

# Reproducible data analysis using R and RStudio

Matthew Galbraith  
Linda Crnic Institute for Down Syndrome

Data Science for Developing Scholars in  
Down Syndrome Research (DS3) 2025

# Useful links for this session

<https://github.com/DS3-2025/installing Updating R-RStudio>

<https://docs.posit.co/ide/user/ide/get-started/>

<https://rstudio.github.io/cheatsheets/base-r.pdf>

<https://posit.co/download/rstudio-desktop/>

<https://rstudio.github.io/cheatsheets/html/rstudio-ide.html>

[https://github.com/DS3-2025/Rproject\\_template](https://github.com/DS3-2025/Rproject_template)

<https://support.rstudio.com/hc/en-us/articles/200526207-Using-RStudio-Projects>

<https://r4ds.had.co.nz/workflow-projects.html>

<https://tidyverse.tidyverse.org/>

<https://ggplot2.tidyverse.org/>

<https://ggplot2-book.org/>

<https://raw.githubusercontent.com/rstudio/cheatsheets/main/data-visualization.pdf>

<https://www.data-to-viz.com/caveats.html>

<https://r-graph-gallery.com/index.html>

# R

## Base R Cheat Sheet

### Getting Help

#### Accessing the help files

##### ?mean

Get help of a particular function.

**help.search('weighted mean')**

Search the help files for a word or phrase.

**help(package = 'dplyr')**

Find help for a package.

#### More about an object

##### str(iris)

Get a summary of an object's structure.

##### class(iris)

Find the class an object belongs to.

### Using Packages

#### install.packages('dplyr')

Download and install a package from CRAN.

#### library(dplyr)

Load the package into the session, making all its functions available to use.

#### dplyr::select

Use a particular function from a package.

#### data(iris)

Load a built-in dataset into the environment.

### Working Directory

#### getwd()

Find the current working directory (where inputs are found and outputs are sent).

#### setwd('C://file/path')

Change the current working directory.

**Use projects in RStudio to set the working directory to the folder you are working in.**

### Vectors

#### Creating Vectors

|                   |             |                             |
|-------------------|-------------|-----------------------------|
| c(2, 4, 6)        | 2 4 6       | Join elements into a vector |
| 2:6               | 2 3 4 5 6   | An integer sequence         |
| seq(2, 3, by=0.5) | 2.0 2.5 3.0 | A complex sequence          |
| rep(1:2, times=3) | 1 2 1 2 1 2 | Repeat a vector             |
| rep(1:2, each=3)  | 1 1 1 2 2 2 | Repeat elements of a vector |

#### Vector Functions

|  |  |
|--|--|
| <b>sort(x)</b><br>Return x sorted.       | <b>rev(x)</b><br>Return x reversed.    |
| <b>table(x)</b><br>See counts of values. | <b>unique(x)</b><br>See unique values. |

#### Selecting Vector Elements

##### By Position

|                   |                                  |
|-------------------|----------------------------------|
| <b>x[4]</b>       | The fourth element.              |
| <b>x[-4]</b>      | All but the fourth.              |
| <b>x[2:4]</b>     | Elements two to four.            |
| <b>x[-(2:4)]</b>  | All elements except two to four. |
| <b>x[c(1, 5)]</b> | Elements one and five.           |

##### By Value

|                             |                                 |
|-----------------------------|---------------------------------|
| <b>x[x == 10]</b>           | Elements which are equal to 10. |
| <b>x[x &lt; 0]</b>          | All elements less than zero.    |
| <b>x[x %in% c(1, 2, 5)]</b> | Elements in the set 1, 2, 5.    |

##### Named Vectors

|                   |                            |
|-------------------|----------------------------|
| <b>x['apple']</b> | Element with name 'apple'. |
|-------------------|----------------------------|

### Programming

#### For Loop

```
for (variable in sequence){  
  Do something  
}
```

##### Example

```
for (i in 1:4){  
  j <- i + 10  
  print(j)  
}
```

#### While Loop

```
while (condition){  
  Do something  
}
```

##### Example

```
while (i < 5){  
  print(i)  
  i <- i + 1  
}
```

#### If Statements

```
if (condition){  
  Do something  
} else {  
  Do something different  
}
```

##### Example

```
if (i > 3){  
  print('Yes')  
} else {  
  print('No')  
}
```

#### Functions

```
function_name <- function(var){  
  Do something  
  return(new_variable)  
}
```

##### Example

```
square <- function(x){  
  squared <- x*x  
  return(squared)  
}
```

### Reading and Writing Data

Also see the **readr** package.

| Input                                  | Output                               | Description  |
|--|--------------------------------------|--|
| <b>df &lt;- read.table('file.txt')</b> | <b>write.table(df, 'file.txt')</b>   | Read and write a delimited text file.  |
| <b>df &lt;- read.csv('file.csv')</b>   | <b>write.csv(df, 'file.csv')</b>     | Read and write a comma separated value file. This is a special case of read.table/write.table. |
| <b>load('file.Rdata')</b>              | <b>save(df, file = 'file.Rdata')</b> | Read and write an R data file, a file type special for R.                                      |

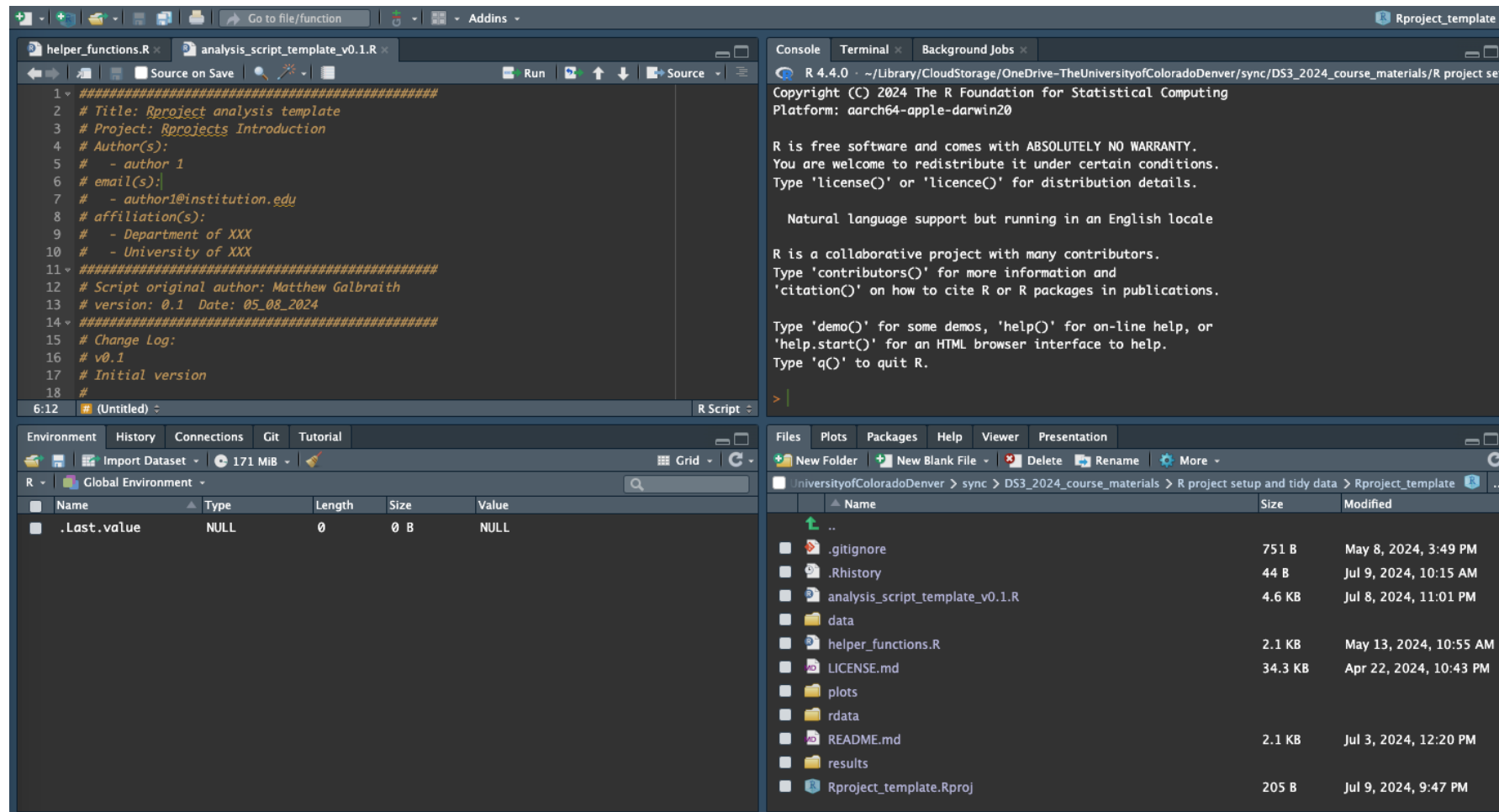
| Conditions | <b>a == b</b> | Are equal | <b>a &gt; b</b> | Greater than | <b>a &gt;= b</b> | Greater than or equal to | <b>is.na(a)</b>   | Is missing |
|------------|---------------|-----------|-----------------|--------------|------------------|--------------------------|-------------------|------------|
|            | <b>a != b</b> | Not equal | <b>a &lt; b</b> | Less than    | <b>a &lt;= b</b> | Less than or equal to    | <b>is.null(a)</b> | Is null    |

# RStudio



## Integrated Development Environment (IDE) for R

- Edit + execute R code
- Syntax highlighting, code completion, debugging
- View output, plots, help, environment



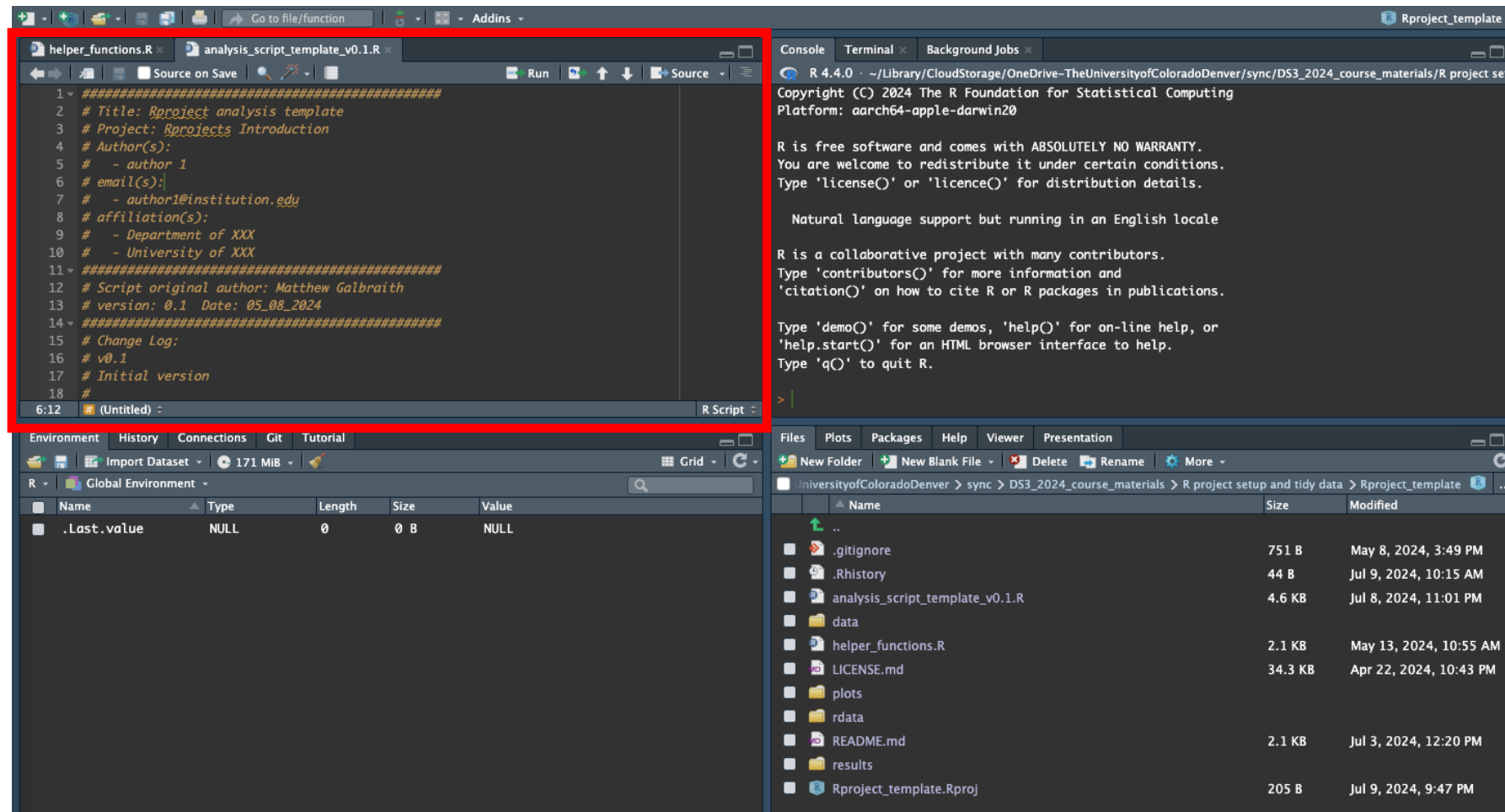
# RStudio



## Integrated Development Environment (IDE) for R

- Edit + execute R code
- Syntax highlighting, code completion, debugging
- View output, plots, help, environment

Source  
pane



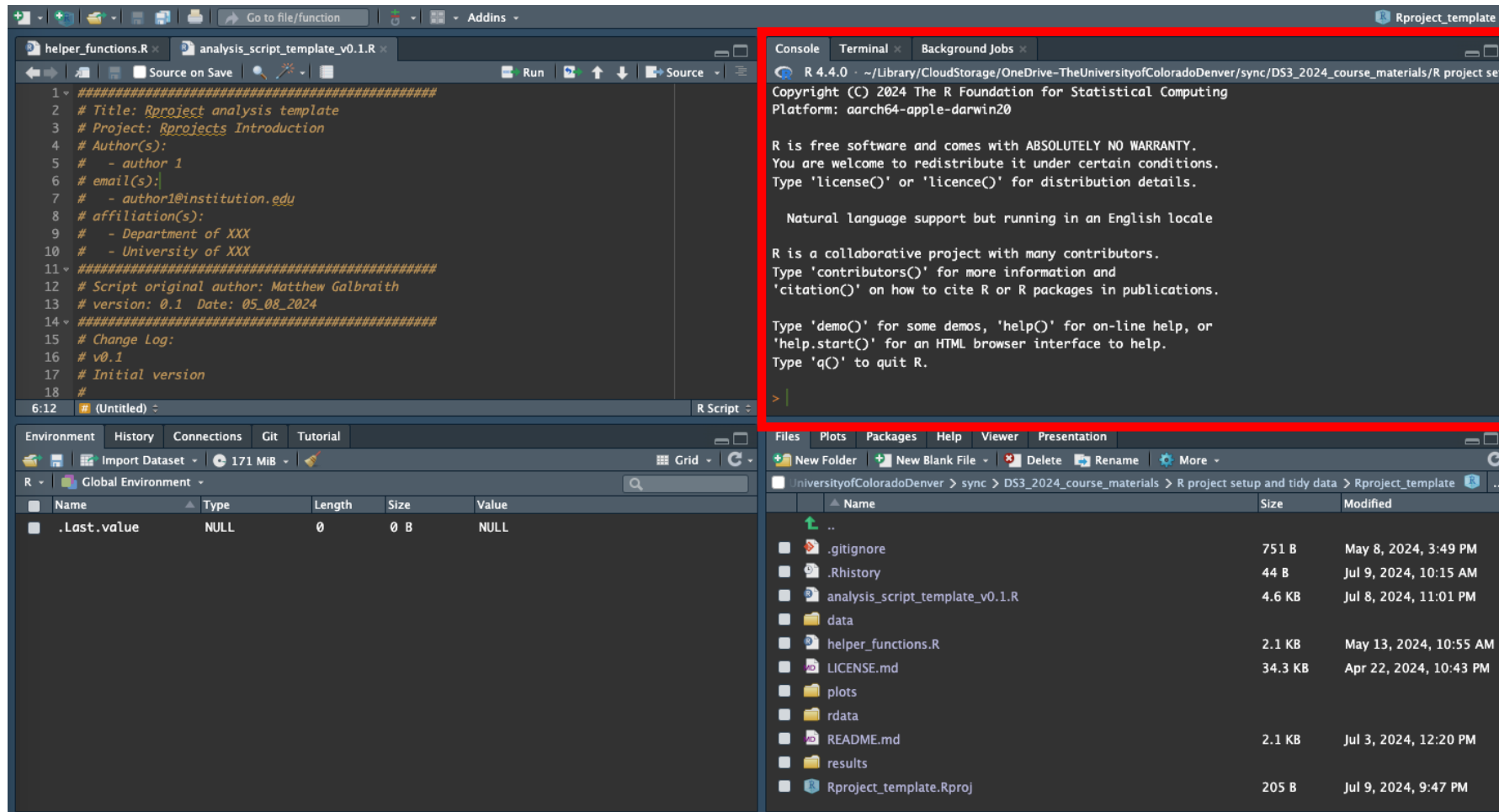
Panes can be rearranged and customized

# RStudio



## Integrated Development Environment (IDE) for R

- Edit + execute R code
- Syntax highlighting, code completion, debugging
- View output, plots, help, environment



Console  
pane

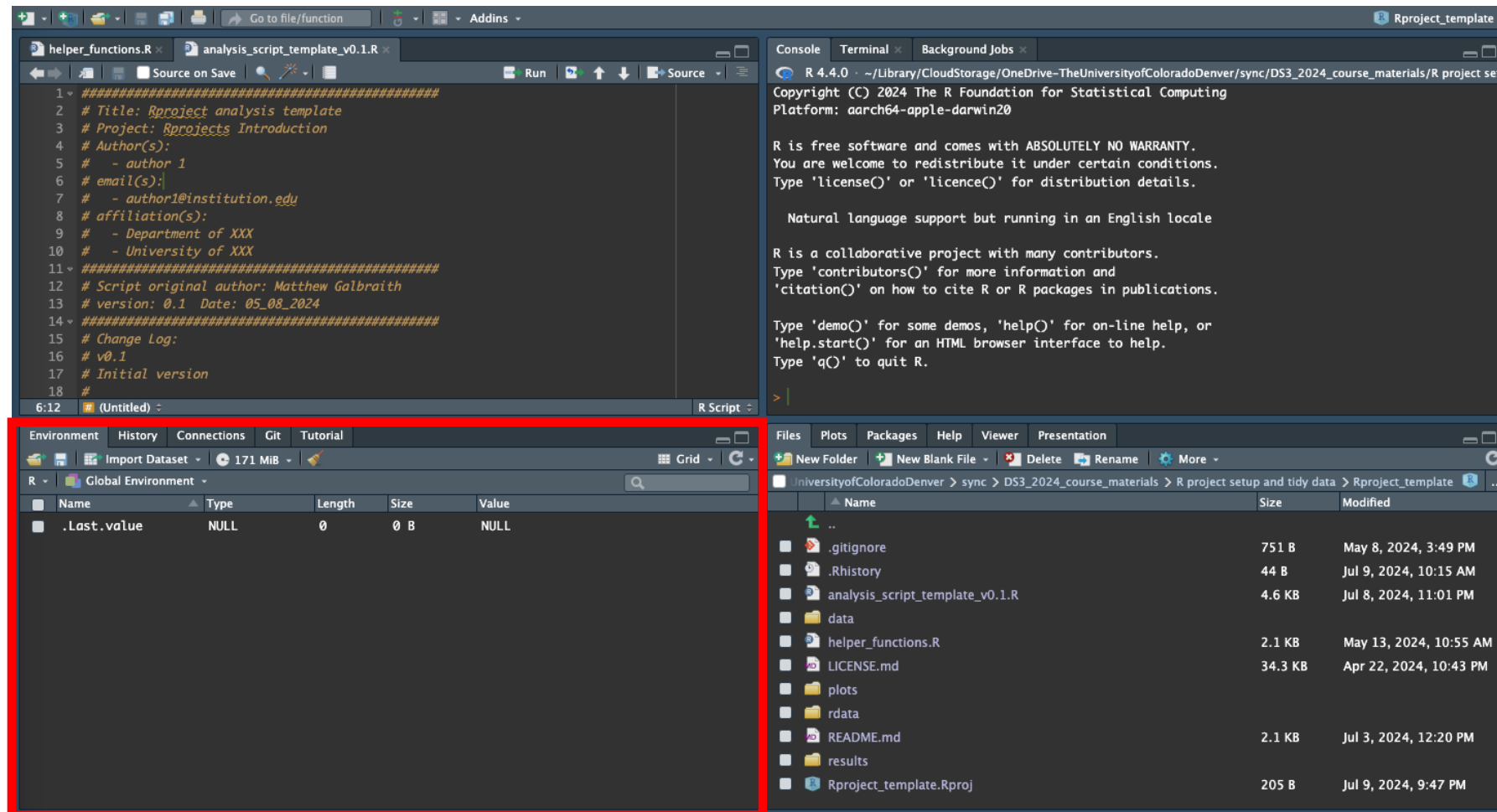
# RStudio



## Integrated Development Environment (IDE) for R

- Edit + execute R code
- Syntax highlighting, code completion, debugging
- View output, plots, help, environment

Environment  
etc pane



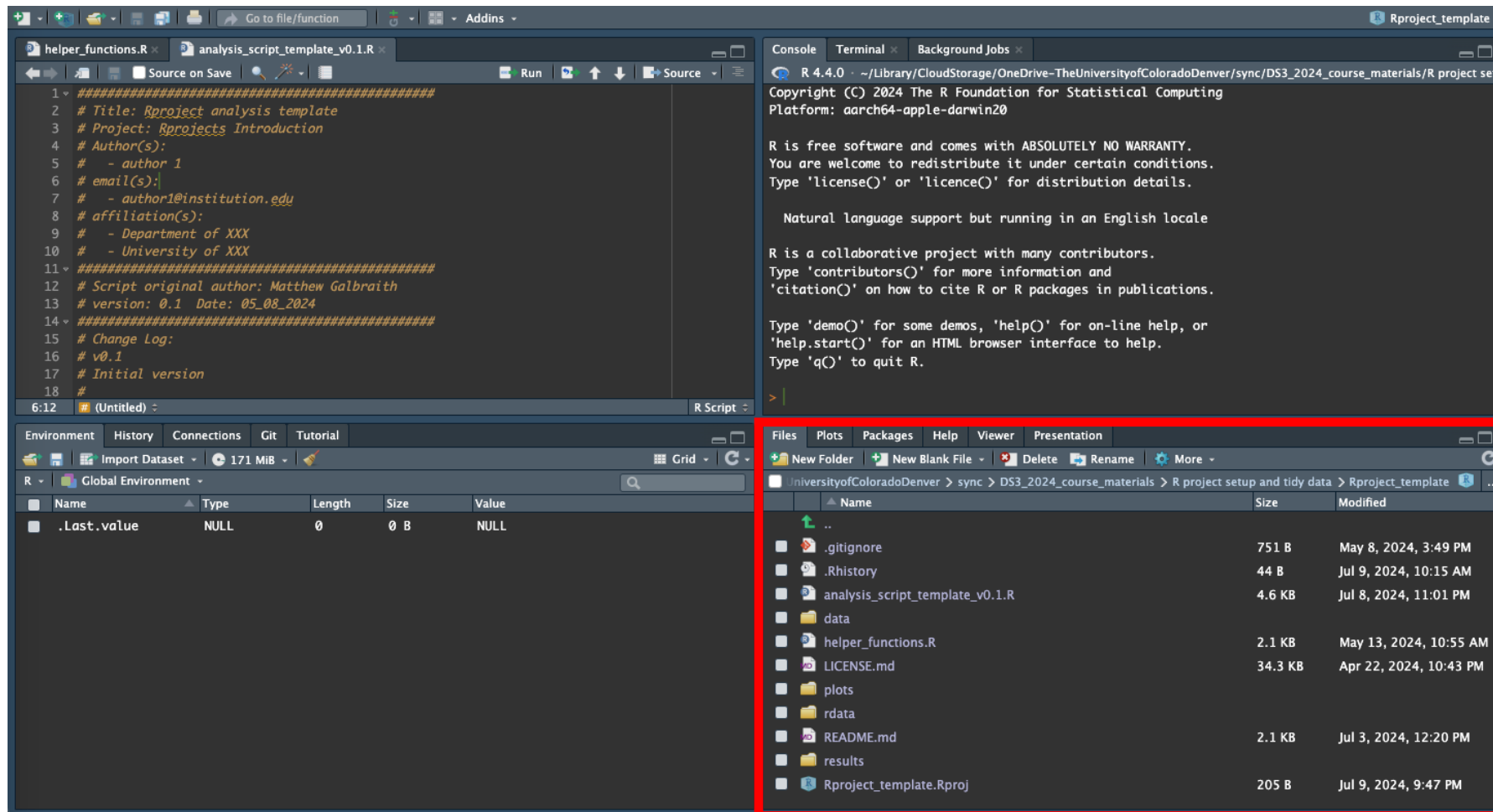


# RStudio



## Integrated Development Environment (IDE) for R

- Edit + execute R code
- Syntax highlighting, code completion, debugging
- View output, plots, help, environment



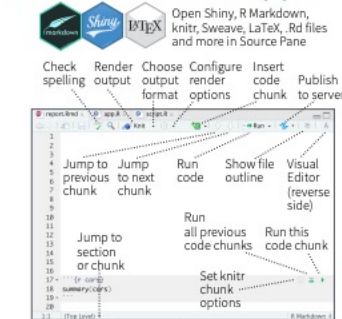
Files, Plots, Help  
etc pane



# RStudio cheatsheet

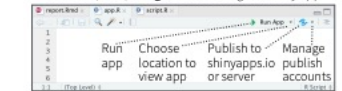
## RStudio IDE : : CHEATSHEET

### Documents and Apps

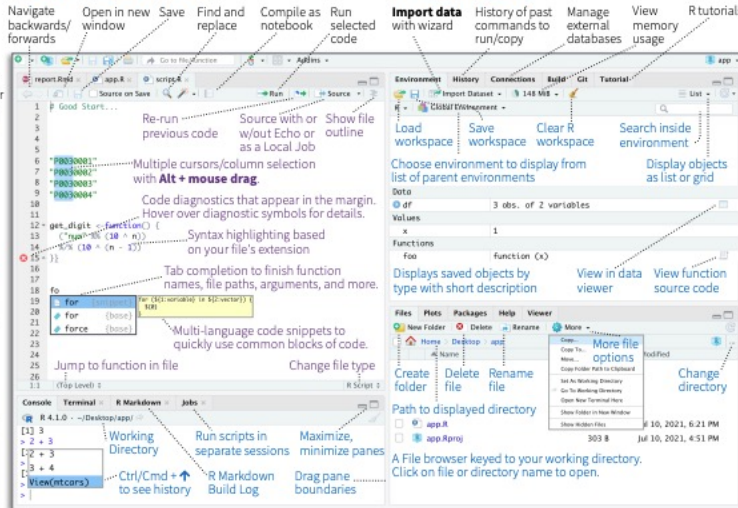


Access markdown guide at **Help > Markdown Quick Reference**  
See reverse side for more on **Visual Editor**

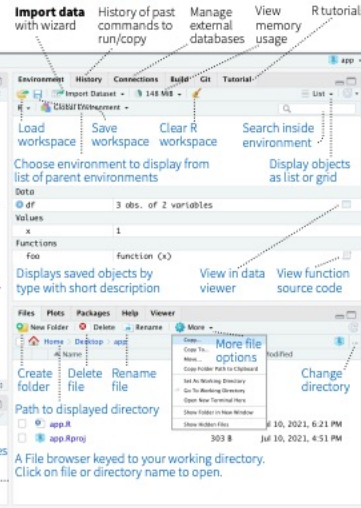
RStudio recognizes that files named **app.R**, **server.R**, **ui.R**, and **global.R** belong to a shiny app



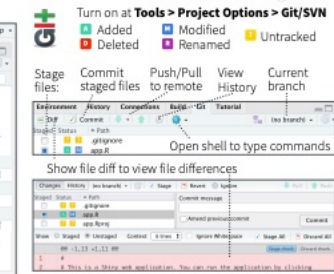
### Source Editor



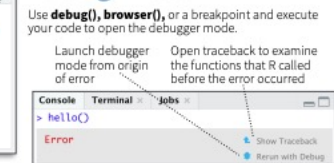
### Tab Panes



### Version Control



### Debug Mode

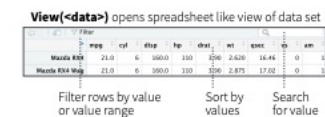
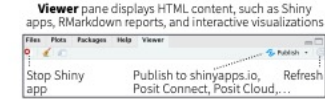
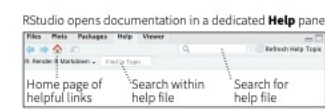
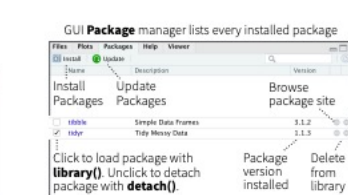
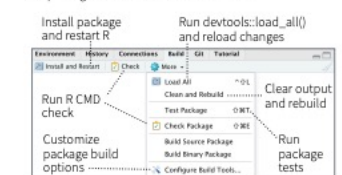


### Package Development

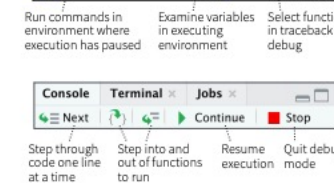
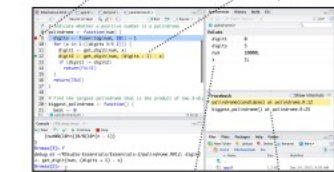


Roxygen guide at **Help > Roxygen Quick Reference**

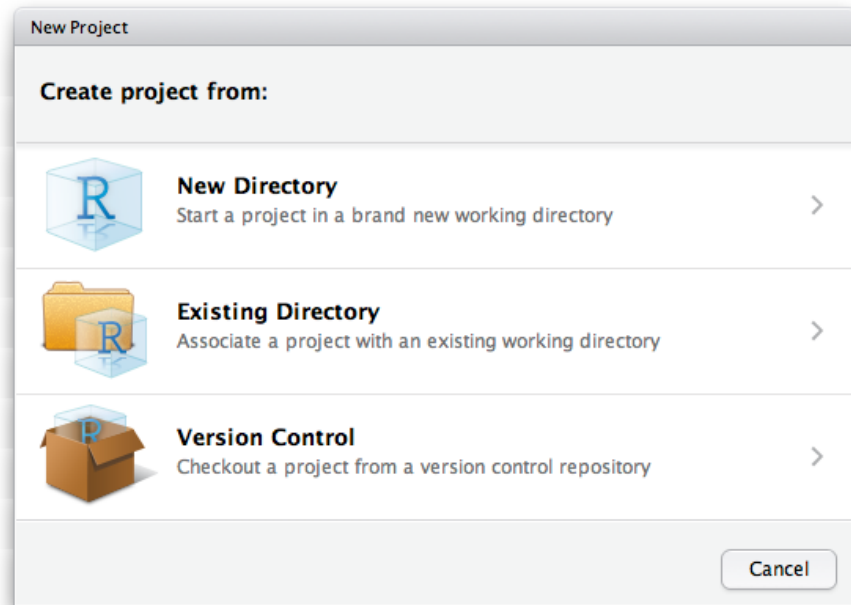
See package information in the **Build Tab**



Click next to line number to add/remove a breakpoint. Highlighted line shows where execution has paused.



# Reproducible data analysis: RStudio Projects



## Project\_directory

- /data
- /results
- /plots
- /rdata
- analysis\_script.R
- helper\_functions.R
- project.Rproj

- Usually open project via .Rproj file
- Automatically sets your working directory
- Self-contained set of directories, scripts, and data files (very important for multiple projects)

## Organizing your Rstudio Projects

- Only **/data** and R scripts are required - everything else can be recreated (incl. earlier versions)
- Treat **/data** directory as read-only
- Analysis outputs go to **/results** or **/plots** (with version info)
- R workspace and large intermediate files stored in **/rdata**
- Additional directories added as needed, eg /Archive
- Compatible with manual or other version control

<https://support.rstudio.com/hc/en-us/articles/200526207-Using-RStudio-Projects>

<https://r4ds.had.co.nz/workflow-projects.html>

[https://github.com/DS3-2024/Rproject\\_template](https://github.com/DS3-2024/Rproject_template)

# Reproducible data analysis: Package management using renv



renv

- `install.packages("renv")`
- `renv::init()` to initialize a new project-local environment with a private R library
- `renv::install()` to install packages after initialization
- `renv::snapshot()` to save the state of your project to `renv.lock`
- `renv::restore()` to restore the state of your project from `renv.lock`

## Project\_directory

```
- /data
- /results
- /plots
- /rdata
- /renv
- renv.lock
- helper_functions.R
- analysis_script.R
- project.Rproj
```

**Allows for fully self-contained R projects  
(Usually) takes care of installing packages**

# Reproducible data analysis: Best Practice

- Main analysis R script with standardized workflow (see next slides)
- Write functions to reduce repetition or increase clarity
- Define functions in dedicated script(s) (e.g. helper\_functions.R) keeps main script workflow clean + makes editing easier
- Comment your code!! (but don't just write what the code is doing)
- Adopt a coding style for consistency (e.g. [Tidyverse style guide](#))
- Track versions and log changes + filenames with versions
- Inspect the data!!!!
- Check each step!!
- Visualize your data!!!!
- Keep track of any random seeds used
- Optional: use R notebooks/R markdown/Quarto/Roxygen to allow generation of reports with code (html, pdf, etc)

# Best Practice for R : :: CHEAT SHEET

## Software

**RStudio**  
Write code in the **RStudio IDE**

**quarto**  
Use **quarto** for literate programming

**git**  
Use **git** to version-control your code and analysis

**GitHub**  
Use **GitHub** to collaborate with other people

## Packages

Packages should be loaded in one place with successive calls to `library()`

Use the **tidyverse** for normal wrangling, plotting etc

Use **tidymodels** for modelling and machine learning

Use **shiny**, **bslib** and **bsDash** for app development

Use **rlang** packages like **rlang**, **cli** and **glue** for low-level programming

Use **renv** in long-term projects to track dependency packages

GitHub Stars are a good proxy for a package's quality. Not sure whether to use a package? If it has >200 stars on GitHub it's probably good!

## Databases

- Use **DBI** and **odbc** to connect to SQL
- Use **helpers** functions to create connections

```

connect_to_db <- function(db) {
  DBI::connect(
    odbc::odbc(), database = db
  ) # Hard-code common options here
} # Connect using the helper
con <- connect_to_db("mwr")
    
```

## Learning More

- For common data science tasks, see [R for Data Science \(pdf\)](#)
- For package development, see [R Packages \(pdf\)](#)
- For advanced programming, see [Advanced R \(pdf\)](#)
- For app development, see [Mastering Shiny](#)

## Projects

### PROJECT CREATION

- Create a new project in RStudio using `File > New Project > New Directory`
- Do not put projects in a single, local folder like `C:\Users\User\OneDrive\Documents`
- Don't put projects in locations controlled by OneDrive / iCloud (these don't play well with Git)

### PROJECT STRUCTURE

Most projects should be structured like this:

```

my-project/
├── .gitignore
├── .Rprojfile
├── R/
│   └── @1-import.R
├── @2-tidy.R
├── @3-sql.R
├── @4-csv.R
├── @5-run-all.R
├── @6-records of package versions, data cleaning
├── @7-env/
├── @8-run_lock
├── my-project.Rproj
└── README.md
    
```

*Ignore files that git which is not to track*

*Scripts in R/ should define functions for use elsewhere*

*Use folders SQL/, data/, etc. for other file types*

*Use a top-level R script to run everything*

*Records of package versions, data cleaning*

*env/: init()*

*A .Rproj file makes this directory an RStudio project*

*Write the main facts about the project here*

## Functions

### WRITING FUNCTIONS: WORKFLOW

- Repetitive, complex code; purpose clarified by comments
- Complex logic abstracted into functions
- Repetition reduced; clearer code; less need for comments

```

operate_on <- function(x) {
  complex_operation(x)
}

a <- operate_on(a)
b <- operate_on(b)
c <- operate_on(c)
d <- operate_on(d)
    
```

## Getting Help

### CREATE A REPREX

- A **minimal, reproducible example** should demonstrate the issue as simply as possible
- Copy your example code and run `reprex::reprex()` to embed errors/ messages/output as comments
- Use your reprex in a question on Teams or Stackoverflow

```

print("Hello ~ world!")
# Error in "Hello ~ world!": non-numeric argument to binary operator
    
```

This reprex minimizes the error when attempting to use + for Python-style string concatenation

## Styling

For other styling guidance, refer to the [Tidyverse style guide](#)

### NAMING THINGS

- Use **lower\_snake\_case** for most objects (functions, variables etc)
- Title\_Snake\_Case** may be used for column names
- Use only **synthetic** names where possible (include only numbers, letters, underscores and periods, and don't start with a number)

### WHITESPACE

- Add spaces** after commas and around operators like `10 * x`, `x + y`, `+`, `-`, `/`, and `<`
- Indentation increases** should always be by exactly 2 spaces
- Add linebreaks** when lines get longer than 80 characters
- When there are many arguments in a call, **give each argument its own line** (including the first one!)

```

# Good (lots of spaces, indents always by 2):
d<- iris |>
  mutate(
    Sepal.Area = Sepal.Width * Sepal.Length,
    Petal.Area = Petal.Width * Petal.Length )

# Good (inconsistent spacing and indentation):
d<-iris |>
  mutate(Sepal.Area=Sepal.Width*Sepal.Length,
    Petal.Area=Width*Petal.Length)
    
```

## Etiquette When Asking Questions

**Don't** Post screenshots of your code

**Do** Use `reprex::reprex()` and paste your code as text

Use `debug()` or `tidyselect::to_include()` to include a data sample

**Ensure your code only fails where you're expecting it to**

Ignore messages or warnings

## Best Practice for RStudio

NB, `usethis::use_description()` + `usethis::use_namespace()` will turn this structure into a package!

CC BY-SA Jacob Scott - [github.com/jws11](#) - Updated: 2023-05-01

# Reproducible data analysis: Standardize your workflow

Title, Project, Author(s)

```
#####  
# Title: Rproject analysis template  
# Project: Rprojects Introduction  
# Author(s):  
#   - author 1  
# email(s):  
#   - author1@institution.edu  
# affiliation(s):  
#   - Department of XXX  
#   - University of XXX  
#####  
# Script original author: Matthew Galbraith  
# version: 0.1   Date: 05_08_2024  
#####  
# Change Log:  
# v0.1  
# Initial version  
#
```

Keep a change log

- What was changed and why?
- When?

# Reproducible data analysis: Standardize your workflow

Explain the purpose of the script + data types used

```
### Summary:
# Description of data wrangling and/or analysis being performed.
#

### Data type(s):
#   A. Meta data
#       Where/who did this data come from?
#       What is the source of the original data and where is it stored?
#   B. Data type 1
#       Where/who did this data come from?
#       What is the source of the original data and where is it stored?
#   C. Data type 2
#       Where/who did this data come from?
#       What is the source of the original data and where is it stored?
#

### Workflow:
#   1. Step 1 description
#   2. Step 2 description
#   3. Step 3 description
#   4. Step 4 description
#

## Comments:
# Any further relevant details?
#
```

Outline the workflow steps (especially for longer scripts)



# Reproducible data analysis: Standardize your workflow

Load package libraries and custom functions

Packages should be loaded in one place with successive calls to `library()`

```
# 0 General Setup -----
# Initialize and install packages with renv
# renv::init()
## 0.1 Load required libraries ----
library("tidyverse")
library("readxl") # read .xlsx files
library("openxlsx") # data export to Excel workbooks
library("skimr") # data summary and validation
library("janitor") # data cleaning etc
library("ggrepel") # labelling points in plots
library("ggforce") # sina plots etc
library("patchwork") # arranging plots
library("tidyHeatmap") # tidy interface to ComplexHeatmap
library("plotly") # generating interactive plots
library("tictoc") # timer
library("conflicted") # force all conflicts to become errors
conflict_prefer("filter", "dplyr")
conflict_prefer("select", "dplyr")
conflict_prefer("count", "dplyr")
library("here") # generate path to current project directory
#
source(here("helper_functions.R")) # load helper functions
#
```

Use comments to explain what packages are being used for

Note use of the *here* package which defines and stores project directory path

Note use of the *conflicted* package which helps manage function conflicts



# Reproducible data analysis: Standardize your workflow

Define file locations and other global variables  
(all input files should be in /data)

```
## 0.2 Set required parameters ----  
# Input data files  
meta_data_file <- here("data", "meta_data_file.txt") # comments/notes on this file?  
data_type1_file <- here("data", "data_type1_file.txt") # comments/notes on this file?  
data_type2_file <- here("data", "data_type1_file.txt") # comments/notes on this file?  
# Other parameters  
standard_colors <- c("Group1" = "#F8766D", "Group2" = "#00BFC4")  
out_file_prefix <- "analysis_script_template_v0.1_"  
# End required parameters ###
```

All plots and results exports should use *out\_file\_prefix* (and go to /plots or /results)  
Include script version number in *out\_file\_prefix*

# Reproducible data analysis: Standardize your workflow

Read in and inspect meta data (sample and/or experiment information)

```
# 1 Read in and inspect data ----  
## 1.1 Read in meta data ----  
meta_data <- meta_data_file |>  
  read_tsv() |>  
  janitor::clean_names(case = "none")  
# inspect  
meta_data  
meta_data |> skimr::skim()  
#
```

How many samples do you expect?

How many measurements do you expect?

Which identifiers are unique?

# Reproducible data analysis: Standardize your workflow

Read in and inspect your main data

```
## 1.2 Read in data type 1 ----
data_type1 <- data_type1_file |>
  read_tsv() |>
  janitor::clean_names(case = "none")
# inspect
data_type1
data_type1 |> skimr::skim()
#

## 1.3 Read in data type 2 ----
data_type2 <- data_type2_file |>
  read_tsv() |>
  janitor::clean_names(case = "none")
# inspect
data_type2
data_type2 |> skimr::skim()
#
```

How many samples do you expect?

How many measurements do you expect?

Which identifiers are unique?

# Reproducible data analysis: Standardize your workflow

Join meta data with main data

```
## 1.3 Join meta data with data type 1 and data type 2 ----  
combined <- meta_data |>  
  inner_join(data_type1) |>  
  inner_join(data_type2)  
# check number of rows returned
```

Do you have all sample rows after join? CHECK

# Reproducible data analysis: Standardize your workflow

Analysis specific sections:

- Data exploration
- Statistical testing
- Plot results
- Export results
- Plot individual features

```
# 2 Data exploration / summary ----  
# check data distribution(s), outliers etc  
  
# 3 Analysis ----  
# statistical testing  
# assemble and export results  
  
# 4 Plot results ----  
# plot results summaries  
  
# 5 Plot individual features ----  
# plot interesting/significant features
```

# Reproducible data analysis: Standardize your workflow

Finally: save and/or reload workspace

```
#####  
# save workspace ----  
save.image(file = here("rdata", paste0(out_file_prefix, ".RData")), compress = TRUE, safe = TRUE) # saves entire workspace (can be slow)  
# To reload previously saved workspace:  
# load(here("rdata", paste0(out_file_prefix, ".RData")))
```

This can save time when resuming, but make sure to resave if something is changed or errors fixed!!

# Into the Tidyverse



<https://tidyverse.tidyverse.org/>

Base R: `install.packages("tidyverse")`

renv: `renv::install("tidyverse")`

## Core tidyverse packages:

[ggplot2](#), for data visualisation.

[dplyr](#), for data manipulation.

[tidyr](#), for data tidying.

[readr](#), for data import.

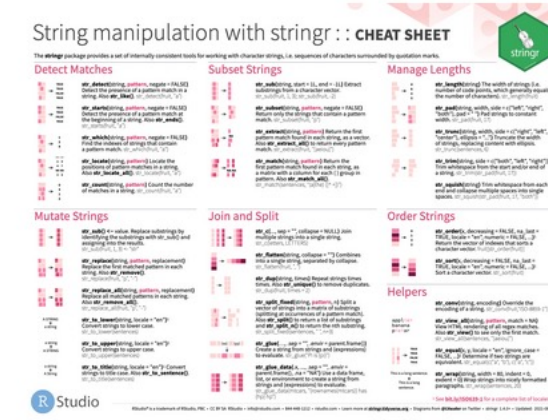
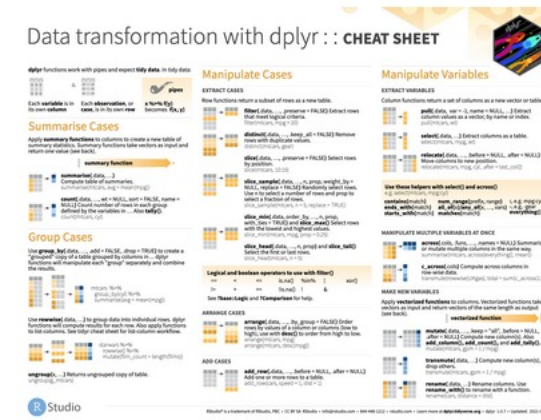
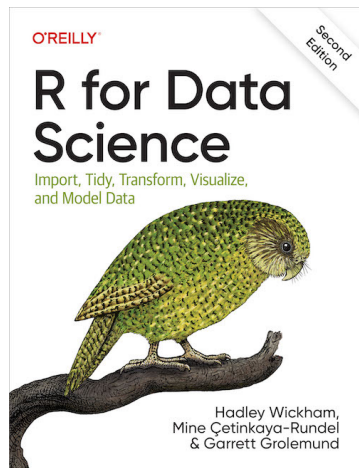
[purrr](#), for functional programming.

[tibble](#), for tibbles, a modern re-imagining of data frames.

[stringr](#), for strings.

[forcats](#), for factors.

[lubridate](#), for date/times.



<https://r4ds.hadley.nz/>

<https://posit.co/resources/cheatsheets/>



# Into the Tidyverse: Pipes



## Pipe operator `%>%`

- Avoids nesting
  - Minimizes need to create objects and functions
  - Structure sequences of operations left-to-right or **top-to-bottom**
  - Easy to inspect and add steps anywhere
- 
- `x %>% f` is equivalent to `f(x)`
  - `x %>% f(y)` is equivalent to `f(x, y)`
  - `x %>% f %>% g %>% h` is equivalent to `h(g(f(x)))`
- 
- Keyboard shortcut in RStudio: cmd/ctrl+shift+m
  - Note: R 4.1.0 introduced a native pipe operator `|>` with some minor differences  
<https://www.tidyverse.org/blog/2023/04/base-vs-magrittr-pipe/>

# Into the Tidyverse: syntax

## Base R “dollar sign” syntax

Example - summary statistics:

one continuous variable:

```
mean(mtcars$mpg)
```

one categorical variable:

```
table(mtcars$cyl)
```

two categorical variables:

```
table(mtcars$cyl, mtcars$am)
```

one continuous, one categorical:

```
mean(mtcars$mpg[mtcars$cyl==4])
```

```
mean(mtcars$mpg[mtcars$cyl==6])
```

```
mean(mtcars$mpg[mtcars$cyl==8])
```

## Tidyverse syntax

Example - summary statistics:

one continuous variable:

```
mtcars %>% dplyr::summarize(mean(mpg))
```

one categorical variable:

```
mtcars %>%
```

```
  dplyr::group_by(cyl) %>%
```

```
  dplyr::summarize(n())
```

two categorical variables:

```
mtcars %>%
```

```
  dplyr::group_by(cyl, am) %>%
```

```
  dplyr::summarize(n())
```

one continuous, one categorical:

```
mtcars %>%
```

```
  dplyr::group_by(cyl) %>%
```

```
  dplyr::summarize(mean(mpg))
```

# Into the Tidyverse

## Z-score calculation with base R:

```
x <- sweep(sweep(t(dat), 1,  
  apply(t(dat), 1, mean, na.rm=T), FUN = "-"),  
  1, apply(t(dat), 1, sd, na.rm=T), FUN = "/")
```

- hard to decipher (learning barrier)
- have to enter target object name in several places

## Z-score calc with tidyverse + scale():

```
zscores <- dat |>  
  select(LabID, Analyte, Value) |>  
  pivot_wider() |>  
  scale()
```

- Somewhat easier to decipher, but not obvious that this calculates Z-scores, even looking at `?scale` defaults (center = TRUE, scale = TRUE)

## Manual Z-score calc with tidyverse:

```
zscores <- dat |>  
  select(LabID, Analyte, Value) |>  
  group_by(Analyte) |>  
  mutate(  
    zscore = (Value - mean(Value, na.rm = TRUE)) / sd(Value, na.rm = TRUE)  
  ) |>  
  ungroup()
```

- Naming of new variable
- Easier to see how calculation was performed
- Easy to keep both original and transformed values for comparison

# Into the Tidyverse

## Z-score calculation with base R:

```
x <- sweep(sweep(t(dat), 1,  
  apply(t(dat), 1, mean, na.rm=T), FUN = "-"),  
  1, apply(t(dat), 1, sd, na.rm=T), FUN = "/")
```

- hard to decipher (learning barrier)
- have to enter target object name in several places

## Even more verbose tidyverse version:

```
zscores <- dat |>  
  select(LabID, Analyte, Value) |>  
  group_by(Analyte) |>  
  mutate(  
    mean = mean(Value, na.rm = TRUE),  
    sd = sd(Value, na.rm = TRUE),  
    zscore = (Value - mean) / sd  
  ) |>  
  ungroup()
```

- Naming of new variable
- Easier to see how calculation was performed
- Easy to keep both original and transformed values for comparison

# Into the Tidyverse: important packages



tibble

## Tibbles = enhanced data frames

- Easier preview of data
- Concise summary information including data types



readr



readxl

## Importing delimited data

- Easy reading in of data from .txt, .csv, .tsv, .xlsx
- Guessing of column types
- Will not convert strings
- Imported as tibble

```
> mpg %>% as.data.frame()
  manufacturer    model displ  year  cyl  trans drv  cty   hwy fl   class
1      audi      a4      1.8  1999    4 auto(l5) f   18   29 p compact
2      audi      a4      1.8  1999    4 manual(m5) f   21   29 p compact
3      audi      a4      2.0  2008    4 manual(m6) f   20   31 p compact
4      audi      a4      2.0  2008    4 auto(av) f   21   30 p compact
5      audi      a4      2.8  1999    6 auto(l5) f   16   26 p compact
6      audi      a4      2.8  1999    6 manual(m5) f   18   26 p compact
7      audi      a4      3.1  2008    6 auto(av) f   18   27 p compact
...
83     ford    explorer 4wd    5.0  1999    8 auto(l4) 4   13   17 r   suv
84     ford   f150 pickup 4wd    4.2  1999    6 auto(l4) 4   14   17 r pickup
85     ford   f150 pickup 4wd    4.2  1999    6 manual(m5) 4   14   17 r pickup
86     ford   f150 pickup 4wd    4.6  1999    8 manual(m5) 4   13   16 r pickup
87     ford   f150 pickup 4wd    4.6  1999    8 auto(l4) 4   13   16 r pickup
88     ford   f150 pickup 4wd    4.6  2008    8 auto(l4) 4   13   17 r pickup
89     ford   f150 pickup 4wd    5.4  1999    8 auto(l4) 4   11   15 r pickup
90     ford   f150 pickup 4wd    5.4  2008    8 auto(l4) 4   13   17 r pickup
[ reached 'max' / getOption("max.print") -- omitted 144 rows ]
```

vs.

```
> mpg
# A tibble: 234 × 11
  manufacturer model    displ  year  cyl trans      drv  cty   hwy fl   class
  <chr>         <chr>    <dbl> <int> <int> <chr>    <chr> <int> <int> <chr> <chr>
1 audi         a4        1.8  1999    4 auto(l5) f    18   29 p    compact
2 audi         a4        1.8  1999    4 manual(m5) f    21   29 p    compact
3 audi         a4        2    2008    4 manual(m6) f    20   31 p    compact
4 audi         a4        2    2008    4 auto(av) f    21   30 p    compact
5 audi         a4        2.8  1999    6 auto(l5) f    16   26 p    compact
6 audi         a4        2.8  1999    6 manual(m5) f    18   26 p    compact
7 audi         a4        3.1  2008    6 auto(av) f    18   27 p    compact
8 audi         a4 quattro  1.8  1999    4 manual(m5) 4    18   26 p    compact
9 audi         a4 quattro  1.8  1999    4 auto(l5) 4    16   25 p    compact
10 audi        a4 quattro  2    2008    4 manual(m6) 4    20   28 p    compact
# ... with 224 more rows
```

# Into the Tidyverse: important packages



dplyr

## Data manipulation

- `mutate()` adds new variables that are functions of existing variables.
- `select()` picks variables based on their names.
- `filter()` picks rows based on their values.
- `summarize()` reduces multiple values down to a single summary.
- `arrange()` changes the ordering of the rows.
- `group_by()` perform group-wise operations.

```
> mtcars %>% as_tibble(rownames = "Model")
# A tibble: 32 x 12
  Model      mpg   cyl  disp    hp  drat    wt   qsec    vs    am  gear  carb
  <chr>    <dbl> <dbl> <dbl>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 Mazda RX4      21     6  160    110   3.9   2.62  16.5     0     1     4     4
2 Mazda RX4 Wag   21     6  160    110   3.9   2.88  17.0     0     1     4     4
3 Datsun 710     22.8    4  108     93   3.85   2.32  18.6     1     1     4     1
4 Hornet 4 Drive  21.4    6  258    110   3.08   3.22  19.4     1     0     3     1
5 Hornet Sportabout 18.7    8  360    175   3.15   3.44  17.0     0     0     3     2
6 Valiant       18.1    6  225    105   2.76   3.46  20.2     1     0     3     1
7 Duster 360    14.3    8  360    245   3.21   3.57  15.8     0     0     3     4
8 Merc 240D     24.4    4  147     62   3.69   3.19   20      1     0     4     2
9 Merc 230     22.8    4  141     95   3.92   3.15  22.9     1     0     4     2
10 Merc 280     19.2    6  168    123   3.92   3.44  18.3     1     0     4     4
# ... with 22 more rows
```

VS.

```
> mtcars %>% as_tibble(rownames = "Model") %>%
+   pivot_longer(mpg:carb, names_to = "feature", values_to = "value")
# A tibble: 352 x 3
  Model      feature  value
  <chr>    <chr>    <dbl>
1 Mazda RX4 mpg      21
2 Mazda RX4 cyl       6
3 Mazda RX4 disp    160
4 Mazda RX4 hp     110
5 Mazda RX4 drat     3.9
6 Mazda RX4 wt     2.62
7 Mazda RX4 qsec    16.5
8 Mazda RX4 vs       0
9 Mazda RX4 am       1
10 Mazda RX4 gear     4
# ... with 342 more rows
```

+ add additional variables



tidyr

## Reshaping data

- Conversion to/from Tidy data where each column is a variable and each row is an observation.
- `pivot_longer()` converts to Tidy/long format.
- `pivot_wider()` converts to wide format.
- `tibble::column_to_rownames(var = "id_col")` converts to data frame (required for some functions).

```
> mpg
# A tibble: 234 x 11
  manufacturer model      displ  year   cyl trans      drv    cty   hwy fl    class
  <chr>         <chr>    <dbl> <int> <int> <chr>   <chr> <int> <int> <chr> <chr>
1 audi         a4        1.8  1999     4 auto(l5) f      18    29 p    compact
2 audi         a4        1.8  1999     4 manual(m5) f      21    29 p    compact
3 audi         a4         2    2008     4 manual(m6) f      20    31 p    compact
4 audi         a4         2    2008     4 auto(av) f      21    30 p    compact
5 audi         a4        2.8  1999     6 auto(l5) f      16    26 p    compact
6 audi         a4        2.8  1999     6 manual(m5) f      18    26 p    compact
7 audi         a4        3.1  2008     6 auto(av) f      18    27 p    compact
8 audi         a4 quattro  1.8  1999     4 manual(m5) 4      18    26 p    compact
9 audi         a4 quattro  1.8  1999     4 auto(l5) 4      16    25 p    compact
10 audi         a4 quattro   2    2008     4 manual(m6) 4      20    28 p    compact
# ... with 224 more rows
```

# Into the Tidyverse: important packages



stringr

## Character string manipulations

- `str_detect(x, pattern)` looks for match to the pattern; commonly used with `dplyr::filter()`
- `str_extract(x, pattern)` extracts the text of the match; commonly used with `dplyr::mutate()`
- `str_replace(x, pattern, replacement)` replaces the matches with new text; commonly used with `dplyr::mutate()`



forcats

## Managing factors

- R uses factors to handle categorical variables
- Often important to control the ordering of factors eg for plotting or modelling
- `fct_relevel()` changes the order of a factor as specified by a character vector
- `fct_inorder()` changes the order of a factor as specified by current order ; commonly used with `dplyr::arrange()`

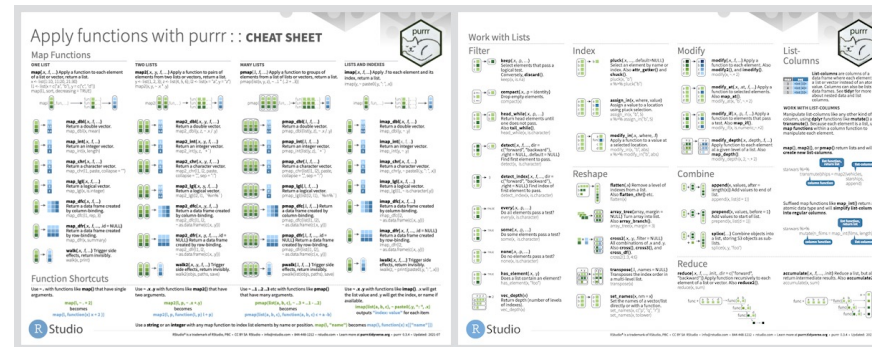


# Into the Tidyverse: important packages



## Functional programming tools for iterating with functions and vectors

- `map()` family of functions to replace for loops
  - see the [Iteration](#) chapter of R for Data Science to learn more
- <https://github.com/rstudio/cheatsheets/blob/master/purrr.pdf>



## Summarizes key statistical model information in tidy format

- `tidy()` summarizes information about model components.
- `glance()` reports information about the entire model.
- `augment()` adds information about observations to a dataset (eg residuals).
- Works with 100+ model objects.
- Plays well with the `nest/unnest` functions in `tidyr` and the `map` functions in `purrr`

[https://broom.tidymodels.org/articles/broom\\_and\\_dplyr.html](https://broom.tidymodels.org/articles/broom_and_dplyr.html)

# Into the Tidyverse: Visualization using ggplot2



## Publication-quality data visualization

- Implements a “grammar of graphics”
- Start by defining the data to be plotted (“aesthetics”):  

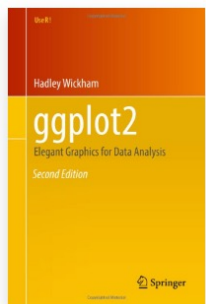
```
ggplot(aes(x, y, color, fill, shape, alpha, linetype))
```
- Then add layers (“geoms”) to specify how data is plotted, eg:  

```
+ geom_point()
```
- Add additional geom layers, eg:  

```
+ geom_boxplot()
```
- Can split into separate plots, eg male vs. female, by “faceting”:  

```
+ facet_wrap(~ Sex)
```
- Finally add title and modify theme:  

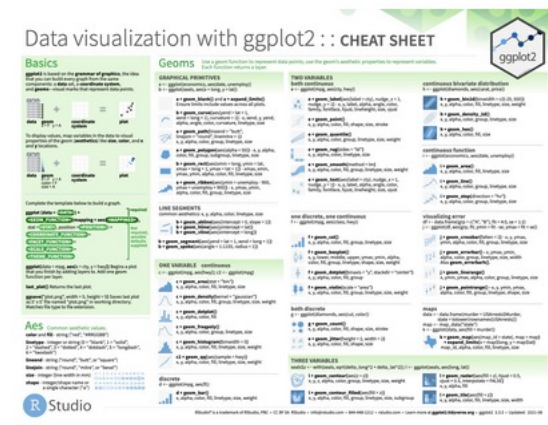
```
+ labs(title = "Plot title", subtitle = "plot details")  
+ theme(aspect.ratio = 1)
```



<https://raw.githubusercontent.com/rstudio/cheatsheets/main/data-visualization.pdf>

<https://www.data-to-viz.com/caveats.html>

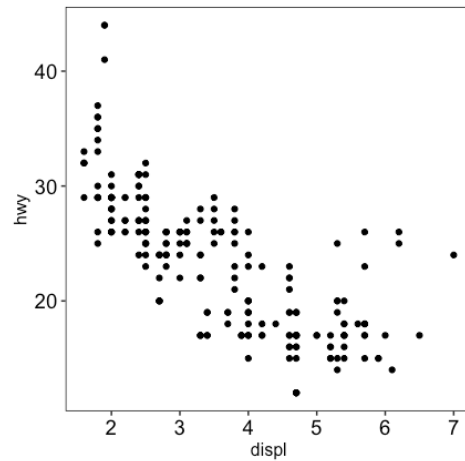
<https://r-graph-gallery.com/index.html>



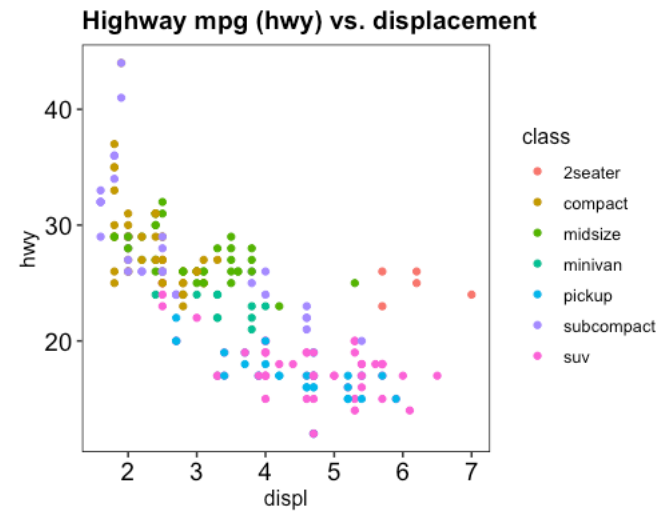
# Into the Tidyverse: Visualization using ggplot2



```
mpg %>%  
  ggplot(aes(displ, hwy)) +  
  geom_point()
```



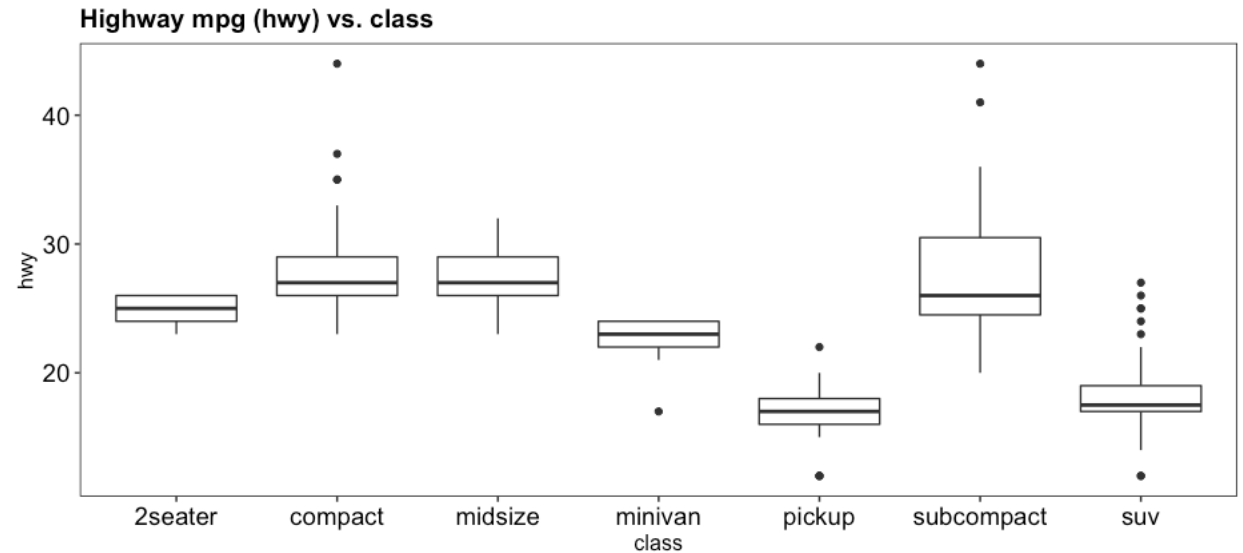
```
mpg %>%  
  ggplot(aes(displ, hwy, color = class)) +  
  geom_point() +  
  theme(aspect.ratio = 1) +  
  labs(title = "Highway mpg (hwy) vs. displacement")
```



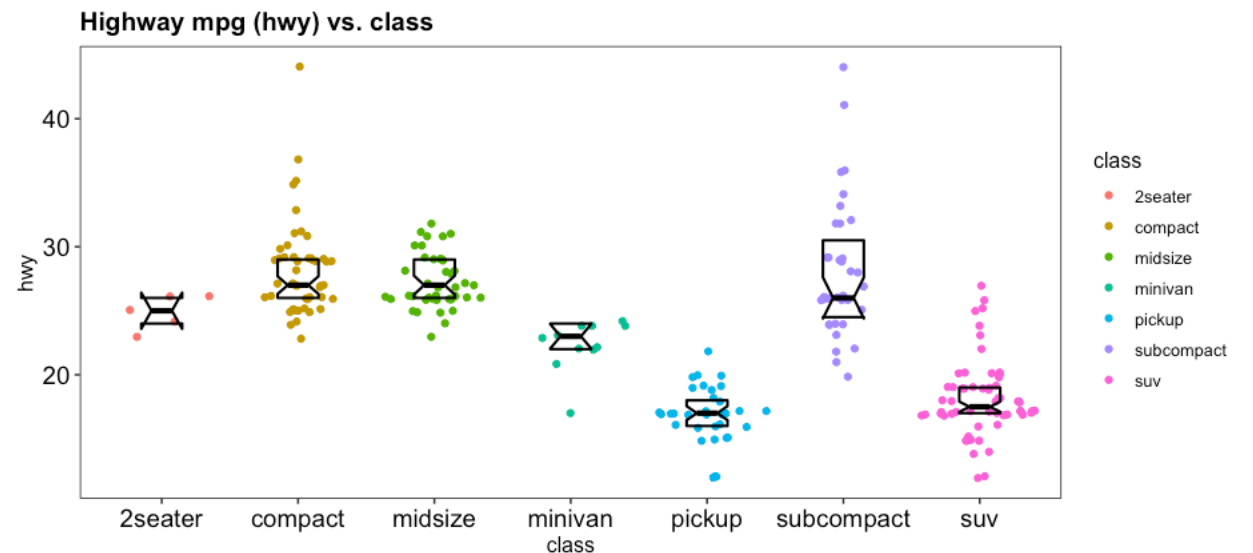
# Into the Tidyverse: Visualization using ggplot2



```
mpg %>%  
  ggplot(aes(class, hwy)) +  
  geom_boxplot() +  
  labs(title = "Highway mpg (hwy) vs. class")
```



```
mpg %>%  
  ggplot(aes(class, hwy, color = class)) +  
  ggforce::geom_sina() +  
  geom_boxplot(  
    notch=TRUE, varwidth=FALSE,  
    outlier.shape=NA, coef=FALSE,  
    width=0.3, color="black",  
    fill="transparent", size=0.75  
  ) +  
  labs(title = "Highway mpg (hwy) vs. class")
```

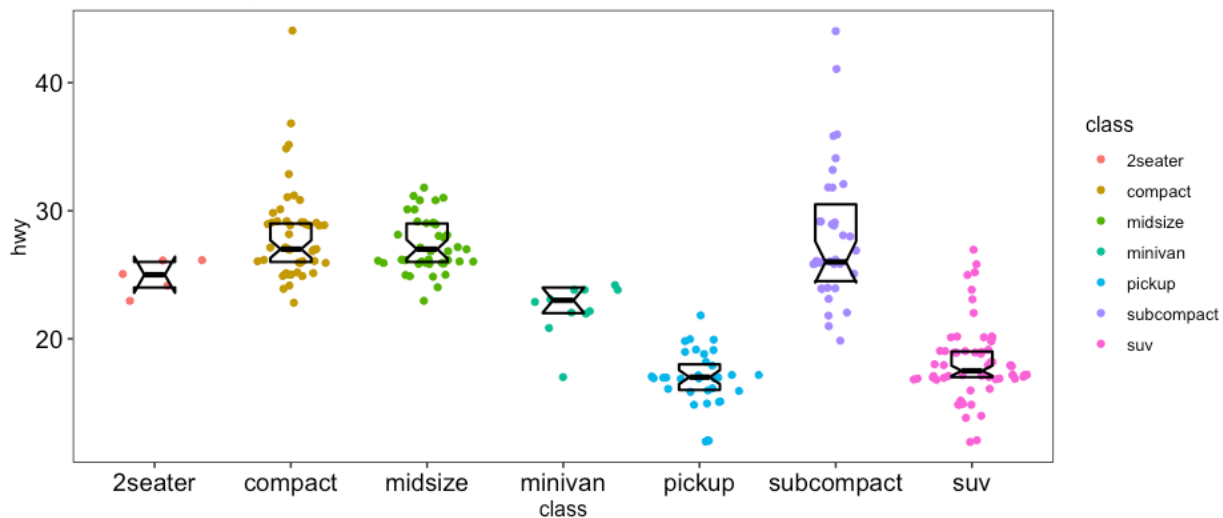


# Into the Tidyverse: Visualization using ggplot2

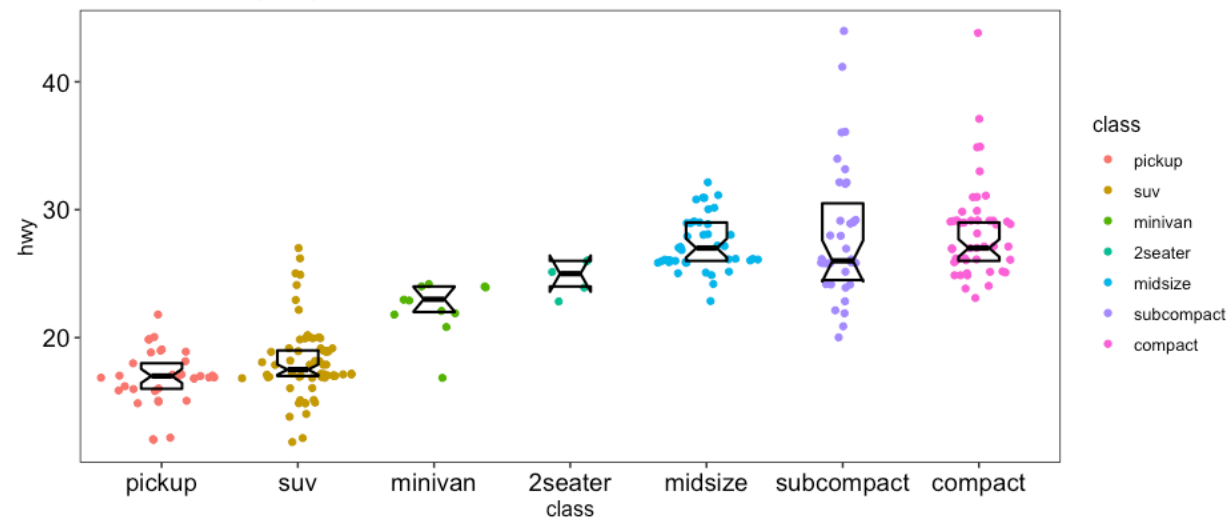


```
mpg %>%
  group_by(class) %>%
  mutate(mean = mean(hwy)) %>%
  ungroup() %>%
  arrange(mean) %>%
  mutate(class = fct_inorder(class)) %>%
  ggplot(aes(class, hwy, color = class)) +
  ggforce::geom_sina() +
  geom_boxplot(
    notch=TRUE, varwidth=FALSE, outlier.shape=NA, coef=FALSE, width=0.3, color="black", fill="transparent", size=0.75
  ) +
  labs(title = "Highway mpg (hwy) vs. class")
```

Highway mpg (hwy) vs. class

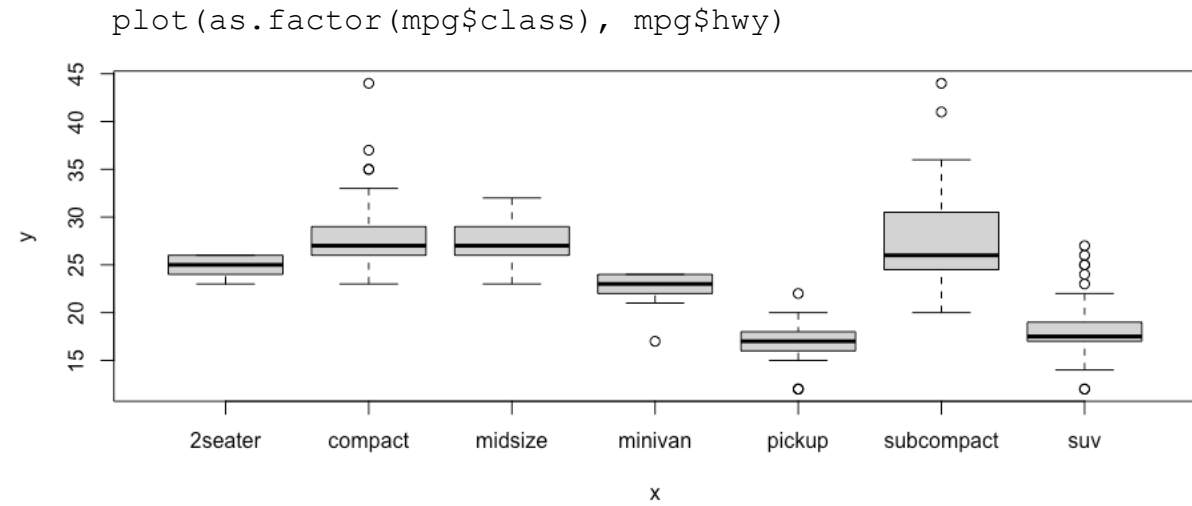
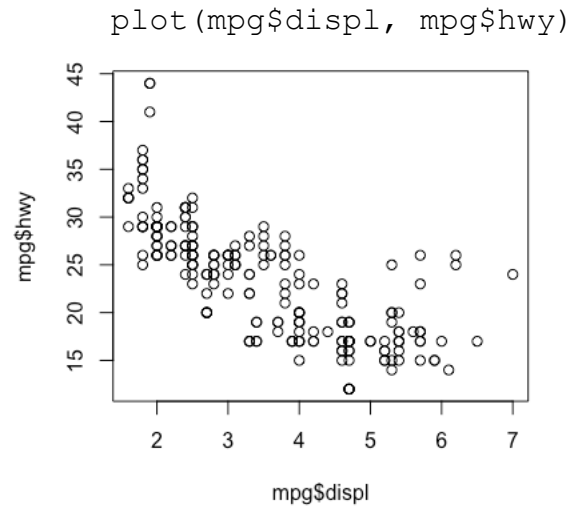


Highway mpg (hwy) vs. class



# Into the Tidyverse: Visualization using ggplot2

## Equivalent plots using base R



# Other commonly used packages

## **ggforce**

Additional plot geoms, including geom\_sina()

<https://ggforce.data-imaginist.com/>

## **ggrepel**

Repel overlapping text labels

<https://ggrepel.slowkow.com/index.html>

## **ggrastr**

Rasterize ggplot objects or layers

[https://cran.r-project.org/web/packages/ggrastr/vignettes/Raster\\_geoms.html](https://cran.r-project.org/web/packages/ggrastr/vignettes/Raster_geoms.html)

## **ggsignif**

Test and/or add significance brackets to plots

<https://cran.r-project.org/web/packages/ggsignif/vignettes/intro.html>

## **tidyheatmap**

Tidy interface to ComplexHeatmap

<https://cran.r-project.org/web/packages/tidyHeatmap/vignettes/introduction.html>

## **janitor**

cleaning variable names

<https://cran.r-project.org/web/packages/janitor/vignettes/janitor.html>

## **rstatix**

Tidy and pipe-friendly statistics, e.g. t-test, Wilcoxon/Mann-Whitney

<https://cran.r-project.org/web/packages/rstatix/index.html>

## **patchwork**

Assemble multiple ggplot objects

<https://patchwork.data-imaginist.com/index.html>