Introducción a R

Elementos básicos, vectores, matrices, dataframes, tibble

Nicolás Sidicaro
Marzo 2025

Contenidos

0.1. Código abierto vs código cerrado

Definiciones

Código abierto (Open Source):

- Software cuyo código fuente está disponible públicamente
- Cualquier persona puede ver, modificar y distribuir el código
- Desarrollado colaborativamente por comunidades de programadores
- Ejemplos: R, Python, Linux, Firefox

Código cerrado (Closed Source):

- Software cuyo código fuente no es accesible públicamente
- Solo los desarrolladores originales pueden modificarlo
- Generalmente comercial y con licencias restrictivas
- Ejemplos: SPSS, SAS, Microsoft Office

Diferencias principales

Aspectos clave que distinguen código abierto y cerrado

Aspecto	Código abierto	Código cerrado
Acceso al código	Público	Privado
Costo	Generalmente gratuito	Generalmente de pago
Desarrollo	Comunidad colaborativa	Equipo interno
Actualizaciones	Frecuentes, impulsadas por la comunidad	Planificadas por la empresa
Soporte	Comunidad, foros, documentación	Oficial, atención al cliente
Seguridad	Revisión por muchos ojos	Revisión por equipo interno

Ventajas y desventajas

Código abierto:

- **Ventajas:**
  - Costo reducido o nulo
  - Transparencia y confiabilidad
  - Personalización flexible
  - Comunidad activa de desarrollo
- **Desventajas:**
  - Puede tener documentación inconsistente
  - Soporte no garantizado
  - Interfaces a veces menos pulidas

Ventajas y desventajas

Código cerrado:

- **Ventajas:**
  - Soporte técnico formal
  - Documentación completa y estructurada
  - Interfaces más pulidas y amigables
- **Desventajas:**
  - Costos de licencias elevados
  - Dependencia del proveedor
  - Menor flexibilidad para personalización

Impacto en la ciencia de datos

La tendencia hacia código abierto:

- R y Python dominan el panorama actual
- Aceleración de la innovación mediante colaboración
- Democratización del acceso a herramientas avanzadas
- Mayor reproducibilidad en investigación
- Comunidades especializadas (ej: Tidyverse, PyData)

Herramientas comerciales:

- Siguen siendo relevantes en entornos empresariales
- Ofrecen interfaces más accesibles para no programadores
- Integración con ecosistemas corporativos
- Mayor soporte para implementaciones a gran escala

0.2. Librerías en R

¿Qué son las librerías en R?

- Colecciones de funciones, datos y documentación
- Extienden las funcionalidades básicas de R
- Desarrolladas por la comunidad global
- Más de 19,000 paquetes disponibles en CRAN
- Permiten especializar R para diferentes campos y necesidades

Terminología:

- **Paquete**: unidad de código distribuible
- **Librería**: colección de paquetes
- **Función**: bloque de código que realiza una tarea específica
- **Dependencia**: paquete requerido por otro paquete

Ecosistema de paquetes en R

Repositorios principales:

- **CRAN** (Comprehensive R Archive Network)
  - Repositorio oficial de paquetes
  - Control de calidad estricto
  - Más de 19,000 paquetes disponibles
- **Bioconductor**
  - Especializado en bioinformática y genómica
  - Más de 2,000 paquetes específicos
- **GitHub**
  - Paquetes en desarrollo
  - Versiones experimentales
  - Acceso a las últimas funcionalidades

Gestión de paquetes

Instalación:

```r
# Desde CRAN
install.packages("dplyr")

# Desde GitHub
# install.packages("devtools")
devtools::install_github("tidyverse/dplyr")

# Desde Bioconductor
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install("GenomicRanges")
```

Carga:

```r
library(dplyr)
# o
require(dplyr)
```

Exploración:

```
# Ver funciones disponibles
ls("package:dplyr")

# Acceder a la documentación
help(package = "dplyr")
?select
```

Librerías fundamentales

Tidyverse:

- Colección coherente de paquetes para análisis de datos
- Desarrollada por Hadley Wickham y equipo
- Incluye: ggplot2, dplyr, tidyr, readr, purrr, tibble
- Filosofía común de diseño y sintaxis consistente

Importación de datos:

- readr, readxl, haven, jsonlite, xml2, rvest, DBI

Manipulación de datos:

- dplyr, tidyr, stringr, lubridate, forcats

Librerías fundamentales

Análisis estadístico:

- stats (base), lme4, car, caret, glmnet, survival

Machine learning:

- randomForest, xgboost, e1071, nnet, neuralnet, caret, mlr

Visualización:

- ggplot2, plotly, lattice, ggvis, shiny, leaflet

Reportes y presentaciones:

- rmarkdown, knitr, xaringan, flexdashboard

Tendencia: Metapaquetes

Colecciones de paquetes con filosofía coherente:

- **Tidyverse**: Análisis de datos general
- **Tidymodels**: Modelado estadístico y ML
- **Shiny**: Aplicaciones web interactivas
- **Rmarkdown**: Informes reproducibles
- **Biocondutor**: Bioinformática

Beneficios:

- Ecosistemas coherentes con filosofías de diseño consistentes
- Mejor interoperabilidad entre paquetes
- Instalación simplificada
- Documentación y soporte integrados

0.3. Google Colab

¿Qué es Google Colab?

Google Colaboratory:

- Entorno de notebook basado en la nube
- Basado en Jupyter Notebook
- Ejecución en servidores de Google
- Acceso gratuito a GPUs y TPUs
- Integración con Google Drive
- Principalmente orientado a Python
- Posibilidad de usar R mediante magics

Características principales

Funcionalidades clave:

- **Zero configuration**: No requiere instalación
- **Computación en la nube**: Recursos computacionales potentes
- **Colaboración en tiempo real**: Como Google Docs
- **Integración con ecosistema Google**: Drive, Sheets
- **Recursos gratuitos**: CPU, GPU y TPU limitados
- **Persistencia limitada**: Las sesiones expiran
- **Entorno pre-instalado**: Bibliotecas científicas y ML

Estructura:

- Notebooks con celdas de código y texto (Markdown)
- Ejecución secuencial o selectiva de celdas
- Soporte para gráficos, widgets y HTML

Ventajas y limitaciones

Ventajas:

- No requiere configuración local
- Acceso a hardware potente (GPU/TPU)
- Ideal para aprendizaje y experimentación
- Facilita compartir y colaborar
- Integración con servicios Google
- Gratuito para uso básico

Limitaciones:

- Sesiones con tiempo limitado (12 horas)
- Desconexión por inactividad
- Recursos computacionales restringidos
- Menos privacidad (datos en servidores Google)
- Menos orientado a R que a Python
- Conectividad a internet obligatoria

Comparación con otros entornos

Entornos para data science:

Característica	Google Colab	RStudio	Jupyter
Instalación	No requiere	Local o servidor	Local o servidor
Lenguajes principales	Python (R posible)	R	Múltiples
Acceso a hardware	GPU/TPU gratuito	Local	Local
Colaboración	En tiempo real	Limitada	Limitada
Persistencia	Temporal	Permanente	Permanente
Curva de aprendizaje	Baja	Media	Media
Costo	Gratuito (versión pro disponible)	Gratuito/Pago	Gratuito

1. Elementos básicos de R

Operaciones aritméticas básicas

```
# Suma
5 + 3
```

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

```
# Resta
10 - 4
```

```
## [1] 6
```

```
# Multiplicación
7 * 2
```

```
## [1] 14
```

```
# División
15 / 3
```

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

Operaciones aritméticas básicas

```
# Exponente
2^3
```

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

```
# Módulo (resto de la división)
17 %% 5
```

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

```
# División entera
17 %/% 5
```

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

Variables y asignación

En R, guardamos valores en variables usando el operador de asignación `←` (aunque también se puede usar `=`).

```
# Asignando valores a variables
x ← 10
y ← 5

# Operaciones con variables
z ← x + y
z
```

```
## [1] 15
```

```
# También podemos usar print()
print(z)
```

```
## [1] 15
```

> **Nota:** Por convención en R, se prefiere usar `←` para asignaciones.

2. Tipos de datos

Tipos de datos básicos

```
# Numérico (numeric)
edad ← 25
estatura ← 1.75
class(edad)
```

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

```
class(estatura)
```

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

```
# Carácter (character)
nombre ← "Ana"
apellido ← 'García'  # También se pueden usar comillas simples
class(nombre)
```

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

Tipos de datos básicos

```
# Lógico (logical)
es_estudiante ← TRUE  # También puede ser FALSE
aprobado ← T  # Abreviatura de TRUE (no recomendada)
class(es_estudiante)
```

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

```
# Factor (para variables categóricas)
genero ← factor(c("Masculino", "Femenino", "No binario"))
class(genero)
```

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

> **Nota:** ¿De qué sirve el `factor`? Nos va a permitir ordenar texto de otras maneras. En caso de usar characters, el orden va a ser alfabético y no siempre queremos eso.

3. Vectores

Creación de vectores

Los vectores son la estructura de datos más básica en R. Son colecciones unidimensionales de elementos del mismo tipo.

```
# Crear un vector con la función c() (combine)
numeros ← c(1, 5, 3, 8, 4)
numeros
```

```
## [1] 1 5 3 8 4
```

```
# Vector de caracteres
frutas ← c("manzana", "banana", "naranja", "pera")
frutas
```

```
## [1] "manzana" "banana"  "naranja" "pera"
```

```
# Vector lógico
logicos ← c(TRUE, FALSE, TRUE, TRUE)
logicos
```

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

Creación de vectores

Los vectores son la estructura de datos más básica en R. Son colecciones unidimensionales de elementos del mismo tipo.

```
# Secuencias numéricas
secuencia1 ← 1:10  # Del 1 al 10
secuencia1
```

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

```
secuencia2 ← seq(0, 1, by=0.1)  # De 0 a 1 en incrementos de 0.1
secuencia2
```

```
##  [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

```
# Repetir valores
repetidos ← rep(c(1, 2), times=3)  # Resultado: 1 2 1 2 1 2
repetidos
```

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

Creación de vectores

¿Qué sucede si mezclamos tipos de valores en un vector? Van a quedar en el tipo que R considere más adecuado. Por ejemplo, si mezclamos números con texto, todo pasará a texto.

```
# Mezcla 1
mezcla1 ← c(1,2,3,'cuatro','cinco',6)
mezcla1
```

```
## [1] "1"      "2"      "3"      "cuatro" "cinco"  "6"
```

```
# Mezcla 2
mezcla2 ← c(1,2,3.5,TRUE,FALSE,6)
mezcla2
```

```
## [1] 1.0 2.0 3.5 1.0 0.0 6.0
```

Operaciones con vectores

```r
# Vectores para ejemplos
a ← c(1, 2, 3, 4, 5)
b ← c(6, 7, 8, 9, 10)

# Operaciones aritméticas (elemento por elemento)
a + b  # Suma
```

```
## [1]  7  9 11 13 15
```

```r
a * 2  # Multiplicación por escalar
```

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

```r
a * b  # Multiplicación elemento a elemento
```

```
## [1]  6 14 24 36 50
```

Operaciones con vectores

```
sum(a)      # Suma de todos los elementos
```

```
## [1] 15
```

```
mean(a)     # Media
```

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

```
median(a)   # Mediana
```

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

```
# ¿cuánto sería min(a)? ¿y max(a)?
range(a)    # Rango (mínimo y máximo)
```

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

```
sd(a)       # Desviación estándar
```

```
## [1] 1.581139
```

Indexación de vectores

```r
# Vector para ejemplos
letras ← c("a", "b", "c", "d", "e")

# Acceso a elementos (indexación)
letras[3]        # Tercer elemento
```

```
## [1] "c"
```

```r
letras[c(1, 3)]  # Primer y tercer elemento
```

```
## [1] "a" "c"
```

```r
# Indexación con condiciones
numeros ← c(2, 5, 8, 3, 7)
numeros[numeros > 5]  # Elementos mayores que 5
```

```
## [1] 8 7
```

Indexación de vectores

```
# Indexación negativa (excluye elementos)
numeros[-2]  # Todos excepto el segundo elemento
```

```
## [1] 2 8 3 7
```

```
# Modificar elementos específicos
numeros[2] ← 10
numeros
```

```
## [1]  2 10  8  3  7
```

Indexación de vectores

¿De qué nos sirven entonces los `[]`?

1. Escoger un valor dentro de un vector según su posición (ojo con Python que la primera posición es la `0`)
2. Para filtrar elementos según una condición
3. Para reemplazar elementos que estén en un lugar específico

> **Nota:** En breve vamos a ver que sirve también para otras cosas. Muchos elementos y funciones en R tienen un uso para determinado tipo de dato y otro uso para otros.

Vectorización en R

Una de las fortalezas de R es la vectorización, que permite realizar operaciones en todos los elementos de un vector sin necesidad de loops explícitos.

```r
# Ejemplo de vectorización
x ← 1:5
x
```

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

```r
# Calcular el cuadrado de cada número (sin loops)
x_squared ← x^2
x_squared
```

```
## [1]  1  4  9 16 25
```

> **Importante**: La vectorización en R hace el código más legible y eficiente.

Vectorización en R

Una de las fortalezas de R es la vectorización, que permite realizar operaciones en todos los elementos de un vector sin necesidad de loops explícitos.

```
# Aplicar una función a cada elemento
y ← sqrt(x)  # Raíz cuadrada de cada elemento
y
```

```
## [1] 1.000000 1.414214 1.732051 2.000000 2.236068
```

```
# Operaciones condicionales vectorizadas
ifelse(x > 3, "Grande", "Pequeño")
```

```
## [1] "Pequeño" "Pequeño" "Pequeño" "Grande"  "Grande"
```

> **Importante**: La vectorización en R hace el código más legible y eficiente.

4. Matrices

Creación de matrices

Las matrices son estructuras de datos bidimensionales (filas y columnas) que contienen elementos del mismo tipo.

```
# Crear una matriz con la función matrix()
mat1 ← matrix(1:6, nrow=2, ncol=3)
mat1
```

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

```
# Por defecto, la matriz se rellena por columnas
# Podemos cambiarlo con el parámetro byrow
mat2 ← matrix(1:6, nrow=2, ncol=3, byrow=TRUE)
mat2
```

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

Creación de matrices

Las matrices son estructuras de datos bidimensionales (filas y columnas) que contienen elementos del mismo tipo.

```
# Crear matriz a partir de vectores
fila1 ← c(1, 2, 3)
fila2 ← c(4, 5, 6)
mat3 ← rbind(fila1, fila2)  # Combinar por filas
mat3
```

```
##       [,1] [,2] [,3]
## fila1    1    2    3
## fila2    4    5    6
```

```
columna1 ← c(1, 4)
columna2 ← c(2, 5)
columna3 ← c(3, 6)
mat4 ← cbind(columna1, columna2, columna3)  # Combinar por columnas
mat4
```

```
##      columna1 columna2 columna3
## [1,]        1        2        3
## [2,]        4        5        6
```

Operaciones con matrices

```
# Dimensiones de una matriz
dim(mat1)  # Resultado: 2 3
```

```
## [1] 2 3
```

```
nrow(mat1)  # Número de filas
```

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

```
ncol(mat1)  # Número de columnas
```

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

Operaciones con matrices

```
# Acceso a elementos
mat1[1, 2]  # Elemento en la fila 1, columna 2
```

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

```
mat1[1, ]   # Primera fila completa
```

```
## [1] 1 3 5
```

```
mat1[, 3]   # Tercera columna completa
```

```
## [1] 5 6
```

Operaciones con matrices

```r
# Operaciones aritméticas
A ← matrix(1:4, nrow=2)
B ← matrix(5:8, nrow=2)
A
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

```r
B
```

```
##      [,1] [,2]
## [1,]    5    7
## [2,]    6    8
```

Operaciones con matrices

```
A + B        # Suma de matrices
```

```
##      [,1] [,2]
## [1,]    6   10
## [2,]    8   12
```

```
A * B        # Multiplicación elemento a elemento
```

```
##      [,1] [,2]
## [1,]    5   21
## [2,]   12   32
```

```
A %*% B      # Multiplicación matricial
```

```
##      [,1] [,2]
## [1,]   23   31
## [2,]   34   46
```

```
solve(A)     # Matriz inversa
```

```
##      [,1] [,2]
## [1,]   -2  1.5
## [2,]    1 -0.5
```

Más operaciones con matrices

```
# Matriz para ejemplos
M ← matrix(1:9, nrow=3)
M
```

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

```
# Transpuesta
t(M)         # Transpuesta
```

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

```
# Diagonal principal
diag(M)      # Diagonal principal
```

```
## [1] 1 5 9
```

Más operaciones con matrices

```r
# Crear matriz diagonal
diag(3)      # Matriz identidad 3×3
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
```

```r
# Nombrar filas y columnas
rownames(M) ← c("Fila1", "Fila2", "Fila3")
colnames(M) ← c("Col1", "Col2", "Col3")
M
```

```
##       Col1 Col2 Col3
## Fila1    1    4    7
## Fila2    2    5    8
## Fila3    3    6    9
```

5. Dataframes

Creación de dataframes

Los dataframes son estructuras de datos tabulares que pueden contener diferentes tipos de datos en cada columna, similares a las hojas de cálculo.

```r
# Crear un dataframe a partir de vectores
nombre ← c("Ana", "Carlos", "María", "Juan")
edad ← c(25, 30, 22, 28)
estudiante ← c(TRUE, FALSE, TRUE, FALSE)

personas ← data.frame(nombre, edad, estudiante)
personas
```

```
##   nombre edad estudiante
## 1    Ana   25       TRUE
## 2 Carlos   30      FALSE
## 3  María   22       TRUE
## 4   Juan   28      FALSE
```

Creación de dataframes

Los dataframes son estructuras de datos tabulares que pueden contener diferentes tipos de datos en cada columna, similares a las hojas de cálculo.

```
# Nombres de columnas personalizados
personas2 ← data.frame(
  nombre = c("Ana", "Carlos", "María", "Juan"),
  edad = c(25, 30, 22, 28),
  estudiante = c(TRUE, FALSE, TRUE, FALSE)
)
personas2
```

```
##   nombre edad estudiante
## 1    Ana   25       TRUE
## 2 Carlos   30      FALSE
## 3  María   22       TRUE
## 4   Juan   28      FALSE
```

Explorando dataframes

```r
# Dimensiones
dim(personas)      # Dimensiones
```

```
## [1] 4 3
```

```r
nrow(personas)     # Número de filas
```

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

```r
ncol(personas)     # Número de columnas
```

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

Explorando dataframes

```
# Estructura y resumen
names(personas)    # Nombres de las columnas
```

```
## [1] "nombre"     "edad"       "estudiante"
```

```
str(personas)      # Estructura del dataframe
```

```
## 'data.frame':    4 obs. of  3 variables:
##  $ nombre    : chr  "Ana" "Carlos" "María" "Juan"
##  $ edad      : num  25 30 22 28
##  $ estudiante: logi  TRUE FALSE TRUE FALSE
```

```
head(personas, 2)  # Primeras 2 filas
```

```
##   nombre edad estudiante
## 1    Ana   25       TRUE
## 2 Carlos   30      FALSE
```

Explorando dataframes

```
# Estructura y resumen
summary(personas)  # Resumen estadístico
```

```
##     nombre               edad       estudiante
##  Length:4           Min.   :22.00   Mode :logical
##  Class :character   1st Qu.:24.25   FALSE:2
##  Mode  :character   Median :26.50   TRUE :2
##                     Mean   :26.25
##                     3rd Qu.:28.50
##                     Max.   :30.00
```

Explorando dataframes

Acá vemos otro uso del `[]` cuando lo aplicamos a los data.frame.

```
# Acceso a elementos
personas$nombre        # Acceder a una columna completa
```

```
## [1] "Ana"    "Carlos" "María"  "Juan"
```

```
personas[2, ]          # Segunda fila
```

```
##   nombre edad estudiante
## 2 Carlos   30      FALSE
```

```
personas[, "edad"]     # Columna "edad"
```

```
## [1] 25 30 22 28
```

```
personas[2, "nombre"]  # Valor específico (nombre de la segunda persona)
```

```
## [1] "Carlos"
```

> **Nota:** También podríamos hacer `personas[2,1]` para obtener el último resultado. ¿Por qué?

Manipulación de dataframes

```r
# Filtrado de datos
personas[personas$edad > 25, ]  # Personas mayores de 25 años
```

```
##   nombre edad estudiante
## 2 Carlos   30      FALSE
## 4   Juan   28      FALSE
```

```r
subset(personas, estudiante == TRUE)  # Sólo estudiantes
```

```
##   nombre edad estudiante
## 1    Ana   25       TRUE
## 3  María   22       TRUE
```

```r
# Añadir nuevas columnas
personas$ciudad <- c("Madrid", "Barcelona", "Sevilla", "Valencia")
personas
```

```
##   nombre edad estudiante    ciudad
## 1    Ana   25       TRUE    Madrid
## 2 Carlos   30      FALSE Barcelona
## 3  María   22       TRUE   Sevilla
## 4   Juan   28      FALSE  Valencia
```

Manipulación de dataframes

```r
# Ordenar dataframe
personas_ordenadas ← personas[order(personas$edad), ]  # Ordenar por edad
personas_ordenadas
```

```
##   nombre edad estudiante    ciudad
## 3  María   22       TRUE   Sevilla
## 1    Ana   25       TRUE    Madrid
## 4   Juan   28      FALSE  Valencia
## 2 Carlos   30      FALSE Barcelona
```

```r
# Eliminar columnas
personas$ciudad ← NULL
head(personas, 2)
```

```
##   nombre edad estudiante
## 1    Ana   25       TRUE
## 2 Carlos   30      FALSE
```

6. Tibbles

Tibbles: Dataframes mejorados

Los tibbles son una versión moderna de los dataframes, incluidos en el paquete `tibble` que forma parte del ecosistema `tidyverse`.

Ventajas de los tibbles sobre dataframes:

- No convierten strings a factores automáticamente
- No cambian los nombres de las variables
- No hacen partial matching
- Tienen un método de impresión mejorado (muestran el tipo de datos)
- Son más estrictos (dan errores en lugar de advertencias)

```
# Primero cargar el paquete tibble
library(tibble)

# Crear un tibble
personas_tibble ← tibble(
  nombre = c("Ana", "Carlos", "María", "
  edad = c(25, 30, 22, 28),
  estudiante = c(TRUE, FALSE, TRUE, FALS
)

# Ver el tibble
print(personas_tibble)

# Convertir un dataframe a tibble
df_normal ← data.frame(
  x = 1:3,
  y = letters[1:3]
)
df_tibble ← as_tibble(df_normal)
```

7. Funciones

Funciones en R

Las funciones son bloques de código que realizan tareas específicas. R tiene muchas funciones incorporadas, y también podemos crear nuestras propias funciones.

```
# Funciones matemáticas
abs(-5)        # Valor absoluto
```

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

```
sqrt(16)       # Raíz cuadrada
```

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

```
log(10)        # Logaritmo natural
```

```
## [1] 2.302585
```

```
exp(1)         # Exponencial
```

```
## [1] 2.718282
```

Funciones en R

Las funciones son bloques de código que realizan tareas específicas. R tiene muchas funciones incorporadas, y también podemos crear nuestras propias funciones.

```
# Funciones estadísticas (ya vimos algunas)
mean(c(1, 2, 3, 4, 5))   # Media
```

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

```
median(c(1, 2, 3, 4, 5)) # Mediana
```

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

```
var(c(1, 2, 3, 4, 5))    # Varianza
```

```
## [1] 2.5
```

```
sd(c(1, 2, 3, 4, 5))     # Desviación estándar
```

```
## [1] 1.581139
```

Más funciones incorporadas

```r
# Funciones de ayuda
help(mean)        # Documentación de la función mean
example(mean)     # Ejemplos de uso de la función mean
```

```
##
## mean> x ← c(0:10, 50)
##
## mean> xm ← mean(x)
##
## mean> c(xm, mean(x, trim = 0.10))
## [1] 8.75 5.50
```

Más funciones incorporadas

```
# Funciones de tipo de datos
is.numeric(5)     # ¿Es numérico?
```

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

```
is.character("a") # ¿Es carácter?
```

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

```
as.numeric("5")   # Convertir a numérico
```

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

```
as.character(5)   # Convertir a carácter
```

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

Más funciones incorporadas

```r
# Funciones para vectores
length(c(1, 2, 3))      # Longitud del vector
```

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

```r
sort(c(3, 1, 2))        # Ordenar vector
```

```
## [1] 1 2 3
```

```r
rev(c(1, 2, 3))         # Invertir vector
```

```
## [1] 3 2 1
```

```r
unique(c(1, 2, 2, 3))   # Valores únicos
```

```
## [1] 1 2 3
```

Creación de funciones propias

Podemos crear nuestras propias funciones para tareas específicas:

```
# Definir una función simple
saludar ← function(nombre) {
  paste("¡Hola,", nombre, "!")
}

# Usar la función
saludar("Ana")
```

```
## [1] "¡Hola, Ana !"
```

> **Nota**: Pero esto lo vamos a ver otro día

Resumen: Lo que hemos aprendido

- **Elementos básicos**: Operaciones aritméticas, asignación de variables
- **Tipos de datos**: Numéricos, caracteres, lógicos, factores
- **Vectores**: Creación, indexación, operaciones, vectorización
- **Matrices**: Creación, manipulación, operaciones
- **Dataframes**: Creación, exploración, manipulación
- **Tibbles**: Ventajas y uso
- **Funciones**: Uso de funciones incorporadas y creación de funciones propias

Recursos adicionales

- R for Data Science (Hadley Wickham & Garrett Grolemund): https://r4ds.had.co.nz/
- Documentación oficial de R: https://www.r-project.org/
- RStudio Cheatsheets: https://www.rstudio.com/resources/cheatsheets/

¡Gracias!

Nos vemos en la próxima clase