PrimeroPasosEnR

Fundación SOL

martes, 10 dic , 2024

Tabla de contenidos

Inicio

Una breve guía

Este documento te permitirá dar los primeros pasos en R. La idea es que puedas comprender los aspectos generales de la historia, estructura de datos y objetos básicos en R.

La guía tiene seis capítulos orientados a conocer aquellos códigos que te servirán para *soltar la mano* y no perder la confianza en lo útil que puede llegar a ser aprender sobre pensamiento crítico, programación y datos. El primer capítulo se llama **Para comenzar** está orientado a que tengas las referencias si es que no has instalado el R en tu computador. También contiene orientaciones generales sobre conceptos como IDE y versión.

El segundo capítulo llamado **Primeros pasos** busca entregar una panorámica general sobre los tipos de datos en R. Te ayudará a comprender que que diferencia hay entre un número y una letra. Está más enfocado a la programación básica en R. Cuando estamos comenzando puede ser un capitulo al que volvamos frecuentemente, mientras avanzamos en la práctica de `R`.

El tercer capítulo **Historia** está pensado para que tengas una idea de los orígenes de R y algunos hitos importantes.

En el cuarto capítulo, **¿Qué es R Base?** se revisan conceptos generables de `R` algunos códigos de utilidad y la creación de funciones, algo muy importante en el uso de R.

Para comenzar a entrenar, en el quinto capítulo **Presentaciones con RMD** se proponen algunas utilidades para implementar el codigo y darle estilos a los trabajos escritos o presentaciones.

El capitulo siguiente (sexto capítulo) siempre está en construcción y pronto será publicado

Las referencias te permitirán conocer algunos sitios recomendables para tu autoaprendizaje.

1 Para comenzar

1.1 Instalar R

Si nunca has instalado un programa de computadora en tu vida, quizás sea recomendable que pidas apoyo para la instalación. Si has instalado un programa antes, hacerlo con R no debería ser más complejo.

Un fuente "oficial" para descargar `R` es CRAN (del inglés Comprehensive R Archive Network) algo así como el gran repositorio de R. Puedes acceder haciendo clik acá: CRAN.

En ese sitio puedes descargar R, para Linux, Windows o Mac. Luego puedes trabajar con tu IDE favorito. No sabes qué es un IDE, para más referencias puede visitar la clase abierta la clase abierta de nuestra Escuela de Datos.

2 Un IDE

En pocas palabras el IDE es donde escribes el codigo y trabajar la mayor parte del tiempo. Su aspecto positivo es que permite visualizar resultados y realizar tareas utilizando botones y una interfaz integradas, por lo que puede ser útil cuando estás comenzando.

Uno de los IDE más populares de R, es RStudio. El IDE es un programa, por lo que también lo debes descargar e instalar. Puedes hacerlo acá: POSIT - RSTUDIO. Posit es la empresa que creó RStudio.

También puede usar otros IDE's, o escribir en tu consola o en un libro de notas, el objetivo es que utilices lo que te sea más cómodo para trabajar.

3 Conocer sobre las versiones

Tanto R como RStudio, tienen versiones. Una versión es generalmente asociada a un número. dependiendo de la cantidad de cambios significativos para funcionar que tenga el programa en el tiempo, es probable que tenga muchas o pocas versiones.

Recordemos que R es parte de lo que podemos llamar *software libre* o *código abierto* es decir, puedes descargar el código desde CRAN y modificarlo. Eso ha sido positivo para que muchas personas puedan crear sus propios avances o programas con R.

Puede que tu interés no vaya a esos aspectos de R, de todas formas, saber la versión de tu programa es importante. Si instalaste R hace años y luego no lo seguiste usando y ahora lo quieres usar para seguir estos primeros pasos, puedes tomar en cuenta el código:

```
version
```

```
               _
platform       x86_64-w64-mingw32
arch           x86_64
os             mingw32
crt            ucrt
system         x86_64, mingw32
status
major          4
minor          4.1
year           2024
month          06
day            14
svn rev        86737
language       R
version.string R version 4.4.1 (2024-06-14 ucrt)
nickname       Race for Your Life
```

Soló con que escribas `version` en la consola y presiones `enter`, podrás tener información de la versión. Debes fijarte en `version.string` y podrás saber la versión.

4 Tipos de datos

#Primeros pasos

En R, hay varios tipos de datos básicos que puedes usar. Aquí hay una descripción de los tipos de datos más comunes.

4.1 Números

Los números pueden ser enteros o números decimales.

```
# Número entero
x <- 5
print(x)
```

```
[1] 5
```

```
# Número decimal
y <- 5.5
print(y)
```

```
[1] 5.5
```

4.2 Cadena de texto

Las cadenas de texto se representan usando comillas simples o dobles.

```
texto <- "Hola, mundo!"


print(texto)
```

```
[1] "Hola, mundo!"
```

4.3 Valores lógicos

Los valores lógicos (booleanos) pueden ser TRUE o FALSE. Los valores lógicos se usan comúnmente para realizar operaciones condicionales.

```r
# Valores lógicos
verdadero <- TRUE
falso <- FALSE
print(verdadero)
```

```
[1] TRUE
```

```r
print(falso)
```

```
[1] FALSE
```

4.4 Vectores

Un vector es una secuencia de datos del mismo tipo. Puedes crear vectores usando la función c().

```r
# Vector numérico
numeros <- c(1, 2, 3, 4, 5)
print(numeros)
```

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

```r
# Vector de caracteres
cadenas <- c("uno", "dos", "tres")
print(cadenas)
```

```
[1] "uno"  "dos"  "tres"
```

```r
# Vector lógico
logicos <- c(TRUE, FALSE, TRUE)
print(logicos)
```

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

```r
# Nuevo vector
# ¿Qué ocurre si en un mismo vector dejamos character y numeric?
combinado <- c(1,2,3,"cuatro", "cinco")

# El resultado es un vector de "character".
combinado
```

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

```r
# Confirmamos la clase del vector con la función `class()`
class(combinado)
```

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

```r
# A lo anterior se le ha llamado, "coerción"
```

4.5 Coerción

En R, la coerción de vectores se refiere al proceso de conversión automática de los elementos de un vector a un tipo de datos común cuando los elementos originales son de tipos diferentes. Esto se hace para asegurar que todas las operaciones en los vectores se realicen de manera consistente y sin errores. R sigue una jerarquía específica de tipos de datos para realizar esta conversión.

Jerarquía de Coerción R tiene una jerarquía de tipos de datos que determina cómo se realiza la coerción. La jerarquía de coerción en R es la siguiente (de menor a mayor):

1. Logical (Lógico): TRUE, FALSE
2. Integer (Entero): Números enteros
3. Numeric (Numérico o Double): Números reales
4. Complex (Complejo): Números complejos
5. Character (Carácter): Cadenas de texto
6. List (Lista): Colecciones de elementos

Cuando se combinan elementos de diferentes tipos en un vector, R los convierte automáticamente al tipo de datos "más alto" en la jerarquía.

** lógico -> entero -> numérico -> cadena de texto (logical -> integer -> numeric -> character) **

4.5.1 1. Coerción de Logical a Integer

```r
vec <- c(TRUE, FALSE, 1)
print(vec)  # Output: 1 0 1
```

```
[1] 1 0 1
```

En este ejemplo, TRUE se convierte a 1 y FALSE se convierte a 0.

4.5.2 2. Coerción de Integer a Numeric

```r
vec <- c(1L, 2.5)
print(vec)  # Output: 1.0 2.5
```

```
[1] 1.0 2.5
```

El número entero 1L se convierte a 1.0 para coincidir con el número numérico 2.5.

4.5.3 3. Coerción de Numeric a Character

```r
vec <- c(1.5, "a")
print(vec)  # Output: "1.5" "a"
```

```
[1] "1.5" "a"
```

El número 1.5 se convierte a la cadena de texto “1.5”.

4.5.4 4. Coerción de diferentes tipos en una lista

```r
vec <- list(1, "a", TRUE)
print(vec)  # Output: [[1]] 1 [[2]] "a" [[3]] TRUE
```

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

[[2]]
[1] "a"

[[3]]
[1] TRUE
```

En una lista, los elementos no se fuerzan a un tipo común; cada elemento puede mantener su tipo de datos original.

4.6 Coerción Explícita

Además de la coerción automática, R también permite realizar coerciones explícitas usando funciones específicas como as.numeric(), as.character(), as.integer(), etc.

Ejemplo de Coerción Explícita

```r
vec <- c("1", "2", "3")
numeric_vec <- as.numeric(vec)
print(numeric_vec)  # Output: 1 2 3
```

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

Aquí, *as.numeric(vec)* convierte explícitamente el vector de caracteres *vec* en un vector numérico.

Notas Importantes

NA (Not Available): Durante la coerción, si algún elemento no puede convertirse correctamente, R genera un valor NA y generalmente emite una advertencia.

Factores: Cuando se trabaja con factores, coercionarlos a otros tipos puede requerir pasos adicionales, como primero convertir el factor a carácter antes de convertirlo a numérico.

Ejemplo

```r
factor_vec <- factor(c("1", "2", "3"))
char_vec <- as.character(factor_vec)
numeric_vec <- as.numeric(char_vec)
print(numeric_vec)  # Output: 1 2 3
```

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

4.7 Manipular vectores

Puedes manipular vectores de varias maneras, incluyendo seleccionar elementos, sumar y restar elementos, y más.

```r
# Selección del primer elemento
primer_elemento <- numeros[1]
print(primer_elemento)
```

```
[1] 1
```

```r
# Selección de múltiples elementos
primeros_tres <- numeros[1:3]
print(primeros_tres)
```

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

4.7.1 Suma de vectores

```r
# Suma de vectores
suma <- numeros + c(1, 1, 1, 1, 1)
print(suma)
```

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

```r
# Producto de vectores
producto <- numeros * c(2, 2, 2, 2, 2)
print(producto)
```

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

4.7.2 Filtrar vectores

```r
# Filtrar valores mayores que 2
mayores_que_dos <- numeros[numeros > 2]
print(mayores_que_dos)
```

```
[1] 3 4 5
```

```r
# Filtrar valores iguales a TRUE
verdaderos <- logicos[logicos == TRUE]
print(verdaderos)
```

```
[1] TRUE TRUE
```

4.8 Lista

En R, una lista es una estructura de datos versátil que puede contener elementos de diferentes tipos, por ejemplo, vectores, matrices, data frames, e incluso otras listas. Esto la vuelve óptima para organizar y manipular datos heterogéneos.

Concepto de list()

Una lista en R se crea utilizando la función list(). A diferencia de los vectores, que deben contener elementos del mismo tipo, las listas pueden contener elementos de diferentes tipos. Cada elemento de una lista puede ser referenciado mediante índices o nombres.

Ejemplo de Creación de una Lista

```r
lista1 <- list(
  numero = 1,
  nombre = "Juan",
  vector = c(1, 2, 3),
  matriz = matrix(1:4, nrow = 2),
  lista_anidada = list(a = 10, b = 20)
)


print(lista1)
```

```
$numero
[1] 1

$nombre
[1] "Juan"

$vector
[1] 1 2 3

$matriz
     [,1] [,2]
[1,]    1    3
[2,]    2    4

$lista_anidada
$lista_anidada$a
[1] 10

$lista_anidada$b
[1] 20
```

Acceso a Elementos de una Lista Puedes acceder a los elementos de una lista utilizando el operador de doble corchete [[ ]] o el operador de signo de dólar $.

Acceso por Índice

```
print(lista1[[1]])  # [1] 1
```

```
[1] 1
```

```
print(lista1[[3]])  # [1] 1 2 3
```

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

Acceso por nombre

```
print(lista1$nombre)  # [1] "Juan"
```

```
[1] "Juan"
```

```r
print(lista1$vector)  # [1] 1 2 3
```

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

4.9 data.frame

Un data.frame en R es una estructura de datos fundamental y muy versátil que se utiliza para almacenar datos tabulares. Es similar a una tabla en una base de datos o una hoja de cálculo en Excel, donde cada columna puede contener un tipo diferente de datos (numéricos, caracteres, factores, etc.), pero todos los elementos de una columna deben ser del mismo tipo.

Concepto de data.frame

Un data.frame es esencialmente una lista de vectores de igual longitud, donde cada vector representa una columna de datos y cada elemento dentro del vector representa una fila. Los data.frames son utilizados ampliamente en R para la manipulación y análisis de datos.

Creación de un data.frame

Puedes crear un data.frame utilizando la función data.frame().

```r
# Crear un data frame simple
df <- data.frame(
  nombre = c("Ana", "Luis", "Marta", "Juan"),
  edad = c(23, 35, 29, 40),
  salario = c(50000, 60000, 70000, 80000)
)
print(df)
```

```
  nombre edad salario
1    Ana   23   50000
2   Luis   35   60000
3  Marta   29   70000
4   Juan   40   80000
```

5 Historia

5.1 Un poco de historia de la programación

La narrativa tradicional presenta la historia de la programación como una marcha triunfal hacia la eficiencia y la automatización. Sin embargo, esta visión ignora las complejas relaciones sociales, económicas y políticas que han moldeado este campo.

La historia de la programación es una narrativa intrínsecamente ligada a las dinámicas sociales y económicas que han moldeado nuestro mundo contemporáneo. Desde sus inicios, la programación ha servido no solo como una herramienta técnica, sino también como un reflejo de las estructuras de poder y las relaciones de producción imperantes.

Este contexto histórico ha dejado una huella profunda en la cultura de la programación. La búsqueda de la eficiencia y la optimización, a menudo a expensas de la legibilidad y la accesibilidad del código, refleja la lógica capitalista de maximizar la producción y minimizar los costos.

Además, la programación ha estado tradicionalmente dominada por una élite técnica, en su mayoría hombres blancos de países desarrollados. Esta falta de diversidad ha perpetuado sesgos y desigualdades en el acceso y uso de la tecnología.

En sus albores, la programación surgió en un contexto de industrialización y expansión capitalista, donde la eficiencia y la productividad eran imperativos. Los primeros lenguajes de programación, como Fortran y COBOL, fueron desarrollados para optimizar procesos industriales y administrativos, facilitando así la consolidación de grandes corporaciones y la centralización del control económico. Este enfoque técnico reflejaba una visión mecanicista de la sociedad, donde el ser humano se veía subordinado a las máquinas y a los imperativos del capital.

La programación no es neutral. Las decisiones sobre qué problemas se abordan, cómo se diseñan las interfaces y quién tiene acceso a la tecnología reflejan las agendas de quienes la crean. En muchos casos, estas agendas están ligadas al poder, el control y el beneficio económico. La creciente influencia de las grandes empresas tecnológicas plantea preguntas sobre la privacidad, la manipulación y el impacto social de la programación.

Si bien R es un lenguaje de código abierto, su uso se ha visto influenciado por la creciente mercantilización del conocimiento y la proliferación de software propietario en el ámbito del

análisis de datos. Las grandes empresas tecnológicas han invertido en el desarrollo de herramientas y plataformas comerciales que, si bien se basan en R, buscan capturar y controlar el flujo de datos y análisis.

En este sentido, es crucial mantener una mirada crítica sobre el desarrollo y uso de R, y de cualquier lenguaje de programación. Debemos preguntarnos:**¿quién controla el acceso a estas herramientas? ¿Cómo se utilizan para generar y distribuir conocimiento? ¿Quiénes se benefician de su uso?**

5.2 Algunos puntos adicionales a considerar:

- **El papel del colonialismo:** La historia de la programación está entrelazada con la expansión colonial. La necesidad de administrar y controlar vastos territorios impulsó el desarrollo de tecnologías de la información, como el telégrafo y las máquinas tabuladoras.
- **La brecha digital:** A pesar de los avances tecnológicos, el acceso a la programación y a las herramientas digitales sigue siendo desigual. La brecha digital perpetúa la exclusión y limita las oportunidades para las comunidades marginadas.
- **El impacto ambiental:** La producción de hardware y el consumo energético de las computadoras tienen un impacto significativo en el medio ambiente. Una mirada crítica debe considerar la sostenibilidad de la industria tecnológica.

6 Un poco de historia de R

El lenguaje R, creado en la década de 1990 por Ross Ihaka y Robert Gentleman, emerge en un contexto diferente, marcado por la globalización y la revolución informática. R fue diseñado como una herramienta para el análisis estadístico y la visualización de datos, promoviendo una democratización del conocimiento técnico. Sin embargo, desde una perspectiva marxista, es crucial cuestionar quién controla y se beneficia de estas herramientas. Aunque R es de código abierto y accesible, su adopción está en gran medida mediada por instituciones académicas y corporativas que dominan la producción de conocimiento y, por ende, perpetúan ciertas relaciones de poder.

Además, el auge de lenguajes de programación como R puede interpretarse como parte de una lógica capitalista que busca constantemente innovar y optimizar, pero que también puede conducir a una precarización laboral en sectores tecnológicos. La constante demanda de nuevos conocimientos y habilidades en programación refleja una dinámica de explotación donde el trabajador debe adaptarse continuamente para mantenerse relevante en el mercado laboral.

Desde una perspectiva crítica, es esencial analizar cómo la evolución de los lenguajes de programación no solo responde a necesidades técnicas, sino también a intereses económicos y políticos. La estandarización de ciertos lenguajes, la propiedad intelectual y las licencias de software son aspectos que revelan cómo el poder se configura y se mantiene en el ámbito tecnológico. En este sentido, la historia de la programación es una ventana para comprender las intersecciones entre tecnología, economía y sociedad, y para cuestionar las estructuras que determinan quién tiene acceso y control sobre las herramientas que moldean nuestro mundo.

Desde sus albores, la programación ha sido moldeada por intereses hegemónicos. El surgimiento de las primeras computadoras y lenguajes de programación estuvo impulsado por la competencia geopolítica durante la Guerra Fría, con fines militares y de control social. Este contexto bélico impregnó la cultura de la programación, priorizando la eficiencia y la optimización por encima de consideraciones éticas y sociales.

La división del trabajo intelectual y manual, característica del capitalismo, se replica en el ámbito de la programación. Los "arquitectos" del software, en su mayoría provenientes de élites académicas, diseñan sistemas que luego son implementados por programadores, quienes a menudo enfrentan condiciones laborales precarias y alienantes. Esta división jerárquica perpetúa la desigualdad y limita la participación democrática en la creación tecnológica.

El lenguaje R, aunque aparentemente neutral, no escapa a esta lógica. Su desarrollo, ligado a la estadística y el análisis de datos, refleja la creciente importancia del control y la manipulación

de la información en la sociedad capitalista. La proliferación de datos masivos (Big Data) y la inteligencia artificial plantean nuevas formas de explotación y vigilancia, que deben ser analizadas desde una perspectiva crítica.

Es crucial desmitificar la idea de que la tecnología es neutral o inherentemente beneficiosa. La programación, como herramienta, puede ser utilizada para fines emancipatorios o para reforzar las estructuras de opresión. Nuestro deber como intelectuales orgánicos es promover una programación crítica, que cuestione el statu quo y busque la transformación social.

6.0.1 Les invitamos a reflexionar sobre las siguientes preguntas:

- ¿Cómo podemos construir una tecnología que esté al servicio de la clase trabajadora y no de la acumulación incesante de capital?
- ¿Cuál es el objetivo de aprender programación en un contexto donde la Inteligencia Artificial puede resolver problemas básicos de forma eficiente?

Recordemos las palabras de Marx:

"Los filósofos no han hecho más que interpretar de diversos modos el mundo, pero de lo que se trata es de transformarlo".

6.1 Orígenes de R

R es un lenguaje y entorno de programación para análisis estadístico y gráficos. Surgió a principios de la década de 1990 como una implementación del lenguaje S, que fue desarrollado en los Laboratorios Bell por John Chambers y sus colegas. R fue creado por Ross Ihaka y Robert Gentleman en la Universidad de Auckland, Nueva Zelanda, y fue lanzado como software libre bajo la licencia GNU General Public License en 1995.

6.1.1 Principales Hitos

- 1995: Primera versión pública de R.
- 2000: Se lanza la versión 1.0.0 de R, marcando su madurez - como herramienta de análisis estadístico.
- 2004: R se convierte en el lenguaje más utilizado en el Proyecto de Análisis Estadístico de Google.
- 2010: Revolution Analytics (ahora parte de Microsoft) comienza a ofrecer soporte comercial para R.
- 2015: Microsoft adquiere Revolution Analytics, impulsando el uso de R en el ámbito empresarial.

- 2017: RStudio (el IDE más usado para R) lanza RStudio Server Pro, facilitando el uso de R en entornos de producción.
- 2023: Rstudio (Antes así se llamaba la empresa y el IDE) cambia su nombre a POSIT y comienza a desarrollar RStudio con orientación al uso multilenguaje.

6.2 Usabilidad

R es conocido por su facilidad para el análisis estadístico y la visualización de datos. Su usabilidad se ve reforzada por una comunidad activa que contribuye con paquetes adicionales que extienden su funcionalidad. R es especialmente apreciado en la academia y en la industria por su capacidad de manejar grandes conjuntos de datos y por sus capacidades gráficas avanzadas.

6.3 Usos Más Populares

Análisis Estadístico: R ofrece una amplia gama de técnicas estadísticas, incluyendo modelos lineales y no lineales, pruebas estadísticas clásicas, análisis de series temporales, clasificación y agrupamiento.

Visualización de Datos: Paquetes como ggplot2 permiten crear visualizaciones de datos sofisticadas y personalizadas. Ciencia de Datos: R es ampliamente utilizado para el análisis de datos en la ciencia de datos, incluyendo minería de datos y machine learning.

Para profundizar sobre los enfoques críticos puedes revisar el trabajo de Wendy Hui Kyong Chun, Safiya Noble, Judith Sutz, Pedro Demo,

6.3.1 Algunos textos y libros:

- Cyber-Marx: Cycles and circuits of struggle in high technology capitalism - Nick Dyer-Witheford.
- La Condición de la Posmodernidad - David Harvey
- A dependência tecnologica segundo a dialéctica da dependência de Ruy Mauro Marini
- Desenvolvimento e dependência - Ruy Mauro Marini

También puede ser útil visitar Revista Chasqui de Ecuador

7 ¿Qué es R Base?

R base se refiere al conjunto fundamental de funciones y paquetes que se instalan con R. Este núcleo incluye funciones para manipulación básica de datos, operaciones aritméticas, funciones estadísticas básicas, y gráficos simples. R base proporciona las herramientas necesarias para comenzar con el análisis de datos, y su funcionalidad puede ser ampliada mediante la instalación de paquetes adicionales desde CRAN (Comprehensive R Archive Network).

7.1 Características de R Base

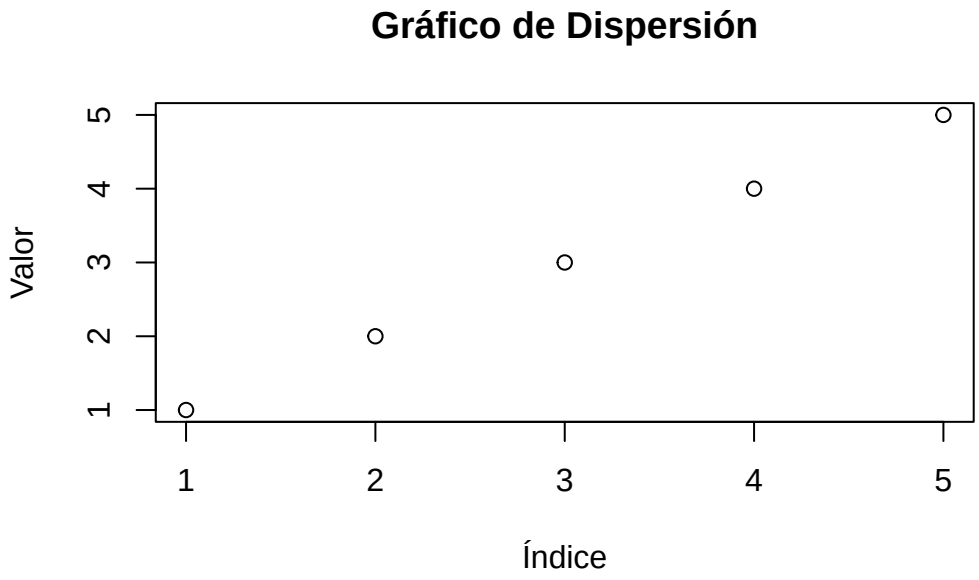
Manipulación de Datos: Funciones para manejar estructuras de datos como vectores, matrices, listas y data frames. Estadísticas Básicas: Funciones para cálculos estadísticos como media, mediana, varianza, y desviación estándar. Gráficos: Herramientas para crear gráficos básicos como histogramas, diagramas de dispersión, y gráficos de líneas. Programación: Funciones de control de flujo, bucles, y la capacidad de definir funciones personalizadas.

7.2 Ejemplo R base

```r
# Crear un vector numérico
numeros <- c(1, 2, 3, 4, 5)

# Calcular la media del vector
media_numeros <- mean(numeros)

# Crear un gráfico de dispersión
plot(numeros, main="Gráfico de Dispersión", xlab="Índice", ylab="Valor")
```



7.3 Manipulaciones de tablas con R base

```r
# Crear un data.frame o tabla

df <- data.frame(
  nombre = c("Ana", "Luis", "Marta", "Juan"),
  edad = c(23, 35, 29, 40),
  salario = c(50000, 60000, 70000, 80000)
)

# Crear un data frame
tabla1 <- data.frame(dia = c("Lunes", "Martes", "Miercoles", "Jueves"),
           obs = c( 10, 11, "hola", 22))

tabla1
```

```
       dia  obs
1    Lunes   10
2   Martes   11
3 Miercoles hola
4   Jueves   22
```

Algo importante a tener en cuenta es que para construir un data.frame o tabla es necesario

tener vectores de una longitud similar. Los vectores pueden tener distintos elementos como ´numeric´o ´character´ sin embargo, se aplicarán las reglas de coerción.

```
# Para ver una columna específica de una tabla podemos utilizar el operador ´$´

class(tabla1$dia)
```

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

```
class(tabla1$obs)
```

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

```
# En este caso utilizamos un función de la familia ´as.´ para convertir el vector en numer
#esta palabra se convierte en NA pues no se puede transformar a numeric.
class(as.numeric(tabla1$obs))
```

```
Warning: NAs introducidos por coerción
```

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

```
# Convertir numerico obs
tabla1$obs <- as.numeric(tabla1$obs)
```

```
Warning: NAs introducidos por coerción
```

```
# Revisamos la clase de la columna del objeto.
class(tabla1$obs)
```

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

```
#Promedio
mean(tabla1$obs, na.rm = TRUE)
```

```
[1] 14.33333
```

7.3.1 Manipulación de tablas con paquete ´dplyr´

```r
# Cargamos la libreria ´dplyr´ y ´clipr´

library(dplyr)
```

```
Adjuntando el paquete: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```r
library(clipr)
```

```
Welcome to clipr. See ?write_clip for advisories on writing to the clipboard in R.
```

```r
# Operador ´pipe´ tiene uso en "Rbase" y en "tidyverse" con el paquete magrittr

#rbase    # magrittr
#  |>     # %>%

# Creamos un nuevo objeto utilizando la pipe
objeto1 <- tabla1 |> # acá asignamos un nuevo objeto "objeto1", desde "tabla1"
dplyr::filter(!is.na(obs)) |>  # se realiza un filtro para quitar "NA" en col "obs"
dplyr::summarise(# Se realiza resumen con medidas
                 promedio = mean(obs, na.rm = TRUE), # promedio
                 mediana = median(obs, na.rm = TRUE), # mediana
                 numero   = n()) |> #conteo
dplyr::rename(Media = promedio) # se renombra la columna promedio como "Media"

# También es posible utilizar pipe en rbase, pero solo la del estilo " |> "
objeto1$Media |> class()
```

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

```r
# Con clipr se puede copiar el objeto rapidamente para pegar en excel y otro
# clipr::write_clip(objeto1)

# Con esta función se puede conocer el directorio de trabajo actual
getwd()
```

```
[1] "C:/Users/rgalv/Nextcloud/PrimerosPasosR/PrimeroPasosEnR"
```

7.4 Guardar o exportar una tabla o data.frame

Es posible guardar un objeto de R en diversos formatos. En el caso de que sean objetos como tablas de datos o bases de datos, que no se trabajarán otro lenguaje, puede ser recomendable almacenarlos en formato ´.rds´ que es el nativo de R y permite mejor velocidad de lectura y escritura, además menos tamaño en disco.

También es posible guardar los archivos en formato ´.csv´ u otros. Generalmente esto se puede hacer con las funciones ´write.´ o ´save´

```r
# Guardar RDS
saveRDS(objeto1, "objeto1.rds")

# Guardar objeto1 como un csv
write.csv2(x = objeto1,  # corresponde al objeto a exportar
          file = "objeto1.csv", # define el nombre que tendrá el archivo exportado
          dec = ",", # define que para decimales se utilice ´,´
          sep = ";", # define que el separador sea ´;´
          row.names = FALSE)
```

```
Warning in write.csv2(x = objeto1, file = "objeto1.csv", dec = ",", sep = ";",
: attempt to set 'sep' ignored
```

```
Warning in write.csv2(x = objeto1, file = "objeto1.csv", dec = ",", sep = ";",
: attempt to set 'dec' ignored
```

7.4.0.1 Manipulación Rbase

Para los siguientes ejercicios se utilizará el paquete ´cars´ que viene incorporado en R.

```r
# Manipular datos
class(cars$dist)
```

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

```r
# Filtro
cars$speed[cars$speed > 10 & cars$dist > 20]
```

```
 [1] 11 12 12 13 13 13 13 14 14 14 14 15 15 16 16 17 17 17 18 18 18 18 19 19 19
[26] 20 20 20 20 20 22 23 24 24 24 24 25
```

```r
# Crea nuevo objeto a partir de filtro
cars2 <- cars[1:15, 1:2]

# Los valores mayores a 10 de la columna speed se reemplazan con ´NA´
cars2$speed[cars2$speed>10] <- NA

# Para trabajar con esta sintaxis es importante considerar que
# en el interior de los corchetes se sigue el orden [filas, columnas]
# objeto[f,c]

# El dato que está en la tercera fila, de la segunda columna
cars[3,2]
```

```
[1] 4
```

```r
# Filtro de rango de los datos que están entre las filas 1 a la 4 y columnas 1 y 2
cars[1:4, 1:2]
```

```
  speed dist
1     4    2
2     4   10
3     7    4
4     7   22
```

```r
# Usando vector
cars[c(1,4), c(1,2)]
```

```
  speed dist
1     4    2
4     7   22
```

```r
# Para crear una tabla de contingencia se puede utilizar la función ´table()´
table(cars2$speed)
```

```
 4  7  8  9 10
 2  2  1  1  3
```

```r
prop.table(table(iris$Species))
```

```
    setosa versicolor  virginica
 0.3333333  0.3333333  0.3333333
```

```r
table(iris$Species, iris$Petal.Length)
```

```
            1 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.9  3 3.3 3.5 3.6 3.7 3.8 3.9  4
  setosa    1   1   2   7  13  13   7   4   2  0   0   0   0   0   0   0  0
  versicolor 0  0   0   0   0   0   0   0   0  1   2   2   1   1   1   3  5
  virginica  0  0   0   0   0   0   0   0   0  0   0   0   0   0   0   0  0

            4.1 4.2 4.3 4.4 4.5 4.6 4.7 4.8 4.9  5 5.1 5.2 5.3 5.4 5.5 5.6 5.7
  setosa      0   0   0   0   0   0   0   0   0  0   0   0   0   0   0   0   0
  versicolor  3   4   2   4   7   3   5   2   2  1   1   0   0   0   0   0   0
  virginica   0   0   0   0   1   0   0   2   3  3   7   2   2   2   3   6   3

            5.8 5.9  6 6.1 6.3 6.4 6.6 6.7 6.9
  setosa      0   0  0   0   0   0   0   0   0
  versicolor  0   0  0   0   0   0   0   0   0
  virginica   3   2  2   3   1   1   1   2   1
```

7.5 Ciclo for

Un bucle for en R permite ejecutar repetidamente un bloque de código un número específico de veces.

7.5.1 Estructura básica del bucle for

La estructura básica del cilo `for` se describe de la siguiente forma: *for (variable in secuencia){ operaciones }*

```
# for (variable in secuencia) {
#       Código a ejecutar
# }
```

- *Variable*: Es una variable que toma los valores de cada elemento en la secuencia.
- *Secuencia*: Es un vector que contiene los valores que la variable tomará sucesivamente.
- *Código a ejecutar*: Es el bloque de código que se ejecutará en cada iteración del bucle.

Ejemplo 1: Iteración en una secuencia

```
for (i in 1:5) {
  print(i)
}
```

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

En este caso, i tomará cada valor en la secuencia 1, 2, 3, 4, 5, y print(i) se ejecutará en cada iteración, imprimiendo cada número.

Ejemplo 2: Iteración en un vector

```
# Crea un vector numérico
numeros <- c(2, 4, 6, 8, 10)
# Realiza el bucle para imprimir los valores del vector
for (num in numeros) {
  print(num)
}
```

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

```
[1] 10
```

Ejemplo 3: Operaciones dentro del bucle

```r
numeros <- c(1, 2, 3, 4, 5)
for (num in numeros) {
  cuadrado <- num^2 # Operación
  print(paste("El cuadrado de", num, "es", cuadrado))
}
```

```
[1] "El cuadrado de 1 es 1"
[1] "El cuadrado de 2 es 4"
[1] "El cuadrado de 3 es 9"
[1] "El cuadrado de 4 es 16"
[1] "El cuadrado de 5 es 25"
```

Ejemplo 4: Bucles anidados

```r
for (i in 1:5) {
  for (j in 1:5) {
    resultado <- i * j
    print(paste(i, "x", j, "=", resultado))
  }
}
```

```
[1] "1 x 1 = 1"
[1] "1 x 2 = 2"
[1] "1 x 3 = 3"
[1] "1 x 4 = 4"
[1] "1 x 5 = 5"
[1] "2 x 1 = 2"
[1] "2 x 2 = 4"
[1] "2 x 3 = 6"
[1] "2 x 4 = 8"
[1] "2 x 5 = 10"
[1] "3 x 1 = 3"
[1] "3 x 2 = 6"
[1] "3 x 3 = 9"
[1] "3 x 4 = 12"
[1] "3 x 5 = 15"
[1] "4 x 1 = 4"
```

```
[1] "4 x 2 = 8"
[1] "4 x 3 = 12"
[1] "4 x 4 = 16"
[1] "4 x 5 = 20"
[1] "5 x 1 = 5"
[1] "5 x 2 = 10"
[1] "5 x 3 = 15"
[1] "5 x 4 = 20"
[1] "5 x 5 = 25"
```

En este caso, el bucle interno recorre los valores del 1 al 5 para cada valor del bucle externo, generando una tabla de multiplicar.

7.6 Sintaxis básica de una función en R

Las funciones en R son fundamentales para estructurar el código y hacerlo reutilizable. Utilizar paquetes como *purrr* y *kableExtra* nos permite ampliar las capacidades de R para manejar listas y generar tablas con estilo, respectivamente. Al seguir una estructura clara y consistente, podemos desarrollar código R eficiente y fácil de mantener.

Para definir una función en R, Se utiliza la palabra clave `function`. Aquí tienes la estructura básica:

```r
nombre_de_la_funcion <- function(argumento1, argumento2, ...) {
  # Cuerpo de la función
  # Código a ejecutar
  resultado <- argumento1 + argumento2
  return(resultado)
}
```

Ejemplo: Crear una función para sumar dos números

```r
suma <- function(a, b) {
  resultado <- a + b
  return(resultado)
}

# Usar la función
suma(3, 5) # Devuelve 8
```

```
[1] 8
```

7.6.1 Funciones con valores predeterminados

Se pueden establecer valores predeterminados para los argumentos de una función:

```
suma <- function(a = 1, b = 1) {
  resultado <- a + b
  return(resultado)
}

# Usar la función con valores predeterminados
suma() # Devuelve 2
```

```
[1] 2
```

7.6.2 Funciones dentro de funciones

En R, también se puede definir funciones dentro de otras funciones:

```
operacion <- function(a, b) {
  suma <- function(x, y) {
    return(x + y)
  }
  producto <- function(x, y) {
    return(x * y)
  }
  resultado_suma <- suma(a, b)
  resultado_producto <- producto(a, b)
  return(list(suma = resultado_suma, producto = resultado_producto))
}

# Usar la función
operacion(3, 5) # Devuelve una lista con la suma y el producto
```

```
$suma
[1] 8

$producto
[1] 15
```

7.6.3 Funciones anónimas

Las funciones anónimas (también conocidas como funciones lambda) son útiles para operaciones rápidas y se utilizan a menudo con funciones de orden superior como apply, lapply, sapply, entre otras.

```
# Usar una función anónima para elevar al cuadrado los elementos de un vector
vector <- c(1, 2, 3, 4)
resultado <- sapply(vector, function(x) x^2)
print(resultado) # Devuelve c(1, 4, 9, 16)
```

```
[1]  1  4  9 16
```

7.6.4 Ejemplo avanzado con purrr

El paquete purrr de *tidyverse* proporciona una forma funcional de trabajar con listas y vectores. Aquí hay un ejemplo de cómo usar una función personalizada con *map*:

```
library(purrr)

# Definir una función para multiplicar por 2
multiplicar_por_dos <- function(x) {
  return(x * 2)
}

# Usar map para aplicar la función a cada elemento de una lista
lista <- list(1, 2, 3, 4)
resultado <- map(lista, multiplicar_por_dos)
print(resultado) # Devuelve una lista con elementos multiplicados por 2
```

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

[[2]]
[1] 4

[[3]]
[1] 6

[[4]]
[1] 8
```

8 Presentaciones en RMarkdown

#Presentaciones en RMD

RMD es la extensión de los archivos creados en `Rmarkdown` es una combinación de códigos de `R` y `Markdown` (Revisa acá ¿Qué es markdown?)

Para crear una presentacion en RMarkdown, utilizando RStudio, se debe seleccionar un nuevo archivo tipo `RMarkdown`, luego el tipo de presentación que se quiere desarrollar, para este ejemplo seleccionamos **beamer**, luego se abrira una ventana con un codigo escrito similar al siguiente:

```
# ---
# title: "Tu Título"
# author: "Tu Nombre"
# date: "Fecha"
# output: beamer_presentation
# ---
#
# ## Diapositivas con viñeta
#
# - Viñeta 1
# - Viñeta 2
# - Viñeta 3
```

8.1 Funciones con kableExtra

Para crear tablas en formato *LaTeX* o *HTML* con el paquete *kableExtra*, primero definimos los datos y luego aplicamos las funciones del paquete para formatear la tabla.

```
# Para instalar paquete "kableExtra" en caso de ser necesario

paquetes <- c("kableExtra", "DT", "htmlwidgets")

# Función para instalar y cargar paquetes
install_carga <- function(paquetes) {
```

	mpg	cyl	disp	hp	drat	wt
Mazda RX4	21.0	6	160	110	3.90	2.620
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875
Datsun 710	22.8	4	108	93	3.85	2.320
Hornet 4 Drive	21.4	6	258	110	3.08	3.215
Hornet Sportabout	18.7	8	360	175	3.15	3.440

```
  if (!require(paquetes, character.only = TRUE)) {
    install.packages(paquetes)
    library(paquetes, character.only = TRUE)
  }
}

# Aplicar la función a cada paquete de la lista
lapply(paquetes, install_carga)
```

```
[[1]]
NULL

[[2]]
NULL

[[3]]
NULL
```

```
# Carga libreria
library(kableExtra)

# Definir datos
dt <- mtcars[1:5, 1:6]

# Crear una tabla con estilo en LaTeX
kbl(dt, booktabs = TRUE) %>%
  kable_styling(latex_options = "striped")
```

Ejemplo de tabla con estilos

```
# Definir los datos
dt <- mtcars[1:5, 1:6]
```

Tabla 8.1: Tabla de Ejemplo con Estilos

	Grupo 1			Grupo 2		
	mpg	cyl	disp	hp	drat	wt
Mazda RX4	21.0	6	160	110	3.90	2.620
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875
Datsun 710	22.8	4	108	93	3.85	2.320
Hornet 4 Drive	21.4	6	258	110	3.08	3.215
Hornet Sportabout	18.7	8	360	175	3.15	3.440

```
# Crear una tabla con múltiples estilos
kbl(dt, booktabs = TRUE,
    caption = "Tabla de Ejemplo con Estilos") %>%
  kable_styling(
    bootstrap_options = c("striped", "hover", "condensed"),
    full_width = FALSE,
    position = "center",
    font_size = 10
  ) %>%
  add_header_above(c(" ", "Grupo 1" = 3, "Grupo 2" = 3)) %>%
  column_spec(1, bold = TRUE, color = "red",
              background = "yellow") %>%
  column_spec(2, width = "5em") %>%
  row_spec(0, bold = TRUE, color = "white",
           background = "#D7261E") %>%
  row_spec(1:5, background = "#F7F7F7")
```

Explicación de los Estilos

- booktabs: Utiliza el paquete booktabs para mejorar el diseño de la tabla.
- bootstrap_options: Añade varias opciones de estilo como striped (filas alternas de color), hover (resaltar filas al pasar el mouse) y condensed (espaciado reducido).
- full_width: Ajusta la tabla para que no ocupe todo el ancho disponible.
- position: Define la posición de la tabla, en este caso centra la tabla en la página.
- font_size: Cambia el tamaño de la fuente de la tabla.
- add_header_above: Añade un encabezado adicional encima de la tabla para agrupar columnas.

- column_spec: Especifica estilos para columnas individuales. En este caso, la primera columna es negrita, roja y con fondo amarillo, y la segunda columna tiene un ancho específico.
- row_spec: Especifica estilos para filas individuales. La fila de encabezado (0) es negrita, blanca con fondo rojo oscuro, y las filas 1 a 5 tienen un fondo gris claro.

8.2 Tablas dinámicas en R

Podemos crear una tabla dinámica utilizando una combinación de `DT::datatable` y el data.frame que queramos visualizar de manera interactiva. A continuación, se muestra un ejemplo utilizando el conjunto de datos 'mtcars'.

```
# Cargar los datos
data("mtcars")

# Crear una tabla dinámica
DT::datatable(mtcars,
          options = list(pageLength = 5,
                         autoWidth = TRUE,
                         dom = 'Bfrtip',
                         buttons = c('copy',
                                     'csv', 'excel',
                                     'pdf', 'print')))
```

8.2.1 Tabla dinámica con estilos

```
# Instalación y carga de paquetes
#install.packages("DT")
library(DT)

# Cargar los datos
data("mtcars")

# Crear una tabla dinámica con personalización adicional
datatable(mtcars,
          options = list(
            pageLength = 5,  # Número de filas por página
            autoWidth = TRUE,  # Ajuste automático del ancho de las columnas
            dom = 'Bfrtip',  # botones, filtro y paginación
```

```
            buttons = c('copy', 'csv',
                        'excel', 'pdf',
                        'print'), # Botones de exportación
            initComplete = JS(
              "function(settings, json) {",
              "$(this.api().table().header()).css({'background-color':
              'skyblue', 'color': 'black'});",
              "}"
            ),
            className = 'cell-border stripe'  # Clases CSS para las celdas
          ),
          class = 'cell-border stripe') %>%
  formatStyle(
    'mpg',  # Columna a estilizar
    backgroundColor = styleInterval(c(20, 30),
                                    c('yellow', 'orange',
                                      'skyblue'))
  )
```

8.2.2 Cambio idioma de la tabla

```
library(DT)
library(htmlwidgets)

# Cargar los datos
data("mtcars")

# Crear una tabla dinámica con opciones de
# estilo personalizadas y configuraciones en español
datatable(mtcars,
          extensions = 'Buttons',  # Botones de exportación disponibles
          options = list(
            pageLength = 5,  # Número de filas por página
            autoWidth = TRUE,  # Ajuste automático del ancho de las columnas
            dom = 'Bfrtip',  # Botones, filtro y paginación
            buttons = list('copy', 'csv', 'excel',
                           'pdf', 'print'),  # Opciones de exportación
            language = list(
              url =
                '//cdn.datatables.net/plug-ins/1.10.25/i18n/Spanish.json'
```

```
            ), # Agregar un archivo json de configuración del idioma
            initComplete = JS(
              "function(settings, json) {",
              "$(this.api().table().header()).css({'background-color':
              '#D7261E', 'color': 'black'});",
              "}"
            )
          ),
          class = 'cell-border stripe') %>%
  formatStyle(
    'mpg',  # Columna a estilizar
    backgroundColor = styleInterval(c(20, 30), c('yellow',
                                                 'orange', 'red'))
  ) %>%
  htmlwidgets::onRender("
    function(el, x) {
      $(el).find('.dt-buttons').appendTo(
      $(el).closest('.dataTables_wrapper').find('.col-md-6:eq(0)') );
    }
  ")
```

8.2.3 Agregar titulo a la tabla

```
# Cargar los datos
data("mtcars")

# Crear un contenedor para la tabla y la referencia
htmltools::div(
  datatable(mtcars,
            extensions = 'Buttons',  # Botones de exportación disponibles
          options = list(
            pageLength = 5,  # Número de filas por página
            autoWidth = TRUE,  # Ajuste automático del ancho de las columnas
            dom = 'Bfrtip',  # Botones, filtro y paginación
            buttons = list('copy', 'csv', 'excel',
                           'pdf', 'print'),  # Opciones de exportación
            language = list(
              url =
                '//cdn.datatables.net/plug-ins/1.10.25/i18n/Spanish.json'
            ), # Agregar un archivo json de configuración del idioma
```

```
            initComplete = JS(
              "function(settings, json) {",
              "$(this.api().table().header()).css({'background-color':
              '#D7261E', 'color': '#fff'});",
              "}"
            )
          ),
          class = 'cell-border stripe',
          caption = htmltools::tags$caption(
            style = 'caption-side: top;
            text-align: center; font-weight: bold;',
            'Título de la Tabla: Comparación de Características de Vehículos'
          )
          ) %>%
    formatStyle(
      'mpg',  # Columna a estilizar
      backgroundColor = styleInterval(c(20, 30), c('yellow', 'orange', 'red'))
    ) %>%
    htmlwidgets::onRender("
      function(el, x) {
        $(el).find('.dt-buttons').appendTo( $(el).closest(
        '.dataTables_wrapper').find('.col-md-6:eq(0)') );
      }
    "),
  htmltools::tags$p(
    style = 'text-align: center; font-style: italic;',
    'Fuente: mtcars dataset'
  )
)
```

8.2.4 Referencias y filtros en la tabla

```
# Cargar los datos
data("mtcars")

# Crear un contenedor para la tabla y la referencia
htmltools::div(
  datatable(mtcars,
            extensions = 'Buttons',  # Botones de exportación disponibles
          options = list(
```

```
            pageLength = 5,  # Número de filas por página
            autoWidth = TRUE,  # Ajuste automático del ancho de las columnas
            dom = 'Bfrtip',  # Botones, filtro y paginación
            buttons = list('copy', 'csv', 'excel',
                           'pdf', 'print'),  # Opciones de exportación
            language = list(
              url =
                '//cdn.datatables.net/plug-ins/1.10.25/i18n/Spanish.json'
            ), # Agregar un archivo json de configuración del idioma
              initComplete = JS(
                "function(settings, json) {",
                "$(this.api().table().header()).css({
                'background-color': 'skyblue', 'color': 'black'});",
                "}"
              )
            ),
            filter = "top",  # Filtros en la parte superior de la tabla
            class = 'cell-border stripe',
            caption = htmltools::tags$caption(
              style = 'caption-side: top; text-align:
              center; font-weight: bold;',
              'Título de la Tabla: Comparación de Características de Vehículos'
            )
            ) %>%
    formatStyle(
      'mpg',  # Columna a estilizar
      backgroundColor = styleInterval(c(20, 30), c('yellow', 'orange', 'red'))
    ) %>%
    htmlwidgets::onRender("
      function(el, x) {
        $(el).find('.dt-buttons').appendTo( $(el).closest(
        '.dataTables_wrapper').find('.col-md-6:eq(0)') );
      }
    "),
  htmltools::tags$p(
    style = 'text-align: center; font-style: italic;',
    'Fuente: mtcars dataset'
  )
)
```

Referencias

A continuación podrás revisar fuentes útiles y referencias.

Referencias principales para Aprender R

Aquí tienes una lista de recursos esenciales para aprender R, desde libros y tutoriales en línea hasta cursos interactivos y canales de YouTube.

Libros

1. **R for Data Science** por Hadley Wickham y Garrett Grolemund
   - Libro en línea gratuito Un recurso excelente para principiantes y usuarios intermedios que cubre conceptos de manipulación, visualización y modelado de datos con R.
2. **Advanced R** por Hadley Wickham
   - Libro en línea gratuito Ideal para aquellos que ya tienen conocimientos básicos de R y desean profundizar en aspectos más avanzados del lenguaje.
3. **El arte de programar en R** por Julio Santana y Efraín Mateos
   - Enlace en CRAN Cubre una amplia gama de temas, desde los fundamentos hasta técnicas avanzadas de programación en R.

Tutoriales en Línea

1. **Swirl - Learn R, in R**
   - Swirl Un paquete de R que ofrece cursos interactivos directamente en la consola de R.
2. **Coursera: R Programming**
   - Curso en Coursera Un curso ofrecido por la Universidad Johns Hopkins que es parte de la especialización en Ciencia de Datos.
3. **Datacamp: Introduction to R**
   - Curso en Datacamp Curso gratuito introductorio que cubre los conceptos básicos de R.

Canales de YouTube

1. **Data School**
   - Data School Ofrece tutoriales detallados sobre R y técnicas de ciencia de datos.
2. **R Programming for Data Science**
   - R Programming for Data Science Enfocado en la programación en R para ciencia de datos, con numerosos ejemplos prácticos.
3. **StatQuest with Josh Starmer**
   - StatQuest Explicaciones claras y simples sobre estadísticas y R.

Documentación Oficial

1. **The Comprehensive R Archive Network (CRAN)**
   - CRAN La fuente oficial de la documentación de R, incluyendo el manual del usuario y otros recursos útiles.
2. **RStudio Documentation**
   - RStudio Documentation Documentación y guías sobre cómo usar RStudio, el IDE más popular para R.

Blogs y Comunidades

1. **R-Bloggers**
   - R-Bloggers Una agregación de blogs sobre R que cubren una amplia gama de temas, desde tutoriales hasta análisis avanzados.
2. **Stack Overflow**
   - Stack Overflow Una comunidad de preguntas y respuestas donde puedes encontrar soluciones a problemas comunes y preguntar sobre tus propios desafíos en R.

Recursos Adicionales

1. **Cheat Sheets**
    - RStudio Cheat Sheets Hojas de referencia rápidas para diversos paquetes y funciones de R, proporcionadas por RStudio.
2. **Tidyverse**
    - Tidyverse Una colección de paquetes de R diseñados para la ciencia de datos, que incluyen `ggplot2`, `dplyr`, `tidyr`, y más.