Análisiside Datos con R y Python

Guía para el Análisis de Datos con R y Python

RICUIB

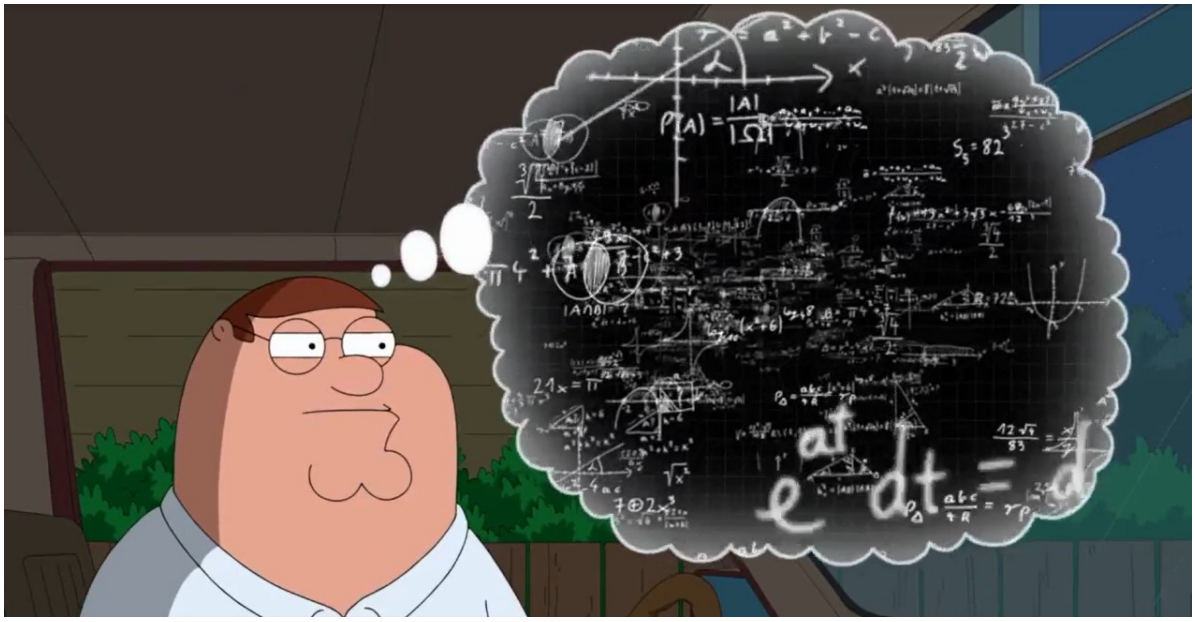
2025-02-01

Tabla de contenidos

Prefacio

Este documento es una versión de unas notas para Análisis de Datos con R

Ha sido elaborado con Quarto RStudio, PBC. (2022). Quarto (Version 1.0).

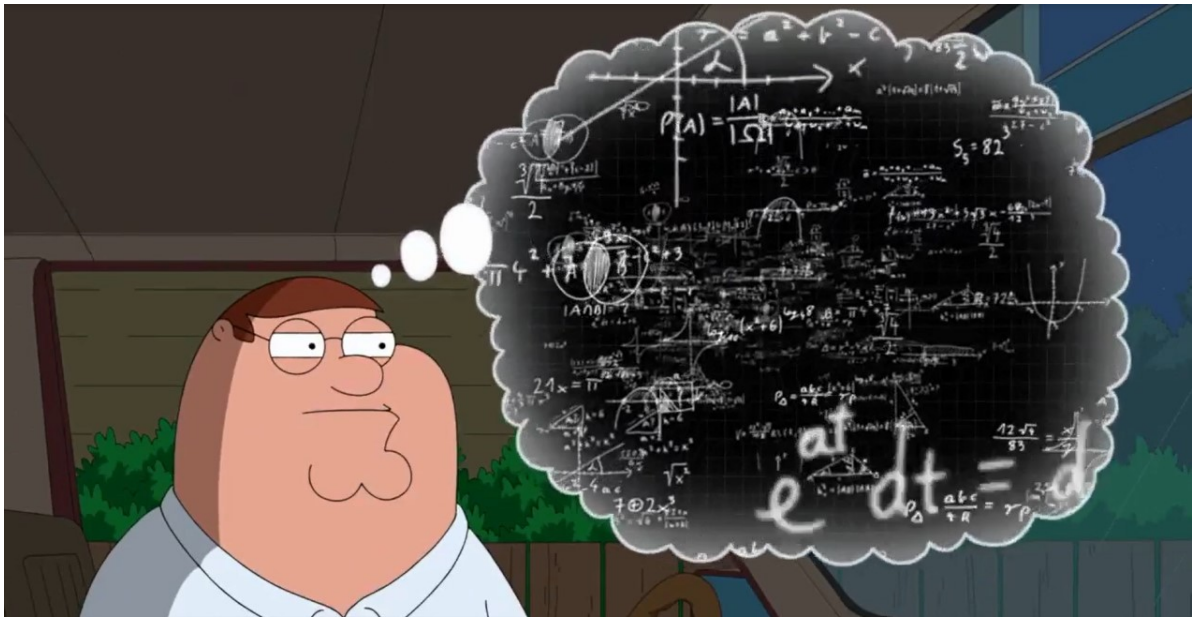


1 Introducción

Estas son unas notas para el análisiside datos vcn R

Parte I

Microcredito I



2 Día 1

2.1 ¿Qué aprenderás?

- Utilizar la interfaz RStudio
- Conocer los tipos de objetos en R
- Instalar y buscar paquetes
- Escribir tus propias funciones y scripts en R
- Realizar los gráficos más adecuados según el tipo de análisis
- Generar informes y garantizar la repetibilidad de los resultados

2.2 ¿Qué es R?

R es un software libre y un lenguaje de programación interpretado, enfocado en la estadística y el análisis de datos.
Características:

- Multiplataforma
- Proyecto abierto y colaborativo

2.3 Paquetes en R

Los paquetes son extensiones de R que contienen código, documentación y datos.

2.3.1 Repositorios comunes

- **CRAN**: Red oficial de distribución

```
install.packages("nombre_paquete")
```

- **Bioconductor**: Orientado a datos biológicos

```
BiocManager::install(c("GenomicFeatures", "AnnotationDbi"))
```

- **GitHub**: Repositorios sin control de calidad oficial

```
devtools::install_github("usuario/repositorio")
```

2.4 Instalación de R

2.4.1 Windows

Descargar R para Windows

2.4.2 Mac

Descargar R para Mac (ARM)

2.4.3 Linux (Ubuntu/Debian)

```
sudo apt update
sudo apt install r-base r-base-dev -y
```

2.5 ¿Cómo usar R?

2.5.1 Consola

Interfaz de línea de comandos sin entorno gráfico.

2.5.2 IDE: RStudio

Entorno integrado de desarrollo para R.
Incluye:

- Editor de código
- Consola
- Historial de comandos
- Explorador de archivos
- Panel de gráficos
- Gestor de paquetes
- Ayuda contextual

2.5.3 Acceso a RStudio vía servidor

- URL: https://bioinformatica.idisba.es/rstudio/
- Usuario y contraseña: enviados por correo electrónico

2.6 ¿Qué es un proyecto?

Un proyecto en R agrupa scripts, datos y resultados:

- En Linux: `./tablas/nombre_archivo`
- En Windows: `.  ablas ombre_archivo`
- Guarda historial de comandos
- Almacena objetos en memoria
- Permite control de versiones (GitHub)
- Permite replicabilidad con `renv`

2.7 Crear un proyecto

Opciones:

- Clonar repositorio desde GitHub
- Usar `renv` para entornos reproducibles
- Crear carpetas como:
  - `scripts/`
  - `practicas/`

2.8 Crear un script

Un script es un conjunto de instrucciones en un archivo `.R`.

- Se escribe código que se ejecutará paso a paso
- Los comentarios comienzan con `#`
- Ejemplo:

```r
# Cargar datos
datos <- read.csv("archivo.csv")
```

2.9 Mover archivos si están en la ubicación incorrecta

Puedes reorganizar manualmente los archivos desde el explorador de RStudio o el sistema operativo.

2.10 Lectura de bases de datos

Formato	Función	Paquete
CSV	`read.csv()`	utils
CSV	`read_csv()`	readr
Excel	`read.xlsx()`	xlsx

Formato	Función	Paquete
SPSS	`read_sav()`	haven

2.10.1 Ejemplo CSV

```
library(readr)
tabla <- read_csv("archivo.csv")
```

```
read.csv(file = "tabla.csv", sep = ",", dec = ".", header = TRUE)
```

Parámetros comunes:

- `sep`: Separador (`,` `;`)
- `dec`: Separador decimal (`.` o `,`)
- `header`: Indica si hay cabecera
- `skip`: Saltar filas
- `check.names`: Validar nombres de columnas
- `comment.char`: Define carácter de comentario (`#`)

2.11 Excel

```
read.xlsx(file = "archivo.xlsx", sheetIndex = 1, header = TRUE)
```

2.12 SPSS

```
library(haven)
datos <- read_sav("archivo.sav")
```

2.13 Almacenar datos en R

```r
tabla <- read.csv("archivo.csv", sep = ",", dec = ".", header = TRUE)
```

También se puede usar = como operador de asignación.

2.14 Práctica 1

Carga las siguientes tablas y asígnales un nombre:

Archivo	Nombre en R
balears_aemet_1980_2023.csv	balears_aemet
Causa_muerte_baleares_1980_2023.xlsx	mort_balears
HealthData.sav	healthdata
antonia_font.txt	antonia_font
zoo.txt	zoo

```r
padro_balears <- read.csv("./tablas/padro_balears_INE_1971_2020.csv", sep = ",", dec = ".", h
```

2.15 ¡Ya tenemos datos!

2.15.1 Visualización básica

```r
head(tabla)
tibble::view(tabla)
colnames(tabla)
print(tabla)
cat("Mensaje")
```

2.15.2 Dimensiones y tipos

```r
dim(tabla)
length(variable)
class(variable)
```

2.16 Descripción básica

```
str(tabla)     # estructura
summary(tabla) # resumen estadístico
```

Con el paquete `Hmisc`:

```
library(Hmisc)
describe(tabla)
```

2.16.1 Funciones de estadística descriptiva

Función	Descripción
`mean()`	Media
`median()`	Mediana
`table()`	Tabla de frecuencias
`range()`	Rango
`var()`	Varianza
`sd()`	Desviación estándar

2.17 Operadores y funciones básicas

Operador	Significado
`+`	Suma
`-`	Resta
`*`	Multiplicación
`/`	División
`^`	Potencia
`==`	Igual
`<`, `<=`	Menor, menor igual
`>`, `>=`	Mayor, mayor igual

Funciones comunes:

- `min()`, `max()`

- `length()`
- `sum()`
- `sort()`
- `grep()`, `gsub()`, `strsplit()`
- `paste()`
- `print()`, `cat()`

2.18 Consejos prácticos

- Siempre comenta tu código para entender qué hace cada bloque.
- Guarda tus scripts con nombres significativos: por ejemplo, `importar_datos.R` es mejor que `script1.R`.
- Usa `View(nombre_tabla)` en RStudio para explorar tablas interactivamente.

2.19 Ejemplo completo: importar y explorar una tabla CSV

```r
# Instalar el paquete si no lo tienes
install.packages("readr")

# Cargar el paquete
library(readr)

# Leer una tabla CSV
datos <- read_csv("datos_salud.csv")

# Ver las primeras filas
head(datos)

# Obtener la estructura de la tabla
str(datos)
```

```r
# Ver un resumen estadístico
summary(datos)

# Mostrar los nombres de columnas
colnames(datos)
```

Este flujo te permitirá empezar a trabajar con datos reales de forma sencilla y ordenada.

3 Día 2

3.1 Práctica 2

1. Tabla `antonia_font`:
    - Visualiza la canción número 15.
    - Usa `grep` para buscar la palabra "lluna" en la columna `lletra`. ¿Cuántas canciones la contienen?
2. Tabla `healthdata`:
    - Muestra los nombres de las columnas.
    - Usa `class()` para indicar el tipo de cada columna.
3. Tabla `balears_aemet`:
    - Visualiza las primeras 15 filas.
    - Usa `summary()` y `describe()` para describir los datos.
    - ¿Hay alguna variable mal introducida?

3.2 Tipos de objetos en R

3.2.1 Vectores

- `numeric` / `double`: valores numéricos decimales.
- `integer`: números enteros.
- `character`: cadenas de texto.
- `logical`: valores `TRUE` o `FALSE`.
- `Date`: fechas (usando `as.Date()`).

3.2.2 Matrices y listas

- Matrices: estructuras bidimensionales homogéneas (`as.matrix()`).
- Listas: estructuras que pueden contener tipos distintos (`list()`).

3.2.3 Data Frames

- Tablas bidimensionales.
- Cada columna puede tener un tipo distinto.

3.3 Práctica 3

1. Identifica el tipo de `balears_aemet$fecha` y `balears_aemet$prec`, y corrígelo.
2. Calcula el índice de masa corporal (IMC): `Weight / Height^2`.
3. Calcula la ráfaga mínima y máxima en "PALMA, PUERTO".

3.4 Manipulación de tablas con `dplyr`

```
library(dplyr)

data %>%
  mutate(nueva_variable = ...) %>%
  select(...) %>%
  filter(...) %>%
  summarize(...) %>%
  arrange(...)
```

3.5 Práctica 4

Crear la variable `vent_tipus` en `balears_aemet`: - fluix: $<$ 5.83 m/s - moderats: 5.83–11.38 m/s - forts: 11.39–19.71 m/s - molt_forts: 19.72–33.33 m/s - huracanats: $>$ 33.33 m/s

Contar cuántos días hay de cada tipo de viento por estación (Palma, Eivissa, Menorca).

3.6 Ejemplo: Clasificar valores lógicos y tipos

```r
# Supongamos que 'antonia_font' tiene una columna 'lletra'

# Buscar canciones que mencionan "lluna"
grep("lluna", antonia_font$lletra, ignore.case = TRUE)

# Comprobar tipo de variable
class(healthdata$Age)
is.numeric(healthdata$Age)

# Convertir fecha
balears_aemet$fecha <- as.Date(balears_aemet$fecha, format = "%Y-%m-%d")
```

3.7 Buenas prácticas

- Revisa siempre los tipos de datos antes de analizarlos.
- Usa `str()` y `summary()` como herramientas diagnósticas.
- Cuando manipules datos meteorológicos o clínicos, asegúrate de trabajar con unidades homogéneas.

3.8 Ejemplo con `dplyr`

```r
library(dplyr)

# Calcular IMC
healthdata <- healthdata %>%
  mutate(BMI = Weight / (Height^2))

# Ver los valores máximos de racha por ubicación
balears_aemet %>%
  filter(nombre == "PALMA, PUERTO") %>%
  summarize(min_racha = min(racha, na.rm = TRUE),
            max_racha = max(racha, na.rm = TRUE))
```

4 Día 3

4.1 Manipulación avanzada de tablas

4.1.1 Funciones `dplyr`:

- `mutate()`: añade variables nuevas.
- `select()`: selecciona columnas.
- `distinct()`: elimina duplicados.
- `filter()`: filtra filas.
- `summarize()`: resumen estadístico por grupo.

4.1.2 Ejemplo:

```
balears_aemet %>%
  group_by(nombre) %>%
  summarize(COR = cor(presMin, prec_num, use="complete.obs"))
```

4.2 Clasificación de precipitaciones

```
mutate(prec_cat = ifelse(prec <= 2, "débil",
                  ifelse(prec <= 15, "moderada",
                  ifelse(prec <= 30, "fuertes",
                  ifelse(prec <= 60, "muy_fuertes", "torrenciales")))))
```

4.3 Introducción a gráficos con `ggplot2`

4.3.1 Tipos de gráficos:

- Univariantes: histogramas, boxplot

- Bivariantes: dispersión, líneas
- Categóricos: barras
- Combinados: boxplot por categoría, columnas

4.3.2 Estructura:

```
ggplot(data, aes(x, y)) + geom_point()
```

4.4 Práctica 5

- Representar `Age` (barras), `Weight` vs `Waist` (puntos), `Age` por género (boxplot), `Weight` (histograma).
- Para `antoniafont`: barras por disco, color y tema personalizado.

4.5 Ejemplo visual completo

```
library(ggplot2)

# Dispersión: Peso vs Cintura
ggplot(healthdata, aes(x = Weight, y = Waist, color = Gender)) +
  geom_point() +
  theme_minimal() +
  labs(title = "Relación entre Peso y Cintura",
       x = "Peso (kg)",
       y = "Cintura (cm)")
```

4.6 Consejo

Cuando hagas gráficos: - Usa `theme_minimal()` o `theme_classic()` para una apariencia profesional. - Siempre nombra los ejes y agrega títulos claros. - Usa `facet_wrap(~ variable)` si necesitas comparar subgrupos.

5 Día 4

5.1 Personalización de gráficos

5.1.1 Escalas:

```
scale_x_continuous(name = "Eje X", limits = c(25,30))
scale_y_continuous(name = "Eje Y", breaks = c(15,20,25), labels = c("quince", "veinte", "vei
scale_color_manual(values = c("red", "blue"))
scale_fill_manual(values = c("blue", "red"))
```

5.1.2 Coordenadas y Zoom:

```
coord_cartesian(ylim = c(30, 35), xlim = c(2,4))
```

5.1.3 Facetas y temas:

- `facet_wrap()`, `facet_grid()`
- `theme_classic()`, `theme_bw()`...

5.2 Exportación

```
save(objeto, file="objeto.RData")
write.csv(tabla, "tabla.csv")
png("grafica.png")
# plot code
dev.off()
```

5.3 Introducción a la estadística

Uso del paquete `compareGroups`:

```
resultado <- compareGroups(var1 ~ var2, data = tabla)
createTable(resultado)
```

5.4 Quarto y RMarkdown

- Transparencia y reproducibilidad científica
- Integración con RStudio
- Múltiples formatos: HTML, PDF, Word
- Automatización de informes

5.4.1 YAML de ejemplo:

```
---
title: "Análisis de tabla"
author: "Tu nombre"
date: today
format:
  html:
    theme: cosmo
    toc: true
    echo: false
---
```

5.5 Práctica 6 y 7

- Crear documento Quarto con análisis `compareGroups` por `Gender`.
- Crear y guardar una gráfica.
- Insertar gráfica guardada.
- Práctica 7: Informe usando StatRadar.

5.6 Ejemplo completo con escalas, etiquetas y exportación

```r
library(ggplot2)

grafica <- ggplot(healthdata, aes(x = Weight, y = Waist, color = Gender)) +
  geom_point() +
  scale_x_continuous(name = "Peso (kg)", limits = c(50, 100)) +
  scale_y_continuous(name = "Cintura (cm)", breaks = seq(60, 120, 10)) +
  labs(title = "Relación Peso-Cintura", subtitle = "Datos por género") +
  theme_classic()

# Guardar la gráfica
png("peso_cintura.png", width = 800, height = 600)
print(grafica)
dev.off()
```

5.7 Quarto y reproducibilidad

Recuerda que los archivos `.qmd` pueden contener texto + código + resultados. Esto hace que tu análisis sea transparente, reproducible y fácil de compartir.