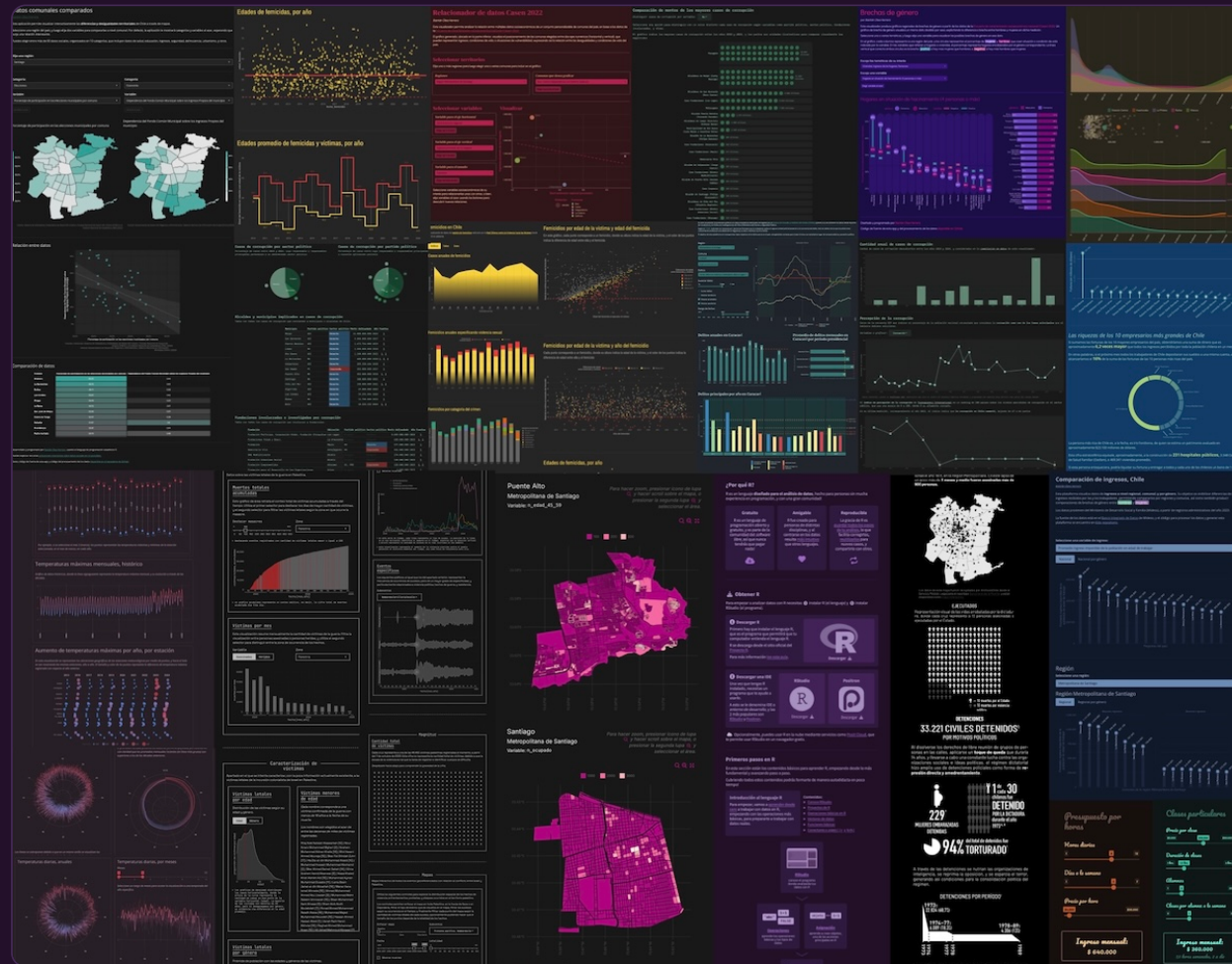
Amigable

R fue diseñado para el análisis de datos, y fue creado por y para personas de diversas disciplinas. Esto lo vuelve en la herramienta apropiada para trabajar con datos, más amigable e intuitivo que otros lenguajes.

Reproducible

La gracia de R es registrar todos los pasos de tu análisis, lo que facilita corregirlos, mejorarlos, reutilizarlos para nuevos casos, y compartirlos con otros.

¿Qué se puede hacer con R?



Muestra de algunas aplicaciones, gráficos, tablas, y mapas hechos con R. También algunos trabajos que he hecho.

RStudio

RStudio es una IDE (entorno de desarrollo integrado) que nos permite escribir código en R de forma más cómoda, organizada, y visual.

- ▶ Es la forma más popular de usar R
- ▶ Lanzado en 2011
- ▶ Software libre (licencia de código abierto AGPL)
- ▶ También existen alternativas, como [Positron](#)



 Descargar RStudio

Conceptos clave



Script

Archivo de texto .R en el que escribimos nuestro código, en pasos, y siguiendo un orden lógico.



Consola

Es la forma directa de interactuar con R, un comando a la vez, con resultados efímeros.



Proyecto

Archivo .Rproj que marca nuestro espacio de trabajo: una carpeta específica que reúne todas las piezas de nuestro análisis.

Paneles de RStudio

Descargar RStudio

1

Scripts

2

Consola

3

Entorno

4

Archivos

The screenshot displays the RStudio interface with the following panels:

- Source Panel:** Shows the script `preprocesar_app.R` with the following code:

```
1 library(dplyr)
2 library(arrow)
3 library(sf)
4 library(ggplot2)
5
6 # cargar datos
7 manzanas <- read_parquet("datos/Cartografia_censo2024_Pais
8
9 manzanas |> glimpse()
10
11 # filtrar
12 manzanas_comuna <- manzanas |>
13   filter(COMUNA == "LA FLORIDA")
14
15 # convertir a sf
16 manzanas_comuna_sf <- manzanas_comuna |>
17   st_as_sf(crs = 4326)
18
19 # definicion
```

The console shows the output of the `glimpse()` function:

```
> manzanas
# A tibble: 216,341 x 218
#   OBJECTID CUT COD_REGION REGION COD_PROVINCIA PROVINCIA
#   <int> <int> <int> <chr> <int> <chr>
1     1  9201     9 LA AR...     92 MALLECO
2     2  9201     9 LA AR...     92 MALLECO
3     3  9201     9 LA AR...     92 MALLECO
4     4  9201     9 LA AR...     92 MALLECO
5     5  9201     9 LA AR...     92 MALLECO
6     6  9201     9 LA AR...     92 MALLECO
7     7  9201     9 LA AR...     92 MALLECO
8     8  9201     9 LA AR...     92 MALLECO
```

- Environment Panel:** Shows the Global Environment with two objects:

Name	Type	Len...	Size	Value
manzanas	tbl_df	218	296....	216341 obs. of...
manzanas_...	tbl_df	218	7.9 ...	3584 obs. of 2...
- Files Panel:** Shows the file explorer for the project `censo_2024_mapas`. The files and folders are:

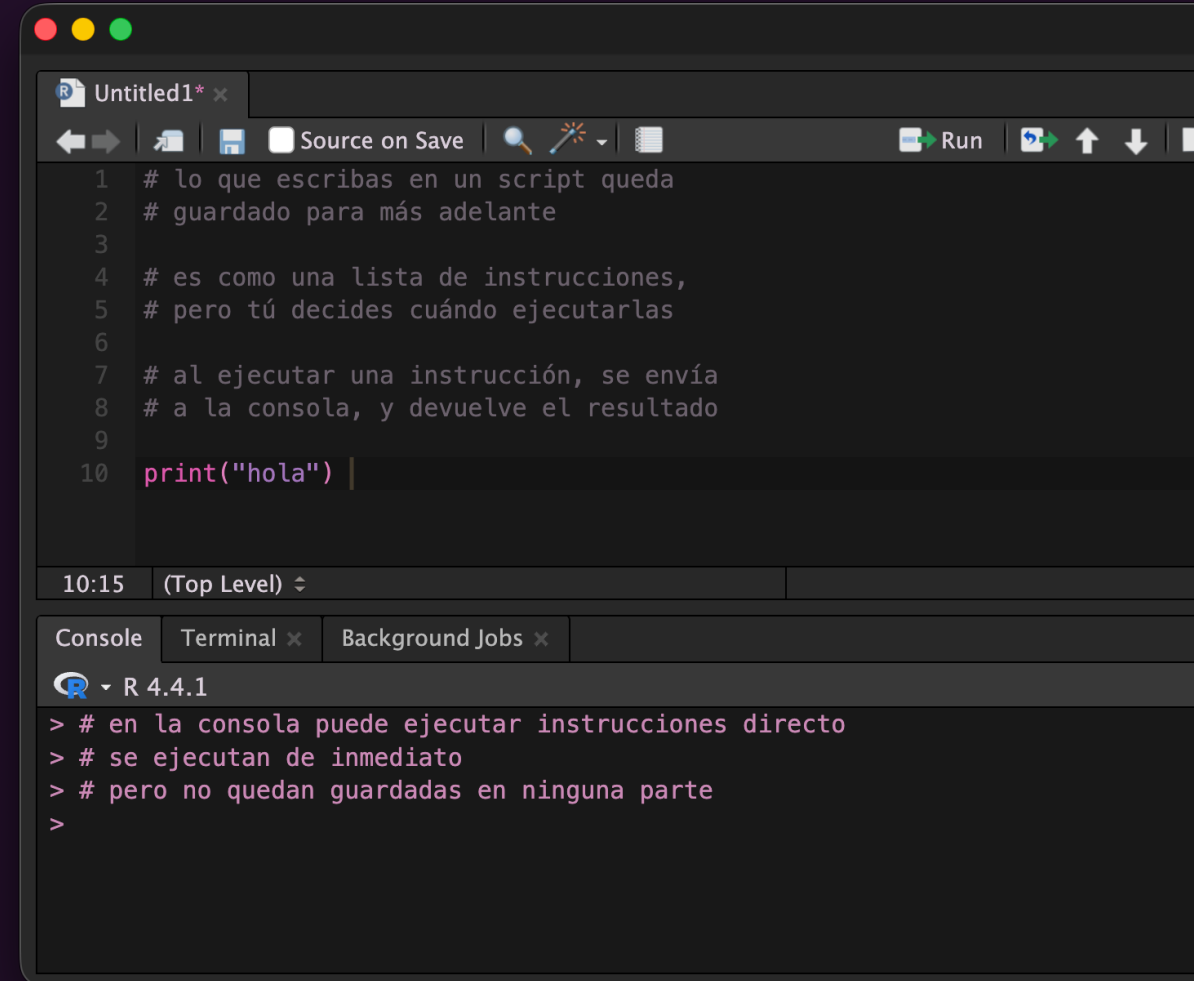
Name	Size	Modified
columnas.rds	4.7 KB	Dec 19, 2024
cut_comunas.rds	16 KB	Dec 19, 2024
datos		
manifest.json	266.7 KB	Jan 11, 2025
mapa_censo_ggplot.R	916 B	Dec 18, 2024
mapa_censo_mapgl.R	1.2 KB	Dec 18, 2024
preprocesar_app.R	1.2 KB	Dec 19, 2024
prueba_duckdb.R	1.2 KB	Dec 18, 2024
readme.md	1.4 KB	Jan 8, 2025
rsconnect		
styles.css	678 B	Dec 19, 2024
www		
- Console Panel:** Shows the R version `R 4.4.1` and the current working directory `~/R/censo_2024_mapas/`.

Conceptos de RStudio

- ❶ **Panel de scripts:** los archivos de texto con nuestro código. Podemos tener varias pestañas. Ejecutamos el código poniendo el cursor en la línea y presionando `control + enter`, o el botón *Run*.
- ❷ **Panel de consola:** en la consola se imprimen los **resultados** del código que ejecutamos. También podemos ejecutar código directamente en ella escribiendo y presionando `enter`.
- ❸ **Panel de entorno:** acá veremos los **objetos** que vayamos creando o cargando, que pueden ser números, texto, tablas de datos, funciones, gráficos y otros.
- ❹ **Panel de archivos:** en este panel podemos navegar los archivos y carpetas de nuestro proyecto y/o computador. La idea es que *todo* esté dentro del proyecto!

Consola y scripts

- ▶ Todos los **comandos** se ejecutan por medio de la **consola** (abajo), que es donde vive R
- ▶ Al ejecutar código en la consola, el código no se guarda, sólo se ejecuta!
- ▶ Para **guardar** el código, escribimos en **scripts** (arriba)
- ▶ Escribimos el código en un script (arriba), y cuando se lo pidamos (botón *Run* o **control** + **enter**), RStudio se encarga de **ejecutar** el código en la consola (abajo)



The screenshot shows the RStudio interface. The top pane is a script editor titled 'Untitled1*' containing R code comments and a print statement. The bottom pane is the console, showing the execution of the code from the script above.

```
1 # lo que escribas en un script queda
2 # guardado para más adelante
3
4 # es como una lista de instrucciones,
5 # pero tú decides cuándo ejecutarlas
6
7 # al ejecutar una instrucción, se envía
8 # a la consola, y devuelve el resultado
9
10 print("hola")
```

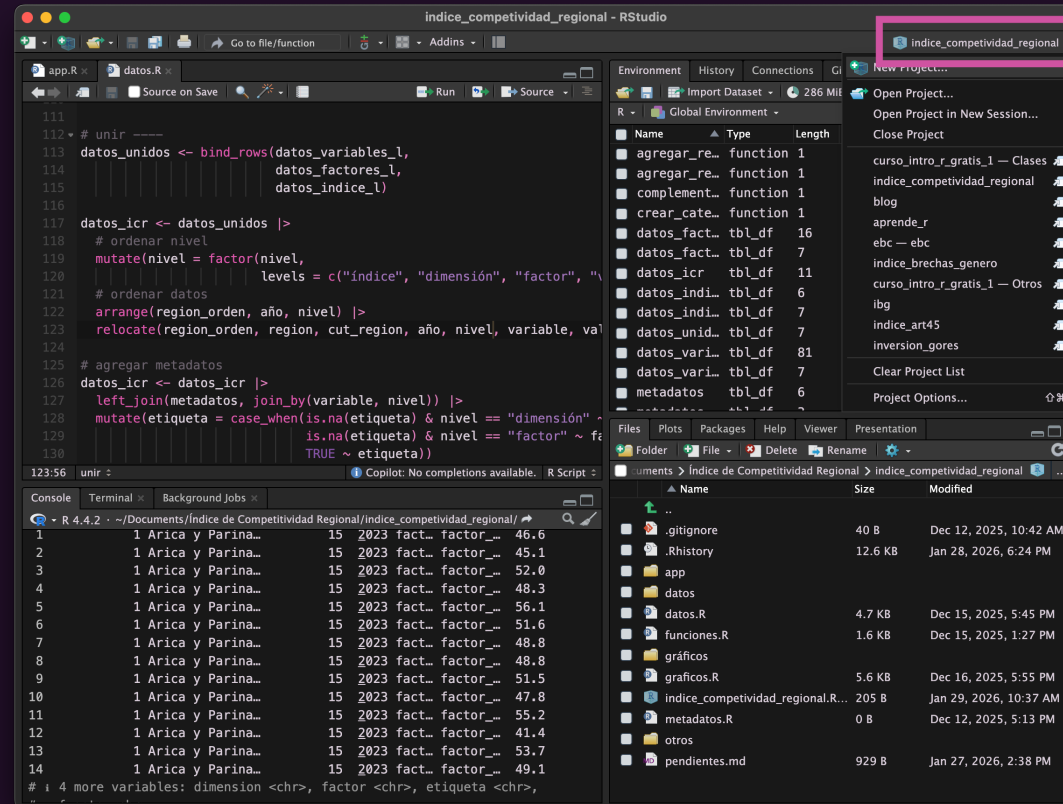
10:15 (Top Level) ↕

Console Terminal × Background Jobs ×

R 4.4.1

```
> # en la consola puede ejecutar instrucciones directo
> # se ejecutan de inmediato
> # pero no quedan guardadas en ninguna parte
>
```

Botones



Cambiar de **proyecto** o crear uno nuevo

Inteligencia Artificial

Nuevo en RStudio: *Posit Assistant*, asistente de inteligencia artificial integrado con tu código y datos.

The screenshot shows the RStudio interface with the **Posit Assistant** pane on the right. The assistant is analyzing R code and data to provide insights. The code in the editor includes data manipulation steps like `mutate`, `distinct`, and `tidyr::pivot_wider`. The assistant's analysis is displayed in the right pane, highlighting data completeness issues and categorical patterns.

Compleitud de datos

Existe una gran disparidad en la completitud entre variables:

- Variables con 100% de cobertura: Identificadores geográficos (región, comuna), año, proveedor internet
- Variables con ~64% de cobertura: firma electrónica, sistemas documentales, transformación digital implementada, encargado TI
- Variables críticas con baja cobertura: firma electrónica tipo (28.7%), sistema facturación electrónico (48.1%), programa doc digital (48.2%)

Patrones en variables categóricas

Velocidad de internet (96.3% válida):

- Mayoría con "Más de 100 Mb" (57.4%)
- Distribución decreciente en velocidades menores

Sistemas implementados (96.4-96.5% válida):

- Altísima adopción de sistemas contables (96.5% implementados)
- Pero solo 44.7% tiene transformación digital implementada

Recursos TI (64.2% válida):

- 70.3% tiene encargado TI a jornada completa
- Solo 36.7% usa servidores cloud

Observación crítica

Los datos de 2020-2022 tienen mayor completitud que años posteriores. Hay un patrón claro donde variables relacionadas con transformación digital y firma electrónica tienen datos faltantes en los últimos años, sugiriendo posibles cambios en la encuesta o recolección de datos.

Más información [en esta publicación.](#)

Introducción a R

Veamos ahora los **elementos más básicos** del lenguaje R, para luego avanzar a su aplicación al **análisis de datos**.



Guía de aprendizaje



Operaciones básicas

 [Ver tutorial](#)

- ▶ Podemos realizar cualquier operación matemática en R.
- ▶ Para **ejecutar** un comando, pon el cursor de texto en la línea que desees ejecutar, o selecciónala, y presiona el botón *Run*, o las teclas **control + enter**.
- ▶ Lo importante es que el cursor de texto **|** esté en cualquier lugar de la **línea**.
- ▶ El **resultado** siempre aparecerá en la **consola**.

```
2 + 2 #suma
```

```
[1] 4
```

```
50 * 100 #multiplicación
```

```
[1] 5000
```

```
4556 - 1000 #resta
```

```
[1] 3556
```

```
6565 / 89 #división
```

```
[1] 73.76404
```

```
10^4 #potencias
```

```
[1] 10000
```

Comentarios

- ▶ Si ejecutamos operaciones que son válidas, obtenemos un **error**!
- ▶ Los comentarios nos permiten poner texto en cualquier parte del script sin que afecte los cálculos.
- ▶ También podemos poner un comentario al final de una línea sin que afecte el código
- ▶ Usa los comentarios para describir lo que buscas hacer con el código

```
1 # sumar 4 números
2 1 + 1 + 1 + 1
3 # quizás esto debería
4 # ser de otra forma, porque
5 # la verdad quedó bien mal...
6
7 1 * 4 # así queda mucho mejor
```

Tipos de datos

Lo que podemos hacer siempre va a depender del **tipo** de cada objeto.

En R existen varios tipos:

Numéricos

```
1 2 3 4 5.1 5.2 5.333
```

Pueden ser decimales (*doubles*) o enteros (*integers*)

Caracter (texto)

```
"ésta es una cadena de texto"
```

Lógicos (verdadero o falso)

```
TRUE FALSE TRUE
```

Objetos

Hasta ahora hemos trabajado con datos sueltos. Si queremos volver a usar un dato o resultado, debemos guardarlo en un **objeto**.

Se le llama **objeto** a cualquier dato, variable, o elemento que **guardemos** en R.

Para **crear** un objeto, simplemente le damos un nombre y le *asignamos* un valor o contenido.

nombre $\leftarrow$ *contenido*

Para asignar algo a un objeto, usamos el operador **$\leftarrow$**

```
numerito <- 4
```

El objeto **numerito** se **crea** cuando declaramos que va a contener el **valor** **4**, y luego podemos ejecutar **numerito** para obtener su contenido (**4**)

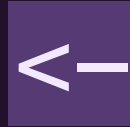
```
numerito
```

```
[1] 4
```

Asignación

[Ver tutorial](#)

Con el *operador de **asignación*** creamos objetos nuevos.

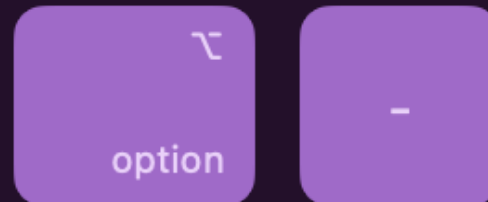


Se escribe con:

Windows:



Mac:



Al asignar algo, **creamos** o **modificamos** un objeto con el valor que le estamos asignando.

Al ejecutar un objeto, obtenemos su valor.

```
1 edad <- 32
2 edad
```

```
[1] 32
```

```
1 animal = "gato"
2 animal
```

```
[1] "gato"
```

Podemos usar el objeto creado para lo que queramos.

Crear objetos es como **asignar variables**, y podemos usar estas variables para llevar a cabo operaciones!

```
1 edad <- 32
2 año <- 2026
3 año - edad
```

```
[1] 1994
```

```
1 perros = 2
2 gatos = 3
3 perros + gatos
```

```
[1] 5
```

```
1 presupuesto = 100000
2 pizza = 15000
3 presupuesto - pizza * 10
```

```
[1] -50000
```

Vectores

Los vectores son **secuencias** de elementos. Un objeto que contiene cero o más datos.

- ▶ Estos elementos son de un **mismo tipo**: numérico, carácter o lógico.
- ▶ Son la forma más básica de registrar e interactuar con varios datos a la vez.

```
1 c(1, 2, 3, 4, 5, 6)
```

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

```
1 c("a", "b", "c", "d")
```

```
[1] "a" "b" "c" "d"
```

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

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

```
1 c(0.2, 0.1, -0.0, -0.1)
```

```
[1] 0.2 0.1 0.0 -0.1
```

Operaciones sobre vectores

Podemos realizar cualquier **operación** sobre los elementos de un vector:

```
1 año <- 2026  
2 edades <- c(54, 34, 65, 21, 32)  
3 año - edades
```

```
[1] 1972 1992 1961 2005 1994
```

Funciones

- Las **funciones** son pequeños programas que permiten realizar distintas operaciones.

```
función(argumento = 123)
```

Algunas funciones comunes:

```
mean() # calcular promedio  
median() # calcular mediana  
sum() # sumar elementos  
min() # valor mínimo  
max() # valor máximo
```

Por ejemplo:

```
1 mean(c(1, 2, 3, 4, 5))
```

```
[1] 3
```

Si no sabemos cómo usar una función, podemos buscar su nombre en el panel **Ayuda** de RStudio, o escribir el nombre de la función precedido de un signo de interrogación: `?funcion`

Paquetes



- **Extensiones** de R que te permiten agregar **nuevas funciones** al lenguaje, **datos**, y más.
- Se instalan desde internet
- Son creados y mantenidos **por la comunidad**
- Se revisan para garantizar su seguridad y estabilidad

```
install.packages()
```

Datos

Para cargar datos, tenemos que:

- ▶ Tener un **archivo** y conocer su **formato** (Excel, Stata, CSV, etc.)
- ▶ ~~Saber **dónde está** el archivo en nuestro computador~~
- ▶ **Guardar** el archivo dentro del proyecto de R 
- ▶ Conocer una **función** y posiblemente un **paquete** que lea ese formato de archivos

Cargar datos

- Las funciones de carga de datos usan como argumento la **ruta** del archivo.

Cargar un archivo en la misma carpeta del proyecto:

```
datos <- read.csv("datos.csv")
```

Cargar un archivo en una subcarpeta del proyecto:

```
datos <- read.csv("carpeta/datos.csv")
```

Paquetes para carga de datos



- ▶ `{readxl}` es uno de los paquetes de lectura de planillas Excel. Para escribir un archivo Excel está `{writexl}`
- ▶ `{readr}` lee y escribe datos de múltiples formatos (csv, rds, rdata), usualmente de forma más veloz y cómoda
- ▶ `{haven}` permite cargar archivos SPSS, Stata, y SAS
- ▶ `{arrow}` es un formato moderno de datos columnares, optimizado para grandes volúmenes y velocidad de carga, y pensado para usarse en distintos softwares

Carga de datos desde Excel

Probemos cargando unos datos!



Descargar datos de pobreza en Excel

Cargamos los datos usando una función del paquete `{readxl}`:

```
1 # cargar paquete
2 library(readxl)
3
4 # cargar datos
5 datos <- read_excel("datos/pobreza/estimaciones_pobreza.xlsx")
6
7 # mirar datos
8 head(datos)
```

```
# A tibble: 6 × 10
  codigo region  comuna      personas_proy personas porcentaje limite_inferior
  <chr>   <chr>   <chr>         <dbl>      <dbl>      <dbl>         <dbl>
1 1101   Tarapacá Iquique      229674    41967.      0.183         0.162
2 1107   Tarapacá Alto Hospic... 138527    45162.      0.326         0.295
3 1401   Tarapacá Pozo Almonte   18290     4563.      0.250         0.199
4 1402   Tarapacá Camiña        1380       308.      0.223         0.138
5 1403   Tarapacá Colchane       1575       473.      0.300         0.187
6 1404   Tarapacá Huara         3072     1185.      0.386         0.319
# i 3 more variables: limite_superior <dbl>, casen <chr>, tipo_estimacion <chr>
```

{dplyr}

- ▶ Paquete de exploración, manipulación y transformación de datos
- ▶ Sus funciones se escriben como verbos
- ▶ Las acciones se encadenan con el operador de conexión o *pipe*: `|>`

```
1 # install.packages("dplyr")  
2 library(dplyr)
```



Ver tutorial



¿Para qué sirve `{dplyr}`?

Para **explorar** y **modificar** los datos.

Seleccionar columnas:

```
1 datos |> select(region, comuna, porcentaje)
```

```
# A tibble: 4 × 3
  region  comuna      porcentaje
  <chr>   <chr>         <dbl>
1 Tarapacá Iquique      0.183
2 Tarapacá Alto Hospicio 0.326
3 Tarapacá Pozo Almonte  0.250
4 Tarapacá Camiña       0.223
```

Filtrar datos:

```
1 datos |> filter(porcentaje > 0.3)
```

```
# A tibble: 4 × 3
  region  comuna      porcentaje
  <chr>   <chr>         <dbl>
1 Tarapacá Alto Hospicio 0.326
2 Tarapacá Colchane      0.300
3 Tarapacá Huara         0.386
4 Atacama  Tierra Amarilla  0.345
```

Ordenar observaciones:

```
1 datos |> arrange(porcentaje)
```

```
# A tibble: 4 × 3
  region  comuna      porcentaje
  <chr>   <chr>         <dbl>
1 Magallanes Río Verde    0.00113
2 Magallanes Laguna Blanca 0.00353
3 Magallanes Cabo De Hornos 0.0158
4 Metropolitana Vitacura    0.0244
```

Crear variables:

```
1 datos |> mutate(ranking = row_number())
```

```
# A tibble: 4 × 3
  comuna      porcentaje ranking
  <chr>         <dbl>   <int>
1 General Lagos 0.586     1
2 Saavedra      0.445     2
3 Ercilla       0.416     3
4 Galvarino     0.399     4
```

Y más!

Conector

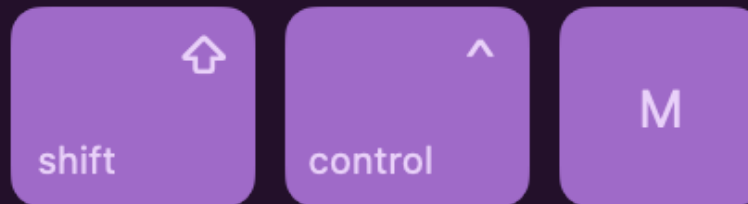
[Ver tutorial](#)

El conector o *pipe* `|>` (o también `%>%`) nos permite encadenar varias funciones de forma más legible.

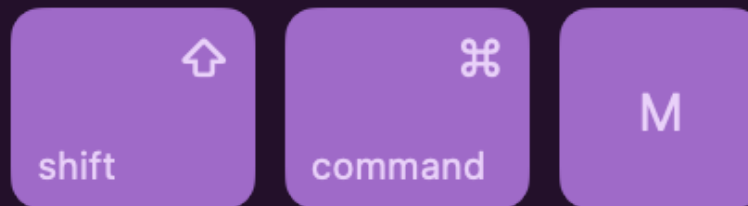
`|>` o `%>%`

Se escribe con:

Windows:



Mac:



Usando el conector

[!\[\]\(1f5fb44c55bdfab3345c8646adddb542_img.jpg\) Ver tutorial](#)

Aplicar una función normalmente:

```
funcion(objeto) # a la función le pasamos un objeto
```

Otra forma de hacerlo con `|>` es:

```
objeto |> funcion() # al objeto luego le aplicamos la función
```

Se lee: *al `objeto` le aplicamos la `funcion()`*

Luego podemos encadenar varias funciones:

```
objeto |> funcion() |> funcion() # al objeto le aplicamos dos funciones
```

También podemos ordenar las instrucciones hacia abajo:

```
1 resultado <- objeto |>
2   funcion() |>
3   funcion()
```

Seleccionar y ordenar

[Ver tutorial](#)

Usaremos los datos de **pobreza** del Ministerio Desarrollo Social y Familia que vimos antes. Si no los cargaste, descárgalos y los cargas así:

```
1 library(dplyr)
2 library(readxl)
3
4 datos <- read_excel("datos/pobreza/estimaciones_pobreza.xlsx")
```

Seleccionar

```
1 datos |>
2   select(region, comuna, personas) |>
3   head()
```

```
# A tibble: 6 × 3
  region    comuna    personas
  <chr>    <chr>      <dbl>
1 Tarapacá Iquique    41967.
2 Tarapacá Alto Hospicio 45162.
3 Tarapacá Pozo Almonte  4563.
4 Tarapacá Camiña       308.
5 Tarapacá Colchane      473.
6 Tarapacá Huara        1185.
```

Ordenar

```
1 datos |>
2   select(region, comuna, personas) |>
3   arrange(desc(personas)) |>
4   head()
```

```
# A tibble: 6 × 3
  region    comuna    personas
  <chr>    <chr>      <dbl>
1 Metropolitana Puente Alto 125235.
2 Metropolitana Santiago  87355.
3 Metropolitana Maipú    86976.
4 Antofagasta  Antofagasta  73103.
5 Metropolitana San Bernardo 64276.
6 Valparaíso   Valparaíso   60901.
```

¿Qué estamos haciendo?

Al aplicar una función sobre un *dataframe*, por ejemplo, `select()` para seleccionar columnas, **no estamos modificando los datos**, sino que estamos **trabajando con una copia** de los datos para no afectar los datos originales.

Todo lo que hagamos con `{dplyr}` serán **pruebas** y **exploración** de los datos, hasta que **decidamos** que los cambios que hagamos deben guardarse, así que entonces los **asignamos** a un **objeto nuevo**.

```
1 # prueba sin consecuencias
2 datos |>
3   select(comuna, personas)
4
5 # creamos un nuevo objeto a partir de los datos originales
6 datos_filtrados <- datos |>
7   select(comuna, personas)
```

Como asignamos los resultados a un **objeto nuevo**, tampoco estamos modificando ni afectando los datos originales. Así mantenemos un proceso **reproducible!**

Comparaciones

Podemos usar cifras u objetos para compararlas entre sí: la respuesta será **TRUE** (verdadero) o **FALSE** (falso).

```
1 edad >= 18
```

```
[1] TRUE
```

```
1 minimo <- 35  
2 edad > minimo
```

```
[1] FALSE
```

```
1 año == 2024
```

```
[1] FALSE
```

```
1 año != 2024
```

```
[1] TRUE
```

Las comparaciones son el principio que luego nos permitirá filtrar datos, crear variables y más!

Comparaciones sobre vectores

También podemos realizar **comparaciones** sobre los valores de un vector:

```
1 edades <- c(54, 34, 65, 21, 32)
2 edades > 40
```

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

En el fondo, cuando filtremos datos, las **columnas** de un *dataframe* serán **vectores**, y las comparaciones se harán sobre cada uno de los valores de la columna, entregando un resultado **TRUE** o **FALSE** para cada fila, que luego usaremos para quedarnos sólo con las filas que cumplen la condición que queremos.

Filtrar datos

[Ver tutorial](#)

- Para filtrar, realizamos una **comparación** entre los valores de una columna y un valor específico.

```
1 library(dplyr)
2
3 datos |>
4   filter(porcentaje > 0.35) |>
5   arrange(desc(porcentaje)) |>
6   head()
```

```
# A tibble: 6 × 10
  codigo region      comuna personas_proy personas porcentaje limite_inferior
  <chr>  <chr>      <chr>      <dbl>    <dbl>      <dbl>      <dbl>
1 15202 Arica y Parin... Gener...      668      392.      0.586      0.435
2  9116 La Araucanía Saave...    12717    5654.      0.445      0.361
3  9204 La Araucanía Ercil...     8422    3503.      0.416      0.361
4  9106 La Araucanía Galva...    12594    5029.      0.399      0.313
5  8314 Biobío Alto ...     6784    2640.      0.389      0.319
6 10306 Los Lagos San J...     7521    2923.      0.389      0.314
# i 3 more variables: limite_superior <dbl>, casen <chr>, tipo_estimacion <chr>
```

Funciones comunes

Algunas de las funciones más comunes de usar en `{dplyr}`, que son también las que usaremos para interactuar con los datos.

- ▶ `head()`, `tail()`, `glimpse()`: vistazos a los datos
- ▶ `select()`: seleccionar columnas o variables de una tabla
- ▶ `slice()`: extraer filas de una tabla por su posición o distintos criterios
- ▶ `pull()`: extraer una columna como vector
- ▶ `filter()`: filtrar observaciones en base a condiciones personalizadas
- ▶ `distinct()`: filtrar observaciones repetidas en una o más columnas
- ▶ `count()`: conteo de casos únicos en una o más columnas
- ▶ `mutate()`: crear columnas entregando valores, funciones sobre columnas, o calculando a partir de otras columnas

Datos de texto



El paquete `{stringr}` se especializa en texto.

```
1 library(stringr)
```

```
1 texto <- "todxs pueden aprender a programar 💕💕"  
2  
3 str_detect(texto, "todos")
```

```
[1] FALSE
```

```
1 str_detect(texto, "aprender")
```

```
[1] TRUE
```

```
1 datos |>  
2   filter(str_detect(comuna, "Alto")) |>  
3   select(region, comuna)
```

```
# A tibble: 4 × 2  
  region      comuna  
  <chr>      <chr>  
1 Tarapacá    Alto Hospicio  
2 Atacama     Alto Del Carmen  
3 Biobío      Alto Biobío  
4 Metropolitana Puente Alto
```

Crear/modificar variables

[Ver tutorial](#)

Con `mutate()` para crear una nueva variable o columna

Le aplicamos `mutate()` a un *data frame* conectando los datos a la función con un conector (`|>` o `%>%`)

```
1 datos |>
2   select(codigo, comuna) |>
3   mutate(saludo = "hola") |>
4   head()
```

```
# A tibble: 6 × 3
  codigo comuna      saludo
  <chr>  <chr>      <chr>
1 1101   Iquique    hola
2 1107   Alto Hospicio hola
3 1401   Pozo Almonte hola
4 1402   Camiña     hola
5 1403   Colchane   hola
6 1404   Huara      hola
```

```
1 datos |>
2   select(comuna, personas) |>
3   mutate(miles = personas/1000) |>
4   head()
```

```
# A tibble: 6 × 3
  comuna      personas  miles
  <chr>      <dbl>  <dbl>
1 Iquique    41967.  42.0
2 Alto Hospicio 45162.  45.2
3 Pozo Almonte  4563.   4.56
4 Camiña       308.   0.308
5 Colchane     473.   0.473
6 Huara       1185.   1.18
```

Aplicar funciones a columnas

[Ver tutorial](#)

Las funciones realizan operaciones sobre todos los datos de una columna, **reemplazando** los datos existentes, o **rellenando** datos de una columna nueva.

Dependiendo de lo que queramos hacer, las funciones que apliquemos con `mutate()` pueden hacer referencia a **una o a varias** columnas:

```
1 datos |>
2   select(comuna, personas) |>
3   arrange(desc(personas)) |>
4   head() |>
5   mutate(total_personas = sum(personas)) |> # aplicada a una columna
6   mutate(letras = nchar(comuna)) |> # aplicada a otra columna
7   mutate(orden = row_number()) |> # hace algo por sí misma
8   mutate(porcentaje = round(personas/total_personas, 2)) # función con argumento
```

```
# A tibble: 6 × 6
  comuna      personas total_personas letras orden porcentaje
  <chr>      <dbl>         <dbl>   <int> <int>         <dbl>
1 Puente Alto 125235.         497846.     11     1          0.25
2 Santiago   87355.         497846.      8     2          0.18
3 Maipú      86976.         497846.      5     3          0.17
4 Antofagasta 73103.         497846.     11     4          0.15
5 San Bernardo 64276.         497846.     12     5          0.13
6 Valparaíso 60901.         497846.     10     6          0.12
```

Variables condicionales

[Ver tutorial](#)

ifelse()

Si ocurre esto, **entonces** retorno lo primero, y **si no**, lo segundo.

```
1 datos |>
2   select(comuna, porcentaje) |>
3   mutate(nivel = ifelse(
4     porcentaje >= 0.3,
5     yes = "alta",
6     no = "baja")) |>
7   head()
```

A tibble: 6 × 3

	comuna <chr>	porcentaje <dbl>	nivel <chr>
1	Iquique	0.183	baja
2	Alto Hospicio	0.326	alta
3	Pozo Almonte	0.250	baja
4	Camiña	0.223	baja
5	Colchane	0.300	alta
6	Huara	0.386	alta

case_when()

Recodificación avanzada con **múltiples condicionales**.

```
1 datos |>
2   select(comuna, porcentaje) |>
3   mutate(miles = case_when(
4     porcentaje >= 0.3 ~ "alta",
5     porcentaje >= 0.2 ~ "media",
6     porcentaje < 0.2 ~ "baja")) |>
7   head()
```

A tibble: 6 × 3

	comuna <chr>	porcentaje <dbl>	miles <chr>
1	Iquique	0.183	baja
2	Alto Hospicio	0.326	alta
3	Pozo Almonte	0.250	media
4	Camiña	0.223	media
5	Colchane	0.300	alta
6	Huara	0.386	alta

Resúmenes de datos

[Ver tutorial](#)

Aplicar una **función** para **resumir** todas las filas de una tabla en un sólo resultado:

```
1 datos |>
2   summarize(total = sum(personas))
```

```
# A tibble: 1 × 1
  total
  <dbl>
1 3368245.
```

Si **agrupamos** los datos, obtendremos una fila de resultado por cada grupo:

```
1 datos |>
2   group_by(region) |>
3   summarize(total = sum(personas)) |>
4   head(n = 4)
```

```
# A tibble: 4 × 2
  region          total
  <chr>          <dbl>
1 Antofagasta    121379.
2 Arica y Parinacota 48086.
3 Atacama        64481.
4 Aysén          15113.
```

Resumiendo datos

Veamos los datos de educación del Censo 2024, que contienen la escolaridad promedio de cada comuna, por sexo.

 Descargar datos de educación en Excel

```
1 educacion <- readxl::read_xlsx("datos/censo/educacion.xlsx")
2
3 educacion |>
4   filter(region != "País") |>
5   filter(sexo == "Total Comuna") |>
6   group_by(region) |>
7   summarize(escolaridad = mean(escolaridad),
8             escolaridad_mayores_edad = mean(escolaridad_mayores_edad),
9             n_comunas = n()) |>
10  head(n = 4)
```

```
# A tibble: 4 × 4
  region                                escolaridad escolaridad_mayores_...1 n_comunas
  <chr>                                <dbl>          <dbl>          <int>
1 Antofagasta                        9.77           11.6             9
2 Arica y Parinacota                  8.6            9.85             4
3 Atacama                            9.5            11.1             9
4 Aysén del General Carlos Ibáñez ... 9.45           11.0            10
# i abbreviated name: 1escolaridad_mayores_edad
```

Cruce de datos o *left join*

 [Ver tutorial](#)

Para cruzar dos tablas de datos
usamos `left_join()` de `{dplyr}`

tabla X				tabla Y		
número	cifra	id		id	letra	nivel
1	30	A	→	A	Z	alto
2	25	B	→	B	X	alto
3	32	C	→	C	Y	bajo

resultado				
número	cifra	id	letra	nivel
1	30	A	Z	alto
2	25	B	X	alto
3	32	C	Y	bajo



Descargar datos de clasificación comunal en Excel

Cruzando datos

Veamos un ejemplo donde cruzamos dos tablas, la de **pobreza** y una nueva de **clasificación comunal** (urbana, mixta, rural según la PNDR), a partir de la columna que tienen en común: **codigo**

```
1 clasificacion <- readr::read_csv2("datos/clasificacion/clasificacion.csv")
```

```
1 clasificacion_b <- clasificacion |>  
2   select(codigo, clasificacion)
```

```
1 datos_clasif <- datos |>  
2   select(codigo, comuna, personas, porcentaje) |>  
3   mutate(codigo = as.numeric(codigo)) |>  
4   left_join(clasificacion_b, by = "codigo")  
5  
6 datos_clasif |> head(n = 3)
```

A tibble: 3 × 5

	codigo	comuna	personas	porcentaje	clasificacion
	<dbl>	<chr>	<dbl>	<dbl>	<chr>
1	1101	Iquique	41967.	0.183	Urbana
2	1107	Alto Hospicio	45162.	0.326	Urbana
3	1401	Pozo Almonte	4563.	0.250	Rural

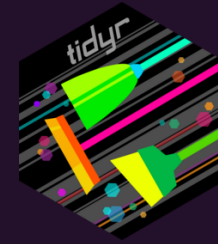
Transformar datos

 [Ver tutorial](#)

```
1 library(tidyr)
```

- ▶ El paquete `{tidyr}` se especializa en **transformar la estructura de los datos**.
- ▶ Esto nos permite **manipular** los datos para que sea más cómodo o lógico realizar ciertas operaciones.
- ▶ Por ejemplo:
- ▶ Si necesitamos sumar varias variables, es más cómodo tenerlas en filas (para poder usar algo como `sum(valores)`) que en columnas (y tener que sumar `a+b+c+d...`)
- ▶ Si queremos comparar varias variables, o mostrarlas a un público, es mejor que estén lado a lado, como en las tablas comunes.

Ejemplo de transformación de datos



- ▶ **Pivotar** una tabla sirve para cambiar su forma:
- ▶ De variables en columnas a filas: `pivot_longer()`
- ▶ De variables en filas a columnas: `pivot_wider()`



Transformando datos

Veamos un ejemplo donde transformamos los datos de estimaciones de población del INE, desde un formato **ancho** (años en columnas) a uno **largo** (años en filas).



Descargar datos de estimaciones de población en Excel

```
1 library(readxl)
2
3 poblacion <- read_xlsx("datos/estimaciones/
4
5 poblacion |>
6   select(1:6) |>
7   head()
```

```
# A tibble: 6 × 6
  edad  sexo `1992` `1993` `1994` `1995`
  <chr> <chr>   <dbl>  <dbl>  <dbl>  <dbl>
1 0     Hombres 151857 148533 146443 144016
2 1     Hombres 153970 151413 148142 146101
3 2     Hombres 153669 153837 151313 148047
4 3     Hombres 147481 153581 153800 151266
5 4     Hombres 140829 147469 153571 153785
6 5     Hombres 134445 140838 147461 153551
```

```
1 library(tidyr)
2
3 poblacion |>
4   pivot_longer(cols = `1992`:`2070`,
5                 names_to = "año",
6                 values_to = "poblacion") |>
7   head()
```

```
# A tibble: 6 × 4
  edad  sexo  año  poblacion
  <chr> <chr>  <chr>    <dbl>
1 0     Hombres 1992    151857
2 0     Hombres 1993    148533
3 0     Hombres 1994    146443
4 0     Hombres 1995    144016
5 0     Hombres 1996    139539
6 0     Hombres 1997    138013
```

Tutoriales de `{dplyr}`

Para que puedan repasar o guiarse con **casos básicos de uso** de `{dplyr}` con **datos reales**, dejo un par de tutoriales:



Tutorial con datos censales



Tutorial con datos sociales

Datos para practicar

Algunos conjuntos de datos limpios para practicar



Clasificaciones comunales



Estimaciones de población



Escolaridad por comuna



Población según género

{ggplot2}

- ▶ Paquete de **visualización de datos**
- ▶ Una de las soluciones más populares a nivel mundial
- ▶ Produce gráficos por medio de **capas**
- ▶ Entre sus beneficios están su flexibilidad, reusabilidad y extensibilidad

```
1 # install.packages("ggplot2")  
2 library(ggplot2)
```



Ver tutorial

Gramática de gráficos



- ▶ **Datos:** variables disponibles para construir el gráfico
- ▶ **Mapeos:** conexión de variables a distintos aspectos de la visualización
- ▶ **Capas:** elementos geométricos agregados unos sobre otros
- ▶ **Escalas:** especificación de la forma en que se mapean las variables a los ejes
- ▶ **Facetas:** dividir la visualización en paneles distintos según una variable
- ▶ **Coordenadas:** proyección de las coordenadas en el plano
- ▶ **Temas:** definición de la apariencia general del gráfico, y específica para cada elemento

Gramática aplicada al código

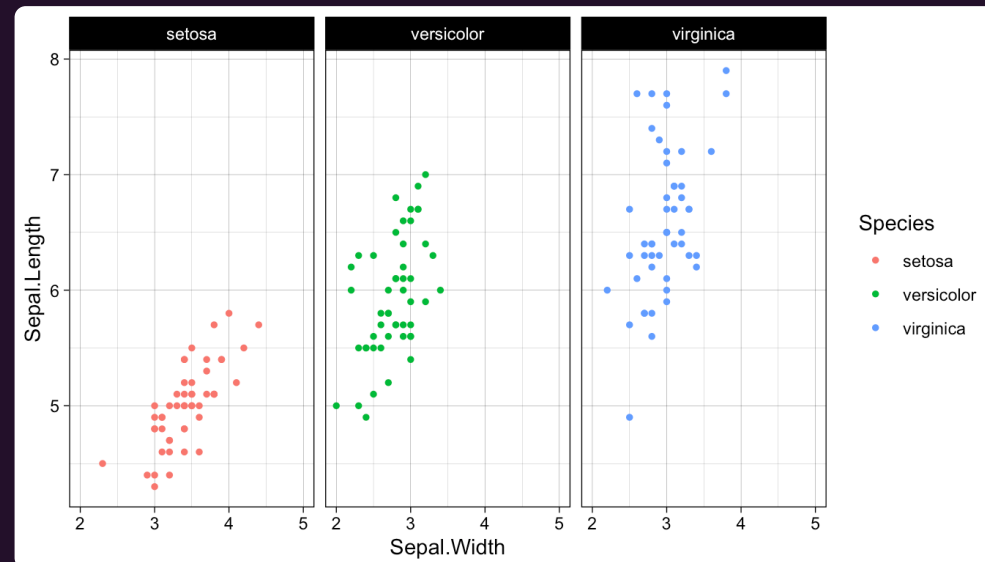
 Ver tutorial

Conceptos que constituyen la **gramática** de los gráficos:

- ▶ Datos **línea 1**
- ▶ Mapeos **línea 2**
- ▶ Capas **línea 3**
- ▶ Escalas **línea 4**
- ▶ Facetas **línea 5**
- ▶ Coordenadas **línea 6**
- ▶ Temas **línea 7**

Conceptos de la gramática de gráficos aplicados al código:

```
1 ggplot(iris) + # datos
2   aes(Sepal.Width, Sepal.Length, color = Species) + #
3   geom_point(size = 1) + # capas
4   scale_color_discrete() + # escalas
5   facet_wrap(~Species) + # facetas
6   coord_cartesian(xlim = c(2, 5)) + # coordenadas
7   theme_linedraw() # temas
```

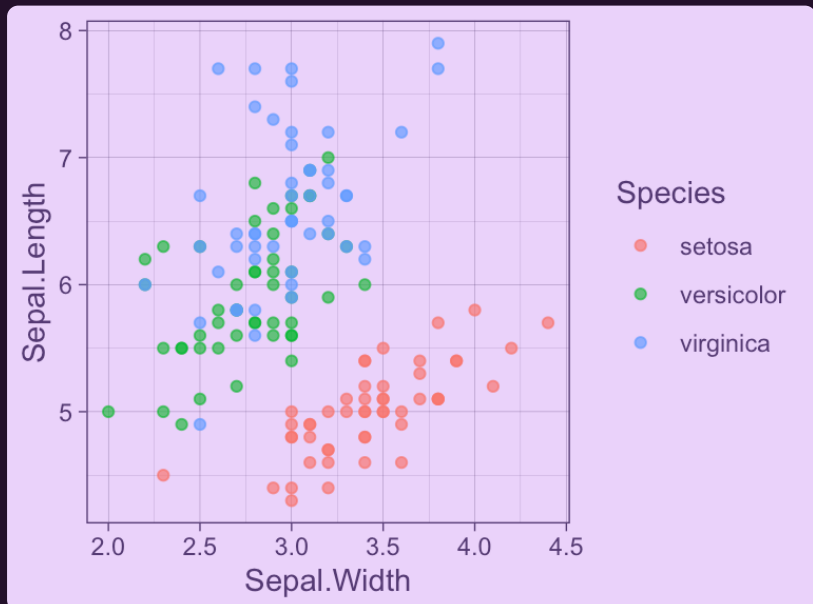


Colores del gráfico

```
1 grafico <- ggplot(iris) +  
2   aes(Sepal.Width, Sepal.Length, color = Species) +  
3   geom_point(alpha = 0.7)
```

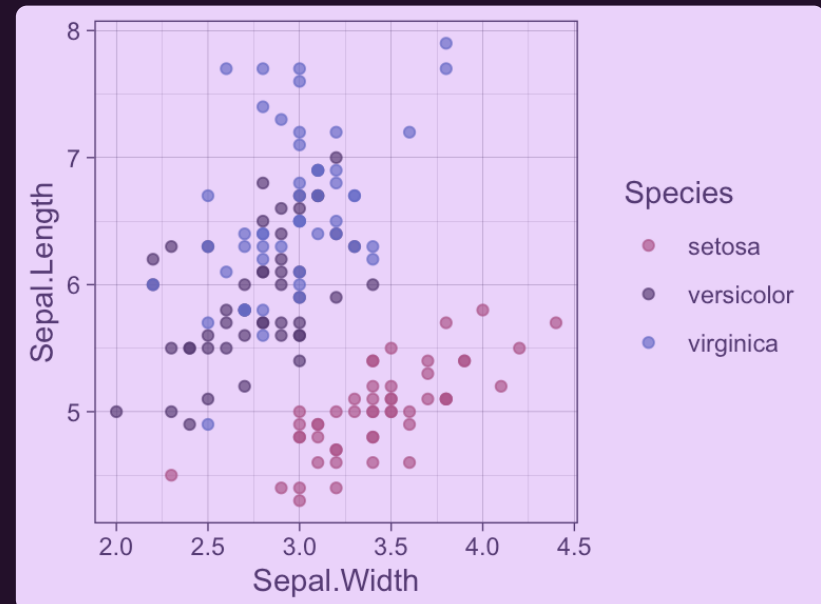
Colores generales del gráfico

```
1 grafico_tema <- grafico +  
2   theme_linedraw(paper = "#EAD2FA",  
3                 ink = "#553A74",  
4                 accent = "#9069C0")  
5 grafico_tema
```



Colores de las categorías

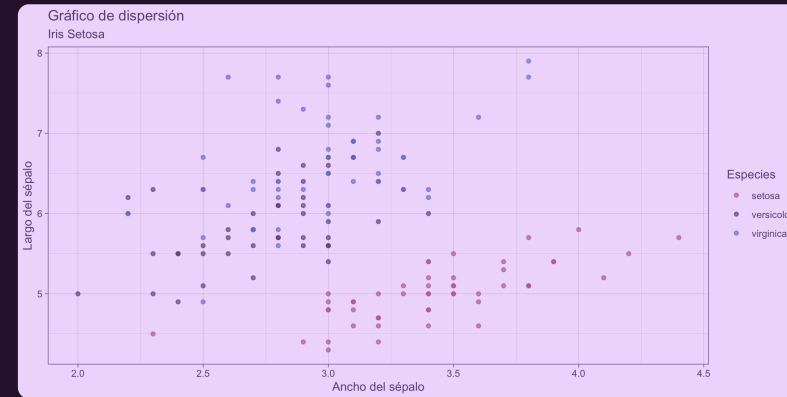
```
1 grafico_color <- grafico_tema +  
2   scale_color_discrete(  
3     palette = c("#AC558A", "#553A74",  
4                 "#666BC7"))  
5 grafico_color
```



Textos del gráfico

Podemos configurar y agregar textos del gráfico con la capa `labs()`:

```
1 grafico_textos <- grafico_color +  
2   labs(x = "Ancho del sépal",  
3       y = "Largo del sépal",  
4       color = "Especies",  
5       title = "Gráfico de dispersión",  
6       subtitle = "Iris Setosa")  
7  
8 grafico_textos
```



Temas

Configuración completa de cualquier aspecto o elemento del gráfico.

Funciones de temas

Agregar un tema prediseñado que cambia la apariencia general del gráfico:

- ▶ `theme_minimal()`
- ▶ `theme_classic()`
- ▶ `theme_void()`
- ▶ `theme_linedraw()`

Componentes del gráfico

En `theme()` podemos cambiar aspectos específicos. Los principales empiezan con:

- ▶ `panel`
- ▶ `grid`
- ▶ `scales`
- ▶ `axis`
- ▶ `strip`
- ▶ `legend`

Tipos de componente

Cada aspecto del gráfico es de un **tipo** distinto, y cada tipo se configura mediante una función:

- ▶ `element_text()`,
txtos
- ▶ `element_line()`,
líneas
- ▶ `element_rect()`,
rectángulos
- ▶ `element_blank()`,
ocultar

Referencia de temas

Hoja de referencia de los elementos de temas en `{ggplot2}`, fuente: [Isabella Benabaye](#)

ggplot2

theme elements reference

Set minimal as the baseline theme:

```
theme_minimal() +  
theme(theme.element = element_type())
```

Use `element_blank()` to remove an element

Axis titles, text, ticks, and lines can be specified per axis using theme inheritance by putting `.x/.y` at the end of the theme element.

`axis.line.y = element_line()`

`axis.title.y = element_text()`

`panel.grid.major = element_line()`

`panel.grid.minor = element_line()`

`axis.text.y`

`axis.text = element_text()`

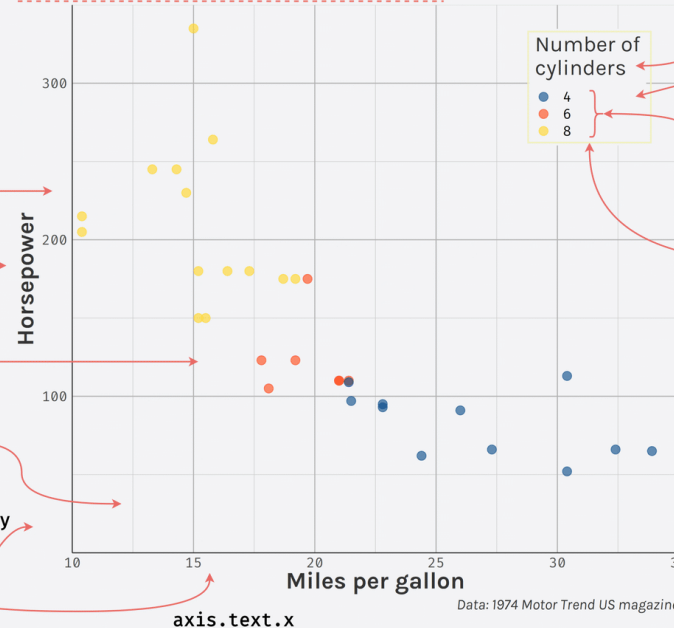
`text = element_text()` ← modifications will be applied to all text elements

```
plot.title.position = "plot"  
plot.caption.position = "plot"  
plot.title = element_text()  
plot.subtitle = element_text()
```

} "plot" means that they will be aligned to the entire plot (instead of the panel)

`plot.margin = margin(25, 25, 25, 25)`

Miles per Gallon & Horsepower of 32 Automobiles (1973-74 models)



`legend.title = element_text()`

`legend.background = element_rect()`

`legend.text = element_text()`

`legend.position = c(.85, .85) / "none" /
"left" / "right" /
"bottom" / "top"`

`plot.background = element_rect()`

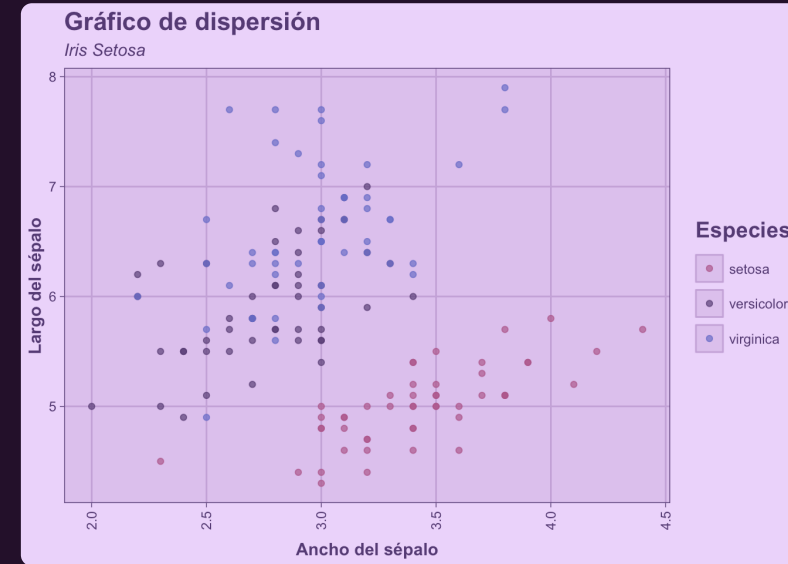
`plot.caption = element_text()`

[isabella-b](#)

Full list of elements at ggplot2.tidyverse.org/reference/theme

Personalización de temas

```
1 grafico_textos +  
2   theme(plot.title = element_text(face = "  
3     plot.subtitle = element_text(famil  
4     panel.grid.major = element_line(li  
5     panel.grid.minor = element_blank()  
6     panel.background = element_rect(fi  
7     axis.title = element_text(face = "  
8     axis.title.x = element_text(margin  
9     axis.text.x = element_text(angle =  
10    legend.title = element_text(face =  
11    legend.text = element_text(margin  
12    legend.key = element_rect(fill = "  
13    legend.key.spacing.y = unit(1.8, "
```



Extensiones de `{ggplot2}`

Existen muchos paquetes que extienden las capacidades de `{ggplot2}`:

- ▶ `{ggforce}`: nuevas geometrías
- ▶ `{patchwork}`: combinación de gráficos
- ▶ `{ggrepel}`: separación de textos superpuestos
- ▶ `{gganimate}`: gráficos animados
- ▶ `{marquee}` y `{ggtext}`: textos con formato, color, y más
- ▶ `{ggiraph}` y `{plotly}`: gráficos interactivos



Acá hay una [lista de extensiones de `{ggplot2}`](#)

Recursos de aprendizaje

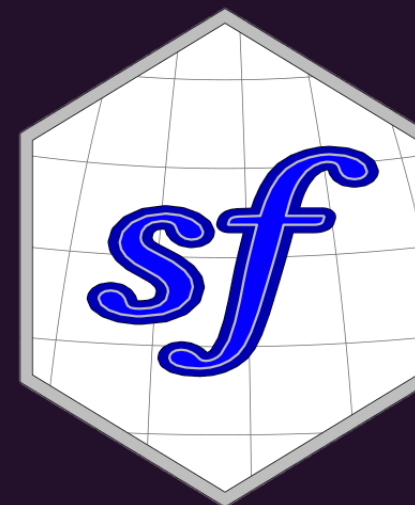
 Tutorial de visualización de datos con `{ggplot2}`

- ▶ [R Graph Gallery](#).
- ▶ [R Charts](#)
- ▶ [A ggplot2 Tutorial for Beautiful Plotting in R](#), tutorial de `{ggplot2}` en inglés

Mapas

- ▶ El paquete `{sf}` entrega herramientas para trabajar con datos geoespaciales en R
- ▶ Permite representar características espaciales en *data frames*
- ▶ Se complementa con `{ggplot2}`

```
1 # install.packages("sf")  
2 library(sf)
```



 Ver tutorial

Mapas de Chile

[Ver tutorial](#)

El paquete `{chilemapas}` ofrece funciones que simplifican la obtención de mapas de Chile en R.

```
1 # install.packages("chilemapas")
2 library(chilemapas)
3
4 mapa_comunas |>
5   filter(codigo_region == "14")
```

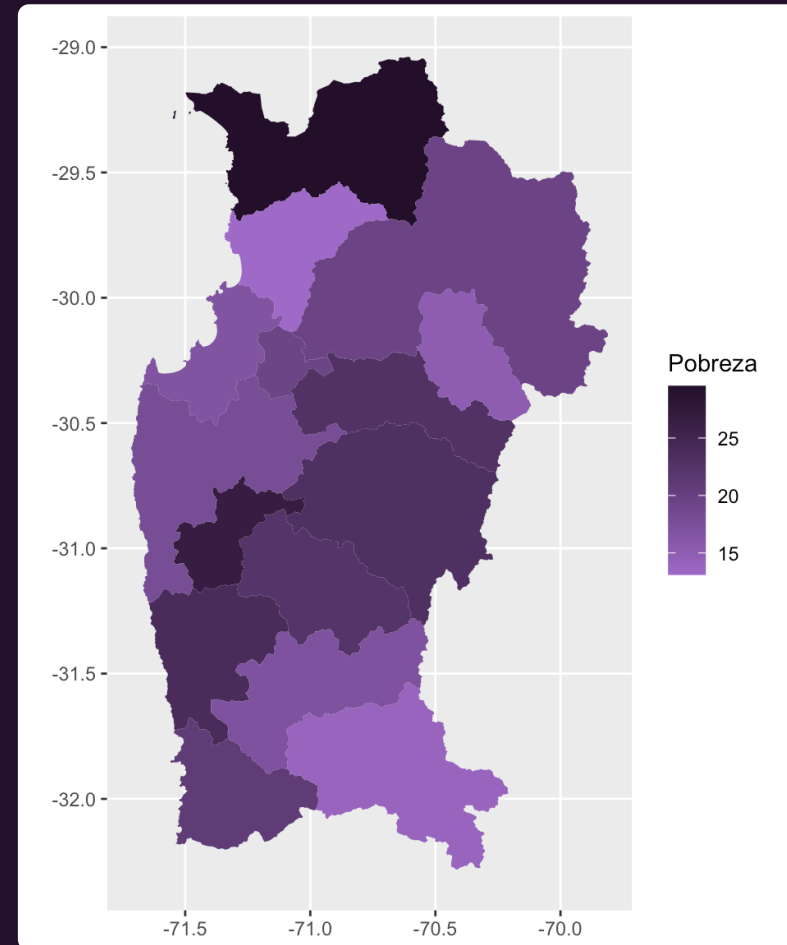
A tibble: 12 × 4

	codigo_comuna <chr>	codigo_provincia <chr>	codigo_region <chr>	geometry <MULTIPOLYGON [°]>
1	14203	142	14	(((-71.65597 -40.35386, -71.657...
2	14204	142	14	(((-71.88915 -40.55144, -71.891...
3	14201	142	14	(((-73.15077 -40.06596, -73.147...
4	14202	142	14	(((-71.81459 -40.08101, -71.820...
5	14104	141	14	(((-72.02965 -39.93203, -72.034...
6	14105	141	14	(((-72.61245 -39.62475, -72.607...
7	14103	141	14	(((-72.32959 -39.51024, -72.327...
8	14106	141	14	(((-72.83588 -39.41328, -72.841...
9	14107	141	14	(((-72.92395 -39.94016, -72.916...
10	14108	141	14	(((-71.81459 -40.08101, -71.815...
11	14102	141	14	(((-73.39203 -39.88653, -73.389...
12	14101	141	14	(((-73.2668 -39.88605, -73.2621...

Visualizando datos en un mapa

Sólo basta con cruzar una tabla con las geometrías

```
1 # cargar datos de pobreza por comuna
2 pobreza <- read_xlsx("datos/pobreza/estima
3   mutate(codigo = as.numeric(codigo))
4
5 # filtrar mapa en una región específica
6 mapa_region <- mapa_comunas |>
7   filter(codigo_region == "04") |>
8   mutate(codigo = as.numeric(codigo_comuna
9
10 # unir el mapa con los datos
11 mapa_datos <- mapa_region |>
12   left_join(pobreza, by = "codigo")
13
14 # visualizar
15 mapa_datos |>
16   ggplot() +
17     aes(geometry = geometry,
18         fill = porcentaje * 100) +
19     geom_sf(linewidth = 0) +
20     scale_fill_gradient(low = "#9F69C7", hig
21     labs(fill = "Pobreza")
```



Datos para practicar

 Estimaciones de población Censo

 Variables educacionales Censo 2024

 Estimación de pobreza multidimensional 2022

 Clasificaciones comunales 2024

Puedes encontrar más datos en mi [repositorio de datos sociales](#), en el [banco integrado de datos del Ministerio de Desarrollo](#), en los [datos abiertos del Estado Chile](#), y [muchos más](#).

Ahora que ya sabes lo básico, todo lo demás se trata de aprender piezas y herramientas que puedes ir agregando a tus análisis! 

Consejos

- ▶ Mantengan su propio **libro de aprendizajes** 
- ▶ Lean bien los **errores** y busquen su significado
- ▶ Si algo sale mal, prueben **paso a paso**
- ▶ También pueden devolverse y **empezar de nuevo**
- ▶ Dense una vuelta y a veces la respuesta llega solita 😊
- ▶ No se vuelvan dependientes de la IA 😞
- ▶ Si quieren usar IA:
- ▶ **Autocompletado** de GitHub Copilot
- ▶ **Mensaje de sistema** para su proveedor de IA
- ▶ Aprendan a **buscar** bien en internet
- ▶ Compartan lo que hacen y socialicen sus aprendizajes! ✨

Gracias ♥

Puedes escribirme cualquier duda o comentario [por aquí](#)



Aprende R



Chat grupal



Curso



Código



Grabaciones