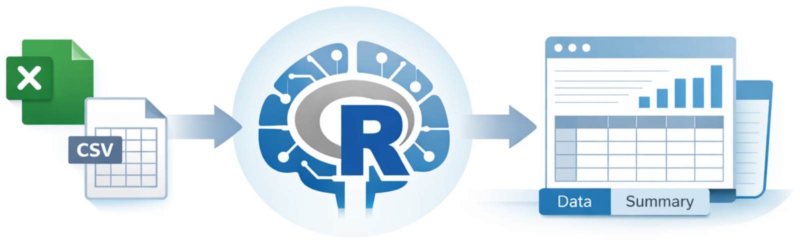
Understanding Data Import and Export in R: Working with CSV and Excel Files

M. Fatih Tüzen

2025-12-26

Table of contents



Introduction

When learning R, most people focus on functions, models, and visualizations. However, many real-world problems start much earlier — at the **data import stage** — and end much later — with **exporting results**.

If data is read incorrectly, no statistical method can save the analysis.

In this post, we focus on the **logic of data import and export in R**, using **CSV and Excel files**. Rather than memorizing functions, we build a mental model for how R interacts with files.

Why Data Import and Export Matters

Data analysis is a workflow:

```
Data source → Import → Analysis → Results → Export → Sharing
```

Errors often occur at the *import* stage:

- wrong delimiters,
- incorrect decimal separators,

- incorrect file paths,
- silently converted data types.

The result?
A model that runs perfectly — on the **wrong data**.

CSV vs Excel: Not a Competition

Before touching R, we should clarify the difference between file formats.

CSV Files

- Plain text files
- Lightweight and fast
- Universally supported
- One table per file
- No formatting, only data

Example:

```
total_bill,tip,sex
16.99,1.01,Female
```

Excel Files

- Binary format (`.xlsx`)
- Can contain multiple sheets
- Store structure and presentation together
- Widely used for reporting and sharing

Key idea:
CSV is a *data transport format.*
Excel is a *communication format.*

Working Directory: Where R Actually Looks

One of the most common beginner mistakes has nothing to do with R syntax.

R does **not** search your entire computer for files. It only looks inside its **working directory**.

```r
getwd()
```

This command shows where R is currently looking.

If a file exists on your computer but not in this directory, R behaves as if the file does not exist.

This is why errors like:

```
cannot open the connection
```

usually indicate a **path problem**, not a coding problem.

The Example Dataset: `tips`

Throughout this post, we use a single dataset: **tips**.

- Restaurant tipping data
- Small and easy to understand
- Contains numeric and categorical variables
- Ideal for demonstrating import/export logic

Data source:
https://raw.githubusercontent.com/mwaskom/seaborn-data/master/tips.csv

Reading CSV Files: The Core Logic

When R reads a CSV file, it needs answers to four questions:

1. How are columns separated?
2. Is the first row a header?
3. What is the decimal separator?
4. How should text be interpreted?

These answers are provided via **function arguments**.

`read.table()`: The Foundation

All CSV-reading functions in base R are built on `read.table()`.

```r
tips <- read.table(
  file = "tips.csv",
  header = TRUE,
  sep = ",",
  dec = ".",
  stringsAsFactors = FALSE
)
```

Understanding this function means understanding CSV import in R.

`read.csv()` and Its Assumptions

`read.csv()` is simply a shortcut for a common case:

- Columns separated by commas
- Decimal separator is a dot

```r
tips <- read.csv("tips.csv")
```

This works perfectly — **if the assumptions match the file**.

The dangerous part? R may not throw an error even if the assumptions are wrong.

> The most dangerous errors are silent ones.

`read.csv2()` and Regional Differences

In many European datasets:

- Columns are separated by semicolons
- Decimals use commas

```
total_bill;tip;sex
16,99;1,01;Female
```

For this structure, `read.csv2()` is designed.

```r
tips2 <- read.csv2("tips_semicolon.csv")
```

Important nuance:
Even if decimals use dots, `read.csv2()` may still work in some cases — but **this is not guaranteed**.

Correct approach:

> Always inspect the file structure before choosing the function.

Writing CSV Files from R

Data analysis rarely ends in R. Results are shared as files.

Writing comma-separated CSV

```r
write.csv(tips, "tips_comma.csv", row.names = FALSE)
```

Writing semicolon-separated CSV

```r
write.csv2(tips, "tips_semicolon.csv", row.names = FALSE)
```

Choosing the correct format depends on **who will read the file next**.

Why We Still Need Excel

CSV is technically superior in many ways. Yet Excel remains dominant in practice.

Why?

- Multiple tables in one file
- Familiar interface for non-technical users
- Common reporting format

Excel is not an analysis tool — but it *is* a powerful delivery tool.

Working with Excel in R: `openxlsx`

The `openxlsx` package allows Excel operations **without requiring Excel itself**.

```r
library(openxlsx)
```

Writing a simple Excel file

```r
write.xlsx(tips, "tips.xlsx", sheetName = "tips")
```

Reading from Excel

```r

tips_excel <- read.xlsx("tips.xlsx", sheet = 1)
```

Multiple Sheets: A Mini Report

Excel shines when organizing related tables.

```r
summary_tips <- aggregate(tip ~ day, data = tips, mean)

wb <- createWorkbook()

addWorksheet(wb, "Raw Data")
writeData(wb, "Raw Data", tips)

addWorksheet(wb, "Summary")
writeData(wb, "Summary", summary_tips)

saveWorkbook(wb, "tips_report.xlsx", overwrite = TRUE)
```

One file.

Multiple views.

Clean structure.

Common Mistakes to Watch For

Most errors are not caused by R, but by assumptions:

- Incorrect working directory
- Wrong delimiter (`sep`)
- Wrong decimal separator (`dec`)
- Reading the wrong Excel sheet
- Overwriting files unintentionally

A healthy habit after every import:

```r
head(data)
str(data)
summary(data)
```

Final Thoughts

If you can:

- read data correctly,
- write data consciously,
- choose file formats intentionally,

you have already crossed one of the most important thresholds in data analysis.

For a complementary discussion, you may also find this article useful:
https://medium.com/p/e730f4a84b3b