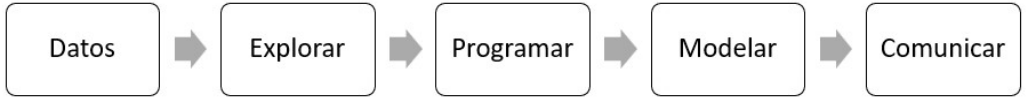
Análisis de encuestas de hogares con R

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 leer los datos y traerlos a R. Una vez cargados, los datos se guardan en un objeto para poder explorarlos y usarlos.

- Ejemplo: cargar un archivo RDS (.rds)

Carga e importación de base de datos

```r
# Importar la base (ejemplo)
datos <- readRDS(
  "../Data/base_personas.rds"
) # readdRDS es una funcion base de R
# Ver las primeras filas
head(datos,5)
```

```
# A tibble: 5 x 14
# Groups:   id_hogar [2]
  id_hogar id_pers parentesco  edad sexo   etnia area  ingreso pobreza
     <dbl>   <dbl> <chr>      <dbl> <chr>  <chr> <chr>   <dbl> <chr>
1      262       1 1             51 Hombre 0     1     542000  3
2      262       1 1             51 Hombre 0     1     536305. 3
3      262       2 2             46 Mujer  0     1     542000  3
4      262       2 2             46 Mujer  0     1     536305. 3
5      265       1 1             26 Mujer  0     1     710556. 3
# i 4 more variables: estrato <dbl>, fep <dbl>, anoest <dbl>,
#   niveduc_ee <hvn_lbll>
```

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
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.

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

Usando funciones en R

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

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

Por ejemplo,

```
seq(1, 10)      # crea los números del 1 al 10
```

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

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_pers", "edad","sexo", "etnia","area",
                           "ingreso", "pobreza", "anoest") %>% rename(
 id = id_pers
)

head(datos2,5)
```

```
# A tibble: 5 x 9
# Groups:   id_hogar [2]
  id_hogar    id  edad sexo   etnia area  ingreso pobreza anoest
     <dbl> <dbl> <dbl> <chr>  <chr> <chr>   <dbl> <chr>    <dbl>
1      262     1    51 Hombre 0     1      542000  3           18
2      262     1    51 Hombre 0     1      536305. 3           18
3      262     2    46 Mujer  0     1      542000  3           12
4      262     2    46 Mujer  0     1      536305. 3           12
5      265     1    26 Mujer  0     1      710556. 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,5)
```

```
# A tibble: 5 x 9
# Groups:   id_hogar [2]
  id_hogar    id  edad sexo   etnia area  ingreso pobreza anoest
     <dbl> <dbl> <dbl> <chr>  <chr> <chr>   <dbl> <chr>    <dbl>
1      262     1    51 Hombre 0     1     542000  3           18
2      262     1    51 Hombre 0     1     536305. 3           18
3      262     2    46 Mujer  0     1     542000  3           12
4      262     2    46 Mujer  0     1     536305. 3           12
5      265     1    26 Mujer  0     1     710556. 3           17
```

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)
```

```
# A tibble: 5 x 9
# Groups:   id_hogar [2]
  id_hogar    id  edad sexo   etnia area    ingreso pobreza anoest
     <dbl> <dbl> <dbl> <chr>  <chr> <chr>     <dbl> <chr>    <dbl>
1    59266     1    52 Hombre 0     1     25383308. 3           20
2    59266     2    45 Mujer  0     1     25383308. 3           20
3    59266     3    13 Hombre 0     1     25383308. 3            7
4    59266     1    52 Hombre 0     1     25377614. 2           20
5    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 funciones como mutate() y case_when() del paquete dplyr, que nos permiten transformar, combinar o recodificar variables sin alterar los datos originales.

Explorar: Transformación de datos

Crear variables derivadas

```
# 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 derivadas

```
# 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
```

```
# A tibble: 2 x 3
  sexo       n ingreso_prom
  <chr>  <int>        <dbl>
1 Hombre 14002      351264.
2 Mujer  14969      330336.
```

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.

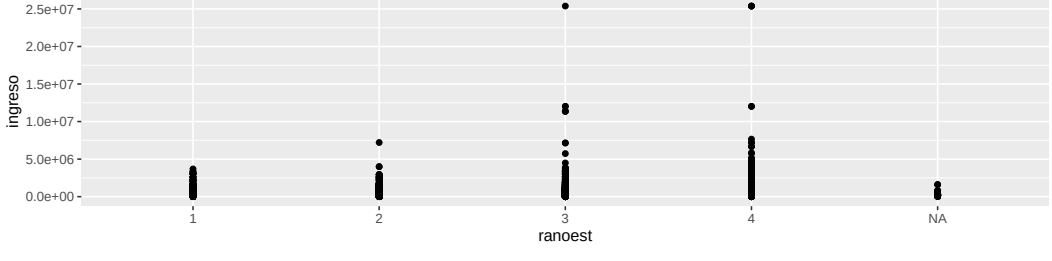
```
head(datos2,5)
```

```
# A tibble: 5 x 11
# Groups:   id_hogar [2]
  id_hogar    id  edad sexo   etnia area  ingreso pobreza anoest grupo_edad
     <dbl> <dbl> <dbl> <chr>  <chr> <chr>   <dbl> <chr>    <dbl> <chr>
1      262     1    51 Hombre 0     1     542000  3           18 Adultez
2      262     1    51 Hombre 0     1     536305. 3           18 Adultez
3      262     2    46 Mujer  0     1     542000  3           12 Adultez
4      262     2    46 Mujer  0     1     536305. 3           12 Adultez
5      265     1    26 Mujer  0     1     710556. 3           17 Adultez
# i 1 more variable: ranoest <chr>
```

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?

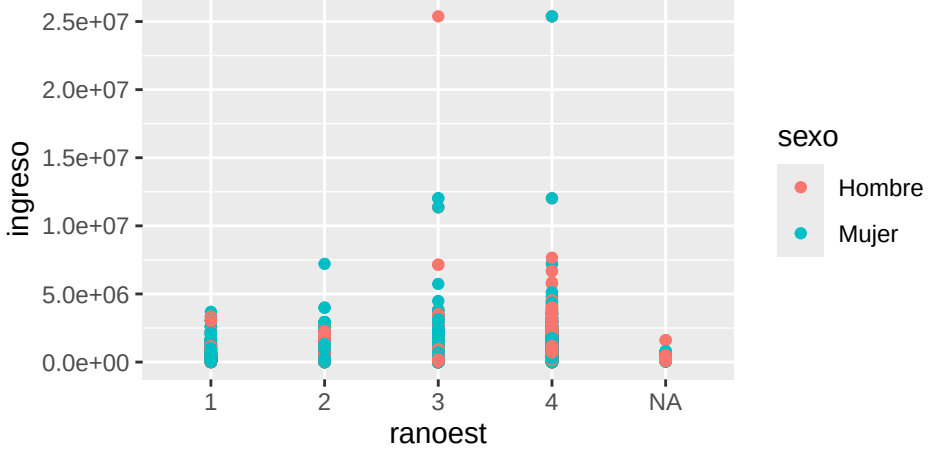
```
ggplot(data = datos2) +
  geom_point(mapping = aes(x = ranoest, y = ingreso))
```



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))
```



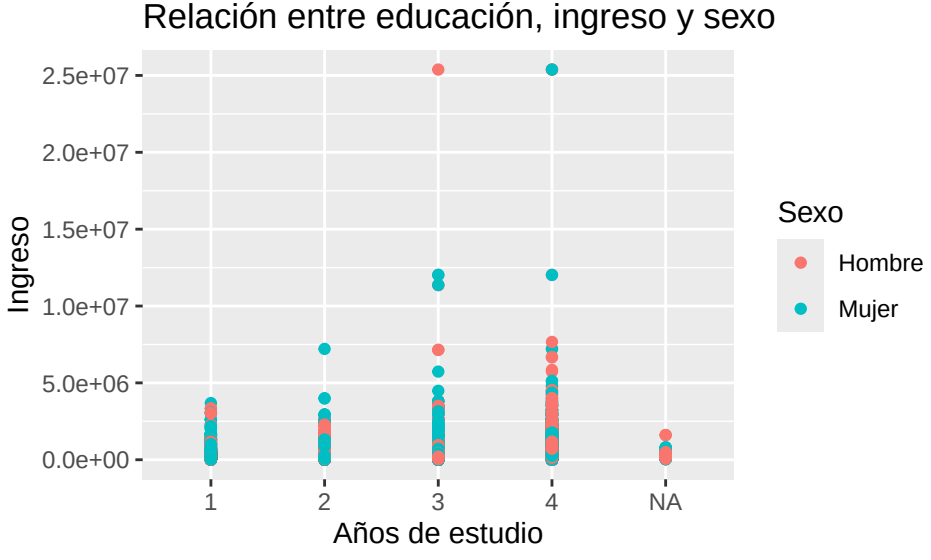
Explorar: Visualización de datos

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

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.
- Boxplot: geom_boxplot() – Comparar distribuciones entre categorías.
- Histograma: geom_histogram() – Distribución de una variable numérica.
- 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
```

```
# A tibble: 2 x 3
  sexo       n ingreso_promedio
  <chr>  <int>            <dbl>
1 Hombre 11124          373801.
2 Mujer  12106          353180.
```

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.

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 = promedi
}
```

Iteración - for

```
resultado
```

```
  pobreza  promedio
1       3 372902.38
2       2 157202.16
3       1  35525.56
```

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

Programar

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, ]
```

```
# A tibble: 1 x 11
# Groups:   id_hogar [1]
  id_hogar    id  edad sexo  etnia area    ingreso pobreza anoest grupo_eda
     <dbl> <dbl> <dbl> <chr> <chr> <chr>     <dbl> <chr>    <dbl> <chr>
1    59266     2    45 Mujer 0     1     25383308. 3           20 Adultez
# i 1 more variable: ranoest <chr>
```

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)
```

```
# A tibble: 5 x 4
  id_hogar n_miembros ingreso_hogar ingreso_pc
     <dbl>      <int>         <dbl>      <dbl>
1      262          4      2156611.    539153.
2      265          5      3541389.    708278.
3      277          6      1439861.    239977.
4      288          7       945416.    135059.
5      289         10      1561527.    156153.
```

Modelar

Utilizar técnicas estadísticas o matemáticas para responder preguntas específicas o hacer predicciones.

Comunicar

Presentar los resultados de forma clara mediante informes, tablas,gráficos o presentaciones.