


# A Short Introduction to Working With Data in R

Jonathan Whiteley

2023-09-06

# Prerequisites

- Access to a copy of the  software
  - ▶ i.e., a “binary executable”
  - ▶ Go to [www.r-project.org](http://www.r-project.org) to get a copy, or ask your system administrator.
- Tidyverse packages installed on the same system as R
  - ▶ Please run this command in R *before* the workshop:

```
install.packages("tidyverse")
```

- Knowledge of common mathematical operations: arithmetic, logarithms, etc.
- Knowledge of basic R concepts, such as *variables*, *objects*, *operators*, *functions*, *packages*, etc.
  - ▶ This is covered in the first workshop: “A Gentle Introduction to R”

# Learning Objectives

- Load tabular data into R
- Explore data to check that it was loaded correctly
- Export data from R to external files
- Data frames
- Clean data
  - ▶ Add & change columns
  - ▶ Edit values systematically
  - ▶ Change data types
- Tidy data
  - ▶ Change the *shape* of a data frame
- Re-use code, reproducible results, automated reports
  - ▶ Scripts
  - ▶ R Markdown, R Notebooks

# Section 1

## Welcome

# Pop Quiz

We will review these *at the end*, so you can see how much you have learned.

- If multiple packages have functions with the same name, how can you specify which one to use?
- Does R store data in memory or temporary files?
- What is the limit to the size of objects and datasets that can be loaded into R?
- TRUE or FALSE: R has rules and conventions for naming functions
- TRUE or FALSE: if you use one package from the tidyverse, you have to use all of them.

Answer in the chat:

What is your favourite emoji? Why do you like to use it so much?

# Introductions

- Name
- Pronouns
- Job title, role
- *optional*: a favourite childhood treat or candy?
- What are you hoping to learn most in today's workshop?

# Disclaimer

- There is often more than one way to achieve a desired result in R
- Some are faster in certain situations
- Some require less code, or are easier to write as code
- Some are more portable (work on multiple systems)
- But there is rarely as single 'best way'.

This workshop focuses on a coherent approach, that can be learned more easily and extended as needed to tackle bigger problems.

Feel free to take what you learn here and experiment, or explore alternatives. Find what works for *you*.

## Section 2

### Loading data into R



# The Working Directory

- When working with external files, it helps to know the current *working directory*
  - ▶ Any paths supplied to R functions will be relative to this path.

```
getwd()
```

- You can change the working directory with this command:

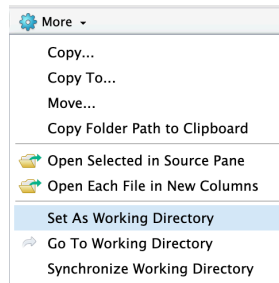
```
setwd('path/to/a/directory')
```

# Set the working directory

- For this workshop, set the working directory to location where you downloaded this presentation and accompanying files.
  - ▶ the directory that *contains* the folder named 'files' that was downloaded along with the files for this workshop.
- Base R on Mac / Linux:
  - ▶ Menu item: "Misc > Change Working Directory..."
  - ▶ CMD+D on Mac; CTL+D on Linux (or Windows)
- In R Studio, you can use the *Files* pane (default bottom-right) to navigate to a directory in your system, and click on "*More > Set As Working Directory*"
  - ▶ or "*Session > Set Working Directory > To Files Pane Location*" in the R Studio menu.

- Base R on Windows:

```
setwd( choose.dir() )
```



# Check your working directory

- Check to see that the working directory is in the right place:

```
file.exists("data_example.csv")
```

```
## [1] TRUE
```

- If the result of the statement above is not “**TRUE**” in your session, try one of the [other approaches](#) to change your working directory, and try again.

# csv files

- 'csv' = **C**omma **S**eparated **V**alues
  - ▶ files in this format have a '.csv' file extension.
- They are:
  - ▶ plain text files
  - ▶ used to represent tabular data, with each *row* on a line, and values in each *column* separated by commas (,)
  - ▶ readable by a wide variety of analysis software (highly portable)
  - ▶ simple—no embedded metadata
- We'll try to load this file into R:  
example\_data.csv
  - ▶ *optional: you can try opening it in a text editor, or spreadsheet software, to see what's in the file.*

# Load a csv file into R

## Section 3

### Exploring your data

# Data frames

## Section 4

### Saving data outside R



# Saving data outside R

## Section 5

### Re-using your code: scripts and other files

# Re-using your code: scripts and other files

## Section 6

### The tidyverse collection of packages

# The tidyverse

```
install.packages("tidyverse")  
help(package="tidyverse")
```

- The **tidyverse** is an “opinionated” collection of packages that are designed to work together.
- All packages share an underlying design philosophy, grammar, and data structures.
  - ▶ *Unlike base R*
  - ▶ Shared naming conventions (e.g., ‘\_’ instead of ‘.’ in function names)
  - ▶ Emphasis on functions that do one thing well
  - ▶ Designed to be combined together to achieve complex operations
- tidyverse is under active development.
  - ▶ New functions and features sometimes replace or supersede old ones.
  - ▶ No guarantee that functions will continue to work the same way in future versions.

# Core tidyverse packages

Today, we will focus on a few of the core tidyverse packages for loading, cleaning, and manipulating data:

- `readr`, `readxl` for **loading** data
- `dplyr` for **manipulating** data (values)
- `tidyr` for **rearranging** data
- `stringr` for working with **strings**

# dplyr: grammar of data manipulation

- dplyr provides many functions, within a coherent framework or *grammar*
- They are intended to help you focus on *what* you want to do, and translate your thoughts into code.
- High-level functions have active names and called “**verbs**” — they describe what they do.
- dplyr and tidyr provide many “**helper functions**” that work *inside* verbs and other functions to make many tasks easier to translate into code.
  - ▶ These functions may not work on their own, outside of dplyr verbs and tidyr functions.

# dplyr verbs

Verbs can be grouped based on the component of the dataset that they work with<sup>1</sup>:

- Rows:
  - ▶ `filter()` chooses rows based on column values.
  - ▶ `slice()` chooses rows based on location.
  - ▶ `arrange()` changes the order of the rows.
- Columns:
  - ▶ `select()` changes whether or not a column is included.
  - ▶ `rename()` changes the name of columns.
  - ▶ `mutate()` changes the *values* of columns and creates new columns.
  - ▶ `relocate()` changes the order of the columns.
- Groups of rows:
  - ▶ `group_by()` defines groups of rows.
  - ▶ `summarise()` collapses a group into a single row.

---

<sup>1</sup><https://dplyr.tidyverse.org/articles/dplyr.html#single-table-verbs>



## dplyr semantics

dplyr verbs and helper functions let you refer to column names of the data frame directly in their arguments as regular variables — without having to quote them. But these names have different meanings (semantics) in different verbs.

- **“select semantics”**: in `select()` and similar functions, a column name refers to its *position* in the data frame.
  - ▶ you can refer to a column as a quoted string in `select()`, and it is interpreted as a reference to the column.
- **“mutate semantics”**: in `mutate()`, a column name refers to a *vector of values*.
  - ▶ you cannot supply a column name as a string in `mutate()`, because it is treated as a vector of length 1, rather than a reference to a column of values.

# A 'pipe' operator



Figure 1: “La Trahison des Images” (“The Treachery of Images”) or “Ceci n'est pas une pipe” (“This is not a pipe”) by René Magritte.



- The `magrittr` package (included with `tidyverse`) provides a “forward-pipe operator”:

```
%>% # ?magrittr::`%>%`
```

- The `magrittr` package is automatically loaded when loading most `tidyverse` packages (e.g., `tidyr`, `dplyr`, `ggplot2`), as these packages all use this operator extensively.
  - ▶ It is often unnecessary to load `magrittr` separately, unless you are **not** using these other packages.

## magrittr's 'forward-pipe' operator

- `%>%` allows you to pass results from an expression on the left-hand side (LHS) as an argument (usually the first) to a *function call* on the right-hand side (RHS).

This expression ...	is equivalent to:
<code>x %&gt;% f()</code>	<code>f(x)</code>
<code>x %&gt;% f(y)</code>	<code>f(x, y)</code>
<code>x %&gt;% f(y, z = .)</code>	<code>f(y, z = x)</code>
<code>x %&gt;% f %&gt;% g %&gt;% h</code>	<code>h(g(f(x)))</code>

- This can make code easier to read, as expressions are written and evaluated from *left to right*, rather than from *inside to outside* nested parentheses.

## R now has a ‘native’ pipe operator

- A pipe operator was introduced in base R in v4.1 (May 2021)<sup>2</sup>:

```
|>      # ?pipeOp
```

- It was inspired by the “forward pipe operator” introduced by `magrittr`, but is more streamlined. See these links for details:
  - ▶ Differences between the base R and `magrittr` pipes
  - ▶ “Understanding the native R pipe `|>`”
- Because it is so new, most code examples online still use ‘`%>%`’ from `magrittr`.
- But ‘`|>`’ is always available *in R  $\geq$  v4.1*, without having to load additional packages.
- This document will use ‘`%>%`’ in the examples, for consistency and because many tidyverse functions were designed to work with it.

---

<sup>2</sup><https://cran.r-project.org/bin/windows/base/old/4.1.0/NEWS.R-4.1.0.html>

# Pipes: exercise

## Section 7

### Clean data

# Clean data

## Section 8

### Tidy data



# Tidy datasets

*“Happy families are all alike; every unhappy family is unhappy in its own way.”*

— Leo Tolstoy

*“Tidy datasets are all alike but every messy dataset is messy in its own way.”*

— Hadley Wickham (doi: [10.18637/jss.v059.i10](https://doi.org/10.18637/jss.v059.i10))

- Tidy datasets provide a standardized way to link the *structure* of a dataset (its physical layout) with its *semantics* (its meaning).

▶ [tidyr vignette](#)

# Section 9

## Review

# Exercise

# Quiz Review

# Section 10

## Backmatter

# Other packages to look at

- `data.table`: a high-performance version of `data.frame` with few dependencies.

Other packages in the tidyverse:

- `lubridate` and `hms`: for date & time values.
- `purrr`: functional programming (FP) tools for working with functions and vectors.
  - ▶ Replace for loops with code that is more efficient and easier to read.

# Writing to Microsoft Excel™ files

Packages that can write to Excel files:

- **xlsx**: read, write, format Excel 2007 (.xlsx) and Excel 97/2000/XP/2003 (.xls) files.
  - ▶ Depends on Java and the rJava package
- **XLConnect**: comprehensive and cross-platform R package for manipulating Microsoft Excel files (.xlsx & .xls) from within R.
  - ▶ Requires a Java Runtime Environment (JRE)
- **openxlsx**: simplified creation of Excel .xlsx files (**not** .xls).
  - ▶ *No dependency* on Java
- **writexl**: portable, light-weight data frame to **xlsx** exporter.
  - ▶ No Java or Excel required

!

I recommend *avoiding* exporting data to Excel files if possible. csv files are easier to read to & write from, and can be read by a wider variety of software (they are more portable).

Automated reports can be produced with R Markdown and output to a variety of more portable formats (pdf, HTML, etc.) instead.

# References

## Cheatsheets:

- [readr/readxl](#)
- [Data transformation with dplyr](#)
- [Data tidying with tidyr](#)