Módulo 3: Importación y exportación de datos

25 de June de 2025

1. Lectura de archivos CSV y TSV:

- **Funciones base**
  - read.csv("archivo.csv", header = TRUE, sep = ",", stringsAsFactors = FALSE)
  - read.table("archivo.txt", header = TRUE, sep = "\t")
- **Argumentos frecuentes:**
  - header: indica si la primera fila son nombres de columnas.
  - sep: separador (, para CSV, "\t" para TSV).
  - stringsAsFactors = FALSE: evita convertir cadenas en factores automáticamente (en versiones recientes de R esto ya no es predeterminado).
  - na.strings = c("","NA"): cómo interpretar valores faltantes.
  - colClasses = c("integer", "character", "numeric", ...): forzar clases de columnas si se conoce de antemano.
- **Paquete readr (parte del tidyverse)**
  - Ventajas: mayor velocidad, detección automática de tipos, funcionalidad de parsers robustos.

1. Lectura de archivos CSV y TSV:

- **Argumentos clave:**
  - col_types: cadena que especifica tipos de columnas (“c” para character, “i” para integer, “d” para double, “l” para lógico).
  - skip = n: omitir primeras n líneas (útil si el archivo tiene metadatos antes de la tabla).
  - n_max = k: leer sólo las primeras k filas (útil para verificar estructura).
- **Manejo de valores faltantes**
  - En read.csv(): na.strings = c(“”,”NA”, “NULL”).
  - En read_csv(): na = c(“”,”NA”).

2. Lectura de archivos Excel:

- **Paquete readxl**
  - Instalación: install.packages("readxl").
  - Cargar: library(readxl).
- **Función principal:**
  - df_excel <- read_excel("datos.xlsx", sheet = "Hoja1", range = "A1:D100")
- **Argumentos clave:**
  - sheet: puede ser nombre o índice de la hoja (1, 2, ...).
  - range: rango de celdas (p. ej., "B2:E50").
  - col_names = TRUE/FALSE: si la primera fila tiene nombres.

2. Lectura de archivos Excel:

- **Otras opciones**
  - Paquete openxlsx: permite mayor flexibilidad (escritura de datos, formatos, estilos).
  - Ejemplo de escritura:
    - write.xlsx(df, file = “salida.xlsx”, sheetName = “Análisis”, rowNames = FALSE)

3. Lectura de otros formatos delimitados:

- **Archivos delimitados no estándar**
  - Si el separador es punto y coma (;):
    - df_psv <- read_delim("archivo.psv", delim = "|") # por ejemplo, pipe-separated
  - Con data.table::fread():
    - Muy rápido, detecta automáticamente el separador en la mayoría de casos.
    - df_dt <- fread("datos_grandes.csv") permite especificar argumentos: sep=";", header=TRUE, na.strings="NA"
- **Ventajas de fread():**
  - La función deduce el tipo de columna más eficientemente.
  - Lee archivos muy grandes en menor tiempo y menor consumo de memoria.

4. Conexión a bases de datos:

- Paquete DBI
- Proporciona una interfaz genérica; luego se cargan controladores específicos según el DBMS (por ejemplo, RSQLite, RMySQL, RPostgres).

4. Conexión a bases de datos:

- **Ejemplo con SQLite:**
  - install.packages("RSQLite")
  - library(DBI)
  - con <- dbConnect(RSQLite::SQLite(), "mi_bd.sqlite") # conecta o crea el archivo
  - dbListTables(con) # muestra tablas presentes en la base
  - df_query <- dbGetQuery(con, "SELECT * FROM tabla_ejemplo WHERE edad > 30")
  - dbDisconnect(con) # cerrar la conexión al terminar

4. Conexión a bases de datos:

- Paquetes específicos
- PostgreSQL: RPostgres o RPostgreSQL.
- MySQL/MariaDB: RMySQL.
- Microsoft SQL Server: odbc con drivers ODBC.
- Leer grandes tablas por partes
- Ideal para no cargar toda la tabla si es muy grande

4. Conexión a bases de datos:

- **Ejemplo:**
  - rs <- dbSendQuery(con, “SELECT columna1, columna2 FROM tabla LIMIT 1000 OFFSET 0”)

5. Exportar datos:

- **Escribir data frames a CSV**
  - Base: write.csv(df, “salida.csv”, row.names = FALSE, fileEncoding = “UTF-8”).
  - readr: write_csv(df, “salida.csv”) (más rápido).

5. Exportar datos:

- **Escribir a Excel**
  - Con writexl:
    - Con openxlsx (más opciones de formato):
- **Escribir a bases de datos:**
  - Usando DBI:

6. Aspectos prácticos de calidad de datos:

- **Verificar tipos de variables**
  - sapply(df, class) devuelve la clase de cada columna

6. Aspectos prácticos de calidad de datos:

- **Detectar valores faltantes y caracteres extraños**
  - summary(df) muestra conteos de NA, mínimos, máximos
  - any(is.na(df)) TRUE si hay al menos un NA en todo el data frame

6. Aspectos prácticos de calidad de datos:

- **Transformar fechas con lubridate**
  - Instalación: install.packages("lubridate"); cargar: library(lubridate).
  - Funciones comunes:
    - fechas1 <- ymd("2020-01-15") # Año-Mes-Día
    - fechas2 <- dmy("15/02/2021") # Día-Mes-Año
    - fechas3 <- mdy_hms("03-12-2022 14:30:00") # Mes-Día-Año hora:minuto:segundo

6. Aspectos prácticos de calidad de datos:

- **Transformar fechas con lubridate**
  - Extraer componentes:
    - year(fechas1) # 2020
    - month(fechas2) # 2
    - day(fechas3) # 3
    - hour(fechas3) # 14
    - wday(fechas3) # día de la semana (1 = domingo)

6. Aspectos prácticos de calidad de datos:

- **Transformar fechas con lubridate**
  - Transformar fechas en variables categóricas (p. ej., “fin de semana” vs “día laborable”):
    - df$dia_semana <- wday(df$fecha, label = TRUE)
    - df$es_finde* <- df$dia_semana %in% c(“Sat”, “Sun”)