Getting Started with R for Mass Spectrometry Data Analysis

ASMS 2021: R Short Course
Saturday/Sunday October 30/31, 2021

The Team

Ryan Benz
Seer, Inc.

Jeffrey Jones
Caltech

N. Heath Patterson
Vanderbilt School of Medicine

Ryan Benz

Education

Chapman University	BS Chemistry, BS Mathematics
UC Irvine PhD	Computational Chemistry & Bioinformatics
UC Irvine	Post-Doc Inst. of Genomics and Bioinformatics

… discovered that lab work was not for me, data was going to be big!

Experience

Applied Proteomics Inc	Bioinformatics Scientist, Director
SoCal Bioinformatics Inc	Bioinformatics consulting, biotech & pharma
Seer, Inc.	Principal Scientist, Director of Data Science

Feel free to connect with me on LinkedIn: https://www.linkedin.com/in/ryan-benz/

Jeff Jones

Education

Cal Poly SLO	BA Microbiology, BA Biochemistry
Univ Arkansas PhD	Analytical Chemistry (FTMS)
UC Irvine	Post-Doc Inst. of Genomics and Bioinformatics

… learned programming and stats (bioinformatics) out of a need to analyze large data sets generated by mass spectrometry…

Experience

Applied Proteomics Inc	blood based biomarkers, colorectal cancer
SoCal Bioinformatics Inc	bioinformatics consulting
Caltech Proteome Exp Lab	bespoke proteomics and bioinformatics

Useful Code

Quantitative Proteomics Analysis Pipeline: *github.com/jeffsocal/keystone*

Nathan “Heath” Patterson

Education

Univ Wyoming	BS Chemistry
U de Montréal	Analytical Chemistry (Imaging Mass Spectrometry)
Vanderbilt	Post-Doc MS Research Center (Caprioli Group)

Took an interest in programming when vendor solutions no longer solved the problem

Experience

Vanderbilt University	*Current:* Faculty in Biochemistry - Research Instructor
Frontier Diagnostics, LLC	*Current:* Director of Bioinformatics

Some of my code

github.com/nhpatterson

Goals for the Short Course

- Learn basic fundamentals of the R programming language
- Learn how to use to the RStudio integrated development environment (IDE)
- Learn about tidy data, what it is, and why it's important for data analysis
- Learn basic fundamentals of the tidyverse ecosystem of R packages and how they can be used to streamline the data analysis process
- Learn how to make data visualizations using the ggplot2 R package

We'll Cover 3 Main Topics

1. Gettings started with R and RStudio (Heath)

2. Introduction to tidy data and the tidyverse (Ryan)

3. Introduction to data visualization in R with ggplot2 (Jeff)

R is a great language for data analysis!

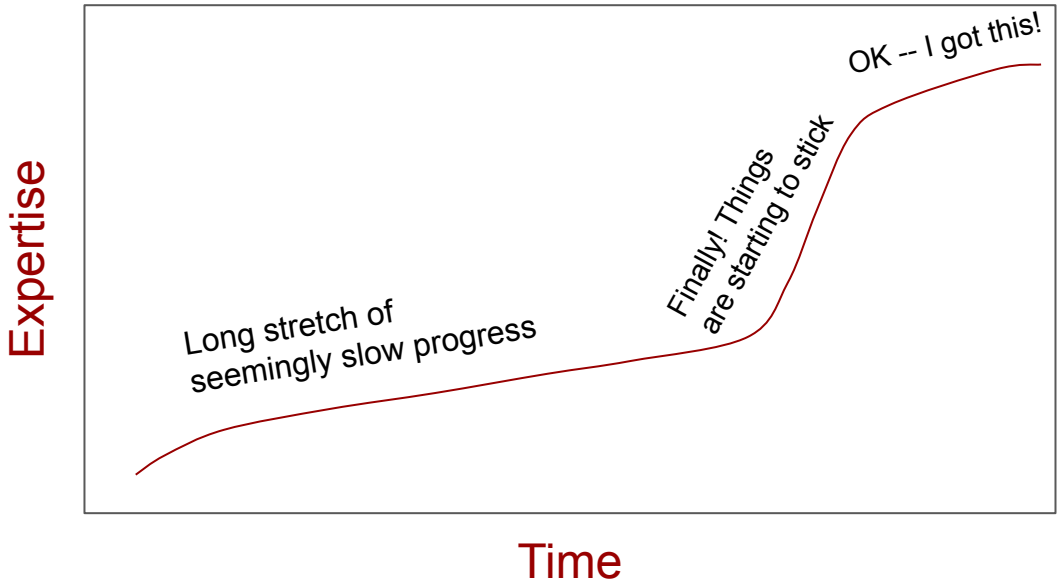
- Many programming languages are general purpose (can be used in any domain), e.g. C/C++, Java, Python
- R is *not* a general purpose programming language, it's a language specifically designed for working with data (that's what scientists do!)
- Because R is geared toward data, its design, structure and continued development is focused on making it easier to work with data
- R has become one of the top languages for data science, and it's popularity and usage continues to grow
- For scientists, R is a great tool to learn

But what about Python?

- Python is a great language, and is another top language for data science
- If you want to become a data scientist or work heavily with data, learning both R and Python is a great idea – both have pros and cons
- But, if you choose Python over R, that’s not a bad choice either
- R might work better for some, Python for others
- If R just isn’t working for you, give Python a shot

The R Learning Curve

10 years ago...

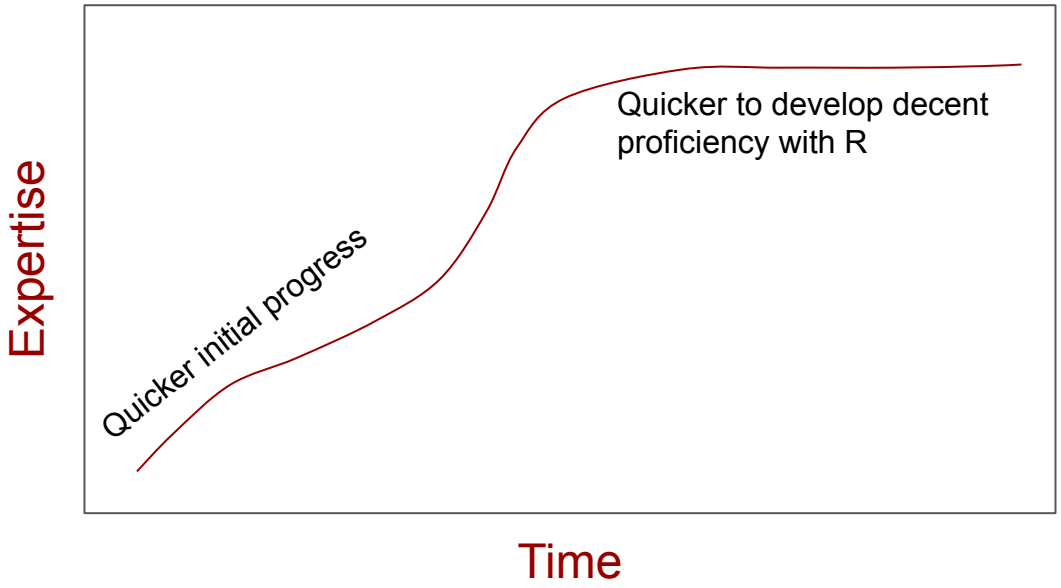


What made R challenging to learn 10 year ago?

- Fewer R resources
- Less mature
- Less interest
- Fewer people in the community
- Data science wasn't "a thing" yet

The R Learning Curve

Today...



Why is R easier to learn today?

- More (high quality) R resources
- Very mature, lots of amazing tools in R
- Lots of interest
- Lots people in the community, and the community is great
- The data science “revolution” has pushed R to develop and evolve, become more user-centric

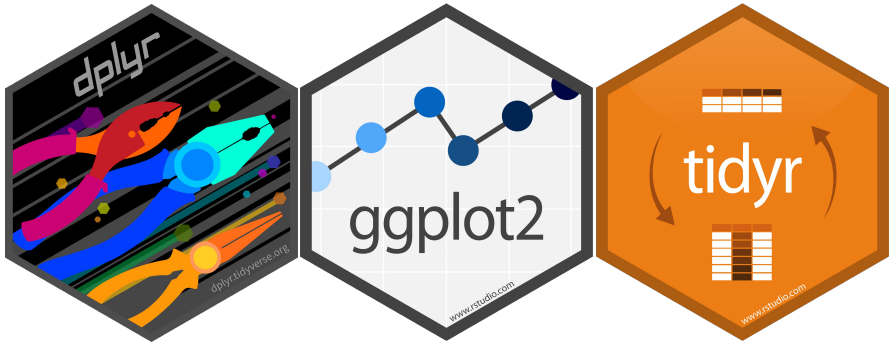
R is still challenging to learn – but it’s worth it!

- R is a quirky language, but the quirks are often there for a reason
  (to make working with data easier)
- R can be frustrating for beginners, and the quirks don’t help
- For many, R might be initially harder to learn than other languages
  (e.g. Python)
- But, once R starts to stick, its data-centric capabilities and tools really shine!

Thoughts about learning R and how to code

- The first step to learning a programming language is to learn its *syntax*
  (the set of rules and symbols that make up structurally correct code)
- Computers are *really* picky, and even the smallest violations of the syntax rules will result in code that doesn't run:
  typos, incorrect names, missing spaces or too many spaces, wrong brackets, …
- Syntax errors are frustrating, and unfortunately, the most common mistake for beginners
- Hang in there, start simple, try to understand very simple cases first, then build and expand on them

The R ecosystem is vast and awesome!



RStudio Desktop

Shiny
Interactive web apps for data

RMarkdown, Blogdown, Bookdown
Reproducible reporting, blogs & books

At the end of the course, are you able to...

- start-up RStudio and make an RStudio project?
- read a formatted text file (e.g. csv file) into R?
- understand basic properties about the data (e.g. # rows, cols)?
- tell someone what tidy data is and why it’s important?
- perform some basic data manipulations and operations on the data?
- make a simple plot from the data?

Schedule

Saturday

Start Time	End Time	Durration	Session
9:00 AM	9:15 AM	15 min	Short course introduction
9:15 AM	10:15 AM	1 hr	Getting Set-up
10:15 AM	10:45 AM	30 min	Coffee Break
10:45 AM	11:30 AM	45 min	Introduction to R and R Studio
11:30 AM	12:00 PM	30 min	Practice session: RStudio Projects
12:00 PM	1:00 PM	1 hr	Lunch
1:00 PM	1:45 PM	45 min	Basic R Data Structures
1:45 PM	2:15 PM	30 min	Reading Data into R
2:15 PM	2:30 PM	15 min	Practice session: Reading & Working with Data
2:30 PM	3:00 PM	30 min	Coffee Break
3:00 PM	3:30 PM	30 min	Useful R Functions
3:30 PM	4:00 PM	30 min	Practice session: Working with R Functions
4:00 PM	5:00 PM	1 hr	Practice session: MS Data Exercises

Schedule

Sunday

Start Time	End Time	Durration	Session
9:00 AM	9:30 AM	30 min	Recap, Day 2 Overview, Review of MS Data Exercises
9:30 AM	10:15 AM	45 min	Introduction to Tidy Data & the Tidyverse
10:15 AM	10:45 AM	30 min	Coffee Break
10:45 AM	11:30 AM	45 min	Data Manipulation with dplyr
11:30 AM	12:00 PM	30 min	Practice session: dplyr examples
12:00 PM	1:00 PM	1 hr	Lunch
1:00 PM	1:15 PM	15 min	Practice session: dplyr examples solutions
1:15 PM	2:00 PM	45 min	Visualization Basics with GGplot2
2:00 PM	2:30 PM	30 min	GGplot2 Extended Syntax
2:30 PM	3:00 PM	30 min	Coffee Break
3:00 PM	3:30 PM	30 min	Practice session: GGplot2 examples
3:30 PM	4:00 PM	30 min	Wrap-up & Next Steps
4:00 PM	5:00 PM	1 hr	General Questions, Office Hours

Logistics

- Feel free to ask questions any time, or grab us at breaks
- As the presenter is presenting, the other team members will be circulating to help with questions or problems
- We'll save the end section of each day for practice exercises and to help with specific questions you might have
- We’ll provide ways to follow-up with us after the course in case your questions aren’t answered during the course, or you have new ones afterwards

Logistics

- Course materials (slides, code, additional info) are on a GitHub repository

https://github.com/ZenBrayn/asms-2021-r-shortcourse

Getting Set-up: Installing R & RStudio

ASMS 2021: R Short Course

Overview: Installing R & RStudio

1. Install R: https://www.r-project.org/
    a. Installers available for Windows, Mac, Linux
    b. (optional): install RTools (Windows) or XCode (Mac)

2. Install RStudio: https://www.rstudio.com/
    a. RStudio Desktop, free version

3. Install R Packages

We'll cover this live during the course

What are R Packages?

- R packages wrap-up code written by others so you can use it in your own projects

- Most R packages provide new functions to deal with specific problems, e.g.
  - ggplot2: helps you make plots
  - mzR: read mass spectrometry data files
  - twitteR: access twitter data

- Packages can be written by anyone, and quality can vary widely

- There are lots of high quality packages, but also be wary of random packages you come across on the internet

Where to get R packages

- CRAN (Comprehensive R Archive Network)
  - Main (and default) repository for R packages
  - Maintainers review packages for basic structure and utility before they are approved (is the package structured properly? can it build and load without problems?)
  - Don't assume that just because a package is on CRAN that is "does the right/correct thing"

- Bioconductor
  - Alternative R package repository for bioinformatics related packages
  - Lots of MS related packages
  - Uses a different installation procedure than CRAN

- GitHub
  - Anyone can host an R package from GitHub -- this is a Pro and a Con
  - Faster way to get your code out to others (no review or submission process)
  - Newer package versions are often on GitHub before they make it to CRAN

Notes on RTools and Command Line Tools

- Most R packages have binary versions that can be directly installed with no additional steps
- If binary versions are not available (e.g. the package is new or just been updated), it may need to be compiled before it can be installed
- RTools (Windows) and Command Line Tools (Mac) provide the necessary programs and libraries you'll need to compile packages
- But… package compilation sometimes doesn't work (for a variety of reasons) and can be frustrating!
- In general, stick with using binary packages

RStudio Cloud - cloud hosted RStudio in a browser

- A cloud hosted version of RStudio, runs in a web browser
  - Same RStudio interface
  - Can run on any device with a modern web browser

- Sign-up for a free account at: https://rstudio.cloud/
  - 25 hrs/month
  - Paid plans with more CPU hours available

- We'll use this as a back-up if anyone has issues getting set-up on their own computer

Introduction to R and RStudio

ASMS 2021: R Short Course

Goals for this module

- Find out where to download R and RStudio
- Become familiar with the RStudio layout
- Learn how to execute lines of code in RStudio from the editor or the console
- Learn about projects and code organization

Why use R to analyze your data?

- Current software solutions do not do exactly what you would like
- Reproducibility
- Control over all aspects of the analysis

Why RStudio?

- **Built for purpose R IDE *(interactive development environment)***

    1. Great for interactive data reading, analysis and plotting
    2. Can help you organize your project and workflow
    3. Also great for developing advanced R programs and packages

https://www.r-project.org/

https://rstudio.com/

Download and installation

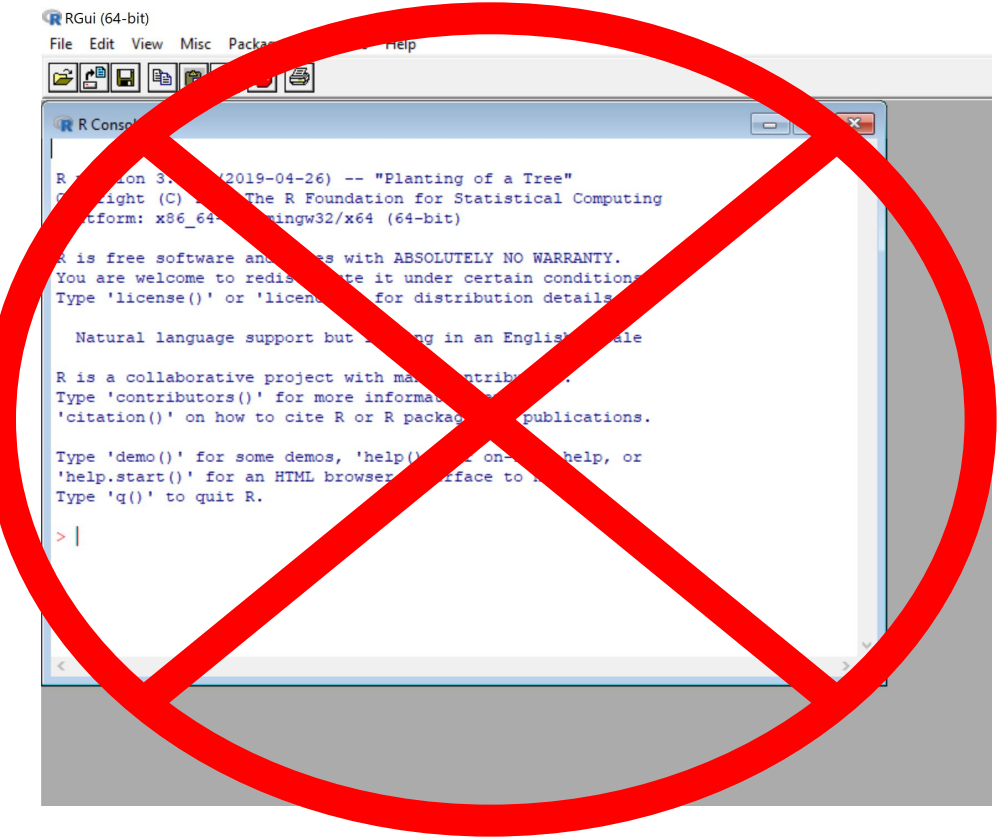
- **Download and install R**
  - The engine of the R language
  - https://www.r-project.org/

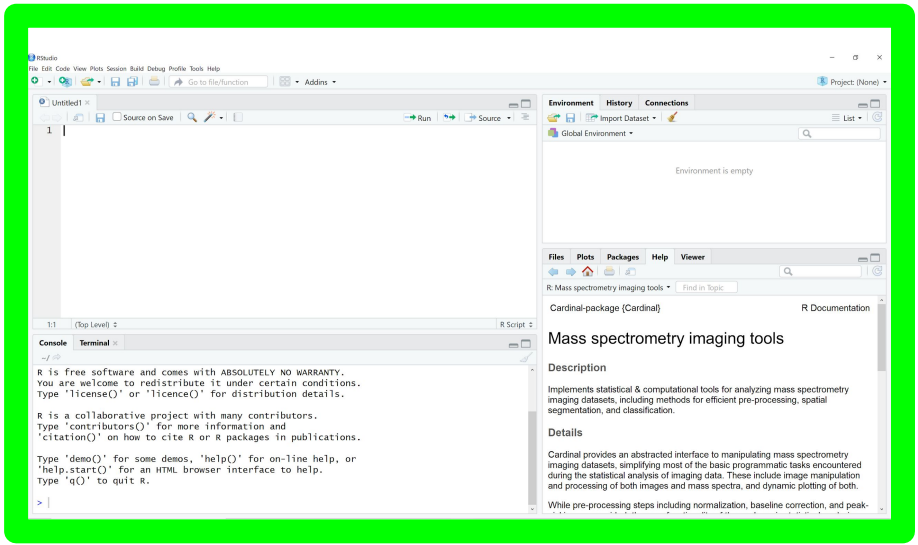
https://www.r-project.org/

- **Download RStudio**
  - The best solution to editing, managing, and running your code
  - https://rstudio.com/

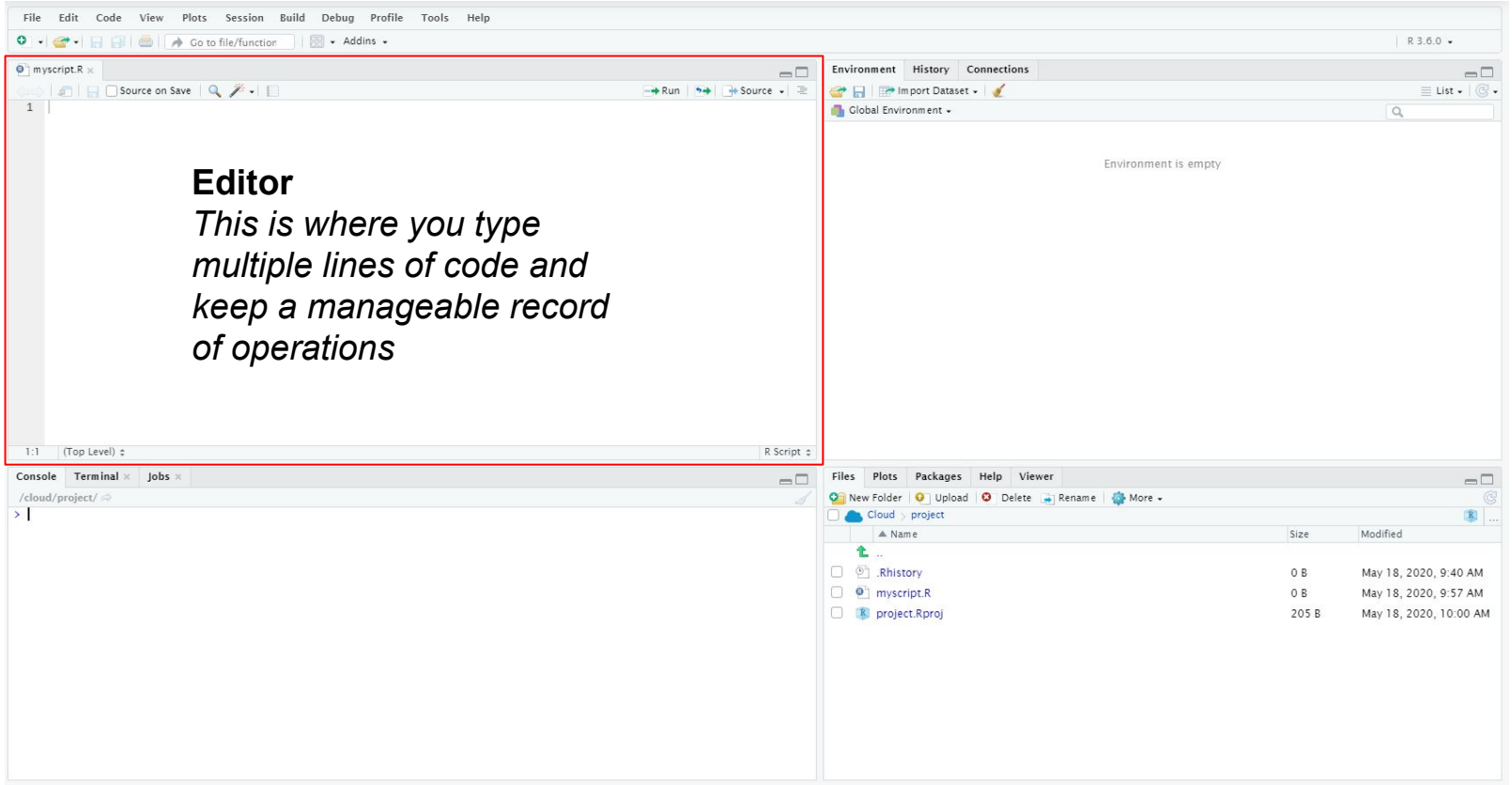
https://rstudio.com/

Opening R vs. opening RStudio IDE

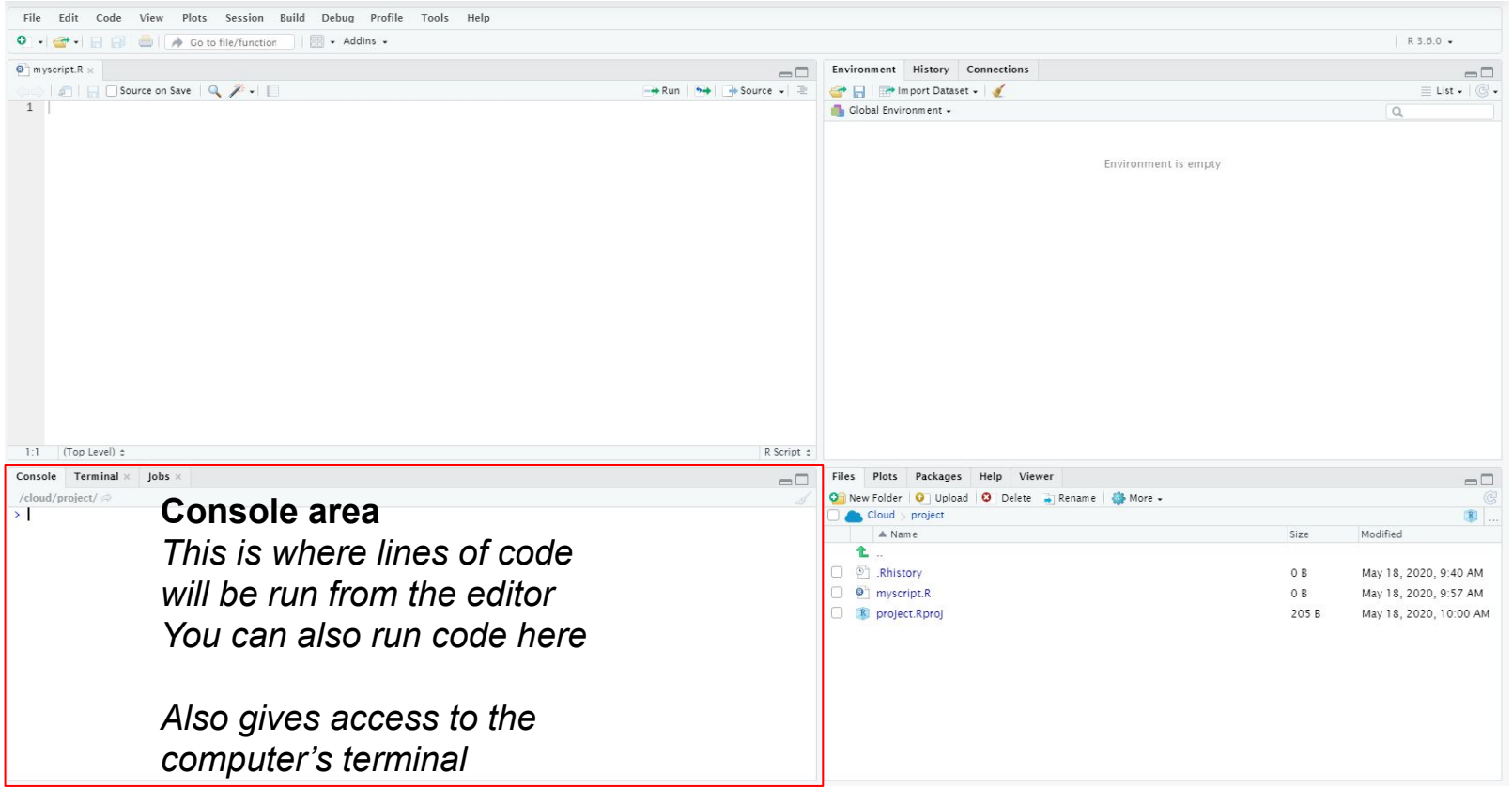




The RStudio pane layout



The RStudio pane layout

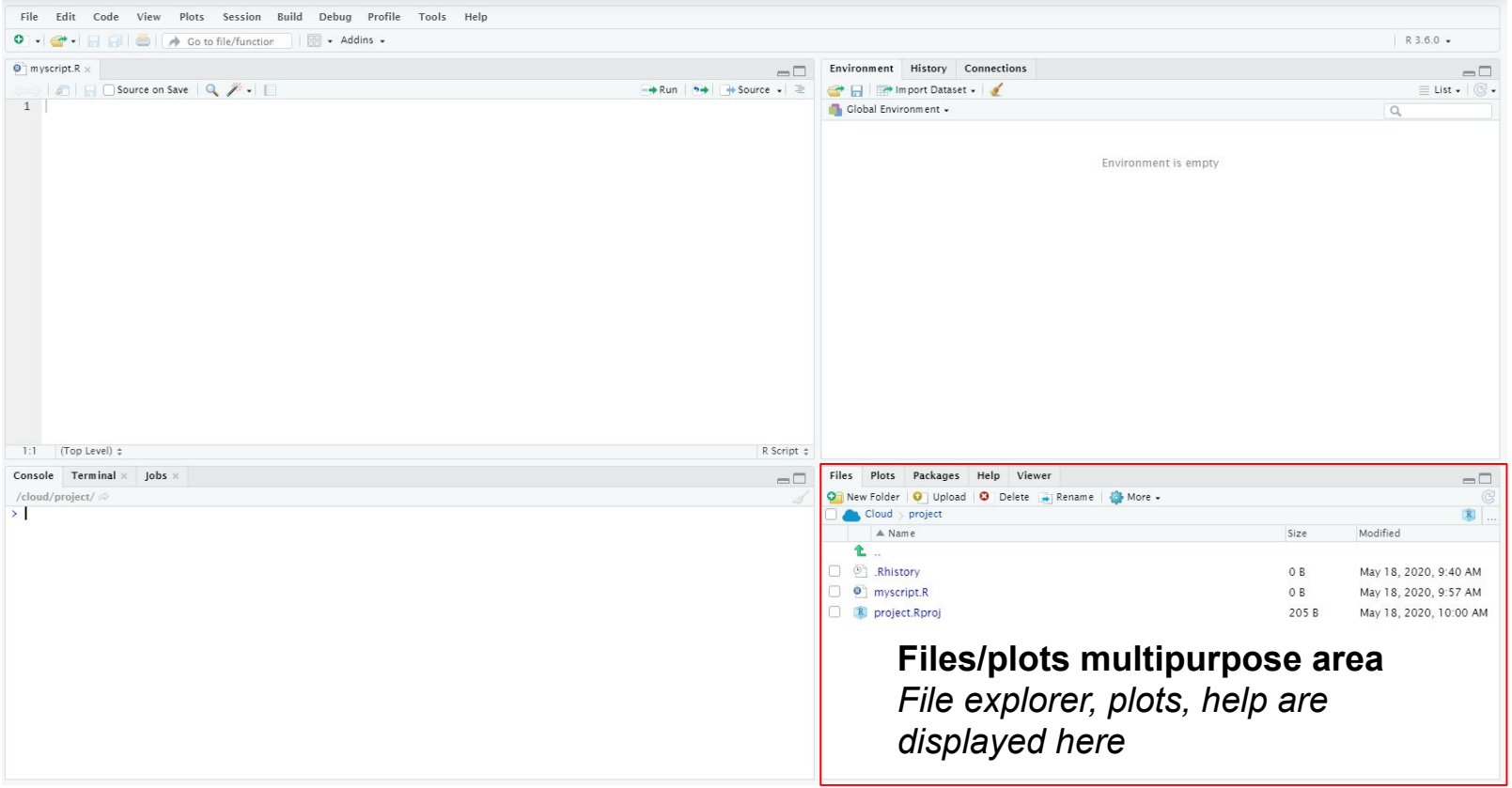


Console area

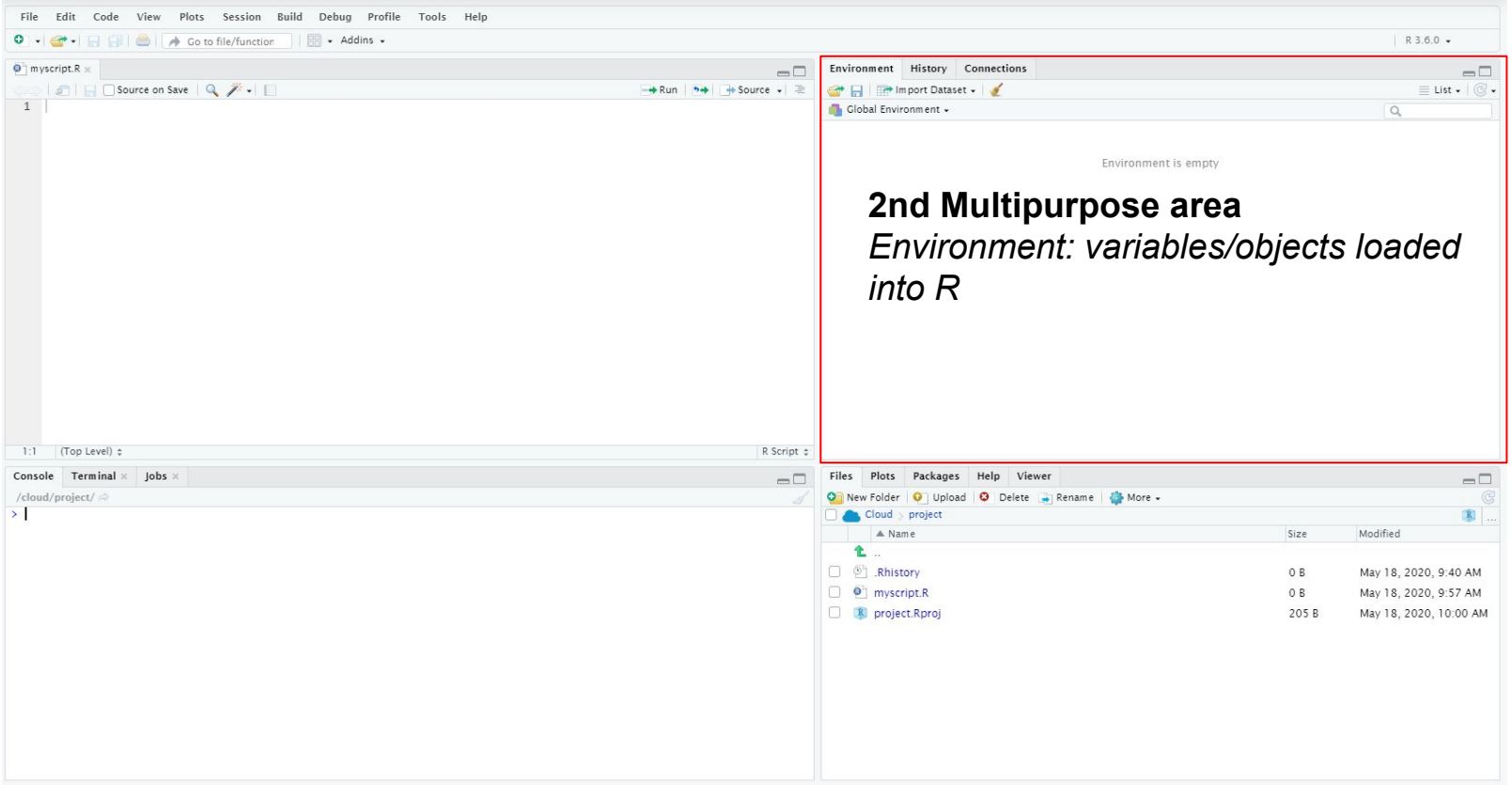
This is where lines of code will be run from the editor
You can also run code here

Also gives access to the computer's terminal

The RStudio pane layout



The RStudio pane layout



Running lines of code in RStudio

- **Run from the editor *(recommended)***
  1. Type in the code in the top-left pane

     ```
     print("Hello world from the editor!")
     ```

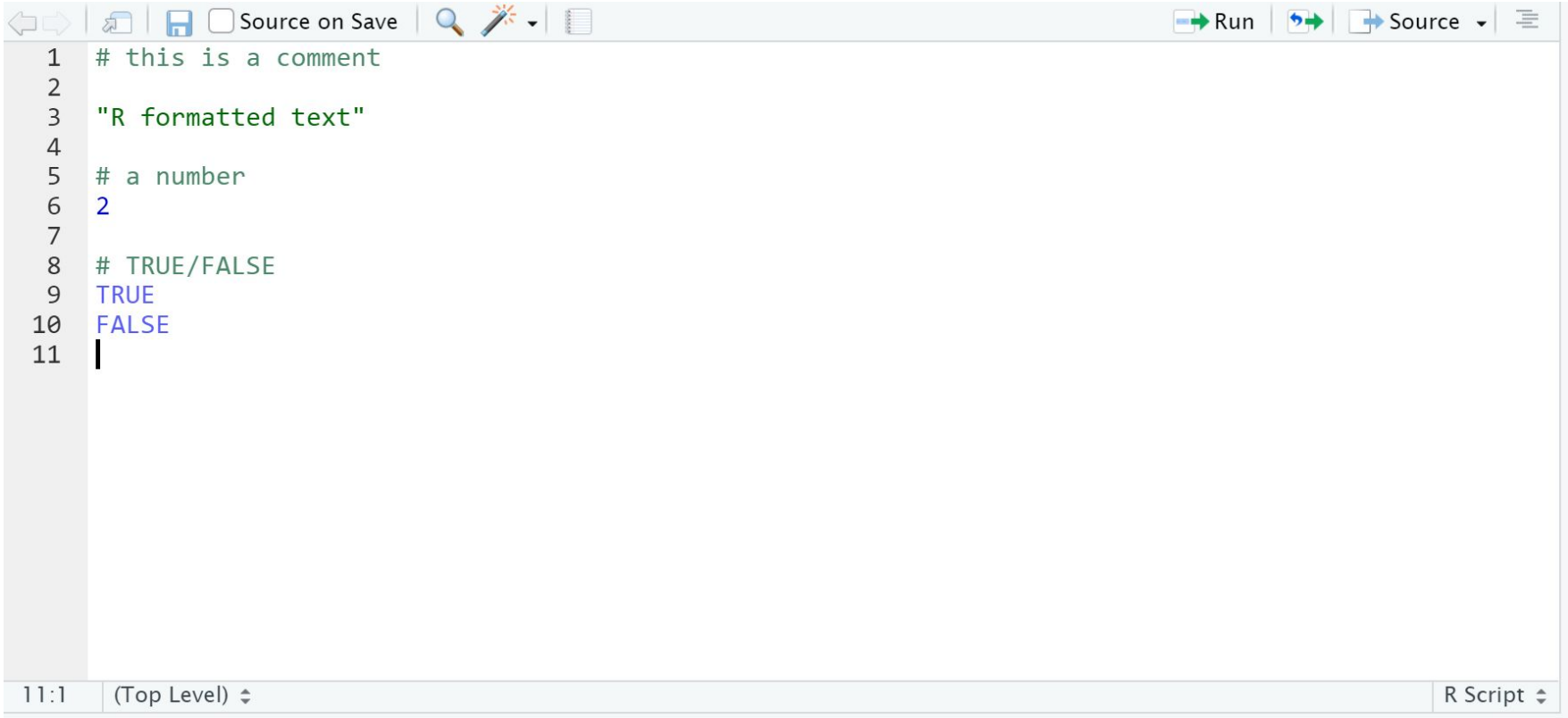
  2. Put editor cursor anywhere on that line
  3. Press Ctrl/CMD+Enter.
  4. Multiple lines: highlight multiple lines then press Ctrl/CMD+Enter

- **Run from the console (*occasionally)***
  1. Type code into console after the '>'

     ```
     print("Hello world from the console!")
     ```

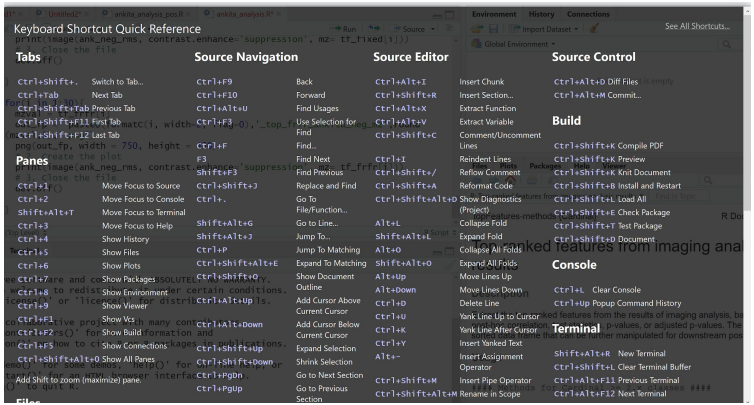
  2. Press Enter.
  3. Multiple lines, not advised, but copy and paste multiple lines into console then press Enter.

Important things to know...



Getting RStudio the way you like...

- Set Theme: Tools → Global options → Appearance
- Text size (on local RStudio): View → Zoom in/ Zoom out
- Keyboard shortcut cheat sheet: alt+shift+k



[RStudio Cheat Sheet]

Setting up your working directory (where files will go when you save, by default)

- This determines the default output for where files will go
  - `getwd() # print the current working directory`
  - `setwd("/directory") # file path to the where you'd like the working directory to be`

- Acceptable file path formatting:
  - iOS/Linux/Windows : "/data/myfolder/"
  - Windows: "C:\\data\\myfolder\\"

- WARNING: Windows style file paths with "\" separating directories will not work in R!

- Unacceptable file path formatting:
  - Windows: "C:\data\myfolder\"

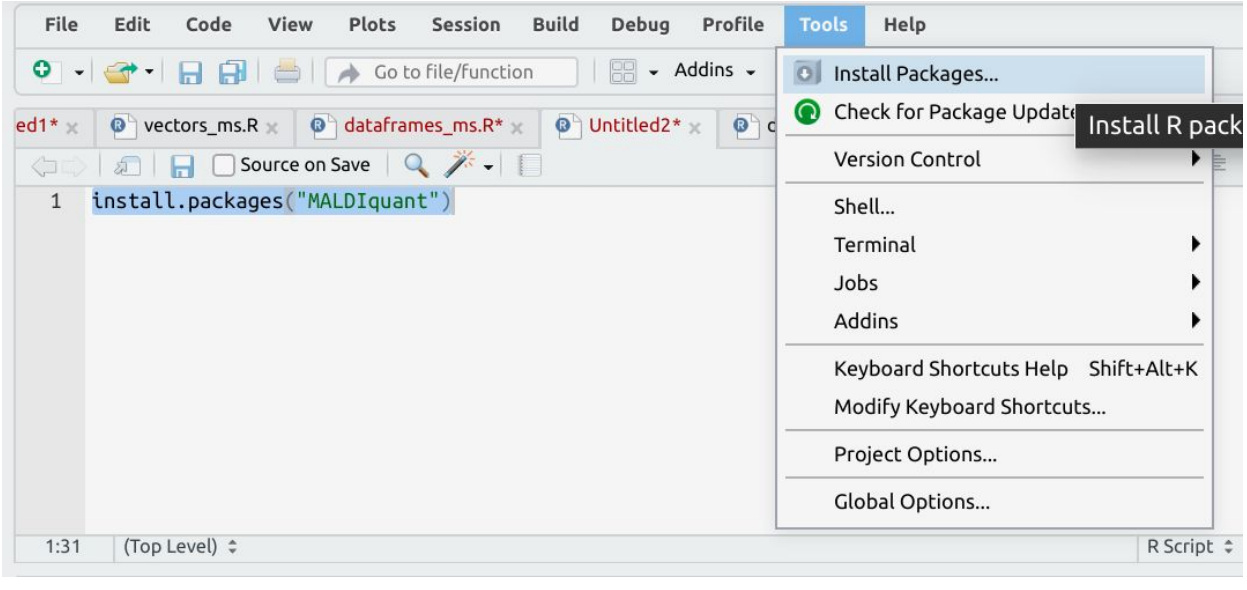
R packages: “fundamental unit of shareable code”

- Most packages are stored and downloaded from CRAN
  (**C**omprehensive **R** **A**rchive **N**etwork)

From Editor/Console

```
install.packages("MALDIquant")
```

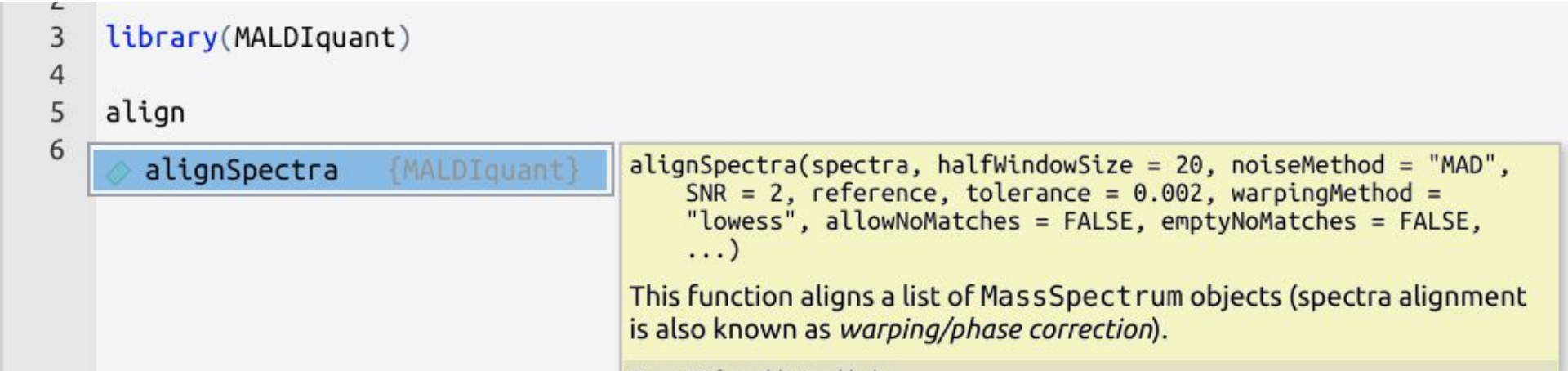
From RStudio Tools → Install Packages...



R packages: loading/using installed packages

- When you load a package, all of its functionality is available without explicit reference to the package…

- Example: `library(MALDIquant)` RStudio will auto-complete functions in MALDIquant



Managing your R code with RStudio Projects

- Projects are highly recommended as they aid to organize and store your R scripts and data

- Starting a project *File → New project... → New directory → New project* then name the project and a directory will be created and RStudio will start working from this directory

- Tip: In a project you may want separate subfolders and plan to have a structure to all of the data

Managing your R code with RStudio Projects

1. Create a new RStudio Project

2. Assemble the required input data & info you need, put them inside the project directory*

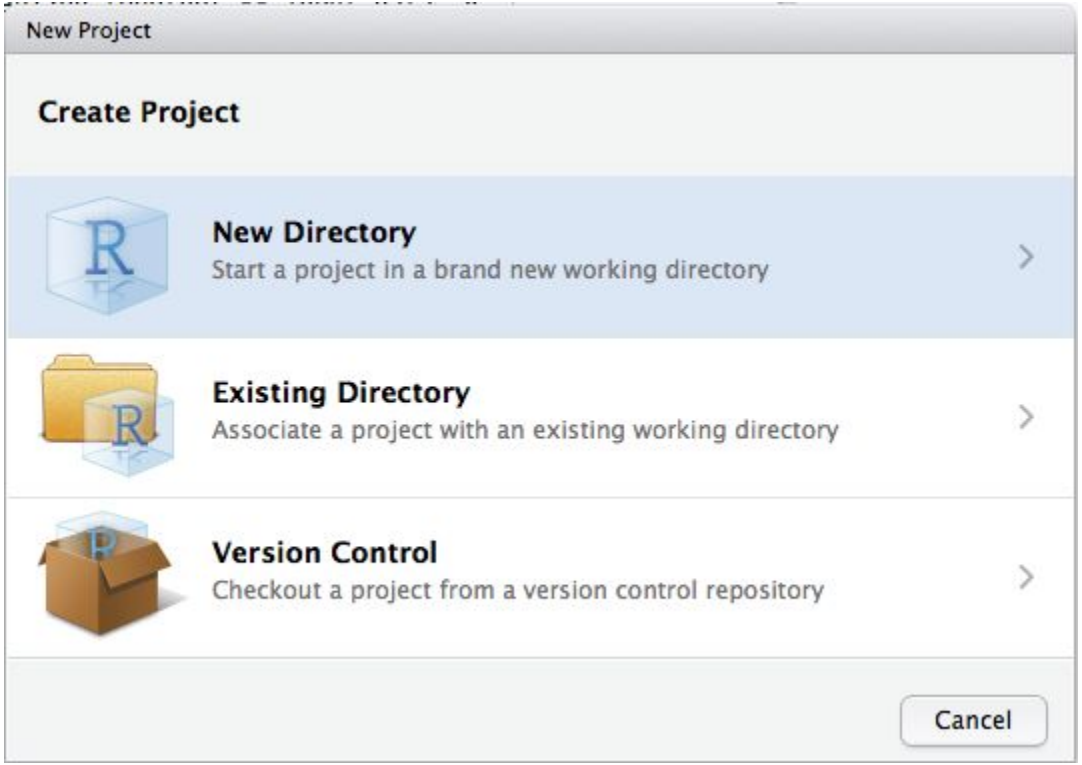
3. Write analysis scripts that do the work; consider naming them with a numeric prefix (e.g. 01_…, 02_…) indicating the order they should be executed in

4. Write, test, write test, write, test…

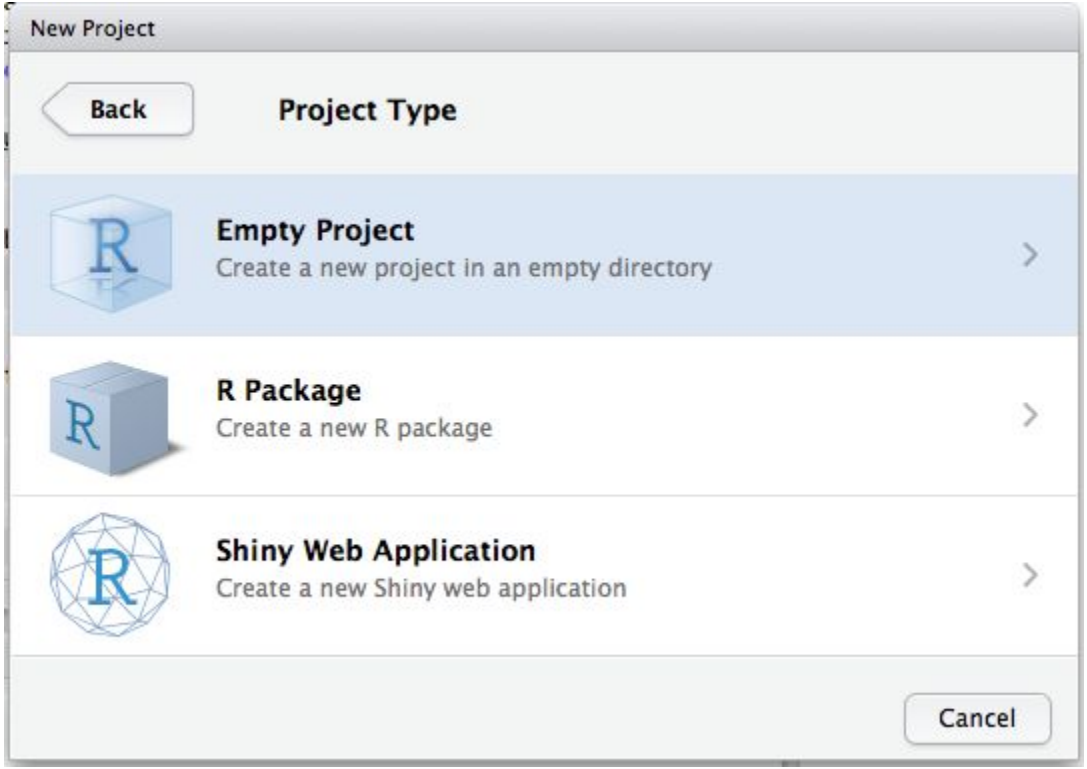
5. Double click the .Rproj file when you want to get back to work

* can depend on how you store data (e.g. networked storage)

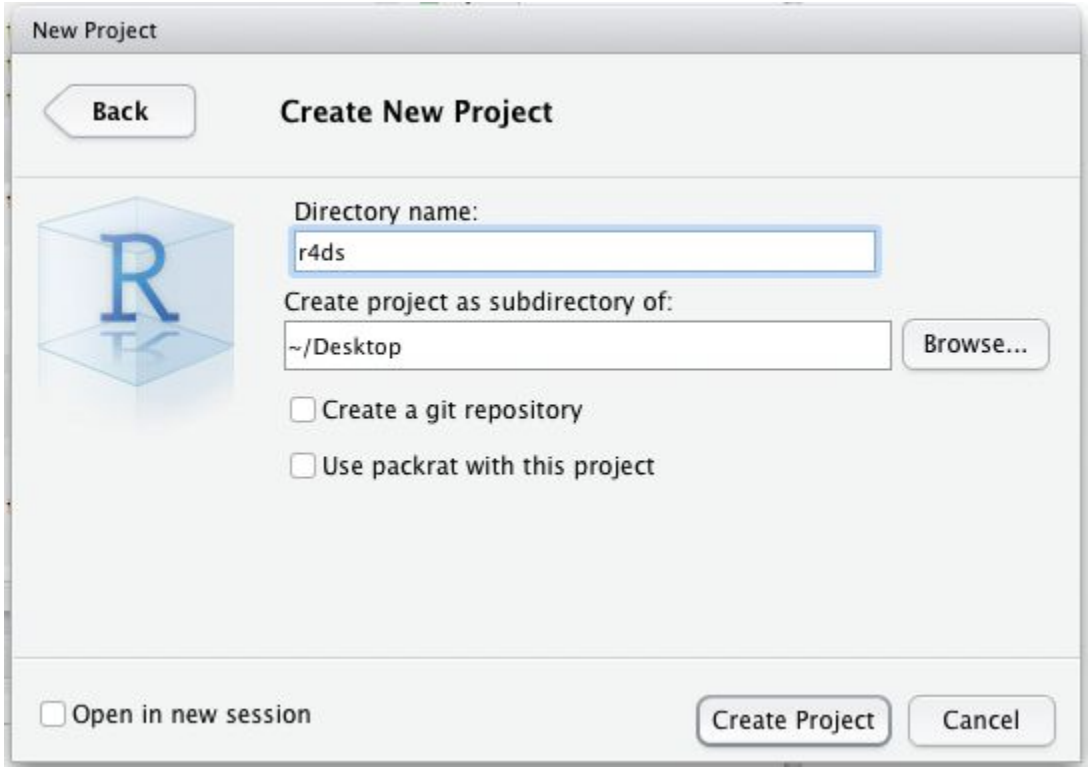
Setting up a project in RStudio



Setting up a project in RStudio



Setting up a project in RStudio



Basics: R as a calculator

- Certain symbols, called operators, do math in R

```
# Addition
10 + 2

# Subtraction
10 - 2

# Multiplication
10 * 2

# Division
10 / 2

#Exponential
10 ^ 2
```

```
# Addition
10 + 2
[1] 12

# Subtraction
10 - 2
[1] 8

# Multiplication
10 * 2
[1] 20

# Division
10 / 2
[1] 5

#Exponential
10 ^ 2
[1] 100
```

Defining a variable

- **Define and use a variable**
  1. Type in and run code in the editor

```
hw_text <- "Hello world stored in a variable!"
```

Stored variable

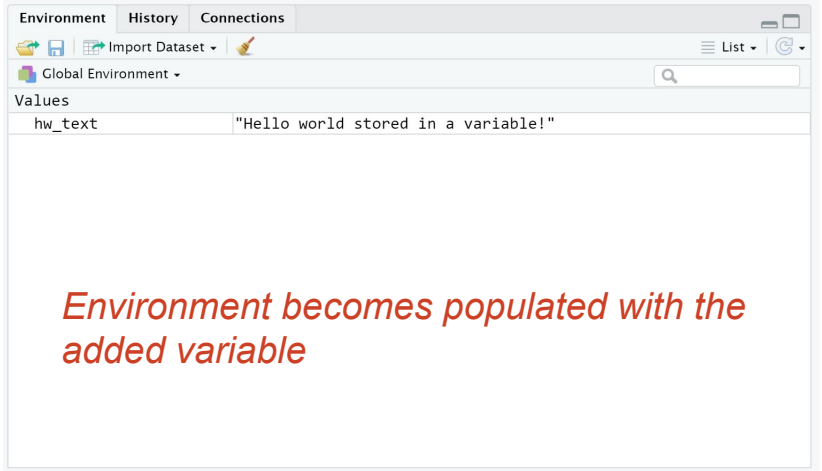
- Name given by user
- Can't lead with a number

Assignment operator

- Assigns text to object, can also be =

Data to be stored

- Hello world text



  2. Run previous print function on variable

```
print(hw_text)
```

Where to get help when you are stuck...a primer

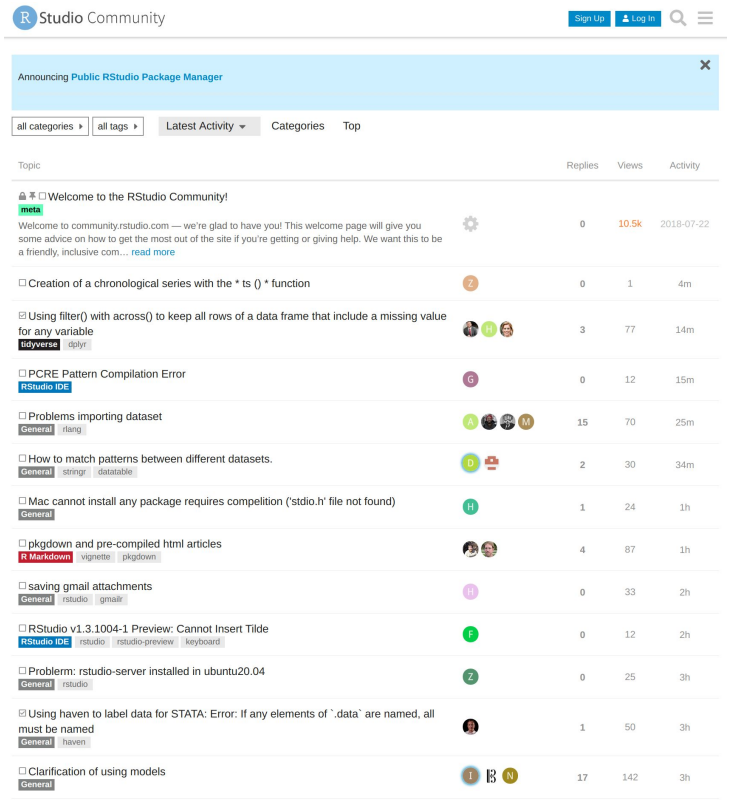
- *First and foremost…don't worry...*a lot or most of the questions you have that are specific to the R programming language or it's packages have been answered.

- The most important part of getting help is asking the right question… This is why it is important to understand the basics we will overview in the next two sections as well as the terms.

- Once you have these in mind, you can google your issue!

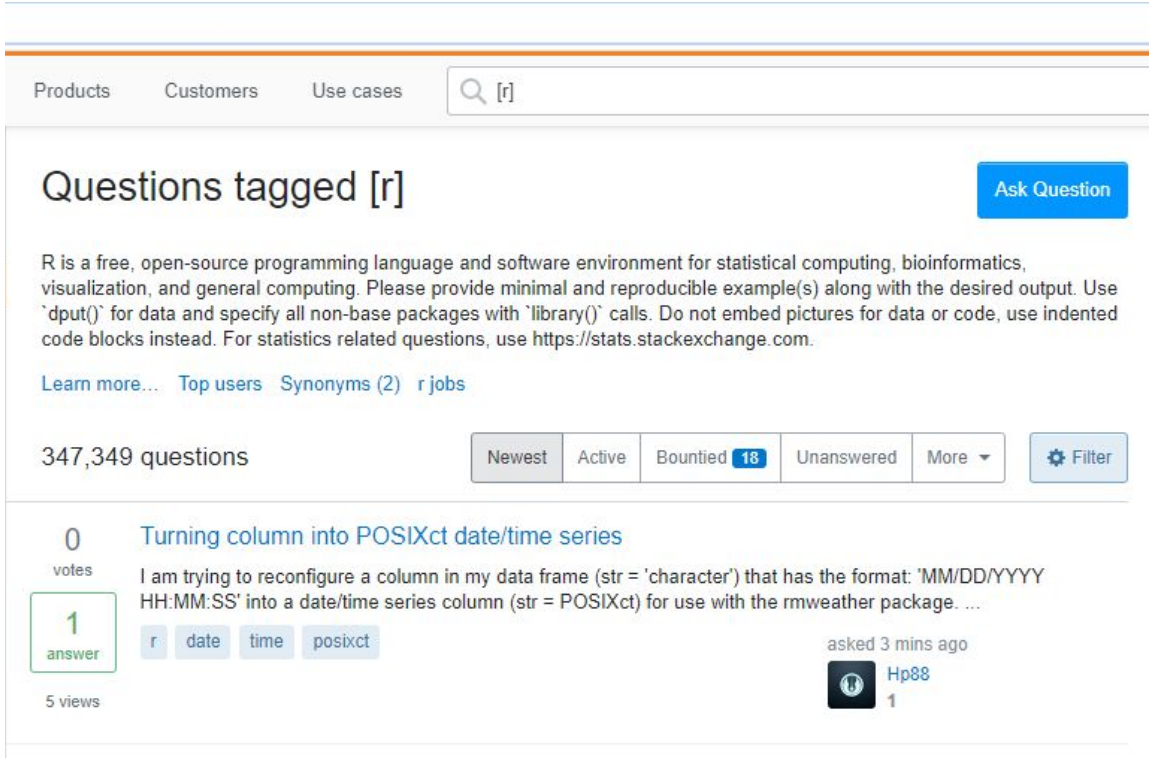
Getting help: [RStudio Community]

- RStudio community is forum with many many users and questions answered, it is especially friendly to new users



Getting help: >347,000 R questions on 'R' stackoverflow.com

- Stackoverflow.com, more extensive but possibly less friendly to newcomers



Helpful RStudio tips: stopping code execution

Situation: I've made a mistake in my code but I have already started executing it

Solution: Use the stop button

This will stop the execution.

Warning: sometimes this will require a restart of the R console

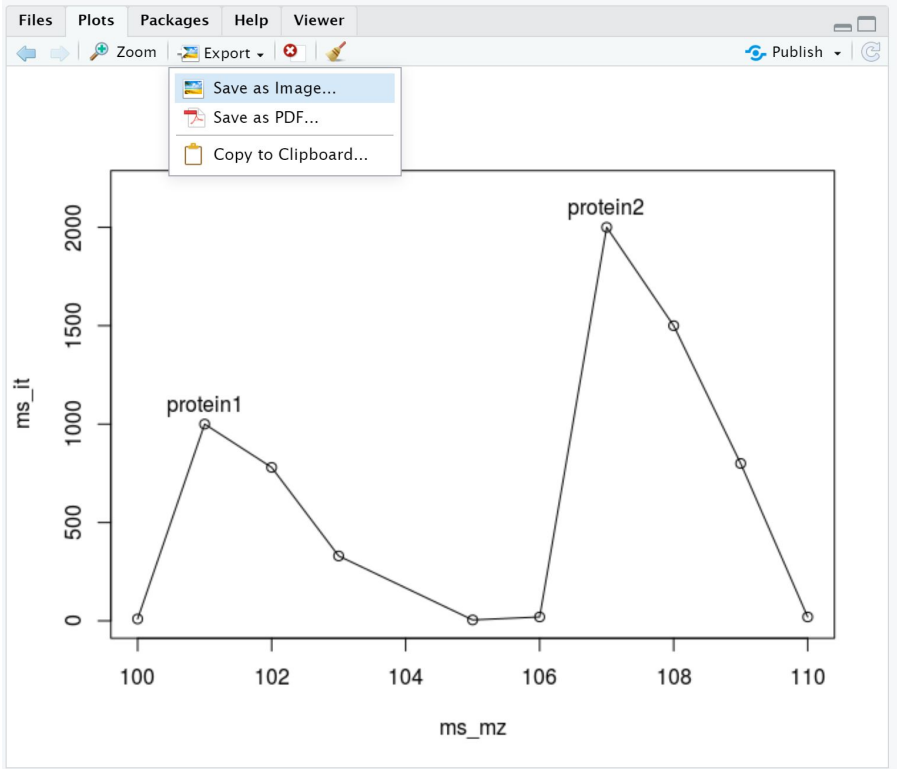
Helpful RStudio tips: exporting plots

Situation: Need high quality figure for publication

Solution: Use the Plot pane's *Export* button

You will have options to export as PDF, image files, vector graphics, etc.

Tip: Check journal specifications on figures and export them at EXACTLY the right size/resolution and you may mitigate submission issues! (maybe, definitely maybe, no guarantees...)



Basic R data structures: vectors and data.frames

ASMS 2021: R Short Course

Goals for this module

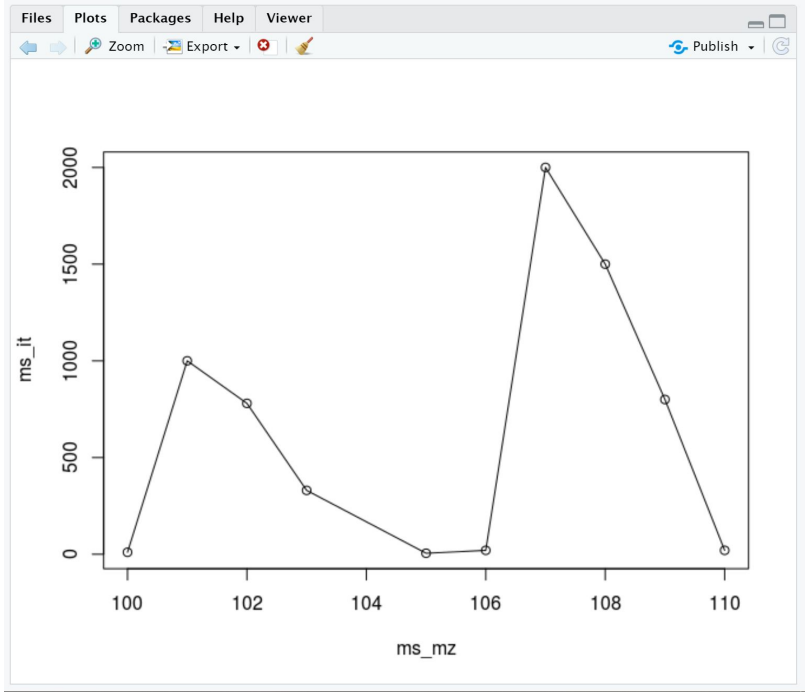
- Learn fundamentals of R vectors: how to construct and use them
- Learn fundamentals of R data.frames: how to construct and use them
- Learn why these data structures are so useful and how they permeate more advanced R functionality in the tidyverse

Defining vectors : pseudo mass spectrum I

- A mass spectrum is composed of *m/z* values and their corresponding intensity values. We can define these as vectors

```
# First we create the m/z vector using c()
# c() "combines" elements into a vector

ms_mz <- c(100,101,102,103,105,106,107,108,109,110)

# Intensity vector

ms_it <- c(10,1000,780,330,5,20,2000,1500,800,20)
```

- The mass spectrum *m/z* and intensity are 'numeric' vectors, it contains numbers, each value is an element



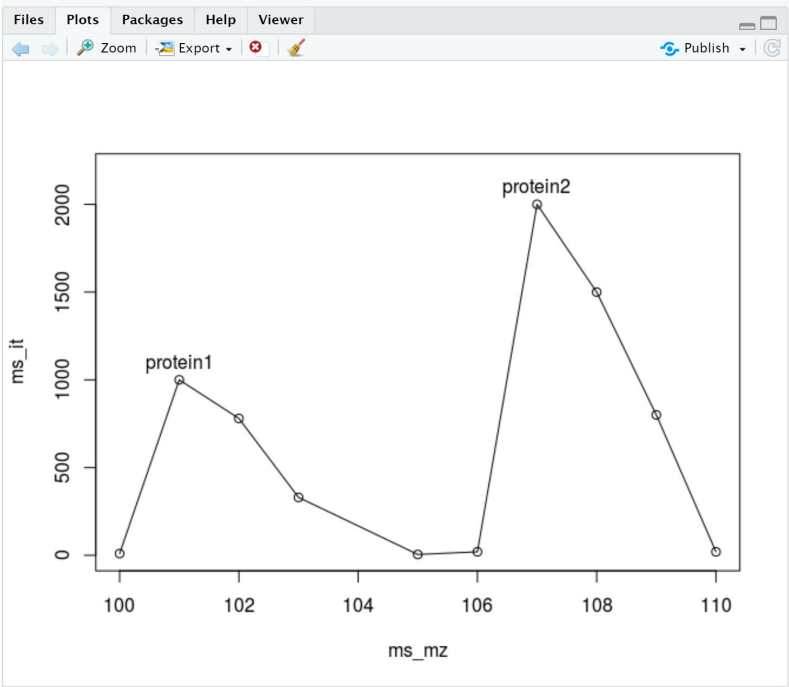

```
# simple plotting
plot(ms_mz, ms_it, type="o")
```

Defining vectors : pseudo mass spectrum II

- Sometimes we have additional information about certain signals in the mass spectrum

```
# additional metadata vector (psuedoIDs)
peak_IDs <- c(NA,"protein1",NA,NA,NA,NA,"protein2",NA,NA,NA)
```

- peak_IDs is a ‘character’ vector, it contains text, not numbers



Working with vectors : subsetting and indexing

R indexing starts at 1, not 0 like many other languages

- Index with integer

```
# simple indexing
# grabs first m/z value
ms_mz[1]

[1] 100
```

- Conditional indexing

```
# indexing based on a condition
mz_above_103 <- ms_mz > 103

# print indexed mzs as sanity check
ms_mz[mz_above_103]
```

- Subset indexing

```
# subset indexing
# grabs first five m/z values
# ':' operator
ms_mz[1:5]
```

- Non-continuous indexing

```
# indexing odd-indexed m/z using integer
vector
ms_mz[c(1,3,5,7,9)]
```

Working with vectors : operations on vectors

- Multiplying a vector by a scalar multiples all elements in a vector by the scalar

```
# multiply intensity values by 2
ms_it * 2
```

- Vector - Vector interaction

```
# adding two vectors
# vectors are operated on element by element
# make sure they are the same length!
ms_mz + ms_it
ms_mz * ms_it
```

- Some basic math on vectors, some examples

```
# common math functions
length(ms_mz)
min(ms_it)
max(ms_it)
mean(ms_it)
sd(ms_it)
```

- Modifying vectors

```
# index vector and replace values
ms_it[1] <- 101.5
```

Data types for vectors

- numeric : c(1.1, 2.1, 3.1) # no quotes must be a number
- integer : c(1,2,3,4) # numeric but all numbers are WHOLE numbers
- character : c(“a”, “b”, “c”, “d”) # always surrounded by QUOTES
- logical : c(TRUE,FALSE,TRUE) # boolean logic, T/TRUE, F/FALSE
- factors : nominal variable (can be any type) that can used for sorting

data.frames : organizing vectors into tables

- In this example, we create psuedo processed protein dataset where we have protein IDs and some values statistics associated

```
# as vectors
proteins <- c("protein1","protein2","protein3","protein4","protein5")
retention_times <- c(10.2,15.2,12.2,9.1,17.3)
sum_ms1_intensities <- c(20000,50000,1000,2000,30000)
n_ms2_transitions <- c(2L,4L,3L,1L,1L)
fdr <- c(0.11,0.03,0.02,0.15,0.06)
seq_coverage <- c(0.63,0.65,0.55,0.90,0.82)
in_ds2 <- c(TRUE,TRUE,FALSE,FALSE,TRUE)
```

- Let's put all these vectors into a data.frame which is a table view of all this information that enables better analysis through

```
# write a data.frame from vectors
protein_id_df <-
data.frame(proteins=c("protein1","protein2","protein3","protein4
","protein5"),
           retention_times = c(10.2,15.2,12.2,9.1,17.3),
           sum_ms1_intensities = c(20000,50000,1000,2000,30000),
           n_ms2_transitions = c(2L,4L,3L,1L,1L),
           fdr = c(0.11,0.03,0.02,0.15,0.06),
           seq_coverage = c(0.63,0.65,0.55,0.90,0.82),
           in_ds2 <- c(TRUE,TRUE,FALSE,FALSE,TRUE))
```

data.frames : accessing variables

- In the previous example we saw the '$' operator, this allows us to access data.frame columns by name

- You may not always need to subset, you may be after a piece of information from a single column in your data.frame

```
# print max peak intensity
max(protein_id_df$sum_ms1_intensities)
# or get intensity range...
range(protein_id_df$sum_ms1_intensities)
```

data.frames : indexing with integer index

- data.frames are indexed using [ ] with [row index, col index]

```
protein_id_df[2,]
```

- Above get the 2nd row, and all the columns. Leaving an index area blank gets all of the values

```
protein_id_df[1:2,3]
```

- Above get the first 2 rows (using ‘:’), and all the columns

data.frames : conditional indexing

- This uses the advantage that multiple vectors are linked in the data table and indexing by a condition based on one vector slices all the rest the same way

```
# conditional indexing
protein_id_df[protein_id_df$fdr < 0.05,]
```

```
  proteins retention_times sum_ms1_intensities n_ms2_transitions  fdr seq_coverage in_ds2
2 protein2            15.2                50000                 4 0.03         0.65   TRUE
3 protein3            12.2                 1000                 3 0.02         0.55  FALSE
```

- One operation to slice the data.frame, while keeping all adjoined information

data.frames : conditional indexing

- Let’s do that same thing with vectors

```
# with vectors
low_fdr_idx <- fdr < 0.05

proteins[low_fdr_idx]
retention_time[low_fdr_idx]
sum_ms1_intensities[low_fdr_idx]
n_ms2_transitions[low_fdr_idx]
fdr[low_fdr_idx]
seq_coverage[low_fdr_idx]
in_ds2[low_fdr_idx]
```

- Interact with 1 object vs interacting with 7 objects. Now imagine you have a data.frame with 100 or even 1000 columns

data.frames/vectors : more conditional operators!

- `>` ,`<`, `>=` and `<=` (see previous slides), compares numerical values

- Exact search: `==`

```
vec_data <- c("p1","p1","p2","p2","p4","p4")
p2_pro <- vec_data == "p2"
```

- Exact NOT search: `!=`

```
not_p2_pro <- vec_data != "p2"
```

- Condition multiple values: `%in%` →

```
p1p4_pro <- vec_data %in% c("p1","p2")
```

data.frames : add a variable using $

- Let’s imagine we had a new piece of information from a second source or from an analysis. We may want to append it as a new column...

```
# adding a variable...
protein_id_df$phosphorylated <- c(TRUE,FALSE,FALSE,FALSE,TRUE)
```

- We can assigned a new column to the data.frame simply by using the $ at the end of the object name and adding a new vector

- *N.B.*: The length of the new column must match the dimension of the data.frame

data.frames : add a variable using cbind

```
# new variables (cols) to add
pathway_result <- data.frame(pathway = c("apoptosis","immune response","regulation","immune
response","apoptosis"),
                             database = c("in-house","in-house","in-house","reactome","reactome"))

cbind(protein_id_df, pathway_result)
```

```
> cbind(protein_id_df, pathway_result)
  proteins retention_times sum_ms1_intensities n_ms2_transitions  fdr seq_coverage in_ds2         pathway database
1 protein1            10.2                20000                 2 0.11         0.63   TRUE       apoptosis in-house
2 protein2            15.2                50000                 4 0.03         0.65   TRUE immune response in-house
3 protein3            12.2                 1000                 3 0.02         0.55  FALSE      regulation in-house
4 protein4             9.1                 2000                 1 0.15         0.90  FALSE immune response reactome
5 protein5            17.3                30000                 1 0.06         0.82   TRUE       apoptosis reactome
```

data.frames : add a new observation (row) with rbind

```
# new variables (cols) to add
protein6_df <- data.frame(proteins = "protein6",
                          retention_times = 15.1,
                          sum_ms1_intensities = 20000,
                          n_ms2_transitions = 4,
                          fdr = 0.01,
                          seq_coverage = 0.22,
                          in_ds2 = FALSE,
                          pathway = "regulation",
                          database = "reactome")

rbind(protein_id_df_pw, protein6_df)
```

	proteins	retention_times	sum_ms1_intensities	n_ms2_transitions	fdr	seq_coverage	in_ds2	pathway	database
1	protein1	10.2	20000	2	0.11	0.63	TRUE	apoptosis	in-house
2	protein2	15.2	50000	4	0.03	0.65	TRUE	immune response	in-house
3	protein3	12.2	1000	3	0.02	0.55	FALSE	regulation	in-house
4	protein4	9.1	2000	1	0.15	0.90	FALSE	immune response	reactome
5	protein5	17.3	30000	1	0.06	0.82	TRUE	apoptosis	reactome
6	protein6	15.1	20000	4	0.01	0.22	FALSE	regulation	reactome

data.frames conclusion

- These structures are very critical and when we discuss the tidyverse you will see how they can be streamlined and manipulated for analysis

- The tibble is the data.frame analogue in the *tidyverse* and has all the same features of the data.frame and more: different printing, subtly different subsetting, and easily embedding lists as columns in tibble

Additional resources (some are more advanced)

- Data structures
- Data Structures | Introduction to R
- 6 Inbuilt Data Structures in R with practical examples

Reading Data Files into R

ASMS 2021: R Short Course

Goals for this module

- Learn how to read data files into R
  - .csv
  - Excel
- Practice with file paths and how RStudio projects help
- Important things to keep in mind when working with .csv files

Getting data into R

- Before you can work with a data set, you need to get it into your R session
- R can handle a large variety of file formats, including
  - Formatted text files (e.g. .csv, .tsv)
  - Most standard open formats (e.g. web formats, json, markup formats)
  - Open and some proprietary binary formats
  - Open and some MS data formats (e.g. mzML)
- You can read data into R using “reader” functions
- Formatted text files and Excel are workhorse formats, will focus on them here

Reading formatted text files

- R has several built-in functions for reading text files
  - `read.csv` - comma separated files
  - `read.delim` - tab separated files
  - `read.fwf` - fixed width column files

- The readr package (part of the tidyverse) has similar functions, but take a more modern approach
  - `read_csv` - comma separated files
  - `read_delim` - tab separated files
  - `read_fwf` - fixed width column files

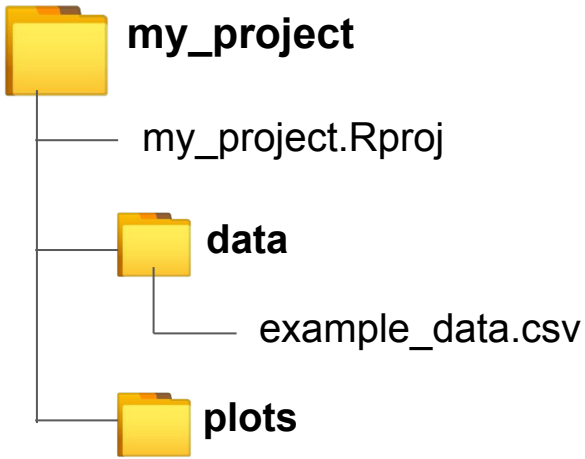
Example: reading a .csv file

```
# load the tidyverse package
# will also load the readr package
library(tidyverse)

# Read an example data file
# and store it in a variable
dat <- read_csv("example_data.csv")
```

`example.csv` needs to be in your R working directory
If you're using an RStudio project, `example.csv` should be in the (main) project directory

For RStudio projects, file paths start at the base directory

Project Directory Structure



R Code

```
read_csv("data/example_data.csv")
```

This is a relative file path to example.csv,
relative to where it exists in the project directory

csv Files: important things to keep in mind

- The first row should be the header row
  - Lists the names of the columns
  - Can contain spaces in the name, but better if not
- Following rows contain the data, each value separated by a column
- Values can contain commas if the entire value is quoted
- The number of values in each row must be the same and match the number of values in the header row

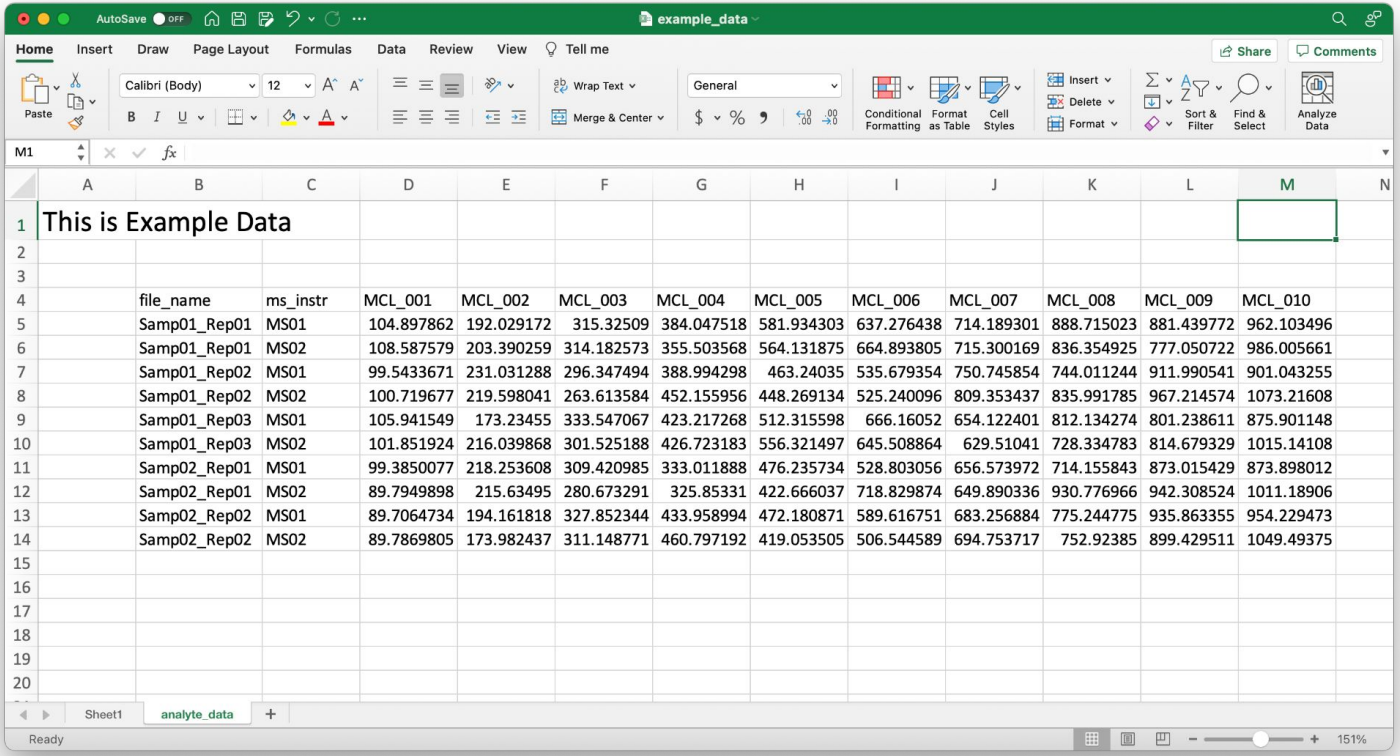
Deviations from the above can result in errors when trying to read a file
If you run into problems, you can open the file in a text editor and diagnose

Reading data from Excel files

- Reading data from Excel files is very common, but a bit more complex due to the additional flexibility Excel provides
  - Data tables can coexist with other data tables, free text, plots, etc.
  - A single Excel file can contain multiple sheets

- Need to be aware of 3 main items
  - Name of the Excel file (and it's file path)
  - Name of the Sheet
  - Cell range of the data table you want to read

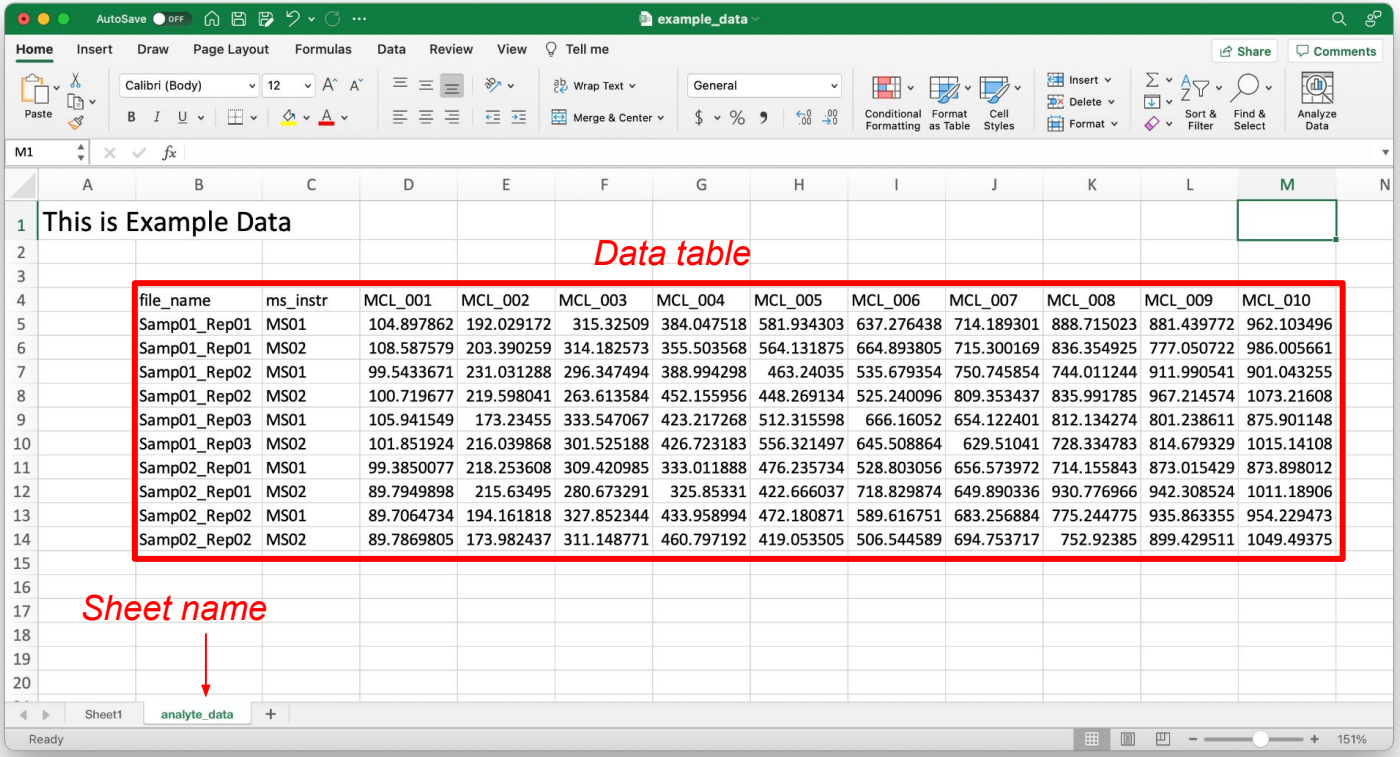
Example: reading data from an Excel file



This is Example Data

	file_name	ms_instr	MCL_001	MCL_002	MCL_003	MCL_004	MCL_005	MCL_006	MCL_007	MCL_008	MCL_009	MCL_010
5	Samp01_Rep01	MS01	104.897862	192.029172	315.32509	384.047518	581.934303	637.276438	714.189301	888.715023	881.439772	962.103496
6	Samp01_Rep01	MS02	108.587579	203.390259	314.182573	355.503568	564.131875	664.893805	715.300169	836.354925	777.050722	986.005661
7	Samp01_Rep02	MS01	99.5433671	231.031288	296.347494	388.994298	463.24035	535.679354	750.745854	744.011244	911.990541	901.043255
8	Samp01_Rep02	MS02	100.719677	219.598041	263.613584	452.155956	448.269134	525.240096	809.353437	835.991785	967.214574	1073.21608
9	Samp01_Rep03	MS01	105.941549	173.23455	333.547067	423.217268	512.315598	666.16052	654.122401	812.134274	801.238611	875.901148
10	Samp01_Rep03	MS02	101.851924	216.039868	301.525188	426.723183	556.321497	645.508864	629.51041	728.334783	814.679329	1015.14108
11	Samp02_Rep01	MS01	99.3850077	218.253608	309.420985	333.011888	476.235734	528.803056	656.573972	714.155843	873.015429	873.898012
12	Samp02_Rep01	MS02	89.7949898	215.63495	280.673291	325.85331	422.666037	718.829874	649.890336	930.776966	942.308524	1011.18906
13	Samp02_Rep02	MS01	89.7064734	194.161818	327.852344	433.958994	472.180871	589.616751	683.256884	775.244775	935.863355	954.229473
14	Samp02_Rep02	MS02	89.7869805	173.982437	311.148771	460.797192	419.053505	506.544589	694.753717	752.92385	899.429511	1049.49375

Example: reading data from an Excel file



file_name	ms_instr	MCL_001	MCL_002	MCL_003	MCL_004	MCL_005	MCL_006	MCL_007	MCL_008	MCL_009	MCL_010
Samp01_Rep01	MS01	104.897862	192.029172	315.32509	384.047518	581.934303	637.276438	714.189301	888.715023	881.439772	962.103496
Samp01_Rep01	MS02	108.587579	203.390259	314.182573	355.503568	564.131875	664.893805	715.300169	836.354925	777.050722	986.005661
Samp01_Rep02	MS01	99.5433671	231.031288	296.347494	388.994298	463.24035	535.679354	750.745854	744.011244	911.990541	901.043255
Samp01_Rep02	MS02	100.719677	219.598041	263.613584	452.155956	448.269134	525.240096	809.353437	835.991785	967.214574	1073.21608
Samp01_Rep03	MS01	105.941549	173.23455	333.547067	423.217268	512.315598	666.16052	654.122401	812.134274	801.238611	875.901148
Samp01_Rep03	MS02	101.851924	216.039868	301.525188	426.723183	556.321497	645.508864	629.51041	728.334783	814.679329	1015.14108
Samp02_Rep01	MS01	99.3850077	218.253608	309.420985	333.011888	476.235734	528.803056	656.573972	714.155843	873.015429	873.898012
Samp02_Rep01	MS02	89.7949898	215.63495	280.673291	325.85331	422.666037	718.829874	649.890336	930.776966	942.308524	1011.18906
Samp02_Rep02	MS01	89.7064734	194.161818	327.852344	433.958994	472.180871	589.616751	683.256884	775.244775	935.863355	954.229473
Samp02_Rep02	MS02	89.7869805	173.982437	311.148771	460.797192	419.053505	506.544589	694.753717	752.92385	899.429511	1049.49375

Example: reading data from an Excel file

```
# Need to load the readxl package
library(readxl)


dat <- read_excel("example_data.xlsx",
                  sheet = "analyte_data",
                  range = "B4:M14")
```

What do you think will happen if you don't specify the sheet and/or range?

Notes about reading data files into R

- You'll usually want to store the output as a variable
  dat <- read_csv("example_data.csv")

- For table-like formats, the data will be read as a data frame

- If a data file is large (~100's MBs to GBs), be careful
  - It might take a (really) long time to read
  - You need to have enough RAM (data is usually read into memory)
  - R may start to have issues with files larger than several GBs

Recap

- Reading csv files
  - Load your RStudio project and make sure your data file is inside the project folder
  - Load the `tidyverse` package
    ```
    library(tidyverse)
    ```
  - Read the file
    ```
    dat <- read_csv("my_data.csv")
    ```

- Reading Excel files
  - Note the file path (RStudio project), sheet name, data range
  - Load the `readxl` package
    ```
    library(readxl)
    ```
  - Read the file
    ```
    dat <- read_excel("my_data.xlsx", sheet = "My Sheet",
                      range = "B2:H26")
    ```

Useful R functions

ASMS 2021: R Short Course

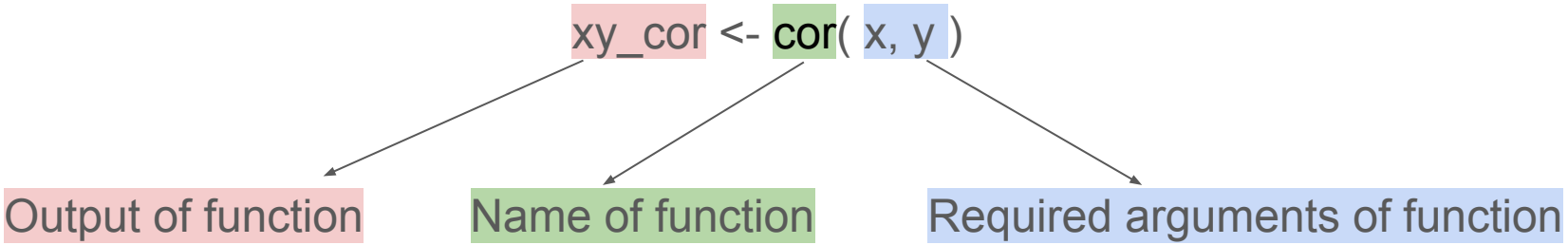
Goals for this module

- Introduction to basic but extremely useful R functions
- Understand structure of a function (name, arguments, return object)
- Understand that functions described here can be made more effective and concise when used in conjunction with the *tidyverse*

Useful R functions and why/how to use them

- Many of the important classes tell you something about the *shape* of the data
- Others do very simple but common *statistics*
- There are many other types of useful functions, it’s impossible to catalogue them all here

Using an R function: cor() example (correlation)



- Object defined by function
- Defined in package
- Separated by comma ,

```
# correlation function example
x <- c(1,2,3,4)
y <- c(2,4,6,10)

xy_cor <- cor(x,y)
```

- **Some arguments are optional**

In arguments you use =, not <-

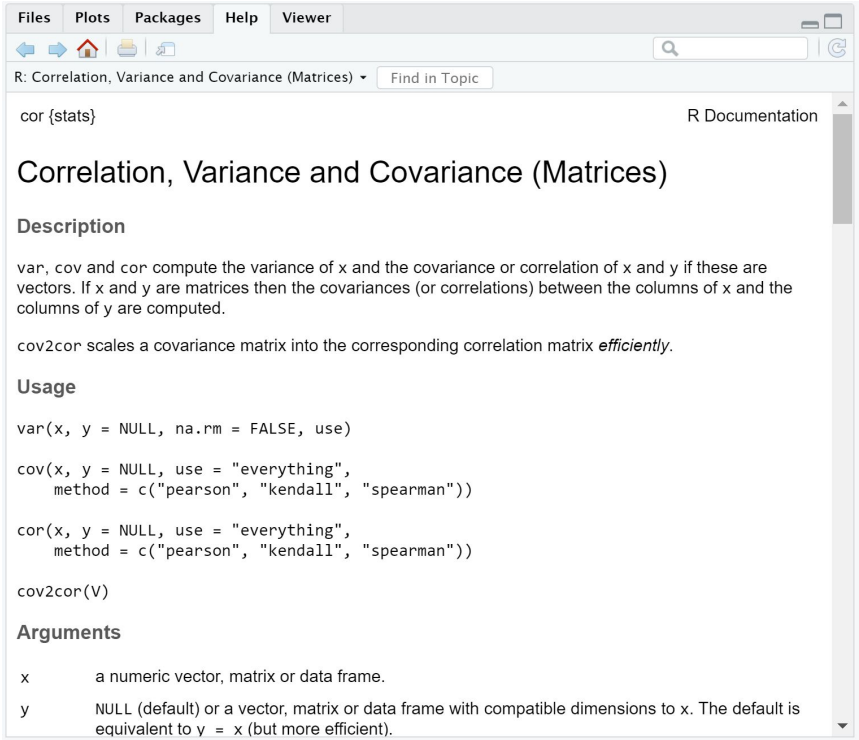
```
xy_cor_spearman <- cor(x,y, method="spearman")
```

Optional argument of function

- Often sets parameters of data but has defaults

Using an R function: getting help for specific function

```
# get docs on cor() function
?cor
```



- **Most R packages are well-documented**

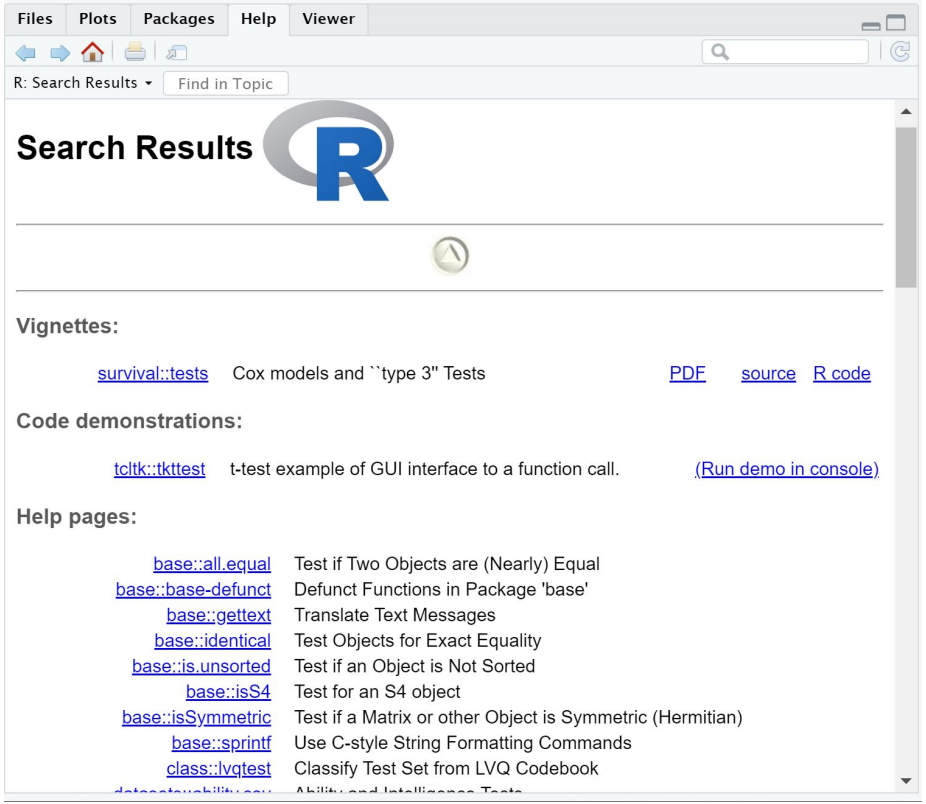
  We can conveniently view documentation documentation for the package within RStudio.

- **Using the ? and ?? commands**
  - ? = find function/package documentation for what has been loaded
  - ?? = search installed packages for key words

Using an R function: exploring functionality

```
# search for functions and package
info about t-testing
??ttest
```

- **Searching with ?? can give a lot of results so check carefully the documentation of a function to make sure it is appropriate**



Construction functions

```r
# functions to construct vectors
char_vec <- c('x','y','z','a','b','c')


# numerical sequencing
every10th <- seq(1,100,10)


# repeat a value (1) multiple times
rep_val <- rep(1, 10)


# remember these vectors can be rolled into data.frames for advanced processing
```

Controlling variable type

```
# control variable type
# some R functions expect variables to be a certain type
mixed_vec <- c("X100200","X100300","X100400","X100500")

numerical_vec <- gsub("X","",mixed_vec)

# get the vector's type
class(numerical_vec)

# coerce to a more appropriate type
as.numeric(numerical_vec)

# check the vector type again using class AFTER coercion
class(as.numeric(numerical_vec))
```

“character”

“numeric”

Functions to learn about the shape of data

```
# functions to get the shape of a dataset
# with vectors, there is only one shape property,
that is the length
length(ms_mz)

# with data.frames, there is much more information
to obtain about the shape
# the basics
nrow(protein_id_df)
ncol(protein_id_df)
```

Number of elements in vector

Number of rows and cols in df

- If we remember the previous data.frame examples, to add columns or rows, we needed to match the existing size of the data.frame, these checks can help us make sure new data is formatted correctly

Functions to learn about the shape AND type of data

- We have reviewed how R vectors and data.frames have types: numeric, character, logical, etc.
- Sometimes we would like to check and summarize these types in a complex data.frame

```
str(protein_id_df)
```



```
> str(protein_id_df)
'data.frame':      5 obs. of  7 variables:
 $ proteins           : Factor w/ 5 levels "protein1","protein2",..: 1 2 3 4 5
 $ retention_times    : num  10.2 15.2 12.2 9.1 17.3
 $ sum_ms1_intensities: num  20000 50000 1000 2000 30000
 $ n_ms2_transitions  : int  2 4 3 1 1
 $ fdr                : num  0.11 0.03 0.02 0.15 0.06
 $ seq_coverage       : num  0.63 0.65 0.55 0.9 0.82
 $ in_ds2             : logi  TRUE TRUE FALSE FALSE TRUE
```

Summary statistics

We can use `mean(), sd(), median(), quantile()` for basic statistics on numerical vectors

```
# get count information for values in a data.frame vector
table(protein_id_df$in_ds2)
```

```
> table(protein_id_df$in_ds2)

FALSE  TRUE
    2     3
```

```
# get sumary statistics
summary(protein_id_df$sum_ms1_intensities)
```

```
> summary(protein_id_df$sum_ms1_intensities)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1000    2000   20000   20600   30000   50000
```

Two sample t-test example

- t.test() # note there is no hyphen in the function name

```
# t.test
set.seed(0)

# make fake data using rnorm for demonstration purposes
ttest_df <- data.frame(condition = c(rep('sample01',3),rep('sample02',3)),
                       variable=c(rnorm(3, mean = 10, sd=2),
                                  rnorm(3, mean = 24, sd=5)))

t.test(variable~condition, data=ttest_df)
```

```
> ttest_df
  condition  variable
1  sample01  7.666859
2  sample01  7.868819
3  sample01  6.872436
4  sample02 29.782685
5  sample02 28.160236
6  sample02 22.863357
```

```
        Welch Two Sample t-test

data:  variable by condition
t = -11.462, df = 10.645, p-value = 2.532e-07
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -19.88254 -13.45472
sample estimates:
mean in group sample01 mean in group sample02
              9.548413              26.217045
```

Check the documentation on t.test…
It's not a one-size-fits-all function and there are many options that may be more appropriate for YOUR data

Randomization in statistics

set.seed(0) # reproducibility in randomized analysis

```
# randomization and sampling
num_vec <- c(1:100)

random_sample <- sample(num_vec, 10)
```

For some machine learning applications, subsampling the data is necessary to build training and test sets

```
# random sample from a normal distribution
rnorm(10, mean = 10, sd=2)
```

Can be great for simulations and many other statistical tasks

Files and directories functions

```
list.files('directory',pattern='*.csv')
```

Find .csv files in ‘directory’

```
list.files('directory', pattern=".csv", recursive=T, full.names=T)
```

Find .csv files
Get full names
Search all subfolders

```
read.csv("path_to_csv.csv", header=T)
```

Read .csv file
First line is imported as column names

```
write.csv(protein_id_df, 'protein_id_df.csv')
```

Save a .csv of a data.frame

Bioconductor: package resource for -omics type data

- Need a separate package installation manager : Bioconductor Install

- Many packages for MS data of all varieties using a common data structure

Function takeaways

- There are many things you can with R with functions: understand data structure, descriptive and inferential statistics, and much, much more

- Additional packages will need to be installed to access much of this functionality

- You will likely need a MS specific package to load in raw MS data for analysis, resources like Bioconductor provide many great packages specific to MS

Exercises

1. Read denver_climate.csv
2. Find month and year with the highest snowfall
3. Find months with no snowfall
4. Find average precipitation per year
5. Convert temperatures to Celsius and add new column to data.frame
6. Find month with the greatest difference between max. and min. temperature
7. Write csv containing Celsius temperature data

Resources

- R for Data Science **Essential**
- R - Functions **Some advanced ideas, exploring making your own functions**
- A Tutorial on Using Functions in R! (and their scoping) **Some advanced ideas, exploring making your own functions**

Tidy Data & The Tidyverse

ASMS 2021: R Short Course

Goals for this module

- Understand what tidy data is
  - How is it different from “messy” data?
  - What makes tidy data “tidy”?
  - Why tidy data is useful for doing data analysis in R
- Learn how to recognize messy data and why it’s hard to work with
  - Understand the things you may need to do when tidying-up messy data
  - Be able to communicate to others what tidy data is
- Learn about the tidyverse and why it’s a great ecosystem for working with data in R
- Learn about some of the fundamental operations useful for converting messy data into tidy data

Quick note about the terms “messy” & “tidy”

Messy

- Has negative connotations but that’s not the intent here
- Messy data is simply data that’s not yet in a form suitable for analysis
- Messy data doesn’t mean bad data

Tidy

- Has positive connotations but that’s not the intent here either
- Tidy data is data in a consistent format that supports the analysis process
- Tidy data doesn’t mean good data

In the R ecosystem, the term “tidy data” has a very specific meaning (that we’ll get to soon)

The Anna Karenina principle...

> Happy families are all alike;
> every unhappy family is unhappy in its own way.
>
> – Leo Tolstoy

... applies to data too

> *Tidy datasets*
> ~~Happy families~~ are all alike;
> every ~~unhappy family~~ is ~~unhappy~~ in its own way.
> *messy dataset* *messy*
>
> – Hadley Wickham

Messy data

Most “real world” data starts out messy...

...which makes it hard to work with.

All tidy data has similar structure...

...making it easier to work with since you know what to expect

Data can be messy in all kinds of ways

Excel spreadsheet with complex formatting

Professor Smith: Data Science Grades Sheet, 1st Trimester 2020

University of DS, Department of Biology

Class Date: 1/6/2020 to 3/27/2020

	Quizzes			Tests		
Name	**Score 1**	**Score 2**	**Score 3**	**Test 1**	**Midterm**	**Final**
al-Jafri, Anas	17 / 25	41 / 50	18 / 25	87 / 100	89 / 100	90 / 100
Drum, Anyssa	19 / 25	40 / 50	19 / 25	88 / 100	87 / 100	92 / 100
Gonzales, David	18 / 25	38 / 50	19 / 25	84 / 100	91 / 100	83 / 100
Granger, Shannan	14 / 25	42 / 50	20 / 25	89 / 100	82 / 100	90 / 100
Kwak, Surabhi	16 / 25	41 / 50	21 / 25	85 / 100	88 / 100	93 / 100
Michels, Breana	17 / 25	46 / 50	18 / 25	80 / 100	85 / 100	82 / 100
Reed, Tyler	19 / 25	39 / 50	17 / 25	78 / 100	94 / 100	88 / 100
Smith, Tyler	16 / 25	40 / 50	19 / 25	79 / 100	90 / 100	86 / 100
Smith, Westin	15 / 25	45 / 50	17 / 25	87 / 100	85 / 100	91 / 100
Vang, Seher	19 / 25	43 / 50	19 / 25	79 / 100	82 / 100	87 / 100

Data tables in PDF (or even images!)

	John Smith	Jane Doe	Mary Johnson
treatmenta	—	16	3
treatmentb	2	11	1

Tidy Data, H. Wickham, J. Stat. Software

Badly formatted text files

```
1, 4, 5, 6, 10
10,000, 900, 874
5,6,8
7473,134187,1398,17
```

Proprietary MS data formats 💾

Data in web pages, plots & figures, ...

Messy data can be hard to work with

- Unclear structure & organization can make it hard to understand what's there
- Data might be optimized for data entry or visual consumption,
  not computer consumption
- Might be represented in a strange ways
  (e.g. numbers combined with words, unclear coding variables)
- Might be in multiple places (e.g. in different files)
- Might be in strange formats (html, pdf, png )

Messy data isn’t necessarily bad data

- Not all data is created or presented with data analysis in mind
- The people who curate the data might not understand the needs of someone who needs to work with it
- The goals for the data might not align directly with data analysis needs (presenting data on a slide, performance or storage requirements)
- ... but messy data might be a sign of lurking data problems too 🕵️

Tidy data in the R ecosystem

Tidy data has consistent structure, arranged in a rectangular table

Variables (columns)
the “things” you are measuring

Observations (rows)
the “things” you are making measurements on

Values (cells)
the values of the measurements

figure: https://r4ds.had.co.nz/tidy-data.html

What is this table about? Variables? Observations?

Last Name	First Name	Age	Major	Graduation Year
Smith	Sally	22	Chemistry	2024
Yamamoto	Akira	24	History	2023
Snow	John	21	Business	2024

(more rows…)

What is this table about? Variables? Observations?

Table of New Students

Last Name	First Name	Age	Major	Graduation Year
Smith	Sally	22	Chemistry	2024
Yamamoto	Akira	24	History	2023
Snow	John	21	Business	2024

(more rows…)

Each **observation** is a **student**

Each **variable** is a piece of **information about the student** (name, age, etc)

Each (individual) **piece of information** is in its own **cell**

A few notes about tidy data

- Tidy data is not the only way
  - Other structures might be needed to optimize for performance or storage
  - Certain fields might follow other data conventions
  - Some types of data might not naturally fit into a rectangular table

- BUT, if your data *can* fit into rectangular structure, tidy data is usually the way to go

- Sometimes the differences between observations and variables is not always clear, and you might swap them depending on the context

- Data tidiness isn’t necessarily black & white, different circumstances might require different levels of tidiness

Prof. Smith is recording class grades in a spreadsheet

Professor Smith: Data Science Grades Sheet, 1st Trimester 2020

University of DS, Department of Biology

Class Date: 1/6/2020 to 3/27/2020

	Quizzes			Tests		
Name	**Score 1**	**Score 2**	**Score 3**	**Test 1**	**Midterm**	**Final**
al-Jafri, Anas	17 / 25	41 / 50	18 / 25	87 / 100	89 / 100	90 / 100
Drum, Anyssa	19 / 25	40 / 50	19 / 25	88 / 100	87 / 100	92 / 100
Gonzales, David	18 / 25	38 / 50	19 / 25	84 / 100	91 / 100	83 / 100
Granger, Shannan	14 / 25	42 / 50	20 / 25	89 / 100	82 / 100	90 / 100
Kwak, Surabhi	16 / 25	41 / 50	21 / 25	85 / 100	88 / 100	93 / 100
Michels, Breana	17 / 25	46 / 50	18 / 25	80 / 100	85 / 100	82 / 100
Reed, Tyler	19 / 25	39 / 50	17 / 25	78 / 100	94 / 100	88 / 100
Smith, Tyler	16 / 25	40 / 50	19 / 25	79 / 100	90 / 100	86 / 100
Smith, Westin	15 / 25	45 / 50	17 / 25	87 / 100	85 / 100	91 / 100
Vang, Seher	19 / 25	43 / 50	19 / 25	79 / 100	82 / 100	87 / 100

Is this data tidy?

It looks nice and neat...

No – this data is not tidy for several reasons

Professor Smith: Data Science Grades Sheet, 1st Trimester 2020

University of DS, Department of Biology

Class Date: 1/6/2020 to 3/27/2020

	Quizzes			Tests		
Name	**Score 1**	**Score 2**	**Score 3**	**Test 1**	**Midterm**	**Final**
al-Jafri, Anas	17 / 25	41 / 50	18 / 25	87 / 100	89 / 100	90 / 100
Drum, Anyssa	19 / 25	40 / 50	19 / 25	88 / 100	87 / 100	92 / 100
Gonzales, David	18 / 25	38 / 50	19 / 25	84 / 100	91 / 100	83 / 100
Granger, Shannan	14 / 25	42 / 50	20 / 25	89 / 100	82 / 100	90 / 100
Kwak, Surabhi	16 / 25	41 / 50	21 / 25	85 / 100	88 / 100	93 / 100
Michels, Breana	17 / 25	46 / 50	18 / 25	80 / 100	85 / 100	82 / 100
Reed, Tyler	19 / 25	39 / 50	17 / 25	78 / 100	94 / 100	88 / 100
Smith, Tyler	16 / 25	40 / 50	19 / 25	79 / 100	90 / 100	86 / 100
Smith, Westin	15 / 25	45 / 50	17 / 25	87 / 100	85 / 100	91 / 100
Vang, Seher	19 / 25	43 / 50	19 / 25	79 / 100	82 / 100	87 / 100

The file contains free form text at the top

Some of this data might be useful but it's not immediately ready for analysis

No – this data is not tidy for several reasons

Professor Smith: Data Science Grades Sheet, 1st Trimester 2020

University of DS, Department of Biology

Class Date: 1/6/2020 to 3/27/2020

	Quizzes			Tests		
Name	**Score 1**	**Score 2**	**Score 3**	**Test 1**	**Midterm**	**Final**
al-Jafri, Anas	17 / 25	41 / 50	18 / 25	87 / 100	89 / 100	90 / 100
Drum, Anyssa	19 / 25	40 / 50	19 / 25	88 / 100	87 / 100	92 / 100
Gonzales, David	18 / 25	38 / 50	19 / 25	84 / 100	91 / 100	83 / 100
Granger, Shannan	14 / 25	42 / 50	20 / 25	89 / 100	82 / 100	90 / 100
Kwak, Surabhi	16 / 25	41 / 50	21 / 25	85 / 100	88 / 100	93 / 100
Michels, Breana	17 / 25	46 / 50	18 / 25	80 / 100	85 / 100	82 / 100
Reed, Tyler	19 / 25	39 / 50	17 / 25	78 / 100	94 / 100	88 / 100
Smith, Tyler	16 / 25	40 / 50	19 / 25	79 / 100	90 / 100	86 / 100
Smith, Westin	15 / 25	45 / 50	17 / 25	87 / 100	85 / 100	91 / 100
Vang, Seher	19 / 25	43 / 50	19 / 25	79 / 100	82 / 100	87 / 100

First and last names are stored together in the same cell

Might want these two pieces of information separate

No – this data is not tidy for several reasons

Professor Smith: Data Science Grades Sheet, 1st Trimester 2020

University of DS, Department of Biology

Class Date: 1/6/2020 to 3/27/2020

	Quizzes			Tests		
Name	**Score 1**	**Score 2**	**Score 3**	**Test 1**	**Midterm**	**Final**
al-Jafri, Anas	17 / 25	41 / 50	18 / 25	87 / 100	89 / 100	90 / 100
Drum, Anyssa	19 / 25	40 / 50	19 / 25	88 / 100	87 / 100	92 / 100
Gonzales, David	18 / 25	38 / 50	19 / 25	84 / 100	91 / 100	83 / 100
Granger, Shannan	14 / 25	42 / 50	20 / 25	89 / 100	82 / 100	90 / 100
Kwak, Surabhi	16 / 25	41 / 50	21 / 25	85 / 100	88 / 100	93 / 100
Michels, Breana	17 / 25	46 / 50	18 / 25	80 / 100	85 / 100	82 / 100
Reed, Tyler	19 / 25	39 / 50	17 / 25	78 / 100	94 / 100	88 / 100
Smith, Tyler	16 / 25	40 / 50	19 / 25	79 / 100	90 / 100	86 / 100
Smith, Westin	15 / 25	45 / 50	17 / 25	87 / 100	85 / 100	91 / 100
Vang, Seher	19 / 25	43 / 50	19 / 25	79 / 100	82 / 100	87 / 100

The main data table is not "by itself"

Can't simply read the data into R

No – this data is not tidy for several reasons

Professor Smith: Data Science Grades Sheet, 1st Trimester 2020

University of DS, Department of Biology

Class Date: 1/6/2020 to 3/27/2020

	Quizzes			Tests		
Name	**Score 1**	**Score 2**	**Score 3**	**Test 1**	**Midterm**	**Final**
al-Jafri, Anas	17 / 25	41 / 50	18 / 25	87 / 100	89 / 100	90 / 100
Drum, Anyssa	19 / 25	40 / 50	19 / 25	88 / 100	87 / 100	92 / 100
Gonzales, David	18 / 25	38 / 50	19 / 25	84 / 100	91 / 100	83 / 100
Granger, Shannan	14 / 25	42 / 50	20 / 25	89 / 100	82 / 100	90 / 100
Kwak, Surabhi	16 / 25	41 / 50	21 / 25	85 / 100	88 / 100	93 / 100
Michels, Breana	17 / 25	46 / 50	18 / 25	80 / 100	85 / 100	82 / 100
Reed, Tyler	19 / 25	39 / 50	17 / 25	78 / 100	94 / 100	88 / 100
Smith, Tyler	16 / 25	40 / 50	19 / 25	79 / 100	90 / 100	86 / 100
Smith, Westin	15 / 25	45 / 50	17 / 25	87 / 100	85 / 100	91 / 100
Vang, Seher	19 / 25	43 / 50	19 / 25	79 / 100	82 / 100	87 / 100

The “Quizzes” and “Tests” labels exist in merged columns, and link to multiple columns below them

This is visual structure, not well suited for computers

No – this data is not tidy for several reasons

Professor Smith: Data Science Grades Sheet, 1st Trimester 2020

University of DS, Department of Biology

Class Date: 1/6/2020 to 3/27/2020

	Quizzes			Tests		
Name	**Score 1**	**Score 2**	**Score 3**	**Test 1**	**Midterm**	**Final**
al-Jafri, Anas	17 / 25	41 / 50	18 / 25	87 / 100	89 / 100	90 / 100
Drum, Anyssa	19 / 25	40 / 50	19 / 25	88 / 100	87 / 100	92 / 100
Gonzales, David	18 / 25	38 / 50	19 / 25	84 / 100	91 / 100	83 / 100
Granger, Shannan	14 / 25	42 / 50	20 / 25	89 / 100	82 / 100	90 / 100
Kwak, Surabhi	16 / 25	41 / 50	21 / 25	85 / 100	88 / 100	93 / 100
Michels, Breana	17 / 25	46 / 50	18 / 25	80 / 100	85 / 100	82 / 100
Reed, Tyler	19 / 25	39 / 50	17 / 25	78 / 100	94 / 100	88 / 100
Smith, Tyler	16 / 25	40 / 50	19 / 25	79 / 100	90 / 100	86 / 100
Smith, Westin	15 / 25	45 / 50	17 / 25	87 / 100	85 / 100	91 / 100
Vang, Seher	19 / 25	43 / 50	19 / 25	79 / 100	82 / 100	87 / 100

Column naming is inconsistent

All refer to a specific quiz or test, but quizzes are referred to by “Score”, tests by the type of test

No – this data is not tidy for several reasons

Professor Smith: Data Science Grades Sheet, 1st Trimester 2020

University of DS, Department of Biology

Class Date: 1/6/2020 to 3/27/2020

	Quizzes			Tests		
Name	**Score 1**	**Score 2**	**Score 3**	**Test 1**	**Midterm**	**Final**
al-Jafri, Anas	17 / 25	41 / 50	18 / 25	87 / 100	89 / 100	90 / 100
Drum, Anyssa	19 / 25	40 / 50	19 / 25	88 / 100	87 / 100	92 / 100
Gonzales, David	18 / 25	38 / 50	19 / 25	84 / 100	91 / 100	83 / 100
Granger, Shannan	14 / 25	42 / 50	20 / 25	89 / 100	82 / 100	90 / 100
Kwak, Surabhi	16 / 25	41 / 50	21 / 25	85 / 100	88 / 100	93 / 100
Michels, Breana	17 / 25	46 / 50	18 / 25	80 / 100	85 / 100	82 / 100
Reed, Tyler	19 / 25	39 / 50	17 / 25	78 / 100	94 / 100	88 / 100
Smith, Tyler	16 / 25	40 / 50	19 / 25	79 / 100	90 / 100	86 / 100
Smith, Westin	15 / 25	45 / 50	17 / 25	87 / 100	85 / 100	91 / 100
Vang, Seher	19 / 25	43 / 50	19 / 25	79 / 100	82 / 100	87 / 100

Each data point for a score is actually two pieces of data:

points received / total points

Also, the data is represented as text since it contains a “/” character

No – this data is not tidy for several reasons

Professor Smith: Data Science Grades Sheet, 1st Trimester 2020

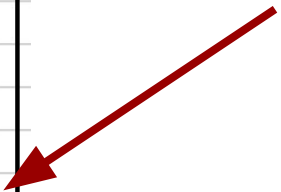
University of DS, Department of Biology

Class Date: 1/6/2020 to 3/27/2020

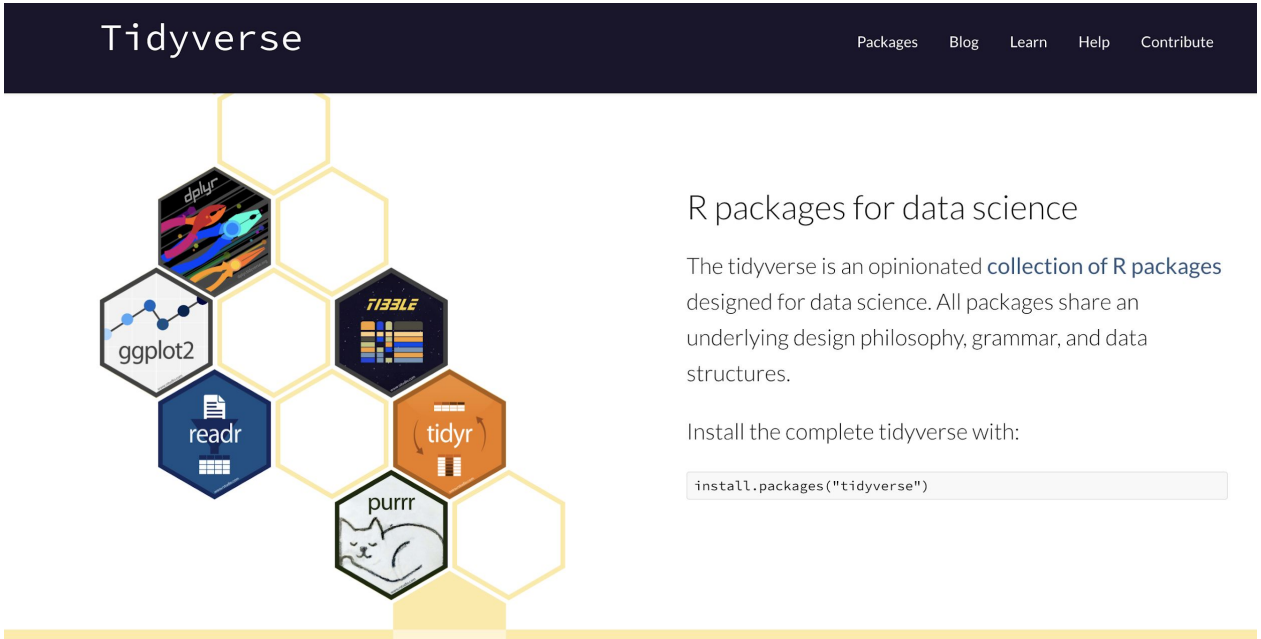
	Quizzes			Tests		
Name	**Score 1**	**Score 2**	**Score 3**	**Test 1**	**Midterm**	**Final**
al-Jafri, Anas	17 / 25	41 / 50	18 / 25	87 / 100	89 / 100	90 / 100
Drum, Anyssa	19 / 25	40 / 50	19 / 25	88 / 100	87 / 100	92 / 100
Gonzales, David	18 / 25	38 / 50	19 / 25	84 / 100	91 / 100	83 / 100
Granger, Shannan	14 / 25	42 / 50	20 / 25	89 / 100	82 / 100	90 / 100
Kwak, Surabhi	16 / 25	41 / 50	21 / 25	85 / 100	88 / 100	93 / 100
Michels, Breana	17 / 25	46 / 50	18 / 25	80 / 100	85 / 100	82 / 100
Reed, Tyler	19 / 25	39 / 50	17 / 25	78 / 100	94 / 100	88 / 100
Smith, Tyler	16 / 25	40 / 50	19 / 25	79 / 100	90 / 100	86 / 100
Smith, Westin	15 / 25	45 / 50	17 / 25	87 / 100	85 / 100	91 / 100
Vang, Seher	19 / 25	43 / 50	19 / 25	79 / 100	82 / 100	87 / 100

Important data points are highlighted in red text

“Information by formatting” is very difficult to deal with



The Tidyverse
An opinionated collection of R packages for data science



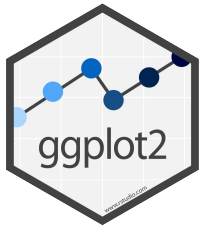
https://www.tidyverse.org

Once you know the general structure of the input data (i.e. tidy data), you can build all kinds of tools to work with it

That's the tidyverse!

The tidyverse covers the fundamental components of the data analysis workflow

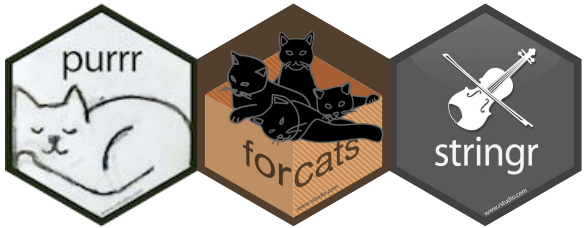
Core tidyverse Packages



Data Visualization

Fundamental Data Manipulation & Pipelines

Read Data
Tidy Data



Manipulate Specific Types of Data

https://www.tidyverse.org

And LOTS of other packages developed by others...

tidyr helps you get to tidy data

tidyr provides many tools, but these are the two most important

Pivoting

country	1999	2000
A	0.7K	2K
B	37K	80K
C	212K	213K

country	year	cases
A	1999	0.7K
B	1999	37K
C	1999	212K
A	2000	2K
B	2000	80K
C	2000	213K

key value

wide(r) form *long(er) form*

Separating

country	year	rate
A	1999	0.7K/19M
A	2000	2K/20M
B	1999	37K/172M
B	2000	80K/174M
C	1999	212K/1T
C	2000	213K/1T

country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172
B	2000	80K	174
C	1999	212K	1T
C	2000	213K	1T

separate combined data into individual columns

https://github.com/rstudio/cheatsheets/blob/master/data-import.pdf

Pivoting case #1:

a variable is spread across multiple columns

	Quizzes			Tests		
Name	**Score 1**	**Score 2**	**Score 3**	**Test 1**	**Midterm**	**Final**
al-Jafri, Anas	17 / 25	41 / 50	18 / 25	87 / 100	89 / 100	90 / 100
Drum, Anyssa	19 / 25	40 / 50	19 / 25	88 / 100	87 / 100	92 / 100
Gonzales, David	18 / 25	38 / 50	19 / 25	84 / 100	91 / 100	83 / 100
Granger, Shannan	14 / 25	42 / 50	20 / 25	89 / 100	82 / 100	90 / 100
Kwak, Surabhi	16 / 25	41 / 50	21 / 25	85 / 100	88 / 100	93 / 100
Michels, Breana	17 / 25	46 / 50	18 / 25	80 / 100	85 / 100	82 / 100
Reed, Tyler	19 / 25	39 / 50	17 / 25	78 / 100	94 / 100	88 / 100
Smith, Tyler	16 / 25	40 / 50	19 / 25	79 / 100	90 / 100	86 / 100
Smith, Westin	15 / 25	45 / 50	17 / 25	87 / 100	85 / 100	91 / 100
Vang, Seher	19 / 25	43 / 50	19 / 25	79 / 100	82 / 100	87 / 100

All of these exams (quizzes and test) are in their own columns

We might consider a single variable called “exam” that collects all the different tests & quizzes into a single column

This will make the table longer

Pivoting case #1:

a variable is spread across multiple columns

We want to take these columns and put their names into a *new* single column (we'll call it exam), and the scores into a new single column (we'll call it score)

Note:
there are 10 students

```
# A tibble: 10 x 7
   Name             `Score 1` `Score 2` `Score 3` `Test 1` Midterm  Final
   <chr>            <chr>     <chr>     <chr>     <chr>    <chr>    <chr>
 1 al-Jafri, Anas   17 / 25   41 / 50   18 / 25   87 / 100 89 / 100 90 / 100
 2 Drum, Anyssa     19 / 25   40 / 50   19 / 25   88 / 100 87 / 100 92 / 100
 3 Gonzales, David  18 / 25   38 / 50   19 / 25   84 / 100 91 / 100 83 / 100
 4 Granger, Shannan 14 / 25   42 / 50   20 / 25   89 / 100 82 / 100 90 / 100
 5 Kwak, Surabhi    16 / 25   41 / 50   21 / 25   85 / 100 88 / 100 93 / 100
 6 Michels, Breana  17 / 25   46 / 50   18 / 25   80 / 100 85 / 100 82 / 100
 7 Reed, Tyler      19 / 25   39 / 50   17 / 25   78 / 100 94 / 100 88 / 100
 8 Smith, Tyler     16 / 25   40 / 50   19 / 25   79 / 100 90 / 100 86 / 100
 9 Smith, Westin    15 / 25   45 / 50   17 / 25   87 / 100 85 / 100 91 / 100
10 Vang, Seher      19 / 25   43 / 50   19 / 25   79 / 100 82 / 100 87 / 100
```

Note:
there are 6 exams

Each row will represent a *single* exam and its score for a given student
We should get 10 students x 6 exams = 60 rows

Pivoting case #1:

a variable is spread across multiple columns

R Code

```
# Load the tidyverse core packages
library(tidyverse)

#
# Step 1: Load the data table
#
dat <- read_csv("ex1_tbl.csv")
print(dat)
```

Output

```
# A tibble: 10 x 7
   Name             `Score 1` `Score 2` `Score 3` `Test 1` Midterm  Final
   <chr>            <chr>     <chr>     <chr>     <chr>    <chr>    <chr>
 1 al-Jafri, Anas   17 / 25   41 / 50   18 / 25   87 / 100 89 / 100 90 / 100
 2 Drum, Anyssa     19 / 25   40 / 50   19 / 25   88 / 100 87 / 100 92 / 100
 3 Gonzales, David  18 / 25   38 / 50   19 / 25   84 / 100 91 / 100 83 / 100
 4 Granger, Shannan 14 / 25   42 / 50   20 / 25   89 / 100 82 / 100 90 / 100
 5 Kwak, Surabhi    16 / 25   41 / 50   21 / 25   85 / 100 88 / 100 93 / 100
 6 Michels, Breana  17 / 25   46 / 50   18 / 25   80 / 100 85 / 100 82 / 100
 7 Reed, Tyler      19 / 25   39 / 50   17 / 25   78 / 100 94 / 100 88 / 100
 8 Smith, Tyler     16 / 25   40 / 50   19 / 25   79 / 100 90 / 100 86 / 100
 9 Smith, Westin    15 / 25   45 / 50   17 / 25   87 / 100 85 / 100 91 / 100
10 Vang, Seher      19 / 25   43 / 50   19 / 25   79 / 100 82 / 100 87 / 100
```

Load the tidyverse packages
Read in the data
Print the data to the screen

Pivoting case #1:

a variable is spread across multiple columns

R Code

```
# .. code continued
# Step 2: Pivot the data longer
# pull all the exams into a single
column
#
dat_long <-
  pivot_longer(dat,
               cols = 2:7,
               names_to = "exam",
               values_to = "score")
print(dat_long)
```

Output

```
# A tibble: 60 x 3
   Name           exam    score
   <chr>          <chr>   <chr>
 1 al-Jafri, Anas Score 1 17 / 25
 2 al-Jafri, Anas Score 2 41 / 50
 3 al-Jafri, Anas Score 3 18 / 25
 4 al-Jafri, Anas Test 1  87 / 100
 5 al-Jafri, Anas Midterm 89 / 100
 6 al-Jafri, Anas Final   90 / 100
 7 Drum, Anyssa   Score 1 19 / 25
 8 Drum, Anyssa   Score 2 40 / 50
 9 Drum, Anyssa   Score 3 19 / 25
10 Drum, Anyssa   Test 1  88 / 100
# … with 50 more rows
```

`pivot_longer` is the function

`cols` says which columns we want to collect into a single column (we do it here by column number)

`names_to` gives the *new* column name where the cols will go

`values_to` gives the *new* column name where the values will go

Pivoting case #1:

a variable is spread across multiple columns

60 rows just like we expected

Each student has 6 rows, one for each exam

```
# A tibble: 60 x 3
   Name           exam    score
   <chr>          <chr>   <chr>
 1 al-Jafri, Anas Score 1 17 / 25
 2 al-Jafri, Anas Score 2 41 / 50
 3 al-Jafri, Anas Score 3 18 / 25
 4 al-Jafri, Anas Test 1  87 / 100
 5 al-Jafri, Anas Midterm 89 / 100
 6 al-Jafri, Anas Final   90 / 100
 7 Drum, Anyssa   Score 1 19 / 25
 8 Drum, Anyssa   Score 2 40 / 50
 9 Drum, Anyssa   Score 3 19 / 25
10 Drum, Anyssa   Test 1  88 / 100
# … with 50 more rows
```

`exam` is a new column

It has the names from the old columns

`score` is a new column

It has the exams score for a *single* exam

By default we just see the first 10 rows (but there are 60 total)

Pivoting case #2:

an observation is scattered across multiple rows

```
# A tibble: 60 x 3
   Name           exam    score
   <chr>          <chr>   <chr>
 1 al-Jafri, Anas Score 1 17 / 25
 2 al-Jafri, Anas Score 2 41 / 50
 3 al-Jafri, Anas Score 3 18 / 25
 4 al-Jafri, Anas Test 1  87 / 100
 5 al-Jafri, Anas Midterm 89 / 100
 6 al-Jafri, Anas Final   90 / 100
 7 Drum, Anyssa   Score 1 19 / 25
 8 Drum, Anyssa   Score 2 40 / 50
 9 Drum, Anyssa   Score 3 19 / 25
10 Drum, Anyssa   Test 1  88 / 100
# … with 50 more rows
```

The long form of the data is *exam*-centric

Each observation (row) represents an exam

Each variable (column) represents something about the exam (student, exam name, score)

What if we want a *student*-centric form of the data?
That's the wide form
(students are the observation)

Pivoting case #2:

an observation is scattered across multiple rows

R Code

```
# Step 2, in reverse: Pivot the data back
# spread the individual tests back into
# separate columns
#
dat_wide <-
  pivot_wider(dat_long,
              names_from = exam,
              values_from = score)
print(dat_wide)
```

Output

```
# A tibble: 10 x 7
   Name            `Score 1` `Score 2` `Score 3` `Test 1` Midterm  Final
   <chr>           <chr>     <chr>     <chr>     <chr>    <chr>    <chr>
 1 al-Jafri, Anas  17 / 25   41 / 50   18 / 25   87 / 100 89 / 100 90 / 100
 2 Drum, Anyssa    19 / 25   40 / 50   19 / 25   88 / 100 87 / 100 92 / 100
 3 Gonzales, David 18 / 25   38 / 50   19 / 25   84 / 100 91 / 100 83 / 100
 4 Granger, Shannan 14 / 25  42 / 50   20 / 25   89 / 100 82 / 100 90 / 100
 5 Kwak, Surabhi   16 / 25   41 / 50   21 / 25   85 / 100 88 / 100 93 / 100
 6 Michels, Breana 17 / 25   46 / 50   18 / 25   80 / 100 85 / 100 82 / 100
 7 Reed, Tyler     19 / 25   39 / 50   17 / 25   78 / 100 94 / 100 88 / 100
 8 Smith, Tyler    16 / 25   40 / 50   19 / 25   79 / 100 90 / 100 86 / 100
 9 Smith, Westin   15 / 25   45 / 50   17 / 25   87 / 100 85 / 100 91 / 100
10 Vang, Seher     19 / 25   43 / 50   19 / 25   79 / 100 82 / 100 87 / 100
```

`pivot_wider` is the function
`names_from` give the *existing* column that contains the *new* columns we want to create
`values_from` gives the *existing* column that contains the values

Pivoting case #2:

an observation is scattered across multiple rows

What constitutes a variable and an observation depends on the circumstances
Pivoting gets you to wide(r) or long(er) forms of the data

back to where we started

```
# A tibble: 10 x 7
   Name             `Score 1` `Score 2` `Score 3` `Test 1` Midterm  Final
   <chr>            <chr>     <chr>     <chr>     <chr>    <chr>    <chr>
 1 al-Jafri, Anas   17 / 25   41 / 50   18 / 25   87 / 100 89 / 100 90 / 100
 2 Drum, Anyssa     19 / 25   40 / 50   19 / 25   88 / 100 87 / 100 92 / 100
 3 Gonzales, David  18 / 25   38 / 50   19 / 25   84 / 100 91 / 100 83 / 100
 4 Granger, Shannan 14 / 25   42 / 50   20 / 25   89 / 100 82 / 100 90 / 100
 5 Kwak, Surabhi    16 / 25   41 / 50   21 / 25   85 / 100 88 / 100 93 / 100
 6 Michels, Breana  17 / 25   46 / 50   18 / 25   80 / 100 85 / 100 82 / 100
 7 Reed, Tyler      19 / 25   39 / 50   17 / 25   78 / 100 94 / 100 88 / 100
 8 Smith, Tyler     16 / 25   40 / 50   19 / 25   79 / 100 90 / 100 86 / 100
 9 Smith, Westin    15 / 25   45 / 50   17 / 25   87 / 100 85 / 100 91 / 100
10 Vang, Seher      19 / 25   43 / 50   19 / 25   79 / 100 82 / 100 87 / 100
```

Separating:

a single value consists of multiple pieces of data

	Quizzes			Tests		
Name	**Score 1**	**Score 2**	**Score 3**	**Test 1**	**Midterm**	**Final**
al-Jafri, Anas	17 / 25	41 / 50	18 / 25	87 / 100	89 / 100	90 / 100
Drum, Anyssa	19 / 25	40 / 50	19 / 25	88 / 100	87 / 100	92 / 100
Gonzales, David	18 / 25	38 / 50	19 / 25	84 / 100	91 / 100	83 / 100
Granger, Shannan	14 / 25	42 / 50	20 / 25	89 / 100	82 / 100	90 / 100
Kwak, Surabhi	16 / 25	41 / 50	21 / 25	85 / 100	88 / 100	93 / 100
Michels, Breana	17 / 25	46 / 50	18 / 25	80 / 100	85 / 100	82 / 100
Reed, Tyler	19 / 25	39 / 50	17 / 25	78 / 100	94 / 100	88 / 100
Smith, Tyler	16 / 25	40 / 50	19 / 25	79 / 100	90 / 100	86 / 100
Smith, Westin	15 / 25	45 / 50	17 / 25	87 / 100	85 / 100	91 / 100
Vang, Seher	19 / 25	43 / 50	19 / 25	79 / 100	82 / 100	87 / 100

The scores are actually two pieces of data

points earned / total points

We can't perform calculations on the scores in the current form

Need to separate the data

Separating:

a single value consists of multiple pieces of data

R Code

```
# .. code continued
# Step 3: separate the scores
# need to use the long form
#
dat_new <-
 separate(dat_long, score,
          into = c("points", "total"),
          sep = " / ", convert = TRUE)
print(dat_new)
```

Output

```
# A tibble: 60 x 4
   Name            exam    points total
   <chr>           <chr>    <int> <int>
 1 al-Jafri, Anas Score 1     17    25
 2 al-Jafri, Anas Score 2     41    50
 3 al-Jafri, Anas Score 3     18    25
 4 al-Jafri, Anas Test 1      87   100
 5 al-Jafri, Anas Midterm     89   100
 6 al-Jafri, Anas Final       90   100
 7 Drum, Anyssa   Score 1     19    25
 8 Drum, Anyssa   Score 2     40    50
 9 Drum, Anyssa   Score 3     19    25
10 Drum, Anyssa   Test 1      88   100
# … with 50 more rows
```

`separate` is the function
`score` in the column where we want to separate the data
`into` gives the *new* column names where the separated data will go
`sep` gives the part of the value that separates the individual pieces of data
`convert` say to convert the data into numbers after separating (they are currently “words”)

Separating:

a single value consists of multiple pieces of data

```
# A tibble: 60 x 4
   Name           exam    points total
   <chr>          <chr>    <int> <int>
 1 al-Jafri, Anas Score 1     17    25
 2 al-Jafri, Anas Score 2     41    50
 3 al-Jafri, Anas Score 3     18    25
 4 al-Jafri, Anas Test 1      87   100
 5 al-Jafri, Anas Midterm     89   100
 6 al-Jafri, Anas Final       90   100
 7 Drum, Anyssa   Score 1     19    25
 8 Drum, Anyssa   Score 2     40    50
 9 Drum, Anyssa   Score 3     19    25
10 Drum, Anyssa   Test 1      88   100
# … with 50 more rows
```

The scores have been converted into two new columns: points & total

We can now perform calculations on these numbers

This wasn't possible before because they were "words"

High-level takeaways

- Real world data is often messy, and we (usually) need to make it tidy before you can start to work with it

- Messiness can occur in different ways
  - A variable is spread across multiple columns → `pivot_longer`
  - An observation is spread across multiple rows → `pivot_wider`
  - Multiple pieces of data exist in a single cell → `separate`
  - … lots of other (often unexpected) ways…

- As you work with data more, you'll begin to see patterns in messy data and how to fix them to get tidy data

Resources

- Paper on tidy data by H. Wickham
  https://vita.had.co.nz/papers/tidy-data.pdf
- R For Data Science, section on tidy data
  https://r4ds.had.co.nz/tidy-data.html
- tidyr package website
  https://tidyr.tidyverse.org
- Data importing cheat sheet
  https://github.com/rstudio/cheatsheets/blob/master/data-import.pdf

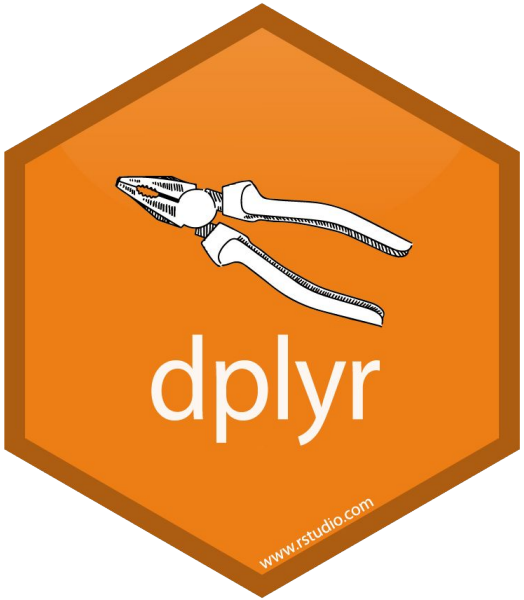
Data Manipulation & Pipelines with dplyr

ASMS 2021: R Short Course

Goals for this module

- Understand the fundamental operations that underlie most common data manipulation challenges
- Learn about the dplyr R package and how it provides the functionality perform these data manipulation operations
- Learn about the pipe: %>%
- Learn how the pipe can be used to combine multiple data manipulation operations together into data pipelines
- Get a basic understanding of how to use dplyr for your own work

dplyr is a tidyverse R package for data manipulation

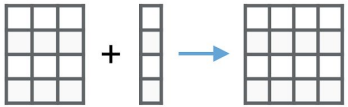
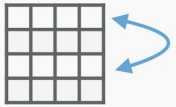
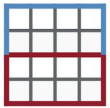
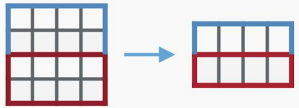


- Defines the fundamental operations that encompass most data analysis tasks
- Assumes you already have tidy data
- Uses a pipeline coding structure to perform complex operations a straight-forward, natural way

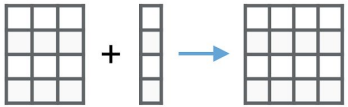
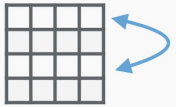
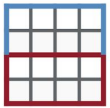
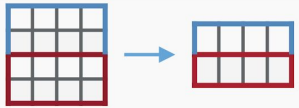
dplyr defines fundamental data manipulation *verbs*

- dplyr formalizes the fundamental operations that occur when working with data into a set of “verbs”
- These verbs are represented as functions that you can use to manipulate data in R
- There are a small number of these verbs, which makes them easy(er) to remember and work with
- Helps you to focus on the question you want to answer rather than the mechanics of how to answer the question

Fundamental data manipulation operations

Operation		Use case
Add a column		compute new data from existing variables, join new data
Pick specific columns		focus in on specific variables
Subset to specific rows		focus on a specific sub-group in the data
Reorder/sort rows		understand the order of the data, find top/bottom observations
Group subsets		want to analyze sub-groups in your data
Summarize rows		compute summary values across multiple rows, useful with grouping

Fundamental data manipulation operations

Operation		Use case	dplyr verb
Add a column		compute new data from existing variables, join new data	`mutate`
Pick specific columns		focus in on specific variables	`select`
Subset to specific rows		focus on a specific sub-group in the data	`filter`
Reorder/sort rows		understand the order of the data, find top/bottom observations	`arrange`
Group subsets		want to analyze sub-groups in your data	`group_by`
Summarize rows		compute summary values across multiple rows, useful with grouping	`summarize`

dplyr verbs represent high-level data manipulation tasks, not the specific “math” operations

- dplyr doesn’t perform specific mathematical operations, statistical summaries, statistical tests, machine learning, data visualization, etc., etc..
- Rather it provides a structural framework inside of which all of these things can be accomplished in an organized, streamlined way
- You’ll still need to know how to perform these specific operations in R, but can then use them in combination with dplyr to structure your analysis

Practice with verbs

Recall the long form of our student exam grades data that we tidied?

What verb would we use to compute the percentage correct?

```
# A tibble: 60 x 4
   Name           exam    points total
   <chr>          <chr>    <int> <int>
 1 al-Jafri, Anas Score 1     17    25
 2 al-Jafri, Anas Score 2     41    50
 3 al-Jafri, Anas Score 3     18    25
 4 al-Jafri, Anas Test 1      87   100
 5 al-Jafri, Anas Midterm     89   100
 6 al-Jafri, Anas Final       90   100
 7 Drum, Anyssa   Score 1     19    25
 8 Drum, Anyssa   Score 2     40    50
 9 Drum, Anyssa   Score 3     19    25
10 Drum, Anyssa   Test 1      88   100
# … with 50 more rows
```

Practice with verbs

Recall the long form of our student exam grades data

What verb would we use to compute the percentage correct?

```
# A tibble: 60 x 4
   Name           exam    points total
   <chr>          <chr>    <int> <int>
 1 al-Jafri, Anas Score 1     17    25
 2 al-Jafri, Anas Score 2     41    50
 3 al-Jafri, Anas Score 3     18    25
 4 al-Jafri, Anas Test 1      87   100
 5 al-Jafri, Anas Midterm     89   100
 6 al-Jafri, Anas Final       90   100
 7 Drum, Anyssa   Score 1     19    25
 8 Drum, Anyssa   Score 2     40    50
 9 Drum, Anyssa   Score 3     19    25
10 Drum, Anyssa   Test 1      88   100
# … with 50 more rows
```

`mutate`

We need to make a *new* column that has the computed value, *percentage correct*

R Code

```
mutate(dat_tidy,
       pct_correct = points / total)
```

Practice with verbs

dplyr applies this formula to every row in the input data – it's like dragging a formula cell in Excel!

R Code

```
mutate(dat_tidy, pct_correct = points / total)
```

Input (dat_tidy)

```
# A tibble: 60 x 4
   Name           exam     points total
   <chr>          <chr>     <int> <int>
 1 al-Jafri, Anas Score 1     17    25
 2 al-Jafri, Anas Score 2     41    50
 3 al-Jafri, Anas Score 3     18    25
 4 al-Jafri, Anas Test 1      87   100
 5 al-Jafri, Anas Midterm     89   100
 6 al-Jafri, Anas Final       90   100
 7 Drum, Anyssa   Score 1     19    25
 8 Drum, Anyssa   Score 2     40    50
 9 Drum, Anyssa   Score 3     19    25
10 Drum, Anyssa   Test 1      88   100
# … with 50 more rows
```

Output

```
# A tibble: 60 x 5
   Name           exam     points total pct_correct
   <chr>          <chr>     <int> <int>       <dbl>
 1 al-Jafri, Anas Score 1     17    25        0.68
 2 al-Jafri, Anas Score 2     41    50        0.82
 3 al-Jafri, Anas Score 3     18    25        0.72
 4 al-Jafri, Anas Test 1      87   100        0.87
 5 al-Jafri, Anas Midterm     89   100        0.89
 6 al-Jafri, Anas Final       90   100        0.9
 7 Drum, Anyssa   Score 1     19    25        0.76
 8 Drum, Anyssa   Score 2     40    50        0.8
 9 Drum, Anyssa   Score 3     19    25        0.76
10 Drum, Anyssa   Test 1      88   100        0.88
# … with 50 more rows
```

A few notes about writing dplyr code

```
mutate(dat_long,
       pct_correct = points / total)
```

The input data frame is always the first argument to dplyr verbs

This second function argument was placed on it's own line!

R doesn't care much about newlines as long as the other syntax is correct.

Feel free to add new lines to let your code breathe!

These refer to column names (i.e. variables) in the input data, or columns we're going to create

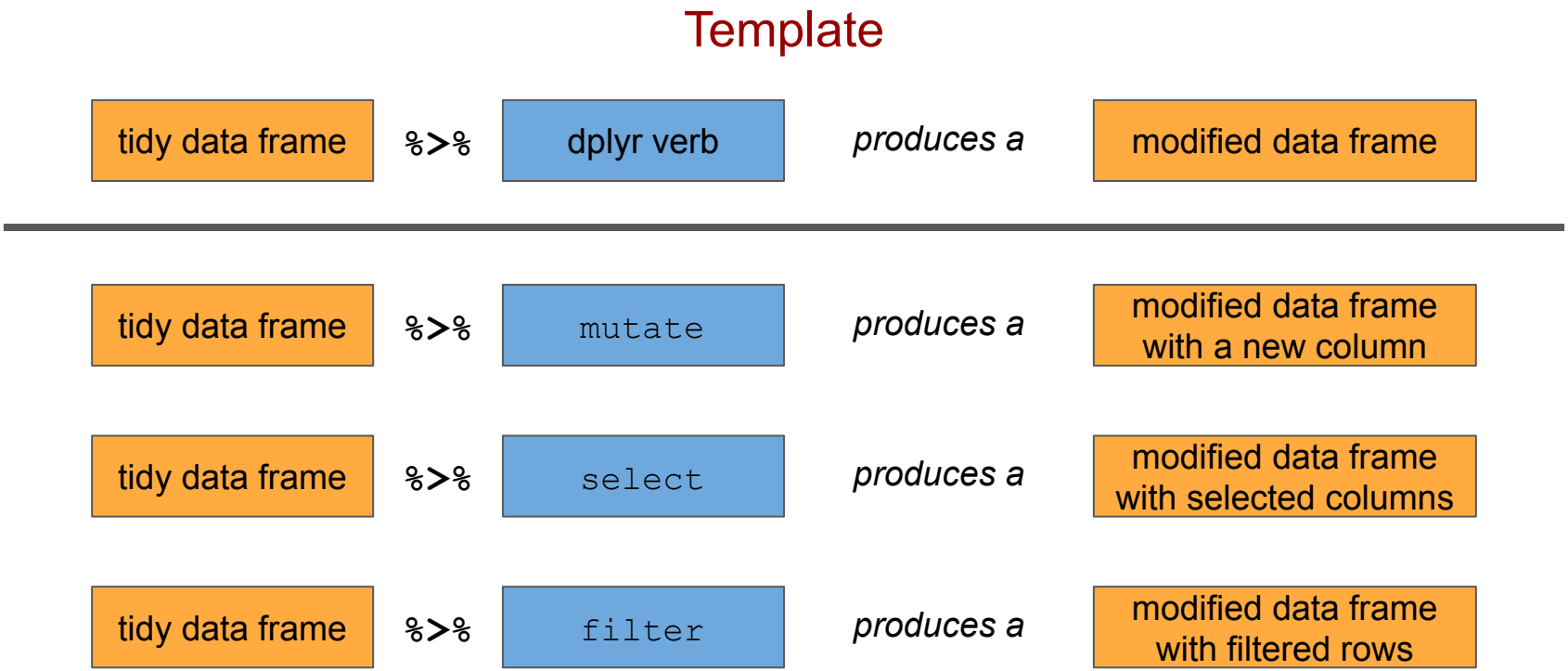
In dplyr code, you don't put quotes ("") around them

The pipe operator: %>%

- The pipe operator takes an input data frame and “feeds” it into a function … the function performs an operation on the input data
- In code, the pipe operator is three characters
- Yes -- you do have to type the three characters (or use a keyboard shortcut)
- Yes -- it looks strange, think of %>% as a pipe, funneling the input data into the function to do some work

Template

The pipe operator: %>%



Practice with pipes

R Code without pipes

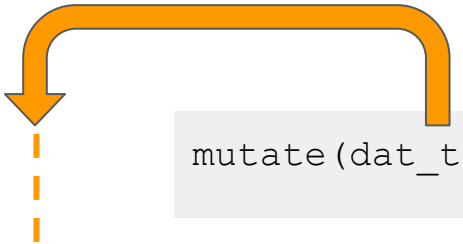
```
mutate(dat_tidy, pct_correct = points / total)
```

R Code *with* pipes

```
dat_tidy %>% mutate(pct_correct = points / total)
```

Both lines of code do exactly the same thing, but are written differently
Can you spot the differences?

Practice with pipes



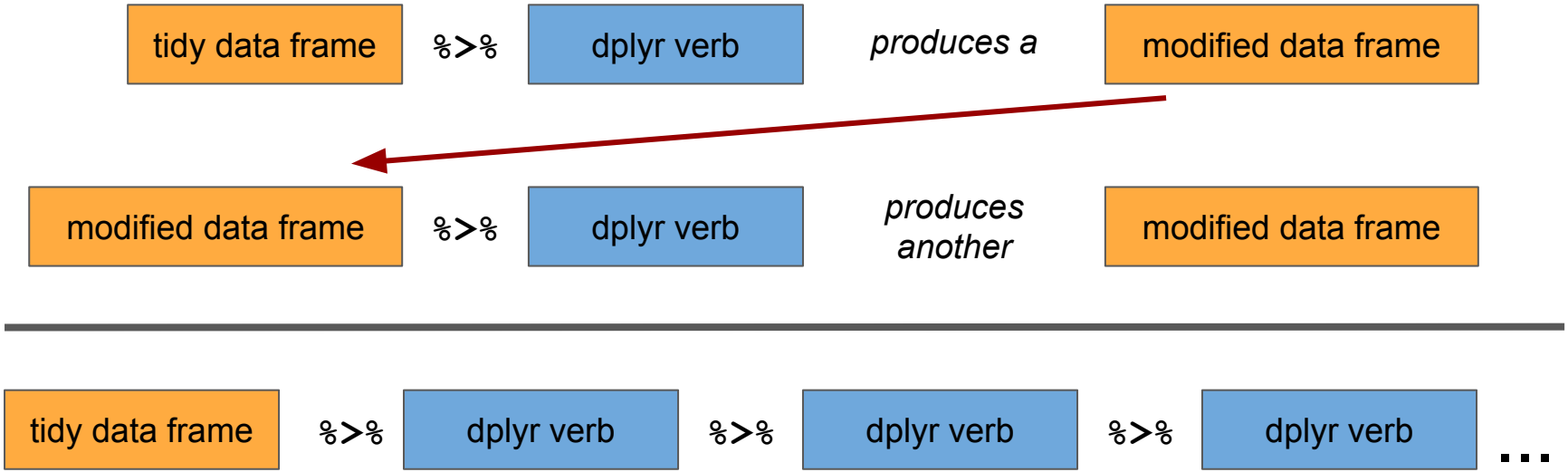
```
mutate(dat_tidy, pct_correct = points / total)
```

```
dat_tidy %>% mutate(pct_correct = points / total)
```

Two different ways to code the same thing

Take the first argument (the input data frame),
move it to the front and add a pipe, %>%

The pipe operator can chain multiple verbs (pipeline!)



You can chain (pipe) together dplyr verbs to produce pipelines that embody complex manipulations

Practice with pipes and pipelines

Goal

- compute the fraction correct for each exam
- get only the exams with > 90% correct
- order the results by the percent correct

Input Data

```
# A tibble: 60 x 4
   Name           exam    points total
   <chr>          <chr>    <int> <int>
 1 al-Jafri, Anas Score 1     17    25
 2 al-Jafri, Anas Score 2     41    50
 3 al-Jafri, Anas Score 3     18    25
 4 al-Jafri, Anas Test 1      87   100
 5 al-Jafri, Anas Midterm     89   100
 6 al-Jafri, Anas Final       90   100
 7 Drum, Anyssa   Score 1     19    25
 8 Drum, Anyssa   Score 2     40    50
 9 Drum, Anyssa   Score 3     19    25
10 Drum, Anyssa   Test 1      88   100
# … with 50 more rows
```

Practice with pipes and pipelines

R Code

```
dat_tidy %>%
  mutate(pct_correct = points / total) %>%
  filter(pct_correct > 0.9) %>%
  arrange(pct_correct)
```

Output

```
# A tibble: 6 x 5
  Name             exam    points total pct_correct
  <chr>            <chr>    <int> <int>       <dbl>
1 Gonzales, David  Midterm     91   100        0.91
2 Smith, Westin    Final       91   100        0.91
3 Drum, Anyssa     Final       92   100        0.92
4 Michels, Breana  Score 2     46    50        0.92
5 Kwak, Surabhi    Final       93   100        0.93
6 Reed, Tyler      Midterm     94   100        0.94
```

Practice with pipes and pipelines

The problem statement maps directly to the code
If you can say it, you can code it!

- compute the fraction correct for each exam
- get only the exams with > 90% correct
- order the results by the percent correct

```
dat_tidy %>%

  mutate(pct_correct = points / total) %>%


  filter(pct_correct > 0.9) %>%


  arrange(pct_correct)
```

group_by + summarize is extremely powerful!

A common analysis task it to compute summaries by groups

- Compute the average test score for each student
- Count the number of analytes detected in each experimental run
- Compute the coefficient of variation for each analyte in a given run

In a dplyr pipeline, `group_by` followed by `summarize` first groups the data by a specified variable, then summarizes the data within those groups

Practice with group_by + summarize

Goal

- compute the fraction correct for each exam
- compute min, mean and max fraction correct for each student

Input Data

```
# A tibble: 60 x 4
   Name           exam    points total
   <chr>          <chr>    <int> <int>
 1 al-Jafri, Anas Score 1     17    25
 2 al-Jafri, Anas Score 2     41    50
 3 al-Jafri, Anas Score 3     18    25
 4 al-Jafri, Anas Test 1      87   100
 5 al-Jafri, Anas Midterm     89   100
 6 al-Jafri, Anas Final       90   100
 7 Drum, Anyssa   Score 1     19    25
 8 Drum, Anyssa   Score 2     40    50
 9 Drum, Anyssa   Score 3     19    25
10 Drum, Anyssa   Test 1      88   100
# … with 50 more rows
```

What should we group by?

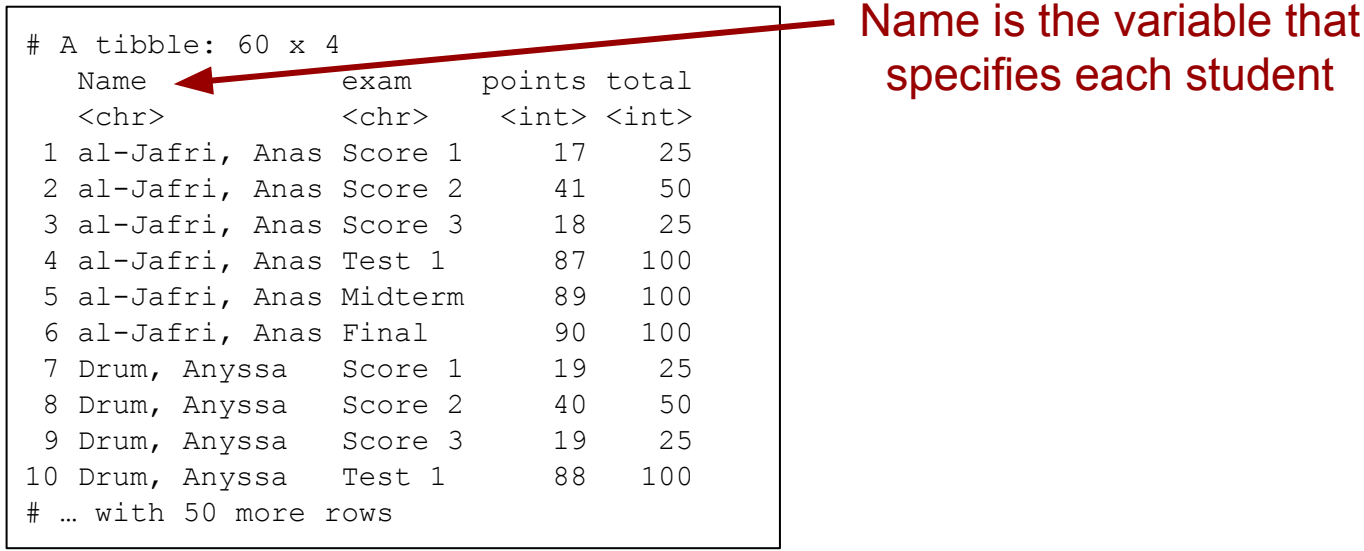
Practice with group_by + summarize

Goal

- compute the fraction correct for each exam
- compute min, mean and max fraction correct for each student

Input Data

```
# A tibble: 60 x 4
   Name           exam    points total
   <chr>          <chr>    <int> <int>
 1 al-Jafri, Anas Score 1     17    25
 2 al-Jafri, Anas Score 2     41    50
 3 al-Jafri, Anas Score 3     18    25
 4 al-Jafri, Anas Test 1      87   100
 5 al-Jafri, Anas Midterm     89   100
 6 al-Jafri, Anas Final       90   100
 7 Drum, Anyssa   Score 1     19    25
 8 Drum, Anyssa   Score 2     40    50
 9 Drum, Anyssa   Score 3     19    25
10 Drum, Anyssa   Test 1      88   100
# … with 50 more rows
```



Practice with pipes and pipelines

R Code

```
dat_tidy %>%
  mutate(pct_correct = points / total) %>%
  group_by(Name) %>%
  summarize(min_score = min(pct_correct),
            max_score = max(pct_correct),
            mean_score = mean(pct_correct))
```

all of the calculations after the `group_by` will occur separately for each student

these are just the names of new variables (columns) that will be created -- you can name them whatever you want

these are the calculations that will be performed on each group (student) -- you can add as many as you need

Output

```
# A tibble: 10 x 4
   Name              min_score max_score mean_score
   <chr>                 <dbl>     <dbl>      <dbl>
 1 al-Jafri, Anas         0.68       0.9      0.813
 2 Drum, Anyssa           0.76      0.92      0.832
 3 Gonzales, David        0.72      0.91      0.803
 4 Granger, Shannan       0.56       0.9      0.802
 5 Kwak, Surabhi          0.64      0.93      0.827
 6 Michels, Breana        0.68      0.92      0.798
 7 Reed, Tyler            0.68      0.94      0.803
 8 Smith, Tyler           0.64       0.9      0.792
 9 Smith, Westin          0.6       0.91      0.802
10 Vang, Seher            0.76      0.87      0.81
```

Review of the dplyr verbs and common use cases

`mutate`

- adds a new column to your data table
- often used when you need to transform/combine with existing data in the table
- example: ratio of two variables, log transform a variable, add variables together

`select`

- subsets your data to a specific set of columns (variables)
- used to focus in on the variables you want, remove ones you don’t need
- mainly an “organizational” task

Review of the dplyr verbs and common use cases

`filter`

- subset your data to a specific set of observations (rows)
- used to filter data, subset based on conditions within the data
- example: filter to values > than a threshold, exclude data from a specific run

`arrange`

- reorders the observations (rows) based on values in the data
- often used to but important data at the top of the table, or to rank observations

Review of the dplyr verbs and common use cases

`group_by`

- groups your data based upon a variable
- usually the variable is categorical-like (a string or a factor)
- doesn't do anything by itself -- need to combine with another verb to do something, often with `summarize`

`summarize`

- summarizes all of the observations to a single value, if used with `group_by`, you get one value per group
- you specify how the observation will be summarized, usually with a function
- does not require `group_by` (but you'll typically want to use it)

High-level takeaways

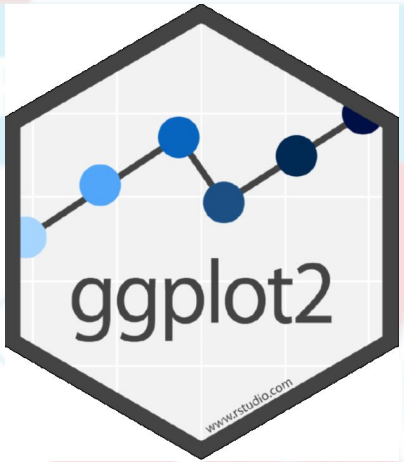
- dplyr provides a general framework for manipulating tidy data
- The fundamental manipulations are specified as verbs (functions)
- These verbs can be combined (piped) together to create data processing pipelines
- Often, these pipelines can often be translated from “human-form” to R code in a straight-forward, natural way
- dplyr helps you to focus on answering the question rather than the mechanics of how to answer the question

Resources

- R for Data Science
  https://r4ds.had.co.nz

- dplyr package website
  https://dplyr.tidyverse.org

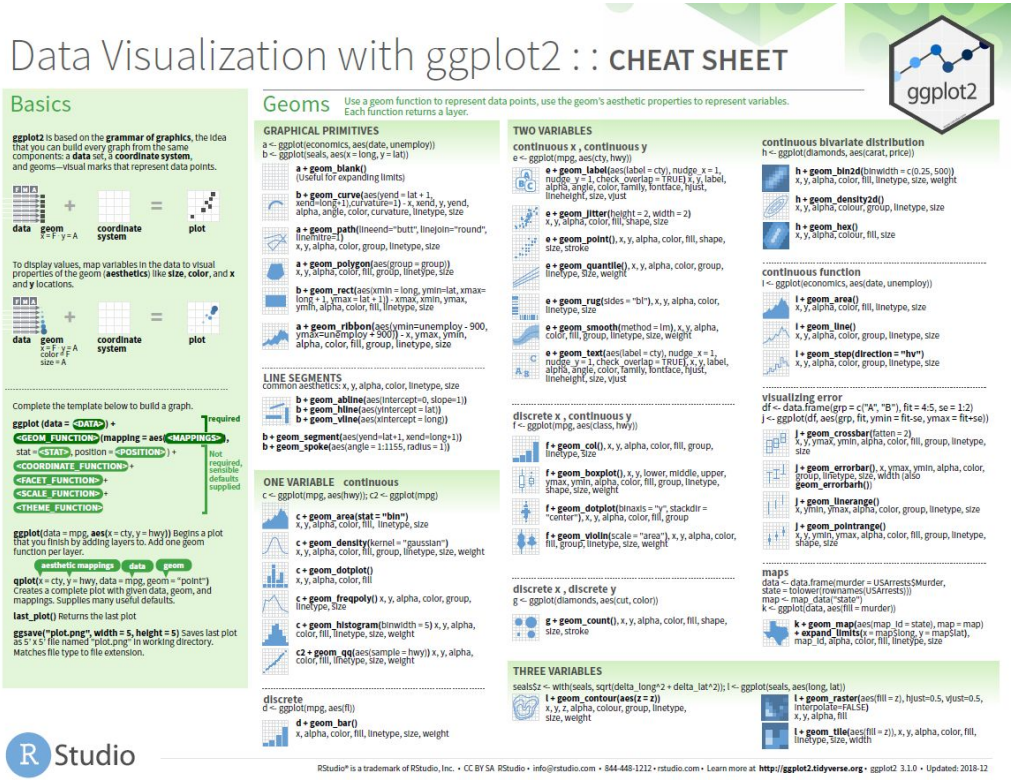
- Data transformation with dplyr cheat sheet
  https://github.com/rstudio/cheatsheets/blob/master/data-transformation.pdf

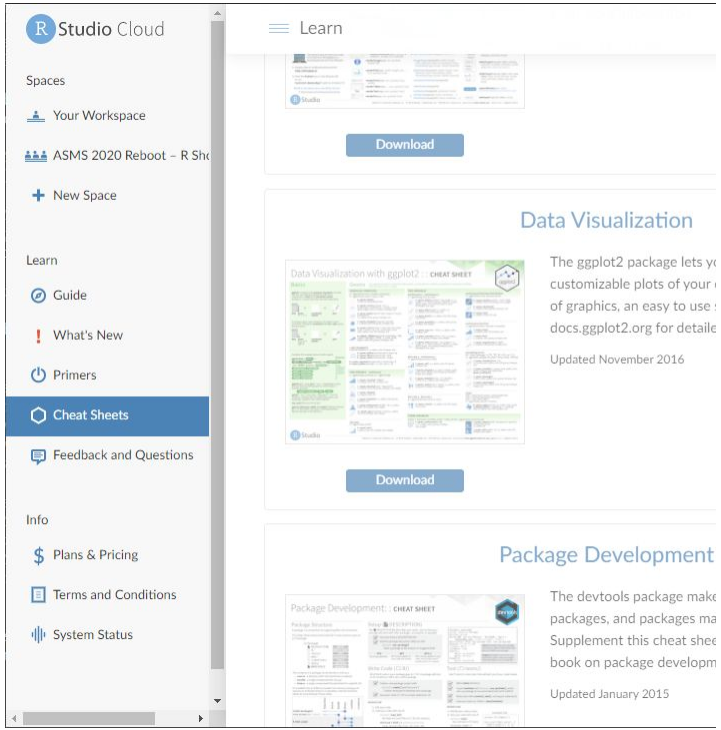


R::ggplot

Visualization Basics

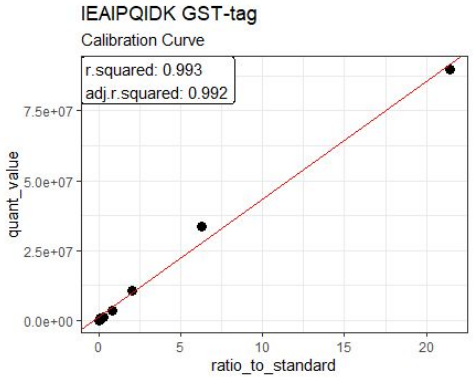
ASMS 2021: R Short Course

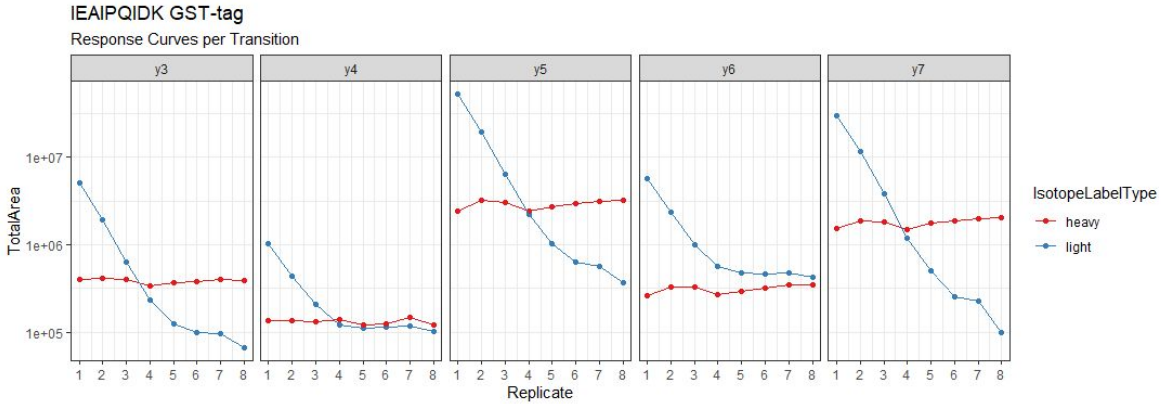


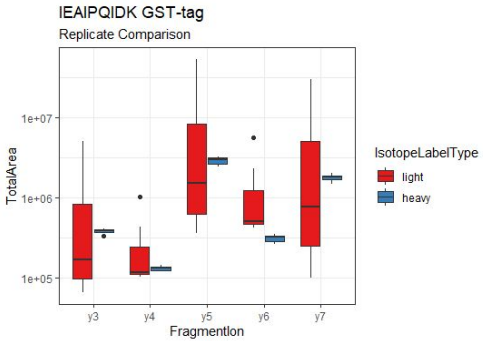


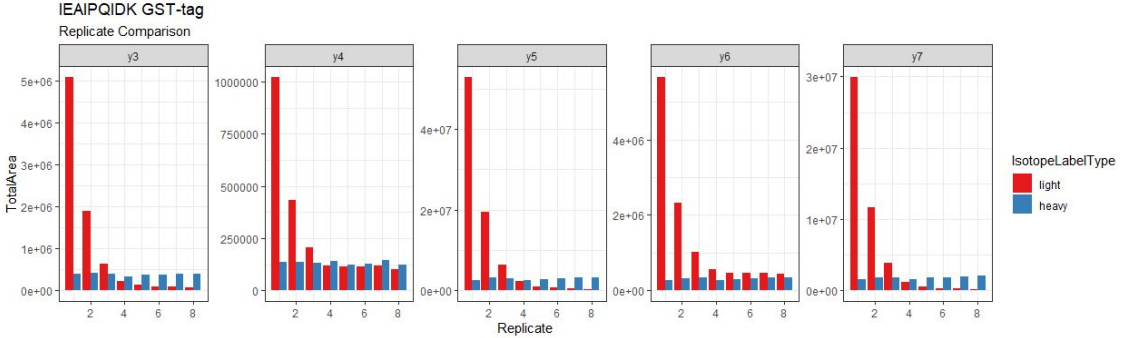
https://github.com/rstudio/cheatsheets/blob/master/data-visualization-2.1.pdf

Sample of Plots Covered in this short course

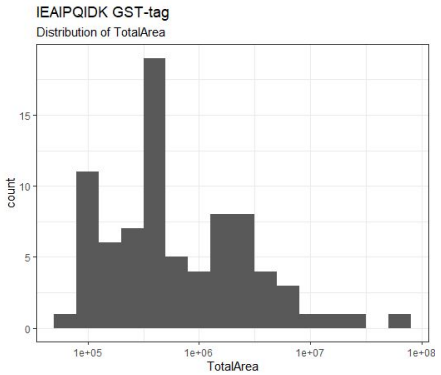


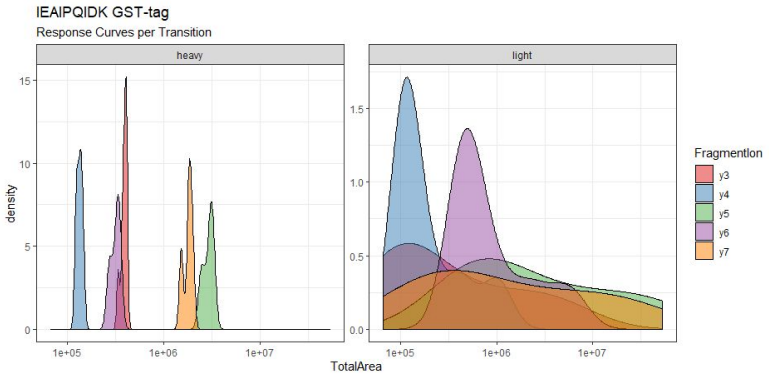


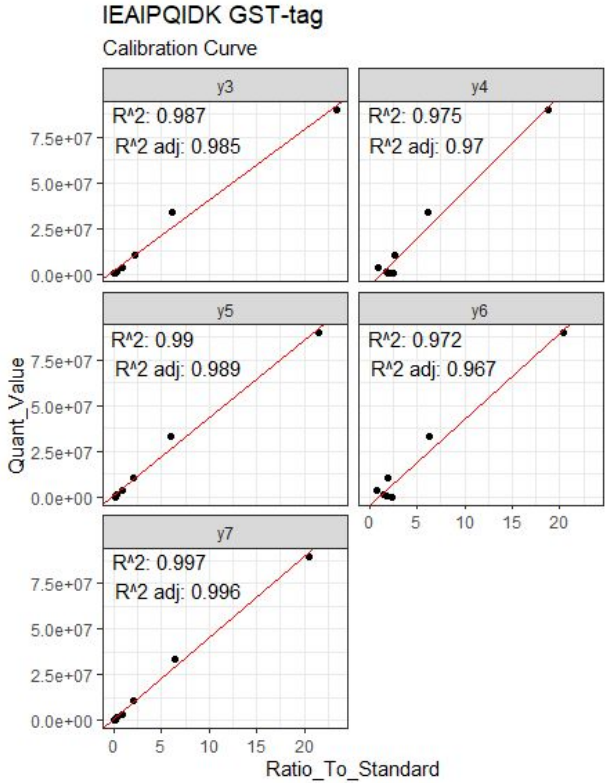




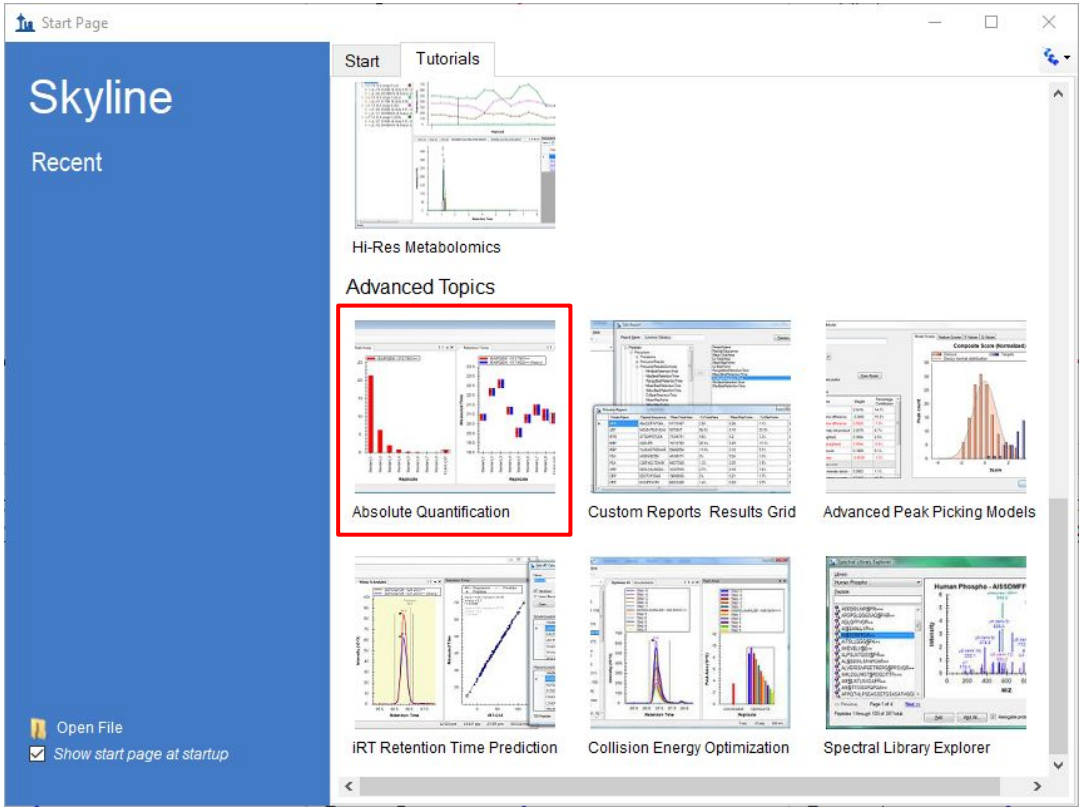
Challenges


IEAIPQIDK GST-tag
Distribution of TotalArea
count
TotalArea
1e+05
1e+06
1e+07
1e+08


IEAIPQIDK GST-tag
Response Curves per Transition
heavy
light
density
TotalArea
Fragmentlon
y3
y4
y5
y6
y7


IEAIPQIDK GST-tag
Calibration Curve
y3
R^2: 0.987
R^2 adj: 0.985
y4
R^2: 0.975
R^2 adj: 0.97
y5
R^2: 0.99
R^2 adj: 0.989
y6
R^2: 0.972
R^2 adj: 0.967
y7
R^2: 0.997
R^2 adj: 0.996
Quant_Value
Ratio_To_Standard

Data Source: Skyline Tutorial -- Absolute Quantification



Follow the tutorial and export the following data along the way

- Transition List
- Report: Calibration Values
- Chromatograms

Exercise 0 covers the data import from CVS and TSV into a RDS tibble object. There are some advanced aspects such as value separation and nested tables.

Goals for this module

- Introduction to data visualization using R package **ggplot2**
  - organizational structure
  - coding syntax

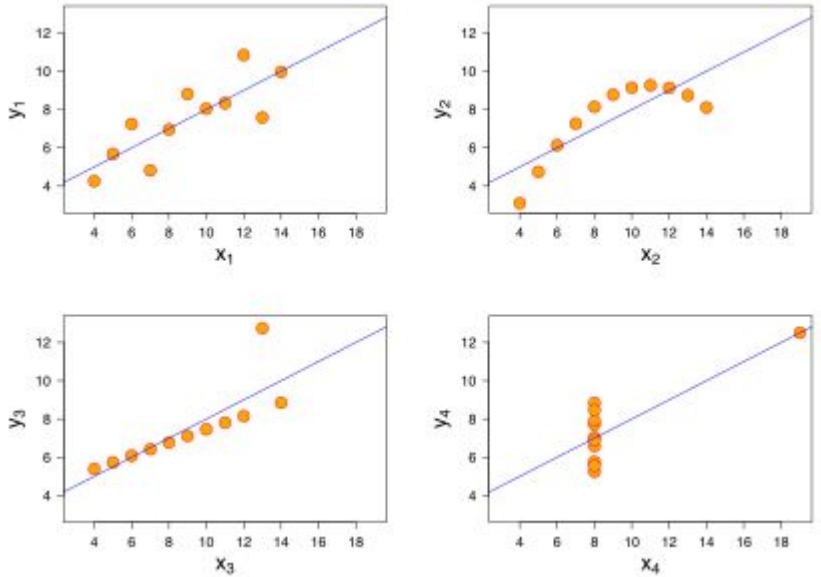
- Demonstrated Application
  - basic plotting

- Practical Application
  - plotting a regression line with R^2 stats

Why visualize your data

Anscombe's quartet is an example that comprises four data sets that have nearly identical simple descriptive statistics, yet have very different distributions and appear very different when graphed.

Property	Value	Accuracy
Mean of *x*	9	exact
Sample variance of *x* :	11	exact
Mean of *y*	7.50	to 2 decimal places
Sample variance of *y* :	4.125	±0.003
Correlation between *x* and *y*	0.816	to 3 decimal places
Linear regression line	*y* = 3.00 + 0.500*x*	to 2 and 3 decimal places, respectively



Why ggplot

based on the grammar of graphics

“the idea that you can build every graph from the same few components”

data	a table of numeric and categorical values (data.frame, tibble)
geoms	a geometric object or visual representation, that can be layered
aesthetics	how variables in the data are mapped to visual properties
coordinate	orientation of the data points (* optional)

```
ggplot(data, aes(x,y)) + geom_point() + coord_cartiesian()

# combined with dplyr makes for a readable process
data %>% ggplot(aes(x,y)) + geom_point()

```

The basic ggplot2 plotting template

The variable name of the data table you want to create a plot with

```
ggplot(data = <DATA>) +
      <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

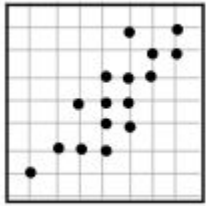
A ggplot2 function that defines the type of geometric representation you want to use, e.g.

```
geom_point
geom_line
geom_histogram
geom_boxplot
```

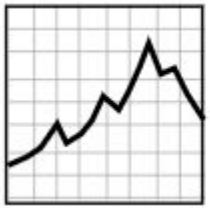
Expression that link columns in <DATA> to aesthetic properties

ggplot **geoms**: simple

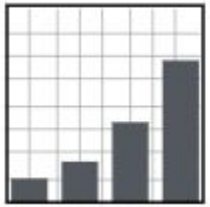
```
1   data %>% ggplot(aes(x,y)) + geom_point()
```



```
1   data %>% ggplot(aes(x,y)) + geom_line()
```

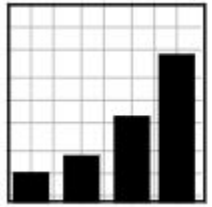


```
1   data %>% ggplot(aes(x)) + geom_bar()
```

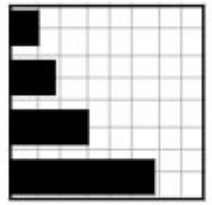


ggplot **coordinates**: data projections

```
1    data %>% ggplot(aes(x)) + geom_bar()  +
2        coord_cartiesian()
```



```
1    data %>% ggplot(aes(x)) + geom_bar()  +
2        coord_flip()
```



```
1    data %>% ggplot(aes(x)) + geom_bar()  +
2        coord_polar()
```

dplyr ~ ggplot

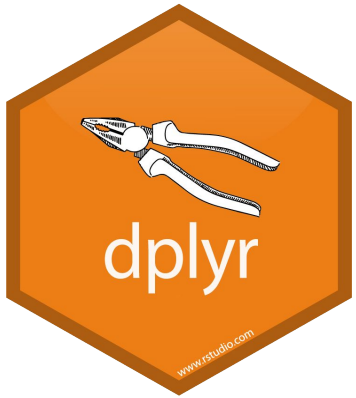
Quick note on tidyverse *joiners* …

dplyr → use the **%>%** to join procedural functions in a specific order

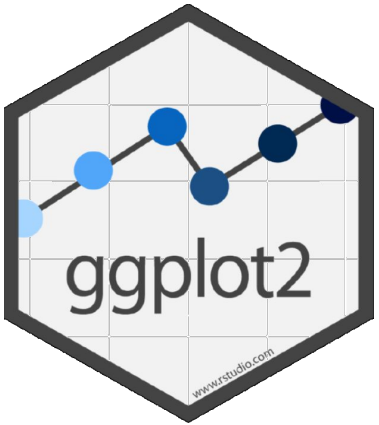
ggplot → use the **+** to join layers and customizations, in any order

* using a **%>%** in a **ggplot** statement results in an error ***"Did you use %>% instead of +?"***

Combining dplyr and ggplot



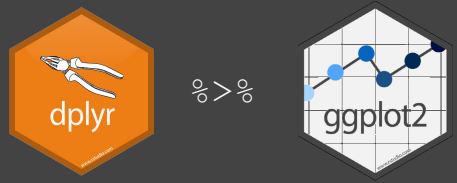
%>%



`data.frame`
`tibble`

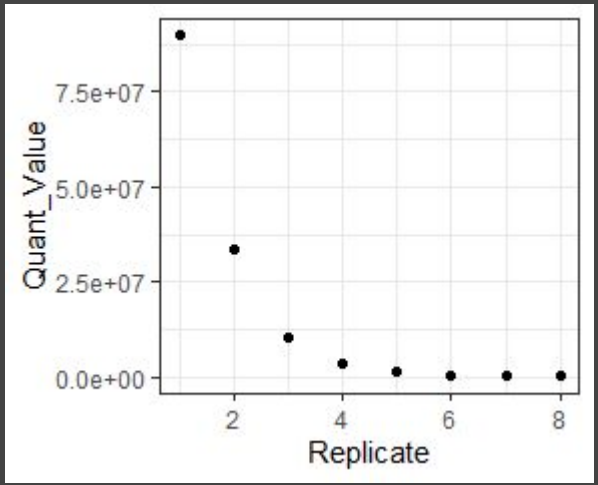
dplyr → ggplot

```
# dat is our original data object
dat <- "path" %>%
    read_csv()

# plot 1
dat %>%
    ggplot(aes(x,y)) +
    geom_point()

# plot 2  ~  var only exists as a factor here
dat %>%
    mutate(var = as.factor(var) ) %>%    # wanted discrete colors
    ggplot(aes(x,y, color=var)) +
    geom_point() +
    geom_line()
```



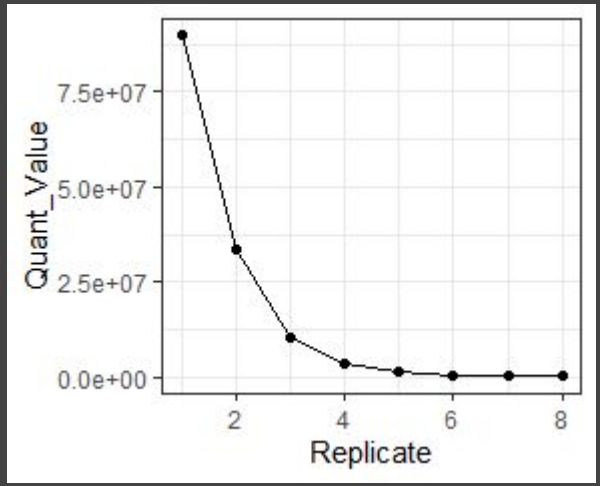
dplyr → ggplot

```
# dat is our original data object
dat <- "./data/raw/samples_1-8_heavy-light_calibration.csv" %>%
    read_csv()

# plot 1
dat %>%
    ggplot(aes(Replicate, Quant_Value)) +
    geom_point()
```



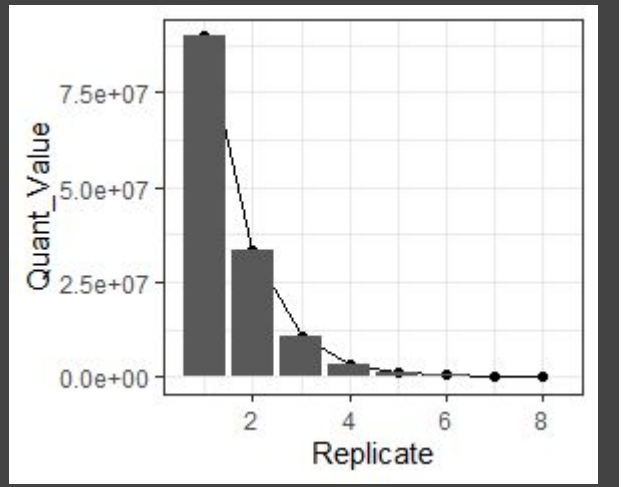
dplyr → ggplot

```
# dat is our original data object
dat <- "./data/raw/samples_1-8_heavy-light_calibration.csv" %>%
    read_csv()

# plot 1
dat %>%
    ggplot(aes(Replicate, Quant_Value)) +
    geom_point() +
    geom_line()
```



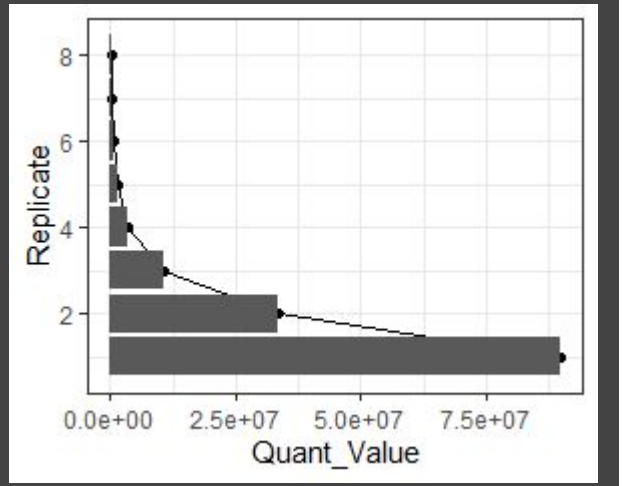
dplyr → ggplot

```
# dat is our original data object
dat <- "./data/raw/samples_1-8_heavy-light_calibration.csv" %>%
    read_csv()

# plot 1
dat %>%
    ggplot(aes(Replicate, Quant_Value)) +
    geom_point() +
    geom_line() +
    geom_bar(stat="identity")
```

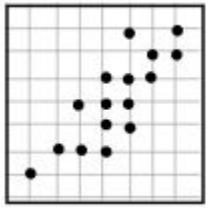


dplyr → ggplot

```
# dat is our original data object
dat <- "./data/raw/samples_1-8_heavy-light_calibration.csv" %>%
    read_csv()

# plot 1
dat %>%
    ggplot(aes(Replicate, Quant_Value)) +
    geom_point() +
    geom_line() +
    geom_bar(stat="identity") +
    coord_flip()
```

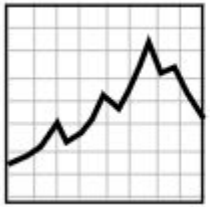


ggplot aesthetics



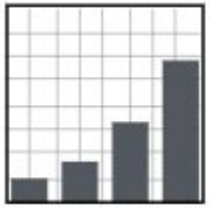
geom_point()

x, y, alpha, color, fill, shape, size



geom_line()

x, y, alpha, color, linetype, size



geom_bar()

x, alpha, color, fill, linetype, size, weight

Position

x = abscissa

y = ordinate

Appearance

Color

alpha = transparency

color = border / lines

fill = volume

Shape

shape = ie. circle, square

size = small to big

dplyr → ggplot

```
# dat is our original data object
dat <- "path" %>% read_csv()

# plot 1
dat %>%
    ggplot(aes(var_x, var_y)) +
    geom_point()
```

```
dat %>%
    ggplot() +
    geom_point(aes(var_x, var_y))
```

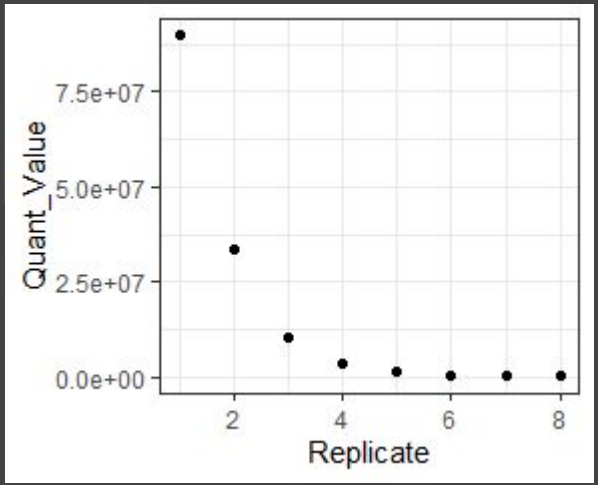
ggplot → invokes the plotting function → takes the **data object** and **aesthetics**

aes → aesthetic, or the plot appearance (x,y, color ...)

geom_point → the geometric function

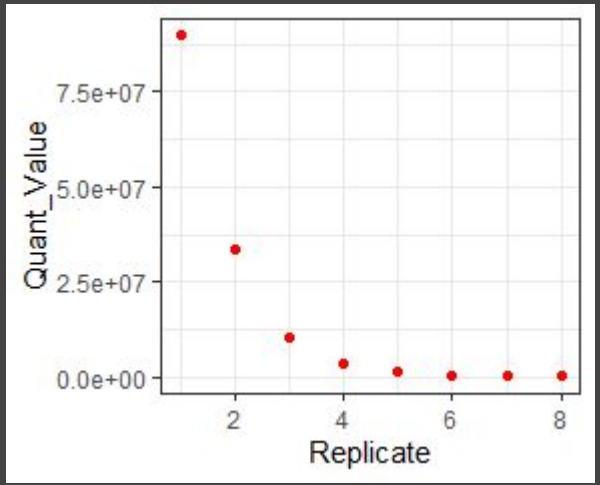
dplyr → ggplot

```
# dat is our original data object
dat <- "./data/raw/samples_1-8_heavy-light_calibration.csv" %>%
    read_csv()

# plot 1
dat %>%
    ggplot(aes(Replicate, Quant_Value)) +
    geom_point()
```



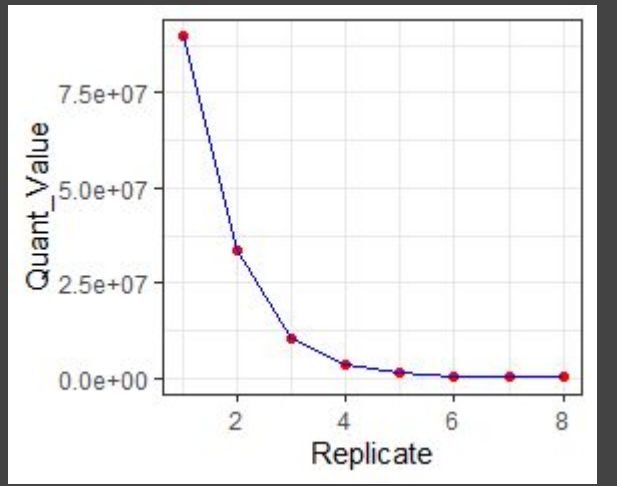
dplyr → ggplot

```
# dat is our original data object
dat <- "./data/raw/samples_1-8_heavy-light_calibration.csv" %>%
    read_csv()

# plot 1
dat %>%
    ggplot(aes(Replicate, Quant_Value)) +
    geom_point(color="red")
```



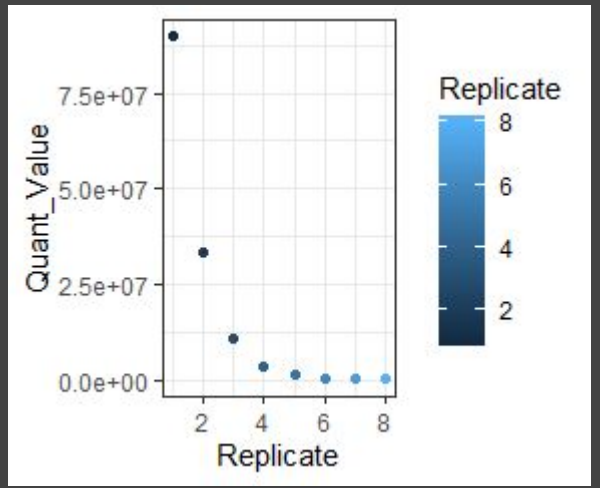
dplyr → ggplot

```
# dat is our original data object
dat <- "./data/raw/samples_1-8_heavy-light_calibration.csv" %>%
    read_csv()

# plot 1
dat %>%
    ggplot(aes(Replicate, Quant_Value)) +
    geom_point(color="red") +
    geom_line(color="blue")
```



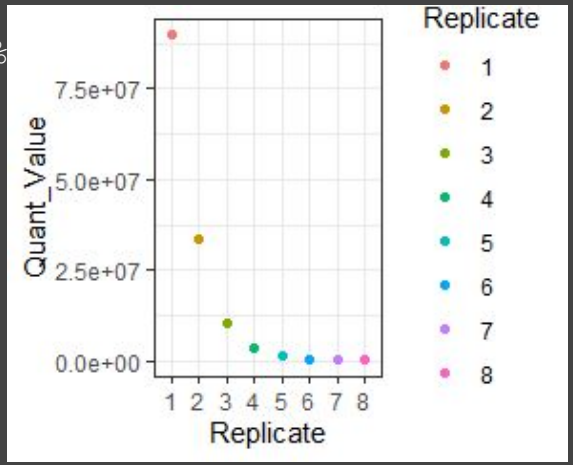
dplyr → ggplot

```r
# dat is our original data object
dat <- "./data/raw/samples_1-8_heavy-light_calibration.csv" %>%
    read_csv()

# plot 1
dat %>%
    ggplot(aes(Replicate, Quant_Value)) +
    geom_point(aes(color = Replicate))
```



dplyr → ggplot

```
# dat is our original data object
dat <- "./data/raw/samples_1-8_heavy-light_calibration.csv" %>%
    read_csv()

# plot 1
dat %>%
    mutate(Replicate = as.factor(Replicate) ) %>%
    ggplot(aes(Replicate, Quant_Value)) +
    geom_point(aes(color = Replicate))
```



dplyr → ggplot

```
1   # mapping variables to aesthetics
2   dat %>%
3       ggplot(aes(X, Y, color = A, fill = B, size = M, shape = N  )) +
4       geom_point()
5
6
7   # custom aesthetics
8   dat %>%
9       ggplot(aes(X, Y), color = "red", fill = "blue",
10                size = 2, shape = 3  ) +
11      geom_point()
12
13
14
15
```

inside the aes()

outside the aes()

dplyr → ggplot

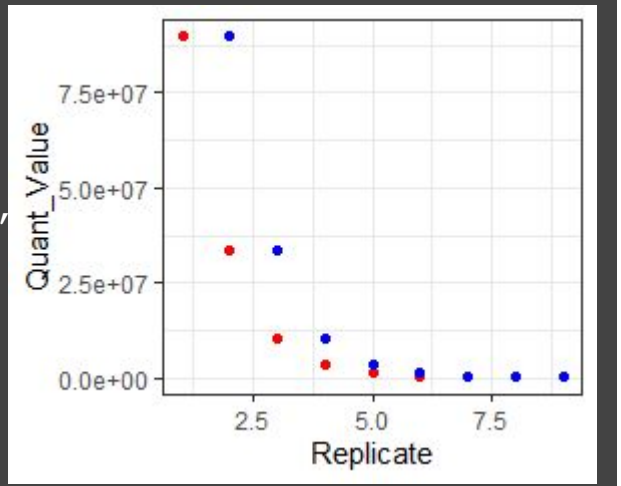
```
1   # mapping variables to aesthetics, globally
2   dat %>%
3       ggplot(aes(X, Y, color = A, fill = B)) +
4       geom_point() +
5       geom_line()
6
7   # mapping variables to aesthetics, locally
8   dat %>%
9       ggplot(aes(X, Y)) +
10      geom_point(aes(color = A)) +
11      geom_line(aes(color = B))
12
13
14
15
```

global aes(), point and line are the *same*

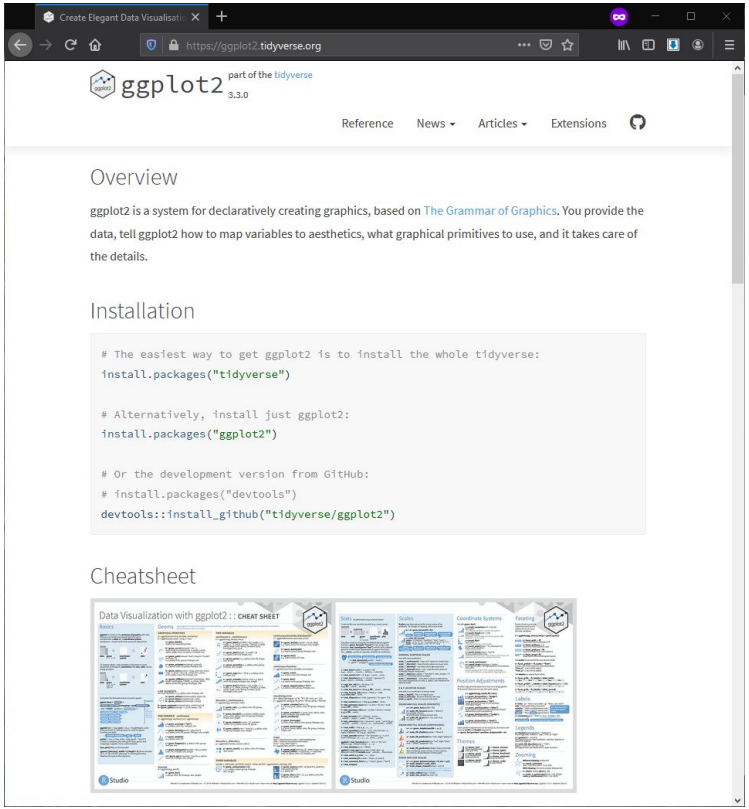
local the aes(), point and line are *different*

dplyr → ggplot

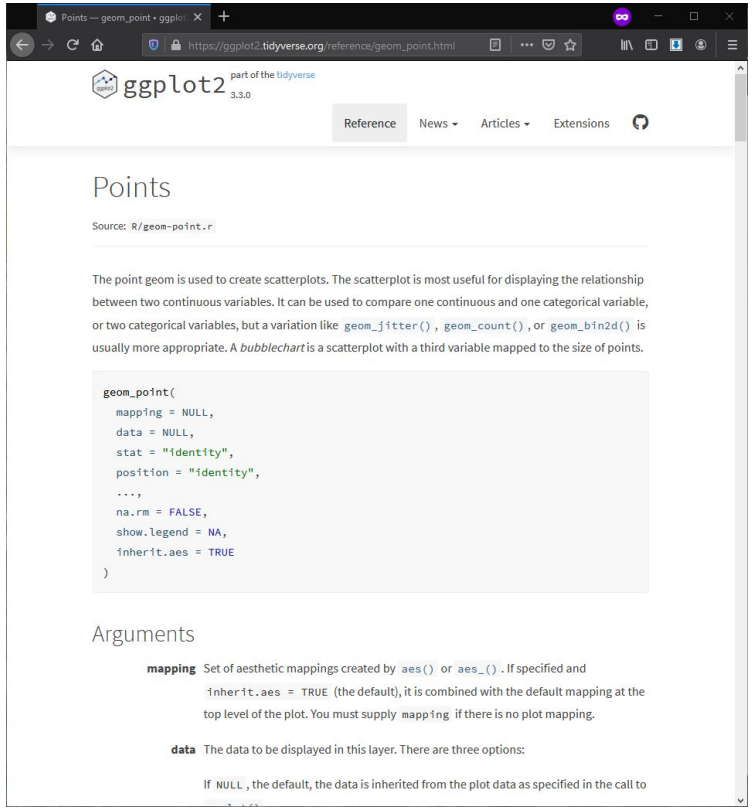
```
# dat is our original data object
dat <- "./data/raw/samples_1-8_heavy-light_calibration.csv" %>%
    read_csv()

# plot 1
dat %>%
    ggplot() +
    geom_point(aes(Replicate, Quant_Value),
               color="red") +
    geom_point(aes(Replicate + 1, Quant_Value),
               color="blue") +
```



ggplot on the web

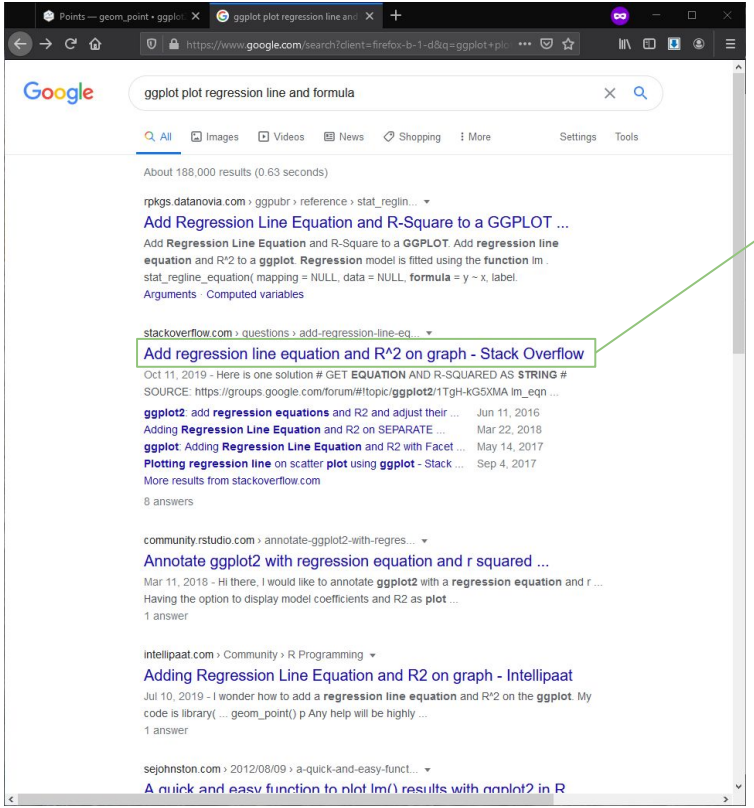


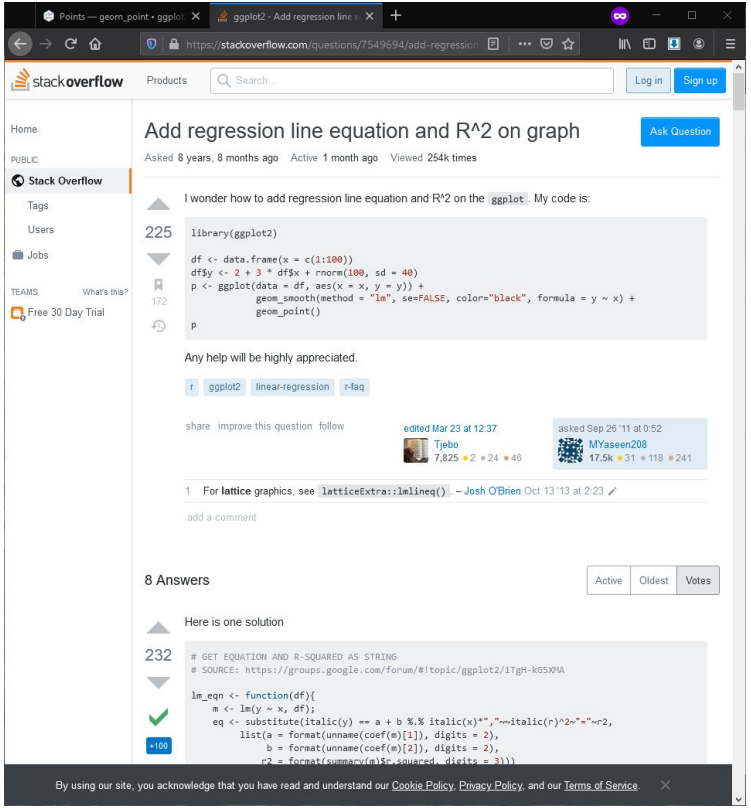
https://ggplot2.tidyverse.org/



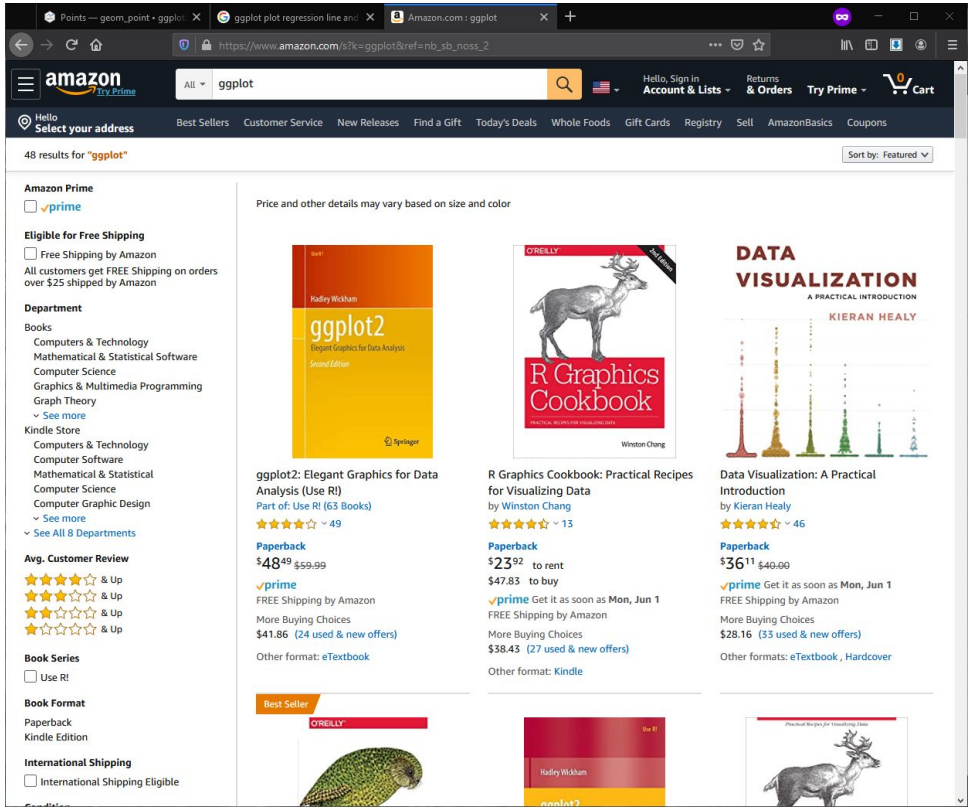
https://ggplot2.tidyverse.org/reference/geom_point.html

ggplot help on the web

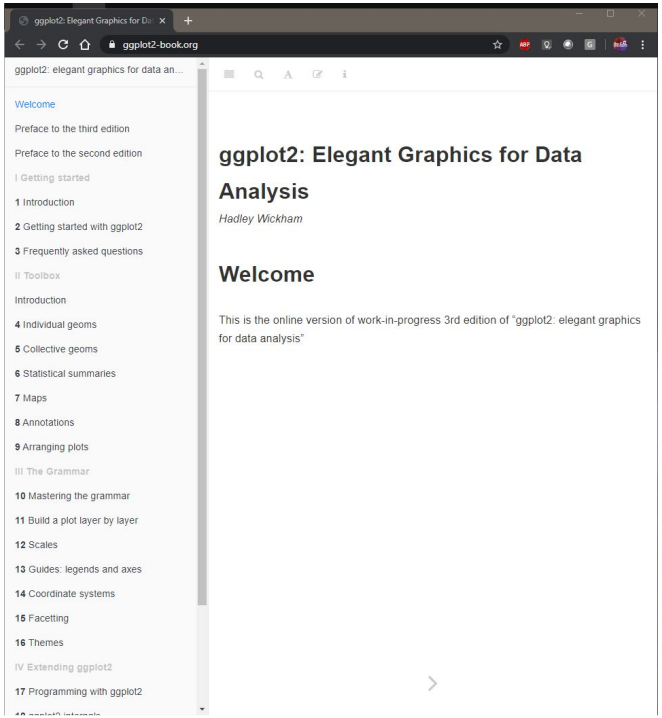




ggplot in print *(or also on the web)*

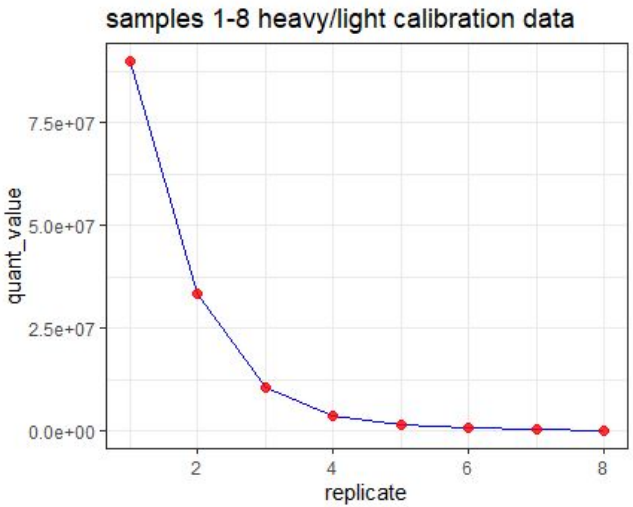


https://ggplot2-book.org/

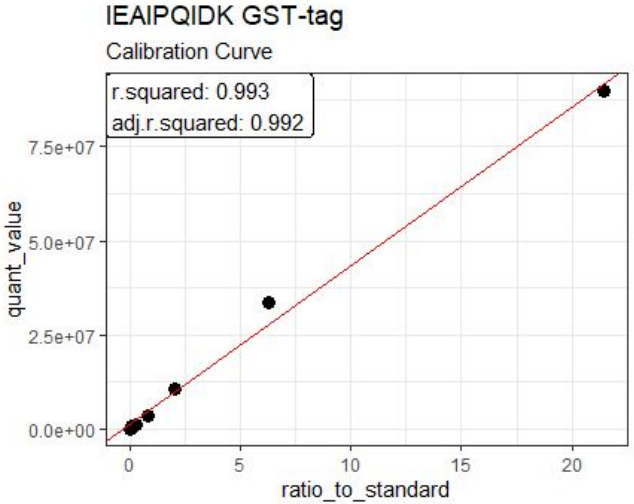


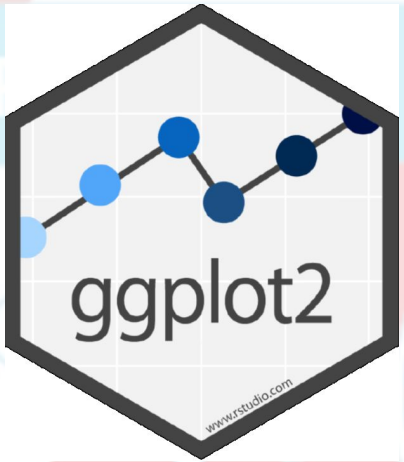
ggplot exercise 1 & 2

`points and lines`



`calibration plot`





R::ggplot

Visualization Extended Syntax

ASMS 2021: R Short Course

Goals for this module

- Advanced **ggplot2** syntax and customization
  - some dplyr tricks

- Demonstrated Application
  - creating multiple comparison plots (facetting)

- Practical Application
  - recreating a plot from another application (getting the right look)

ggplot extended syntax

```
1  p <- dat %>% ggplot(aes(x, y,  color)) + geom_point() +
2        facet_wrap(~m) +
3        scale_color_brewer(palette="Set1")
4        ggtitle(label, subtitle)
```

facet_wrap → create sub-plots based on a class value

scale_color_brewer → change the **color** of lines and points

scale_fill_brewer → change the colors of areas

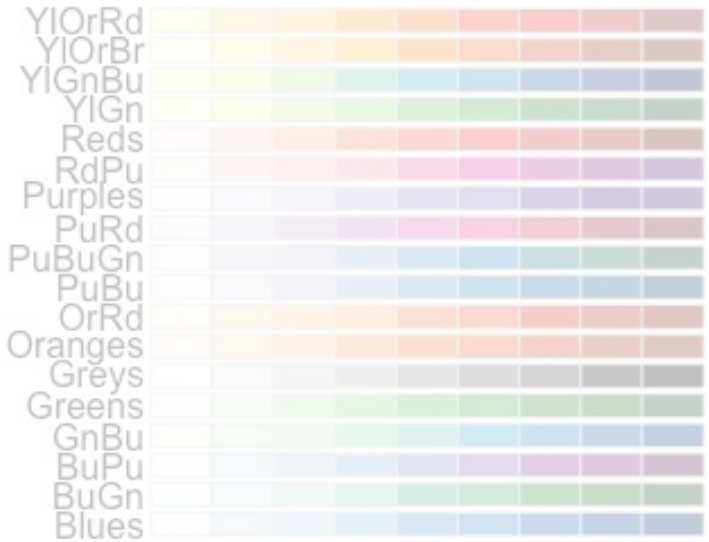
ggtitle → add a **main-title** and **subtitle**

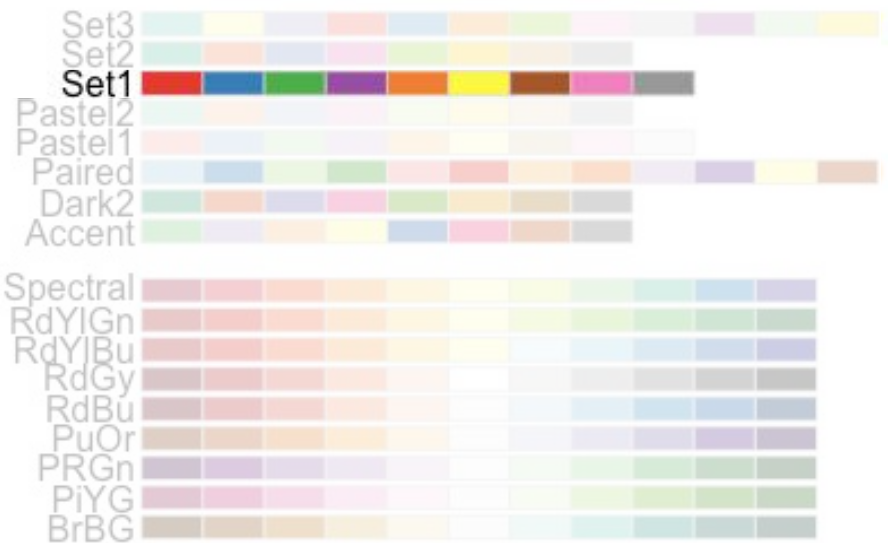
ggsave → save the plot object

```
5  ggsave("plot.pdf", plot = p, width = 5, height = 3)
```

ggplot color palettes *(pre-mixed)*

```
1       scale_color_brewer(palette="Set1")     # lines and points
1       scale_fill_brewer(palette="Set1")      # areas
```





https://ggplot2.tidyverse.org/reference/scale_brewer.html for a more indepth discussion

ggplot **facets**: data partitioning

a

facet_wrap(. ~ a)

wrap facets into a rectangular layout

b

a

facet_grid(b ~ a)

facet into both rows and columns

a

facet_grid(a ~ .)

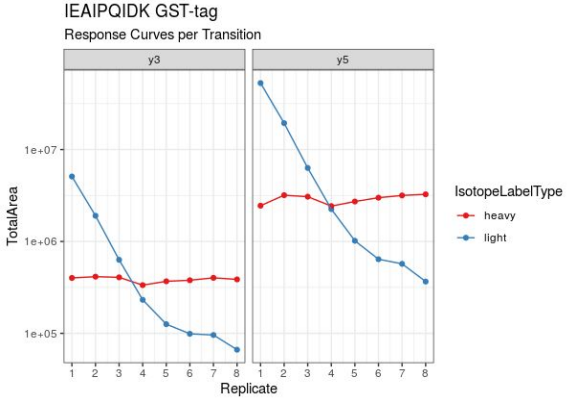
a

facet_grid(. ~ a)

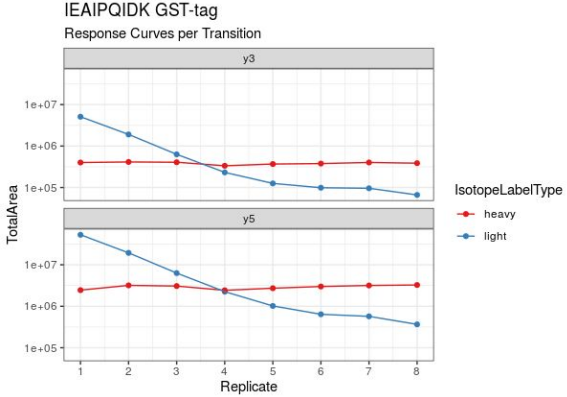
ggplot **facets**: data partitioning

facet_wrap(~a)
facet_grid(~a)

same scales

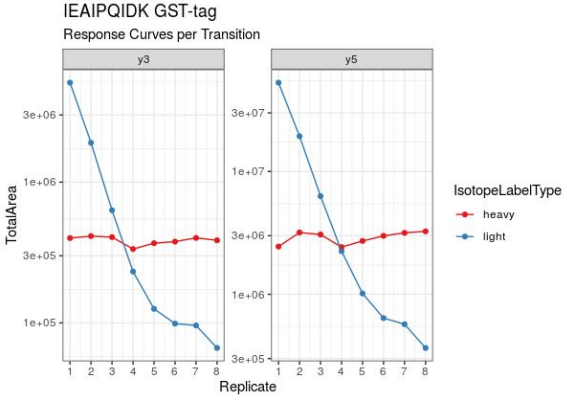


facet_wrap(~a,
ncol = 1)
facet_grid(a~.)



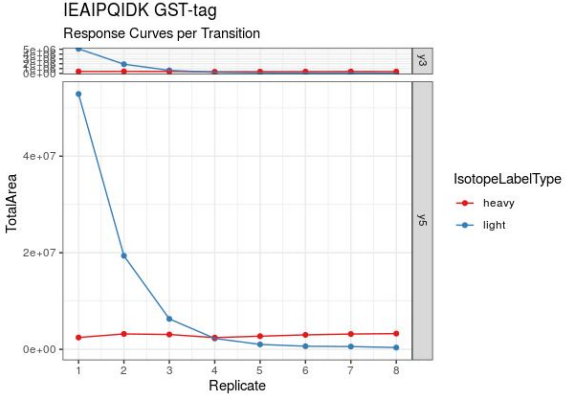
individual scales

facet_wrap(~a,
scales = "free")



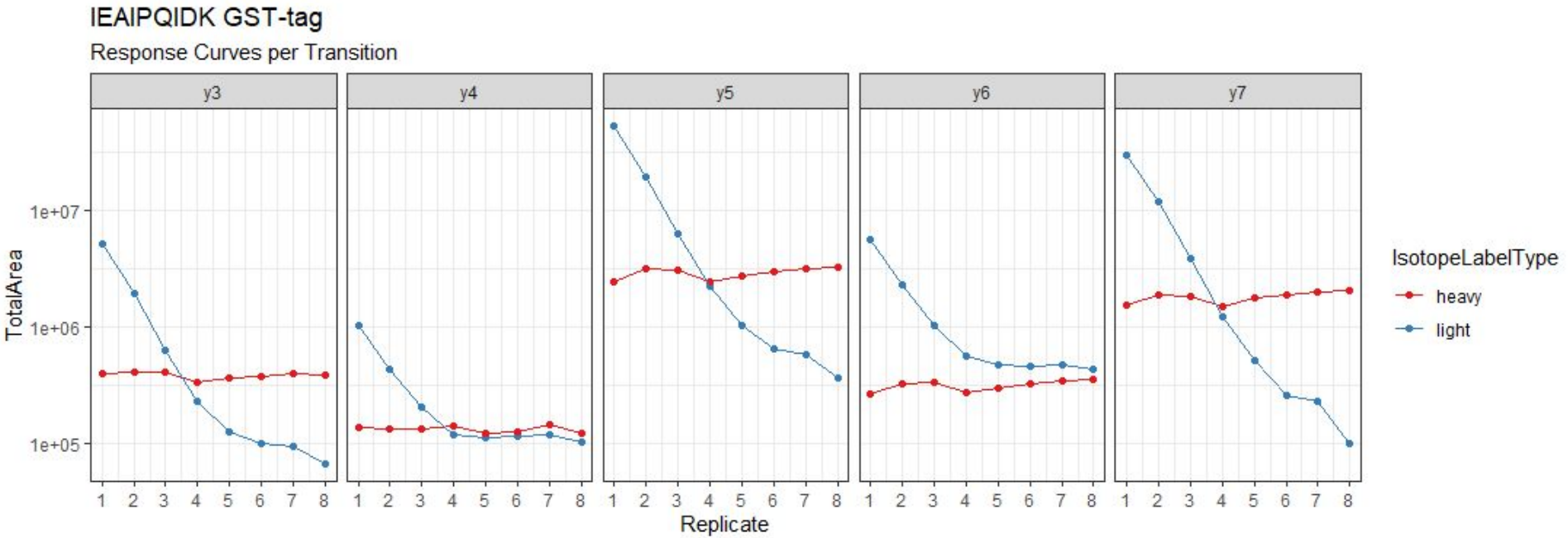
proportional scales

facet_grid(a~.,
space = "free")



ggplot exercise 3

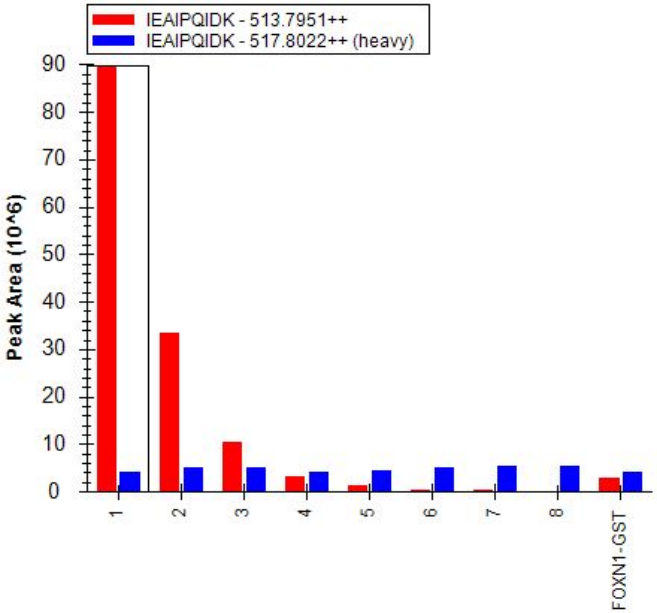
`faceting and coloring by variable`



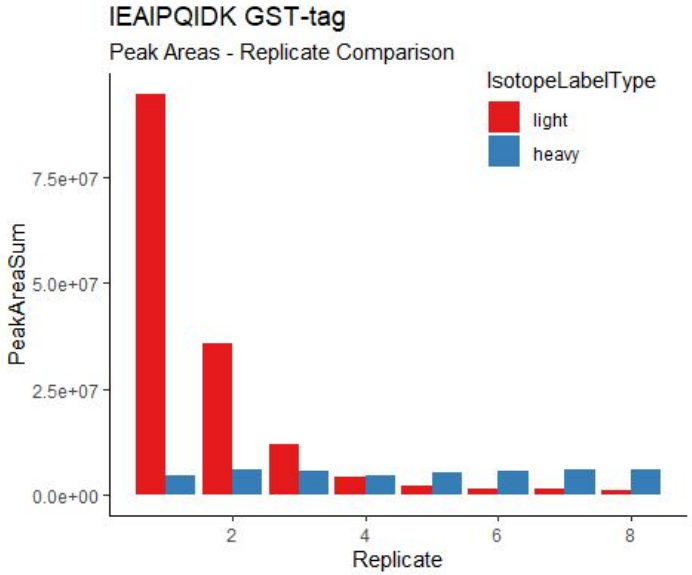
ggplot

replicating standard bar plots

Skyline



R::ggplot



tidyverse: reorder a variable by another variable

Factors are the data objects which are used to categorize the data and store it as levels … defaults are alphanumeric.

`forcats::fct_reorder()`

reorder by another variable such as rank

```
data %>%
    mutate( x = fct_reorder( x, y ))
```

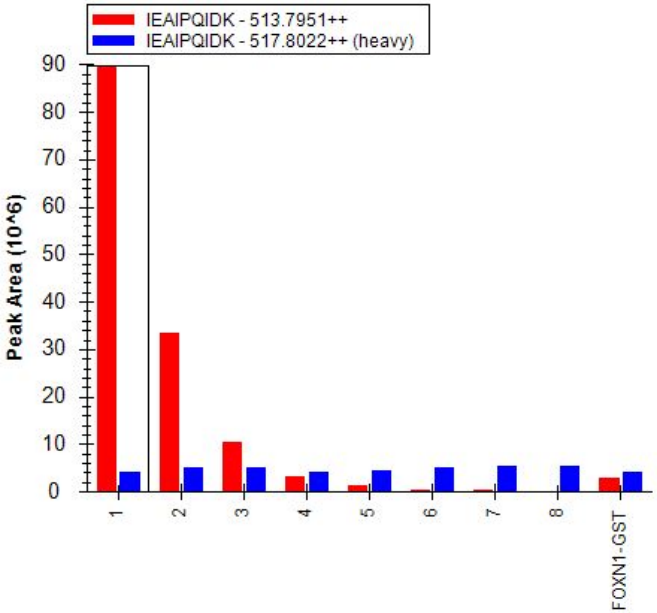
reverse order by alphanumeric sorting

```
data %>%
    mutate( x = fct_reorder( x, desc(x) ) )
```

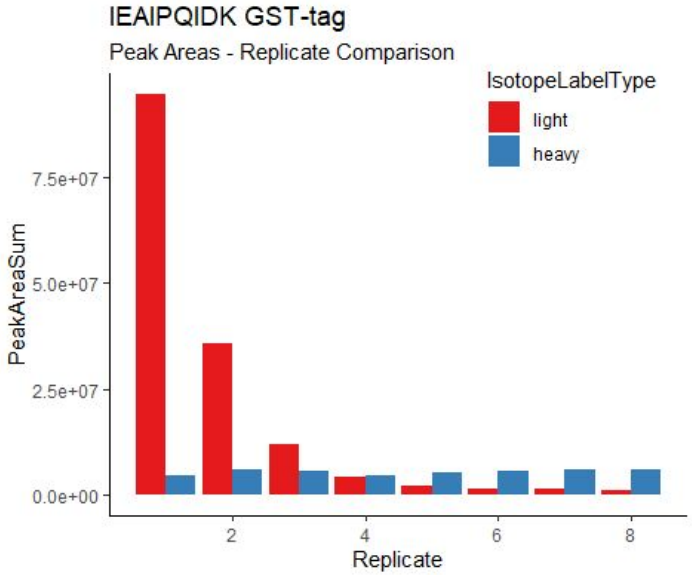
ggplot exercise 4

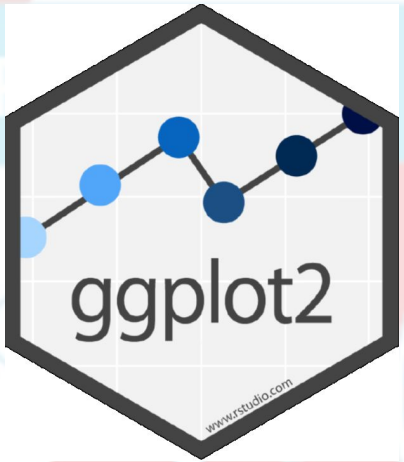
`replicating standard bar plots`

`Skyline`



`R::ggplot`





R::ggplot

Visualization Stats Plotting

ASMS 2021: R Short Course

Goals for this module

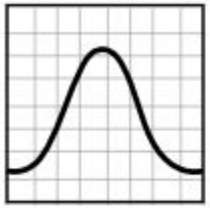
- Additional **ggplot2** plots & advanced challenges
  - looking at distributions

- Demonstrated Application
  - creating a comparative box-plot from multiple variables

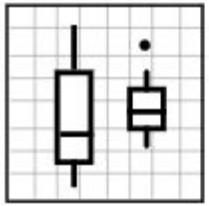
- Practical Application
  - exploring your data in a multitude of ways
  - Anscombe's quartet → the exemplar for why data visualization is important

ggplot **geoms**: distribution statistics

```
data %>% ggplot(aes(x)) + geom_histogram()
```

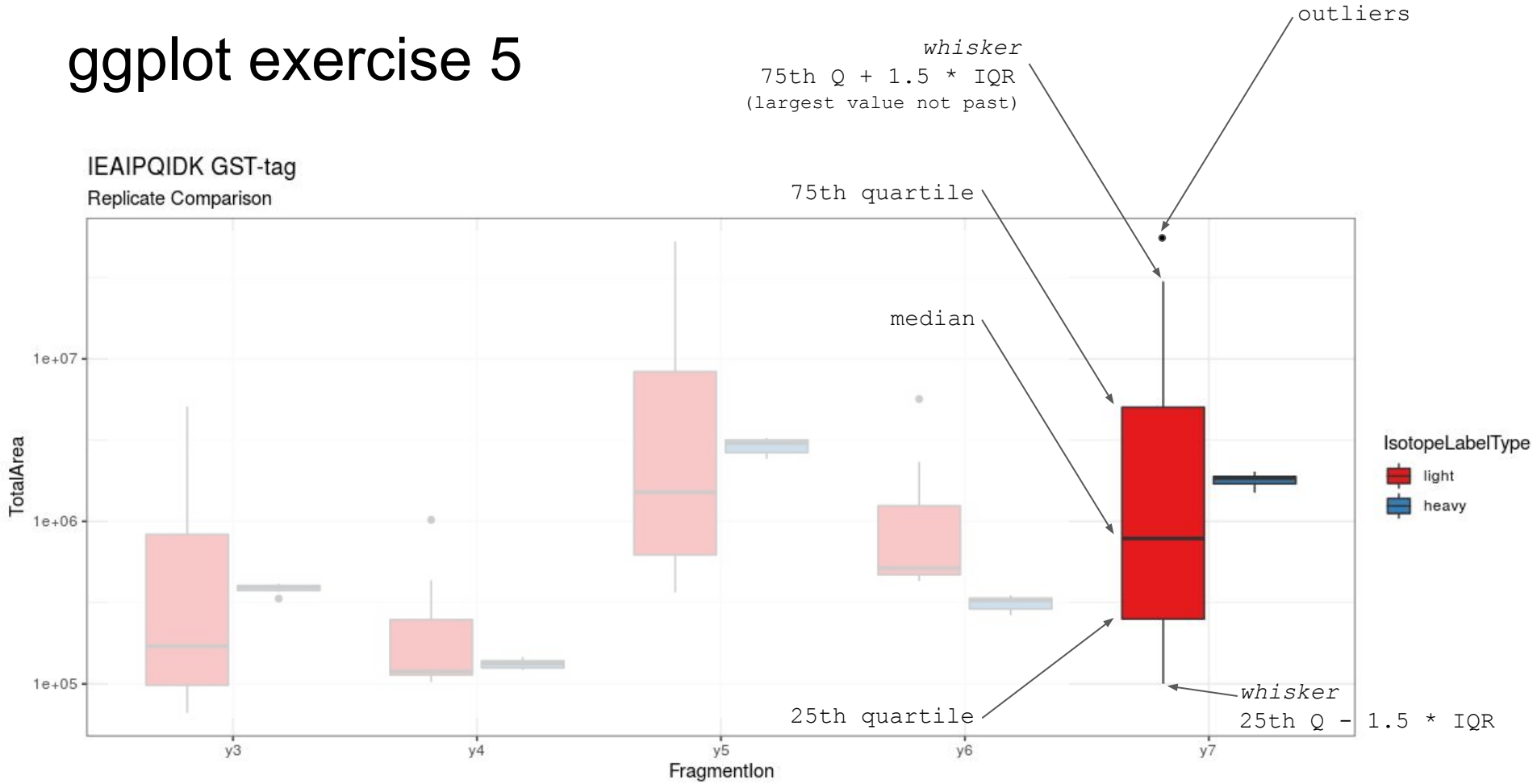


```
data %>% ggplot(aes(x)) + geom_denisty()
```



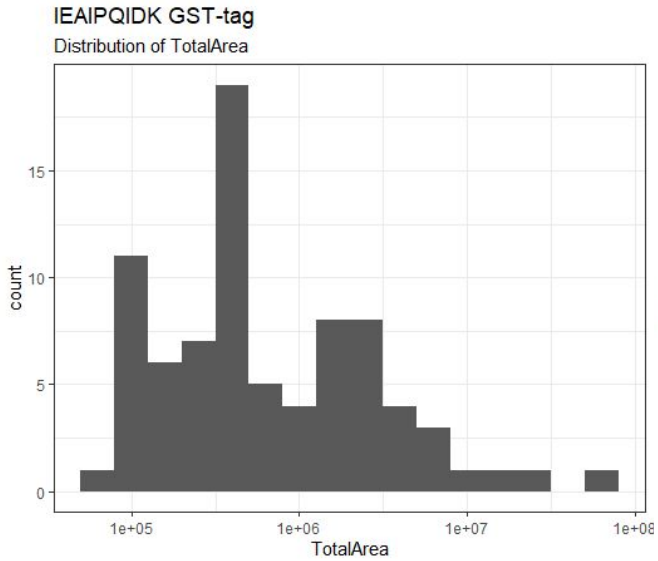
```
data %>% ggplot(aes(x,y)) + geom_boxplot()
```

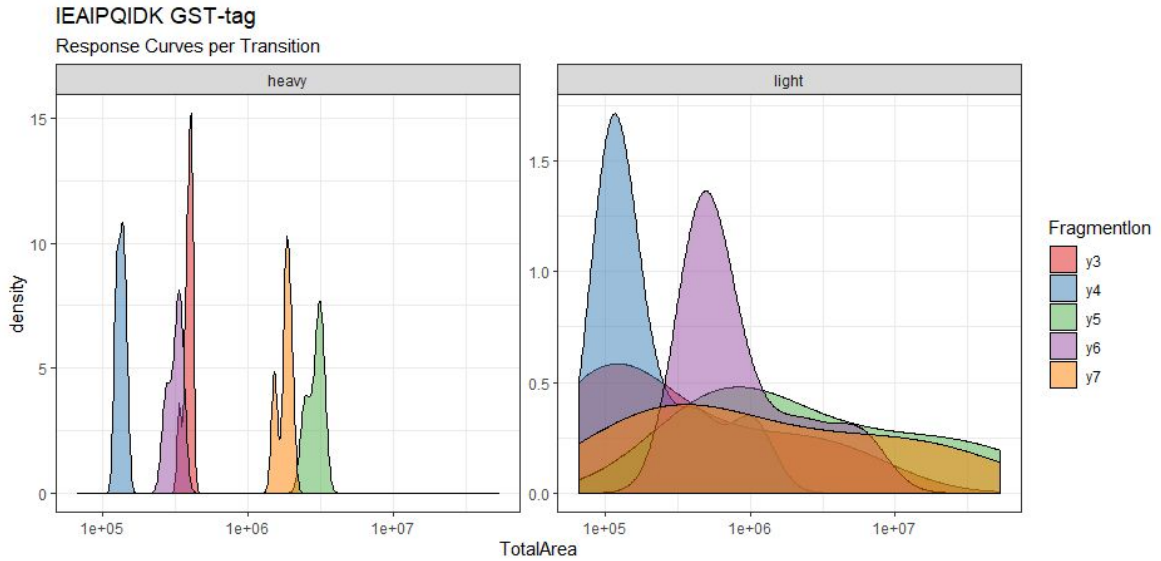
ggplot exercise 5



ggplot exercise 6 *challenge*

histograms and density plots

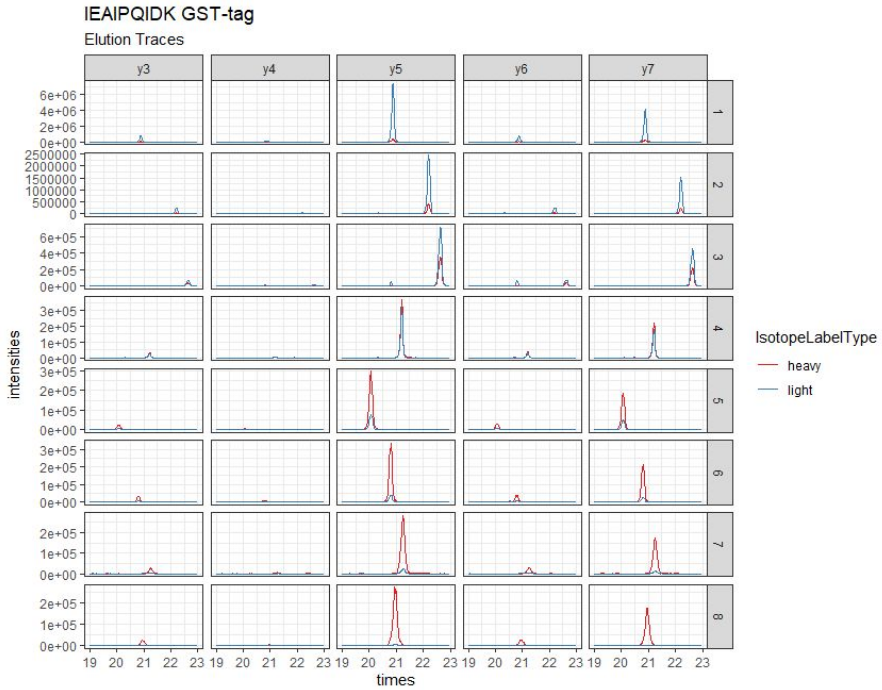


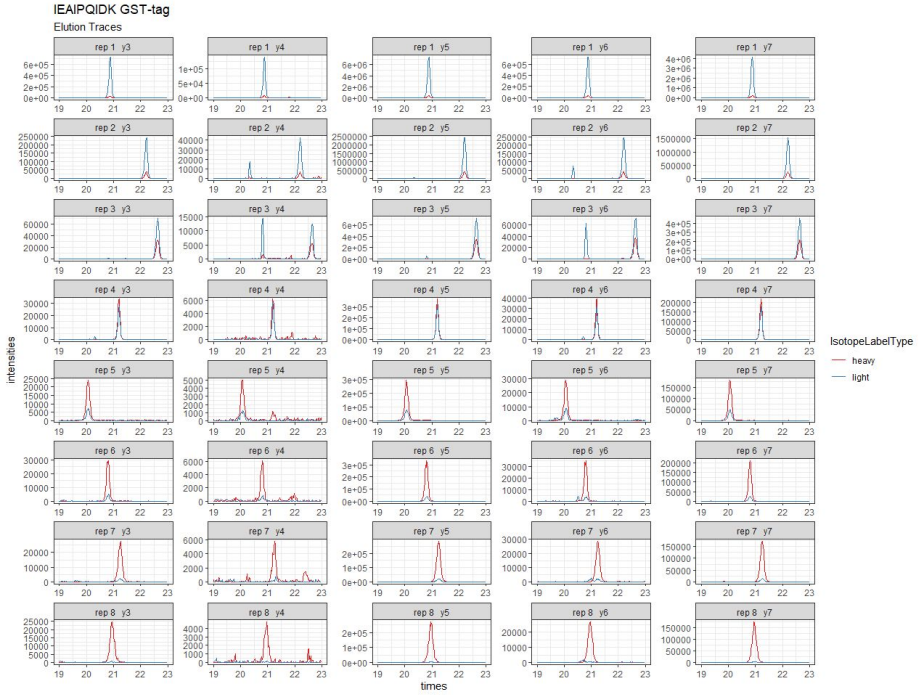


note: these are less informative visualizations, but decent demonstrations

ggplot exercise 7 *challenge*

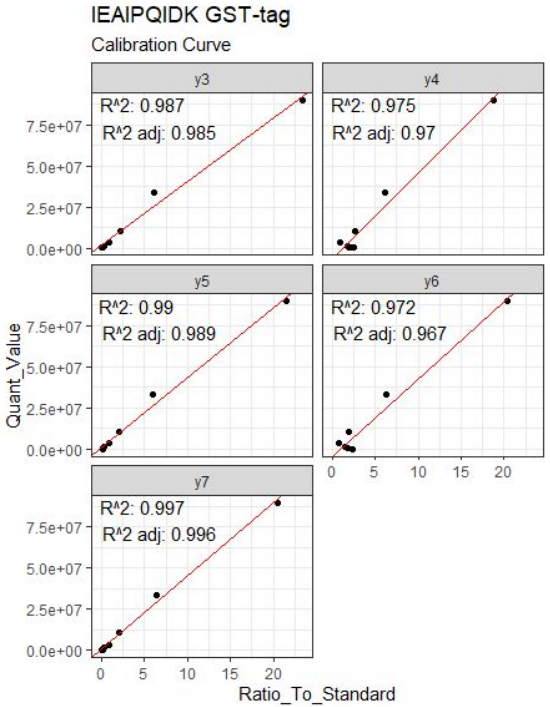
`plot elution profiles: facet_grid and facet_wrap`





ggplot exercise 8 *advanced challenge*

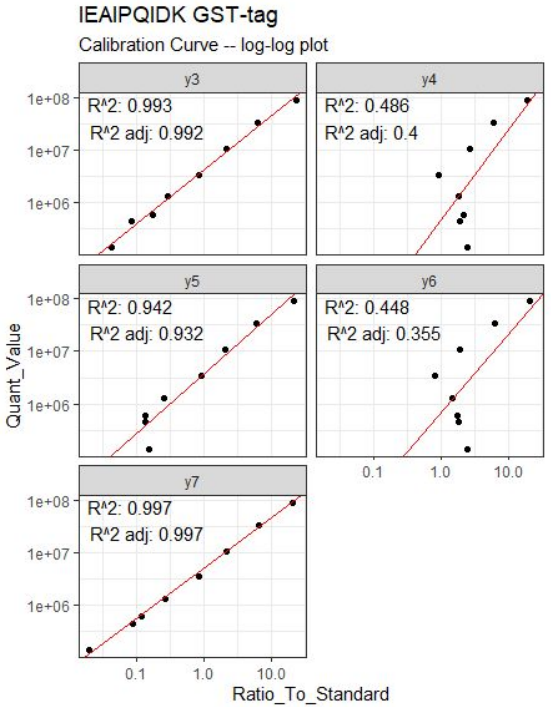
nested calibration model with faceted plot



Extra Credit

Why do the log-log plots have a different regression correlation estimate (R^2)?

Note: Refer to Anscombe's quartet set IV for similar regression issues.



Wrap-up & Next Steps

ASMS 2021: R Short Course

High-level Review

- The basics of R and how to start working with R using RStudio
  - The RStudio interface
  - Using RStudio projects to organize your work
  - Basic R syntax
- Some of the fundamental features and structure of the R language
  - Working with data using vectors and data frames
  - Useful functions in R for getting work done
- What tidy data is, why it’s important & ways to analyze tidy data
  - How the tidyverse R packages and tidy data help streamline the process of working with data
  - How to use dplyr to perform data analysis tasks
- Data visualization in R using ggplot2
  - Background information about visualization concepts with ggplot2
  - How to put together basic (and even more complex) plots with ggplot2

Are you now able to...

- start-up RStudio and make an RStudio project?
- read a .csv or Excel file into R?
- understand basic properties about the data (e.g. # rows, cols)?
- tell someone what tidy data is and why it’s important?
- perform some basic data manipulations and operations on the data?
- make a simple plot from the data?

Resources – Websites

- RStudio: https://rstudio.com/
- The tidyverse: https://www.tidyverse.org/
- ggplot2 documentation: https://ggplot2.tidyverse.org/reference/
- Shiny: https://shiny.rstudio.com/
- RMarkdown: https://rmarkdown.rstudio.com/
- Twitter hash-tags: #rstats, #DataScience

Resources – Books

- Hands on Programming with R: https://rstudio-education.github.io/hopr/
- R for Data Science: https://r4ds.had.co.nz/
- ggplot2: https://ggplot2-book.org/
- R Markdown: https://bookdown.org/yihui/rmarkdown/

People/Orgs to Follow

- RStudio
- Hadley Wickham (@hadleywickham)
- Jenny Bryan (@JennyBryan)
- Julia Silge (@juliasilge)
- Mara Averick (@dataandme)
- David Robinson (@drob)

Where to get help

- Google is a good place to start
  - copy error messages and search with quotes: “*error message here*”
  - try to use R specific/centric terms when possible in your search, e.g. data frame, tidyverse, package names, etc.
- Google will often link you to Stack Overflow https://stackoverflow.com/, you can often get good help here, but it can be a mixed bag
- RStudio Community https://community.rstudio.com/ is a great, friendly place to search for help and solutions, but not as extensive as Stack Overflow (yet?); worth it to just browser the topics
- Join a local R Users group (highly recommended!)

How to ask for help

- It’s very easy to ask for help online now, e.g. on Stack Overflow, but it can also be intimidating and sometimes responses might not always be positive (surprise: people can be mean on the internet 😱!)
- Before asking for help
  - Try to do background research on the problem; be sure to use Google first
  - Make it as easy as possible for people to help you, try to state your problem concisely
  - Read about making reproducible examples: https://www.tidyverse.org/help/
  - Read a site’s rules about posting and asking for help
- If you post a question, be sure to follow-up with anyone who helps; did the suggestions work? Did they help solve the problem? Are you still stuck?

Proposed plan for next steps

- Try to practice with R a little bit every day, much better than longer sessions spread farther apart
- Take simple R code examples that you understand, and try to re-type them yourself (without looking/copy/pasting) – helps with understanding syntax and debugging syntax errors
- Give yourself small data tasks that apply to your daily work so you can practice things that are also useful to what you do
- Take small steps and slowly build on what you know to expand your expertise

Homework

1. Choose a simple data task you've already done (maybe in Excel?),
   e.g. a simple plot, a table of summary stats, computing a value from a data set

2. Try to reproduce the task in R using only R code; you can check your results to make sure they match (doesn't have to be *exactly* the same)

3. Think about ways you might change your data analysis workflow for future projects

We’d love to get your feedback!

What worked for you, what didn’t?

What parts were difficult to understand

What else would you like to learn in a intro class like this?

Slower pace? Faster pace?

More examples and practice sessions?

Thank you very much!