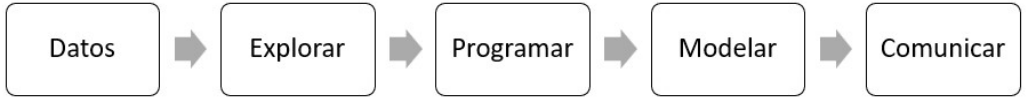
Análisis de encuestas de hogares con R

Curso Básico Rstudio

CEPAL - Unidad de Estadísticas Sociales

Introducción

Cuando trabajamos con datos en R, seguimos un flujo de trabajo que nos permite pasar de datos en bruto a resultados útiles y comprensibles. Este proceso tiene cuatro etapas principales:



Carga e importación de librerias

Antes de trabajar con datos en R, es necesario cargar las librerías, que son conjuntos de funciones ya creadas que nos facilitan el análisis.

- Instalar una librería (install.packages()) se hace solo una vez.
- Cargarla (library()) se debe hacer cada vez que abrimos R o RStudio.
- Algunas librerías importantes para análisis de datos son: tidyverse, dplyr, ggplot2, readr, readxl, entre otras.

Carga e importación de librerias

```r
# Instalar (solo la primera vez):
# install.packages("tidyverse")

library(tidyverse)   # Incluye dplyr, ggplot2, readr, etc.
```

¿Por qué es importante?

Porque sin cargar las librerías, R no reconoce funciones como filter(), ggplot(), read_csv(), etc.

Carga e importación de base de datos

Antes de analizar, debemos importar o leer los datos y traerlos a R. Esto significa que R toma la información que está guardada en un archivo externo (como Excel, CSV, RDS, SPSS, etc.) y la convierte en un objeto dentro del entorno de trabajo.

Una vez los datos están cargados en un objeto (por ejemplo, un data frame llamado datos), podemos visualizarlos, limpiarlos, transformarlos, hacer gráficos o aplicar modelos estadísticos. Es decir, importar los datos es el paso que conecta la información real con el análisis que haremos en R.

Carga e importación de base de datos

Formatps más comunes

Tipo de archivo	Librería	Ejemplo en R
CSV	readr	`datos <- read_csv("data/archivo.csv")`
Texto delimitado ";"	readr	`datos <- read_delim("data/archivo.txt", delim = ";")`
Texto delimitado tab	readr	`datos <- read_tsv("data/archivo.tsv")`
Excel (.xlsx)	readxl	`datos <- read_excel("data/datos.xlsx")`
RDS (.rds)	Base de R	`datos <- readRDS("data/base.rds")`
RData (.RData)	Base de R	`load("data/mi_objeto.RData")`

Carga e importación de base de datos

Otros formatos

Tipo de archivo	Librería Ejemplo en R	
SPSS (.sav)	haven	`datos <- read_sav("data/archivo.sav")`
Stata (.dta)	haven	`datos <- read_dta("data/archivo.dta")`
JSON	jsonlite	`datos <- fromJSON("data/archivo.json")`
CSV grande (.csv.gz)	data.table	`datos <- fread("data/archivo.csv.gz")`

Carga e importación de base de datos

- Ejemplo: cargar un archivo RDS (.rds)

```
# Importar la base (ejemplo)
datos <- readRDS(
  "../Data/base_personas.rds"
) %>% as.data.frame() # readdRDS es una funcion base de R
# Ver las primeras filas
head(datos[, 1:8], 3)
```

id_hogar	id_pers	parentesco	edad	sexo	etnia	area	ingreso
262	1	1	51	Hombre	0	1	542000.0
262	1	1	51	Hombre	0	1	536305.3
262	2	2	46	Mujer	0	1	542000.0

Explorar

Explorar es el primer paso para conocer los datos. Consiste en mirarlos, hacer preguntas, generar ideas rápidas y comprobarlas visualmente o con resúmenes simples. No busca respuestas finales, sino entender qué hay en los datos, detectar patrones, errores o curiosidades que luego podamos analizar mejor.

Explorar: conocimientos básicos

Antes de explorar datos, necesitamos saber cómo funciona R: cómo escribir código, crear objetos, usar funciones, y organizar nuestro trabajo. Estas bases son lo que permite explorar, transformar y modelar datos de forma confiable.

Explorar: conocimientos básicos

R como calculadora

R puede ejecutar operaciones matemáticas directamente:

```
1 + 2
```

```
[1] 3
```

```
3 * 5
```

```
[1] 15
```

```
(10 + 5) / 3
```

```
[1] 5
```

Esto es útil, pero no suficiente si no guardamos los resultados.

Explorar: conocimientos básicos

Crear objetos

```
x <- 3 * 4
x
```

```
[1] 12
```

```
resultado <- (59 + 73 + 2) / 3
resultado
```

```
[1] 44.66667
```

Con el símbolo <- le asignamos un valor a un objeto. Esto permite guardar un resultado con un nombre y reutilizarlo más adelante en el código.

Explorar: conocimientos básicos

Crear objetos

Buenas prácticas al nombrar objetos

- Usar nombres claros y descriptivos:promedio_altura, ventas_2024.
- No usar espacios ni tildes.
- Recomendado: snake_case (todo en minúsculas, separar con _).
- R distingue mayúsculas/minúsculas: Edad edad

Explorar: conocimientos básicos

Tipos de datos

Antes de explorar y analizar una base de datos, es fundamental reconocer qué tipo de información contiene cada variable.
Conocer los tipos de datos nos permite:

- Aplicar las funciones correctas (por ejemplo: sumar, filtrar, agrupar, graficar).
- Evitar errores al transformar o visualizar datos.
- Elegir correctamente cómo interpretar la información (número, texto, fecha, categoría, etc.).
- Preparar los datos adecuadamente para modelos estadísticos

Explorar: conocimientos básicos

Tipos de datos

A continuación, se presentan los tipos de datos más comunes en R:

Tipo de dato	**¿Qué representa?**
`numeric`	Números decimales o enteros
`integer`	Números enteros declarados explícitamente
`character`	Texto: palabras, nombres, frases
`logical`	Valores lógicos: verdadero o falso
`factor`	Categorías o niveles (variables cualitativas)
`Date`	Fechas en formato calendario

Explorar: conocimientos básicos

Tipos de datos

```
# Numeric (numérico)
x_num <- 12.5
class(x_num)
```

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

```
# Integer (entero)
x_int <- 7
class(x_int)
```

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

Explorar: conocimientos básicos

Tipos de datos

```
# Character (texto)
x_char <- "Bogotá"
class(x_char)
```

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

```
# Logical (lógico)
x_log <- TRUE
class(x_log)
```

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

Explorar: conocimientos básicos

Tipos de datos

```
# Factor (categorías)
x_fac <- factor(c("Primaria", "Secundaria", "Universitaria"))
class(x_fac)
```

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

```
levels(x_fac)
```

```
[1] "Primaria"      "Secundaria"    "Universitaria"
```

```
# Date (fecha)
x_date <- as.Date("2025-10-21")
class(x_date)
```

```
[1] "Date"
```

Explorar: conocimientos básicos

Clases de objetos

En R, los datos no se guardan de forma suelta, sino dentro de objetos. Estos objetos pueden tener diferentes estructuras o "clases", y conocerlas es clave para trabajar correctamente.

Explorar: conocimientos básicos

Clases de objetos

Tipo de objeto	Descripción
Vector	Conjunto básico de valores del mismo tipo.
Factor	Vector para datos categóricos con niveles definidos.
Matrix	Arreglo rectangular de datos numéricos o de un solo tipo.
Array	Extiende la idea de matriz a más de dos dimensiones.
Data Frame	Tabla similar a Excel; cada columna puede tener distinto tipo de dato.
Tibble	Versión moderna del data frame (tidyverse), más ordenada y amigable.
List	Contenedor que puede almacenar objetos de cualquier tipo y tamaño.

Explorar: conocimientos básicos

Clases de objetos: Vector

```r
# Vector numérico
edades <- c(25, 30, 28, 40)
edades
```

```
[1] 25 30 28 40
```

```r
# Vector de texto (character)
nombres <- c("Ana", "Luis", "María")
nombres
```

```
[1] "Ana"   "Luis"  "María"
```

```r
# Vector lógico (TRUE / FALSE)
es_mayor_edad <- c(TRUE, TRUE, FALSE, TRUE)
es_mayor_edad
```

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

Explorar: conocimientos básicos

Clases de objetos: Factor

```
sexo <- factor(c("Mujer", "Hombre", "Mujer", "Hombre"))
levels(sexo)       # Niveles del factor
```

```
[1] "Hombre" "Mujer"
```

```
class(sexo)        # "factor"
```

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

Explorar: conocimientos básicos

Clases de objetos: Matriz

```
matriz_ejemplo <- matrix(1:9, nrow = 3, ncol = 3)
matriz_ejemplo
```

1	4	7
2	5	8
3	6	9

Explorar: conocimientos básicos

Clases de objetos: Data Frame

```r
personas <- data.frame(
  nombre = c("Ana", "Luis", "María"),
  edad   = c(23, 30, 28),
  ingreso = c(1200, 1500, 1800)
)

personas
```

nombre	edad	ingreso
Ana	23	1200
Luis	30	1500
María	28	1800

```r
class(personas)  # "data.frame"
```

Explorar: conocimientos básicos

Clases de objetos: Tibble

```
personas_tibble <- tibble(
  nombre = c("Ana", "Luis", "María"),
  edad   = c(23, 30, 28),
  ingreso = c(1200, 1500, 1800)
)
personas_tibble
```

nombre	edad	ingreso
Ana	23	1200
Luis	30	1500
María	28	1800

Explorar: conocimientos básicos

Clases de objetos: Lista

```r
mi_lista <- list(
  numeros = c(1, 2, 3),
  tabla = personas
)

mi_lista
```

```
$numeros
[1] 1 2 3

$tabla
  nombre edad ingreso
1    Ana   23    1200
2   Luis   30    1500
3  María   28    1800
```

Explorar: conocimientos básicos

Usando funciones en R

R trabaja principalmente a través de funciones, que se escriben con la forma:

```
nombre_funcion(argumento = valor)
```

Explorar: conocimientos básicos

Usando funciones en R

```r
# Secuencias y repetición
seq(1, 10, by = 2)      # 1, 3, 5, 7, 9
```

```
[1] 1 3 5 7 9
```

```r
rep(5, times = 4)       # 5 5 5 5
```

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

```r
rep(c("A","B"), each = 3)  # A A A B B B
```

```
[1] "A" "A" "A" "B" "B" "B"
```

Explorar: conocimientos básicos

Usando funciones en R

```
# Resumen rápido de datos
tail(datos[, 1:8], 3)          # Últimas 3 filas
```

	id_hogar	id_pers	parentesco	edad	sexo	etnia	area	ingreso
28969	70947	2	2	60	Mujer	1	2	195000.0
28970	70947	2	2	60	Mujer	1	2	189305.3
28971	70948	1	1	48	Mujer	1	2	900000.0

```
class(datos)             # Clase del objeto (data.frame, vector, etc.)
```

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

```
names(datos)             # Nombres de columnas
```

```
 [1] "id_hogar"   "id_pers"    "parentesco" "edad"       "sexo"
```

Explorar: conocimientos básicos

Usando funciones en R

```
# Resumen rápido de datos

str(datos)              # Estructura del objeto (tipo de dato, columnas, e
```

```
'data.frame':   28971 obs. of  14 variables:
 $ id_hogar  : num  262 262 262 262 265 265 265 265 265 277 ...
  ..- attr(*, "label")= chr "Identificador de hogar"
  ..- attr(*, "format.stata")= chr "%10.0g"
 $ id_pers   : num  1 1 2 2 1 1 2 3 3 1 ...
  ..- attr(*, "label")= chr "Identificador de la persona"
  ..- attr(*, "format.stata")= chr "%10.0g"
 $ parentesco: chr  "1" "1" "2" "2" ...
 $ edad      : num  51 51 46 46 26 26 24 7 7 42 ...
  ..- attr(*, "label")= chr "Edad de la persona"
  ..- attr(*, "format.stata")= chr "%10.0g"
 $ sexo      : chr  "Hombre" "Hombre" "Mujer" "Mujer" ...
 $ etnia     : chr  "0" "0" "0" "0" ...
```

Explorar: conocimientos básicos

Usando funciones en R

```r
# Estadísticas básicas
mean(datos$ingreso, na.rm = TRUE)   # Media
```

```
[1] 340451
```

```r
median(datos$ingreso)               # Mediana
```

```
[1] 222621.1
```

```r
sd(datos$ingreso)                   # Desviación estándar
```

```
[1] 526949.5
```

```r
var(datos$ingreso)                  # Varianza
```

```
[1] 277675726689
```

Explorar: conocimientos básicos

Usando funciones en R

¿Cómo me ayuda Rstudio?

- Si escribes el inicio de una función y presionas TAB, RStudio te sugiere cómo completarla.
- Si presionas F1 sobre una función (como mean o seq), aparece la ayuda explicando qué hace.
- RStudio cierra paréntesis y comillas automáticamente.

Si te olvidas de cerrar algo, aparece un símbolo como "+. Eso significa que R está esperando que completes la instrucción.

Explorar: Transformación de datos

Transformar datos es el "puente" entre mirar y modelar. Con dplyr podemos:

- Seleccionar variables (select, rename, relocate)
- Filtrar observaciones (filter)
- Ordenar filas (arrange)
- Crear variables derivadas (mutate, case_when, if_else)
- Resumir por grupos (group_by + summarise)

Explorar: Transformación de datos

Seleccionar variables

Seleccionar variables es el primer paso para ordenar una base de datos y trabajar únicamente con la información que realmente necesitamos. Muchas veces las bases contienen decenas o cientos de columnas, pero no todas son útiles para el análisis. Con select() podemos quedarnos solo con las variables relevantes; con rename() podemos ponerles nombres más claros o consistentes; y con relocate() podemos mover variables importantes al inicio para facilitar la lectura.

Explorar: Transformación de datos

Seleccionar variables

```
datos2 <- datos %>% select("id_hogar","id_pers", "edad","sexo",
                           "etnia","area","ingreso",
                            "pobreza", "anoest") %>% rename(
 id = id_pers
) %>% as.data.frame()

head(datos2,5)
```

id_hogar	id	edad	sexo	etnia	area	ingreso	pobreza	anoest
262	1	51	Hombre	0	1	542000.0	3	18
262	1	51	Hombre	0	1	536305.3	3	18
262	2	46	Mujer	0	1	542000.0	3	12
262	2	46	Mujer	0	1	536305.3	3	12
265	1	26	Mujer	0	1	710555.7	3	17

Explorar: Transformación de datos

Seleccionar variables

Filtrar observaciones consiste en quedarnos solo con las filas que cumplen ciertas condiciones analíticas (edad, área, empleo, ingresos válidos, etc.). Con filter() expresamos esas reglas de forma legible: combinamos operadores lógicos (&, |, !), conjuntos con %in%, y rangos con between().

Explorar: Transformación de datos

Seleccionar variables

Supongamos que es de nuestro interés es analizar únicamente a las personas que se encuentran en la zona rural, entonces debemos filtrar la base de datos para conservar solo aquellas observaciones cuyo estado laboral es "1".

```
datos_mayores <- datos2 %>%
  filter(area == "1")

head(datos_mayores,4)
```

id_hogar	id	edad	sexo	etnia	area	ingreso	pobreza	anoest
262	1	51	Hombre	0	1	542000.0	3	18
262	1	51	Hombre	0	1	536305.3	3	18
262	2	46	Mujer	0	1	542000.0	3	12
262	2	46	Mujer	0	1	536305.3	3	12

Explorar: Transformación de datos

Ordenar Filas

Ordenar filas nos permite reorganizar la base de datos según una o varias variables, facilitando la identificación de valores extremos, patrones o jerarquías dentro de la información. Con la función arrange() de dplyr, podemos ordenar de forma ascendente o descendente.

```
datos_ord <- datos2 %>% arrange(desc(ingreso))
head(datos_ord, 5)
```

id_hogar	id	edad	sexo	etnia	area	ingreso	pobreza	anoest
59266	1	52	Hombre	0	1	25383308	3	20
59266	2	45	Mujer	0	1	25383308	3	20
59266	3	13	Hombre	0	1	25383308	3	7
59266	1	52	Hombre	0	1	25377614	2	20
58397	1	50	Hombre	0	1	12024306	3	17

Explorar: Transformación de datos

Crear variables derivadas

Crear variables derivadas consiste en generar nuevas columnas a partir de otras ya existentes dentro de la base de datos. Esto es fundamental en el análisis de encuestas porque permite construir indicadores.Para ello utilizamos la funcion mutate() del paquete dplyr, que nos permiten transformar, combinar o recodificar variables sin alterar los datos originales.

Explorar: Transformación de datos

Crear variables con condicionales

Para crear nuevas variables a partir de reglas lógicas, R permite usar estructuras condicionales como ifelse o case_when, que evalúan una condición y ejecutan acciones dependiendo de si es verdadera o falsa.

¿Cuándo usar cada una?

- ifelse() $\rightarrow$ clasificar rápidamente toda una columna con 2 opciones
- case_when() $\rightarrow$ múltiples categorías o reglas, más ordenado y fácil de leer

Explorar: Transformación de datos

Crear variables con condicionales

```
# Crear variable: ¿ingreso alto o no?
datos2$ingreso_alto <- ifelse(datos2$ingreso > 1000,
                              "Alto",
                              "No alto")

# Si tiene más de 12 años de estudio = educación superior
datos2$educ_superior <- ifelse(datos2$anoest > 12,
                               "Sí",
                               "No")
```

Explorar: Transformación de datos

Crear variables con condicionales

```
# Crear grupos de edad (niñez, juventud, adultez, vejez)
datos2 <- datos2 %>%
  mutate(grupo_edad = case_when(
    edad < 12 ~ "Niñez",
    edad >= 12 & edad < 18 ~ "Adolescencia",
    edad >= 18 & edad < 60 ~ "Adultez",
    edad >= 60 ~ "Adulto mayor",
    TRUE ~ NA_character_
  ))
```

Explorar: Transformación de datos

Crear variables con condicionales

```
# Crear grupos de años de educación
datos2 <- datos2 %>%
  mutate(ranoest = case_when(
    anoest == 0  ~ "1", # Sin educacion
    anoest %in% c(1:6) ~ "2",       # 1 - 6
    anoest %in% c(7:12) ~ "3",      # 7 - 12
    anoest > 12 ~ "4",      # mas de 12
    TRUE ~ NA_character_
  ))
```

Explorar: Transformación de datos

Resumir por grupos

Resumir por grupos nos permite obtener indicadores estadísticos (promedios, totales, porcentajes, medianas, etc.) para diferentes categorías dentro de los datos.

```
resumen1 <- datos2 %>%
  group_by(sexo) %>%
  summarise(
    n = n(),
    ingreso_prom = mean(ingreso, na.rm = TRUE)
  )
resumen1
```

sexo	n	ingreso_prom
Hombre	14002	351264.4
Mujer	14969	330336.1

Explorar: Visualización de datos

"Un simple gráfico ha brindado más información a la mente del analista de datos que cualquier otro dispositivo". — John Tukey

Los gráficos permiten ver lo que las tablas no muestran: patrones, diferencias y tendencias de un solo vistazo.

En esta sección aprenderemos a:

- Crear gráficos básicos con ggplot2.
- Representar relaciones entre variables (barras, dispersión, boxplots, histogramas).
- Personalizar colores, ejes y etiquetas para comunicar mejor los datos.

Explorar: Visualización de datos

Para trabajar con gráficos en R usaremos ggplot2, que hace parte del Tidyverse. Antes de crear gráficos, es útil recordar cómo es nuestra base de datos y pensar qué información nos gustaría visualizar.

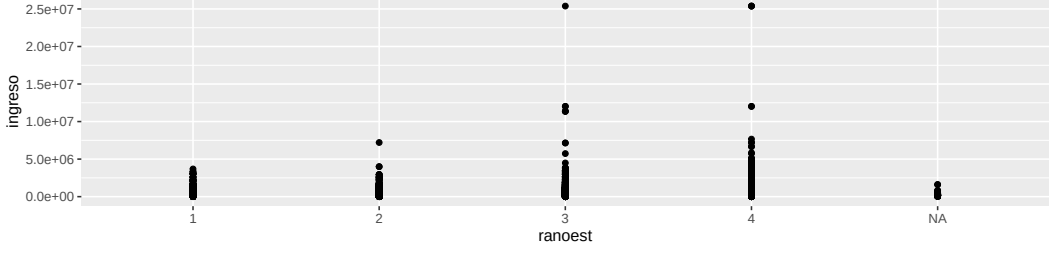
```
head(datos2[, 1:8],5)
```

id_hogar	id	edad	sexo	etnia	area	ingreso	pobreza
262	1	51	Hombre	0	1	542000.0	3
262	1	51	Hombre	0	1	536305.3	3
262	2	46	Mujer	0	1	542000.0	3
262	2	46	Mujer	0	1	536305.3	3
265	1	26	Mujer	0	1	710555.7	3

Explorar: Visualización de datos

Una primera pregunta que podríamos hacernos al observar la base de datos es: ¿las personas con mayor nivel educativo tienen mayores ingresos?

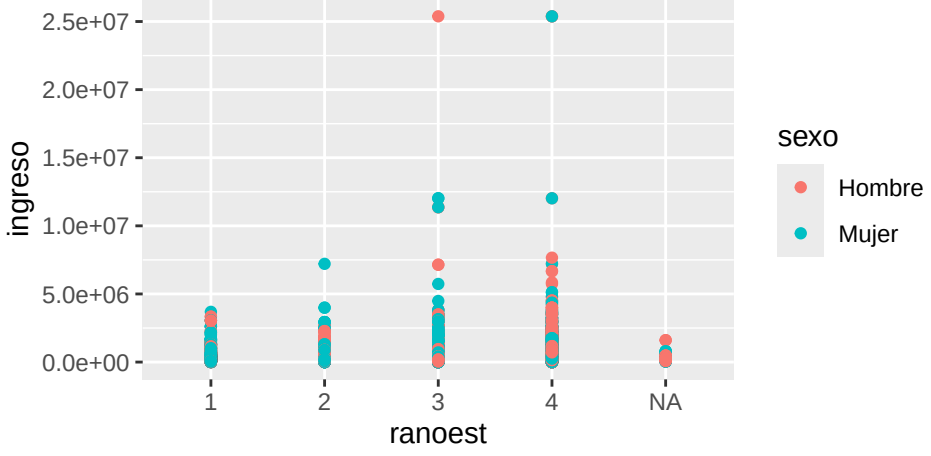
```
ggplot(data = datos2) +
  geom_point(mapping = aes(x = ranoest, y = ingreso))
```



Explorar: Visualización de datos

Si además queremos comparar si existen diferencias entre hombres y mujeres, podemos incorporar la variable sexo al gráfico.

```
ggplot(data = datos2) +
  geom_point(mapping = aes(x = ranoest, y = ingreso, color = sexo))
```



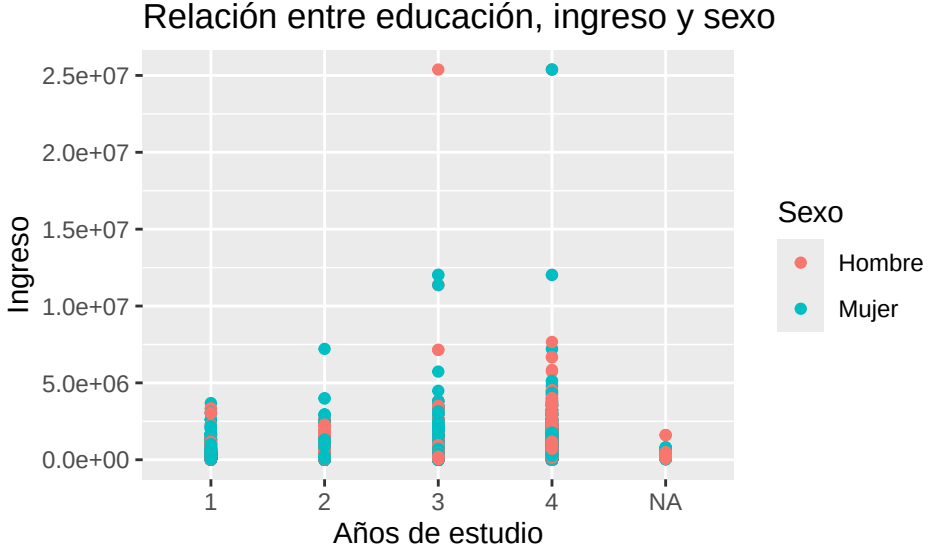
Explorar: Visualización de datos

También podemos añadir un título al gráfico y etiquetas a los ejes para que la información sea más clara y fácil de interpretar.

```
grafico <- ggplot(data = datos2) +
  geom_point(mapping = aes(x = ranoest, y = ingreso, color = sexo)) +
  labs(
    title = "Relación entre educación, ingreso y sexo",
    x = "Años de estudio",
    y = "Ingreso",
    color = "Sexo"
  )
```

Explorar: Visualización de datos

```
grafico
```



Explorar: Visualización de datos

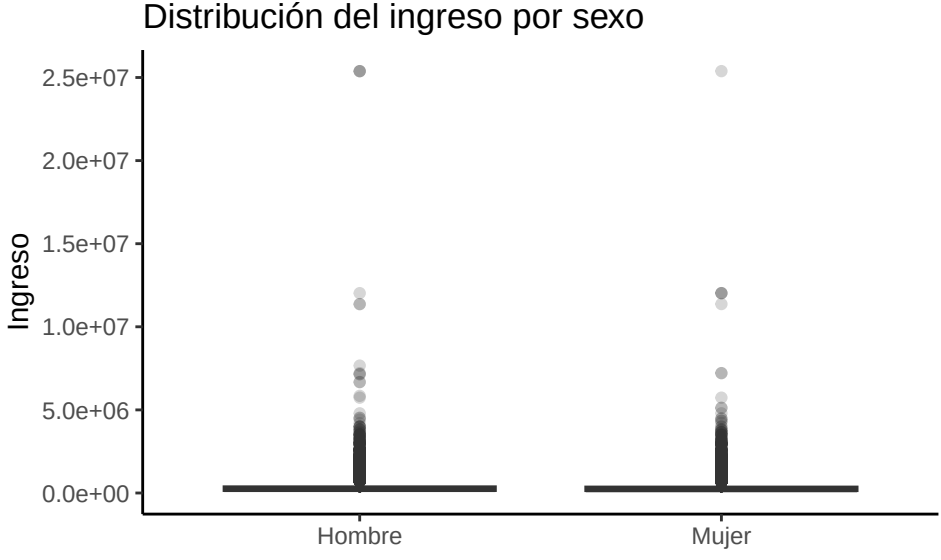
Hasta ahora hemos observado cómo varía el ingreso según los años de estudio. Pero ¿existen diferencias en los ingresos entre hombres y mujeres, independientemente del nivel educativo?

Para explorar esta pregunta podemos usar un boxplot, que nos permite comparar la distribución del ingreso entre ambos grupos. Este gráfico muestra valores como la mediana, los cuartiles y posibles valores atípicos.

```
box_plot<- ggplot(datos2, aes(x = sexo, y = ingreso, fill = sexo)) +
  geom_boxplot(outlier.alpha = 0.2) +
  labs(title = "Distribución del ingreso por sexo",
       x = NULL, y = "Ingreso") +
  theme_classic() +
  theme(legend.position = "none")
```

Explorar: Visualización de datos

```
box_plot
```



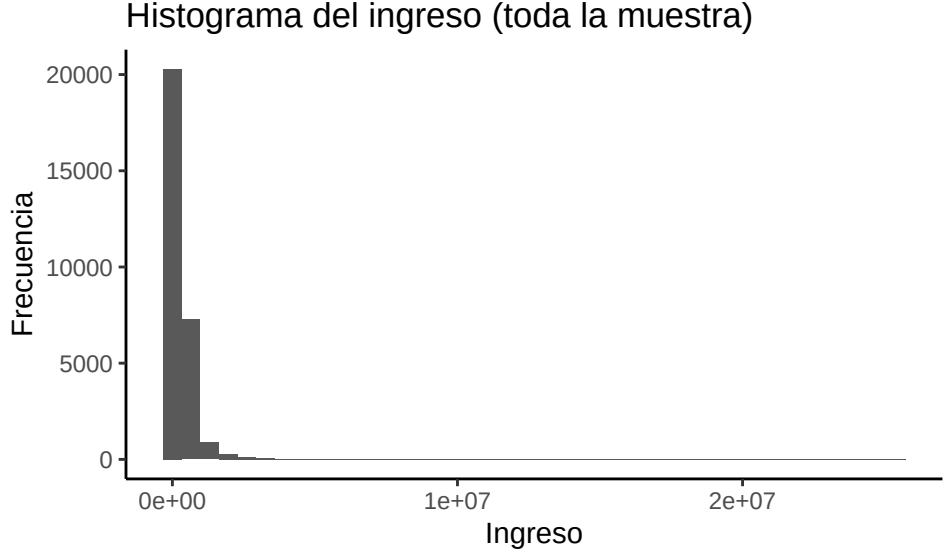
Explorar: Visualización de datos

Finalmente, muchas veces nos interesa ver el comportamiento del ingreso total en toda la muestra. Para ello, un histograma permite observar la forma de la distribución (concentraciones, asimetrías y posibles valores extremos).

```
histo <- ggplot(datos2, aes(x = ingreso)) +
geom_histogram(bins = 40) +
labs(title = "Histograma del ingreso (toda la muestra)",
x = "Ingreso", y = "Frecuencia") +
theme_classic()
```

Explorar: Visualización de datos

```
histo
```



Explorar: Visualización de datos

Otras geoms útiles en ggplot2

- Líneas: geom_line() – Series temporales o secuencias.
- Barras: geom_bar() $\rightarrow$ cuenta filas (stat = "count").
- Barras: geom_col() $\rightarrow$ altura = valor (y) ya calculado.
- Densidad: geom_density() – Distribución suavizada.
- Tendencia: geom_smooth() – Curva/recta ajustada.

Programar

¿Por qué programar y no solo ejecutar código?

- Automatizar tareas repetitivas.
- Asegurar reproducibilidad (que otra persona pueda replicar el análisis).
- Organizar el trabajo para proyectos reales, no solo ejemplos.
- Evitar copiar/pegar mil veces lo mismo.

Pipes

Los pipes son una forma de escribir código en R que permite encadenar varias acciones de manera ordenada y legible, como si leyéramos una receta paso a paso. En lugar de escribir funciones anidadas o crear muchas variables intermedias, los pipes permiten decir:
"Toma estos datos $\rightarrow$ luego filtra $\rightarrow$ luego crea una variable $\rightarrow$ luego ordena".
Por eso se les llama "pipes", porque el resultado de una operación se "envía" a la siguiente.

Programar

Pipes

- Ejemplo: Si queremos conocer cuál es el ingreso mensual promedio de las personas del área urbana y compararlo entre hombres y mujeres, podemos hacerlo usando un pipe.

```
ing_sex <- datos2 %>%
  # 1. Nos quedamos con las personas ocupadas
  filter(area == "1") %>%
  # 2. Agrupamos por sexo
  group_by(sexo) %>%
  summarise(
    # Número de personas ocupadas en cada grupo
    n = n(),
    # Ingreso mensual promedio
    ingreso_promedio = mean(ingreso, na.rm = TRUE)
  )
```

Pipes

```
ing_sex
```

sexo	n	ingreso_promedio
Hombre	11124	373800.8
Mujer	12106	353180.3

Este resultado lo hicimos filtrando únicamente a las personas del área urbana, luego agrupamos la base por sexo, y finalmente calculamos el promedio del ingreso mensual dentro de cada grupo. Todo este proceso se puede hacer en una sola cadena de pasos, sin necesidad de crear muchas variables intermedias.

Programar

Pipes

- Ejemplo: Calculamos ingreso per cápita por hogar y mostramos los 5 hogares con mayor ingreso per cápita —todo en una sola cadena.

```
top5_pc <- datos2 %>%
   # 1) agrupamos por hogar
  group_by(id_hogar) %>%
  # 2) sumamos ingreso del hogar
  summarise(ingreso_hogar = sum(ingreso, na.rm = TRUE),
            # 3) contamos miembros
            miembros      = n(),
            .groups = "drop") %>%
  # 4) ingreso per cápita
  mutate(ingreso_pc = ingreso_hogar / miembros) %>%
  # 5) ordenamos
  arrange(desc(ingreso_pc)) %>%
  # 6) top 5
  slice_head(n = 5)
```

Pipes

```
top5_pc
```

id_hogar	ingreso_hogar	miembros	ingreso_pc
59266	101527539	4	25381885
58397	48097222	4	12024306
57530	34099304	3	11366435
55637	7664305	1	7664305
30726	21636250	3	7212083

Iteración

Iterar es repetir una misma operación sobre un conjunto de elementos (archivos, columnas, grupos, filas) sin copiar/pegar código. En R puedes iterar con bucles como for, while,entre otros.

¿Cuándo iterar?

- Repetir el mismo cálculo por sexo, región o educación.
- Aplicar una función a muchas columnas.
- Leer/limpiar varios archivos.
- Generar y guardar un gráfico por cada grupo.

Iteración - for

Sirve cuando ya sabemos cuántas veces repetir.

- Ejemplo: Para cada nivel de pobreza, calcular el ingreso promedio.

```
pobre <- unique(datos2$pobreza)
resultado <- data.frame(pobreza = character(), promedio = numeric())

for (p in pobre) {
  promedio <- mean(datos2$ingreso[datos2$pobreza == p],
                   na.rm = TRUE)
  resultado <- rbind(resultado,
                     data.frame(pobreza = p, promedio = promedio))
}
```

Iteración - for

```
resultado
```

pobreza	promedio
3	372902.38
2	157202.16
1	35525.56

Esto hace lo mismo que si calculáramos el promedio para cada nivel de pobreza, pero automáticamente.

Programar

Iteración - for

- Ejemplo: Para cada combinación de nivel de pobreza y sexo, calcular el promedio y la desviación estándar.

```
pobrezas <- unique(datos2$pobreza)
sexos <- unique(datos2$sexo)

#Resultado vacío
resultado2 <- data.frame(pobreza=character(), sexo=character(),
promedio=numeric(), sd=numeric())

for (p in pobrezas) {
for (s in sexos) {
x <- datos2$ingreso[datos2$pobreza == p & datos2$sexo == s]
resultado2 <- rbind(resultado2, data.frame(
pobreza = p, sexo = s,promedio= mean(x, na.rm = TRUE),
sd = sd(x, na.rm = TRUE)))}}
```

Iteración - for

```
resultado2
```

pobreza	sexo	promedio	sd
3	Hombre	382279.15	564809.52
3	Mujer	364009.70	495210.72
2	Hombre	168313.48	665892.89
2	Mujer	147682.34	252717.41
1	Hombre	35904.14	32002.01
1	Mujer	35193.11	31716.84

Esto hace lo mismo que calcular esas medidas para cada grupo pobreza $\times$ sexo, pero automáticamente.

Iteración - while

El bucle while sirve para repetir algo mientras se cumpla una condición. Es como decir: "Sigue haciendo esto mientras algo siga siendo verdadero. Cuando deje de serlo, párate."

A diferencia de for, no sabemos cuántas veces se va a repetir. Se detiene cuando la condición ya no se cumple.

Iteración - while

- Ejemplo: Supongamos que queremos encontrar la primera mujer que tenga un ingreso mayor a 15 millones.

```
i <- 1

while (datos2$ingreso[i] <= 15000000 | datos2$sexo[i] != "Mujer") {
  i <- i + 1   # Avanzar a la siguiente persona
}

datos2[i,1:8 ]
```

	id_hogar	id	edad	sexo	etnia	area	ingreso	pobreza
24734	59266	2	45	Mujer	0	1	25383308	3

Iteración - while

- Ejemplo: Encontrar el primer Hombre del área rural con ingreso entre 2 y 4 millones.

```
i <- 1

while (
datos2$sexo[i] != "Hombre" |
datos2$area[i] != "2" |
is.na(datos2$ingreso[i]) |
datos2$ingreso[i] < 2000000 |
datos2$ingreso[i] > 4000000
) {
i <- i + 1 # Avanzar a la siguiente persona
}
```

Iteración - while

```
datos2[i,1:8 ]
```

	id_hogar	id	edad	sexo	etnia	area	ingreso	pobreza
5099	11916	1	52	Hombre	0	2	2565478	3

Programar

Crear funciones

Una función es un bloque de código que:

- Recibe valores de entrada (argumentos).
- Ejecuta instrucciones.
- Devuelve un resultado.

Sirve para no repetir código, ahorrar tiempo y mantener el análisis ordenado.

```
nombre_funcion <- function(argumento1, argumento2) {
  # código que hace algo
  resultado <- argumento1 + argumento2  # ejemplo
  return(resultado)  # opcional, pero recomendado
}
```

Programar

Crear funciones

- Ejemplo: Función para calcular el ingreso per cápita por hogar.

```
ingreso_pc_por_hogar <- function(base) {
base %>%
group_by(id_hogar) %>%
summarise(
n_miembros = n(),
ingreso_hogar = sum(ingreso, na.rm = TRUE),
.groups = "drop"
) %>%
mutate(ingreso_pc = ingreso_hogar / n_miembros)
}
```

Programar

Crear funciones

```
# Usarla
hogares <- ingreso_pc_por_hogar(datos2)
head(hogares,5)
```

id_hogar	n_miembros	ingreso_hogar	ingreso_pc
262	4	2156611	539152.7
265	5	3541389	708277.8
277	6	1439861	239976.8
288	7	945416	135059.4
289	10	1561527	156152.7

Programar

Crear funciones

- Ejemplo: Función para clasificar hogares como pobres según una línea de pobreza usando ingreso per cápita.

```
pobreza_por_hogar <- function(base, linea_pobreza) {
base %>%
group_by(id_hogar) %>%
summarise(
n_miembros = n(),
ingreso_hogar= sum(ingreso, na.rm = TRUE),
.groups = "drop"
) %>%
mutate(
ingreso_pc = ingreso_hogar / n_miembros,
pobre = ifelse(ingreso_pc < linea_pobreza, "Pobre", "No pobre")
)
}
```

Programar

Crear funciones
Usarla (ejemplo: línea de pobreza mensual de 480.000)

```
# Usarla
hogares_lp <- pobreza_por_hogar(datos2, linea_pobreza = 480000)
head(hogares_lp, 5)
```

id_hogar	n_miembros	ingreso_hogar	ingreso_pc	pobre
262	4	2156611	539152.7	No pobre
265	5	3541389	708277.8	No pobre
277	6	1439861	239976.8	Pobre
288	7	945416	135059.4	Pobre
289	10	1561527	156152.7	Pobre