Cómo guardar archivos en R: Usando write.csv y file.path

Introducción

Una de las cosas más importantes cuando laburamos con datos es poder guardar los resultados de nuestros análisis. En R, tenemos varias funciones que nos permiten exportar los datos en distintos formatos. En esta guía, vamos a ver cómo guardar archivos CSV usando la función `write.csv()` y cómo manejar las rutas de archivos de forma práctica con `file.path()`.

Directorio de trabajo

Antes de arrancar a guardar archivos, es importante saber dónde se van a guardar por defecto. R usa un "directorio de trabajo" (working directory o WD) que podemos chequear y cambiar:

```
# Ver el directorio de trabajo actual
getwd()

# Cambiar el directorio de trabajo
setwd("/ruta/a/mi/carpeta")
```

Guardar un archivo CSV básico

La forma más simple de guardar un dataframe como un archivo CSV es:

```
# Crear un dataframe de ejemplo
datos <- data.frame(
  id = 1:5,
  nombre = c("Ana", "Carlos", "María", "Juan", "Laura"),
  edad = c(25, 30, 22, 28, 35)
)

# Guardar el dataframe como CSV
write.csv(datos, "datos_personas.csv")
```

Este comando va a guardar el archivo en el directorio de trabajo que tengamos configurado.

Usando file.path() para crear rutas de archivos

La función `file.path()` está buenísima para crear rutas de archivos de manera robusta y compatible con diferentes sistemas operativos:

```
# Crear una ruta usando file.path()
mi_ruta <- file.path("datos", "resultados", "archivo.csv")
print(mi_ruta)
# En Windows: "datos\\resultados\\archivo.csv"
```

```
# En Unix/Mac: "datos/resultados/archivo.csv"
```

Lo copado de `file.path()` es que maneja automáticamente las barras según el sistema operativo donde estés corriendo el código.

Definiendo una ruta de guardado con outstub

Una práctica re útil es definir una variable que tenga la ruta base donde queremos guardar nuestros archivos. Esto nos permite:

1. Cambiar la ubicación de guardado tocando solo una línea de código
2. Mantener una estructura prolija para los archivos exportados
3. Hacer que el código sea más fácil de leer

Veamos cómo hacerlo:

```
# Definir la variable outstub con la ruta de guardado
outstub <- file.path("resultados", "proyecto_ventas")

# Crear el directorio si no existe
if (!dir.exists(outstub)) {
  dir.create(outstub, recursive = TRUE)
}

# Guardar diferentes archivos usando la misma base
write.csv(datos_clientes, file.path(outstub, "clientes.csv"), row.names = FALSE)
write.csv(datos_ventas, file.path(outstub, "ventas.csv"), row.names = FALSE)
write.csv(datos_productos, file.path(outstub, "productos.csv"), row.names = FALSE)
```

Ejemplo completo: Flujo de trabajo para guardar archivos

Veamos un ejemplo completo que incluye la creación de datos, procesamiento y guardado en una estructura organizada:

```
# Cargar librerías
library(dplyr)

# Crear algunos datos de ejemplo
ventas <- data.frame(
  fecha = as.Date("2025-01-01") + 0:29,
  producto_id = sample(1:10, 30, replace = TRUE),
  cantidad = sample(1:20, 30, replace = TRUE),
  precio_unitario = round(runif(30, 10, 100), 2)
)

# Agregar columna de total
ventas <- ventas %>%
  mutate(total = cantidad * precio_unitario)

# Crear resumen por producto
resumen_productos <- ventas %>%
  group_by(producto_id) %>%
  summarise(
    ventas_totales = sum(cantidad),
    ingresos = sum(total),
    precio_promedio = mean(precio_unitario)
```

```r
  )

# Definir la estructura de directorios para guardar
fecha_analisis <- format(Sys.Date(), "%Y%m%d")
outstub <- file.path("analisis", "ventas", fecha_analisis)

# Crear el directorio si no existe
if (!dir.exists(outstub)) {
  dir.create(outstub, recursive = TRUE)
}

# Guardar los diferentes archivos
write.csv(ventas, file.path(outstub, "datos_ventas_completos.csv"), row.names = FALSE)
write.csv(resumen_productos, file.path(outstub, "resumen_por_producto.csv"), row.names =

# Avisar donde quedaron guardados
cat("Archivos guardados en:", outstub, "\n")
```

Parámetros importantes de write.csv

La función `write.csv()` tiene varios parámetros que pueden venir bien:

```r
write.csv(
  x,                # El dataframe a guardar
  file = "",        # Nombre o ruta del archivo
  append = FALSE,   # Si es TRUE, agrega al archivo en vez de pisarlo
  quote = TRUE,     # Si es TRUE, pone comillas en los campos de texto
  sep = ",",        # Separador de campos (coma por defecto)
  row.names = TRUE, # Si es TRUE, incluye los nombres de las filas
  col.names = TRUE, # Si es TRUE, incluye los nombres de las columnas
  na = "NA"         # Cómo representar los valores NA
)
```

Para la mayoría de los casos, te recomiendo usar `row.names = FALSE` para evitar que te cree una columna extra con los índices de las filas.

Alternativas a write.csv

Además de `write.csv()`, hay otras funciones para guardar archivos:

- `write.csv2()`: Usa punto y coma como separador (útil para nosotros que usamos la coma como separador decimal)
- `write_csv()` del paquete `readr`: Es más rápida que `write.csv()`
- `fwrite()` del paquete `data.table`: Rapidísima para archivos grandes
- `write_excel_csv()` del paquete `readr`: Pensada para que Excel la abra bien

Buenas prácticas

1. **Usar variables para rutas**: Definí las rutas base como variables.
2. **Incluir fechas en los nombres**: Te ayuda a mantener un control de versiones.
3. **Verificar directorios**: Siempre fijate si el directorio existe antes de intentar guardar.
4. **Nombres descriptivos**: Usá nombres de archivo que muestren claramente qué contienen.

5. **Documentar la estructura**: Comentá en tu código qué archivos estás generando y por qué.
6. **Ser coherente**: Mantené una estructura consistente para todas tus exportaciones.

Conclusión

Manejar bien los archivos es clave en el análisis de datos. Usando `write.csv()` junto con `file.path()` y variables como `outstub`, podés armar un sistema ordenado y práctico para guardar tus resultados.

Acordate que la organización de archivos que implementes al principio de un proyecto te puede ahorrar un montón de tiempo a medida que el proyecto crece, sobre todo cuando necesites encontrar y reusar análisis anteriores.