R Fundamentals and Best Practices for Mass Spectrometry Data Analysis

Saturday, November 7 (12:00-3:15pm Eastern)

Ryan Benz, Seer, Inc

Introduction to R & RStudio &
Working with Data Fundamentals

Workshop Overview

- **Day 1 (Sat 11/7):** Introduction to R & Working with Data Fundamentals
  Ryan Benz, Seer Inc.
- **Day 1 – Intermediate Track (Sat 11/7):** Intermediate R
  Mateusz Staniak, University of Wrocław

- **Day 2 (Sat 11/8):** Data Visualization with ggplot2
  Kylie Bemis, Northeastern University

- **Day 3 (Sat 11/14):** Basic Statistics with R & Reproducible Data Analysis
  Meena Choi, Genentech

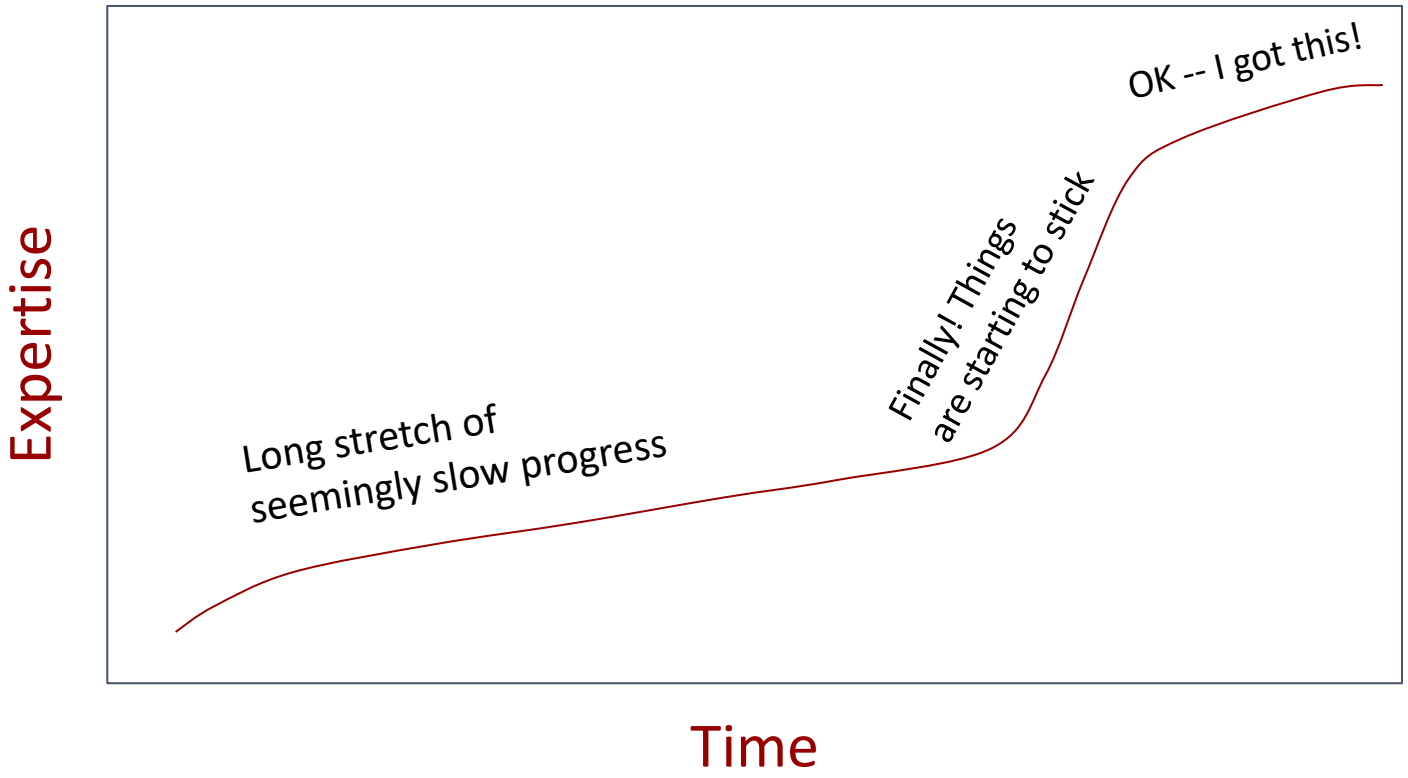
- **Day 4 (Sun 11/15):** Analysis of Proteomics Data with MSstats, MS Imaging with Cardinal
  Meena Choi, Genentech
  Olga Vitek, Northeastern University
  Melanie Föll, University of Freiburg

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

The R Learning Curve

10 years ago...

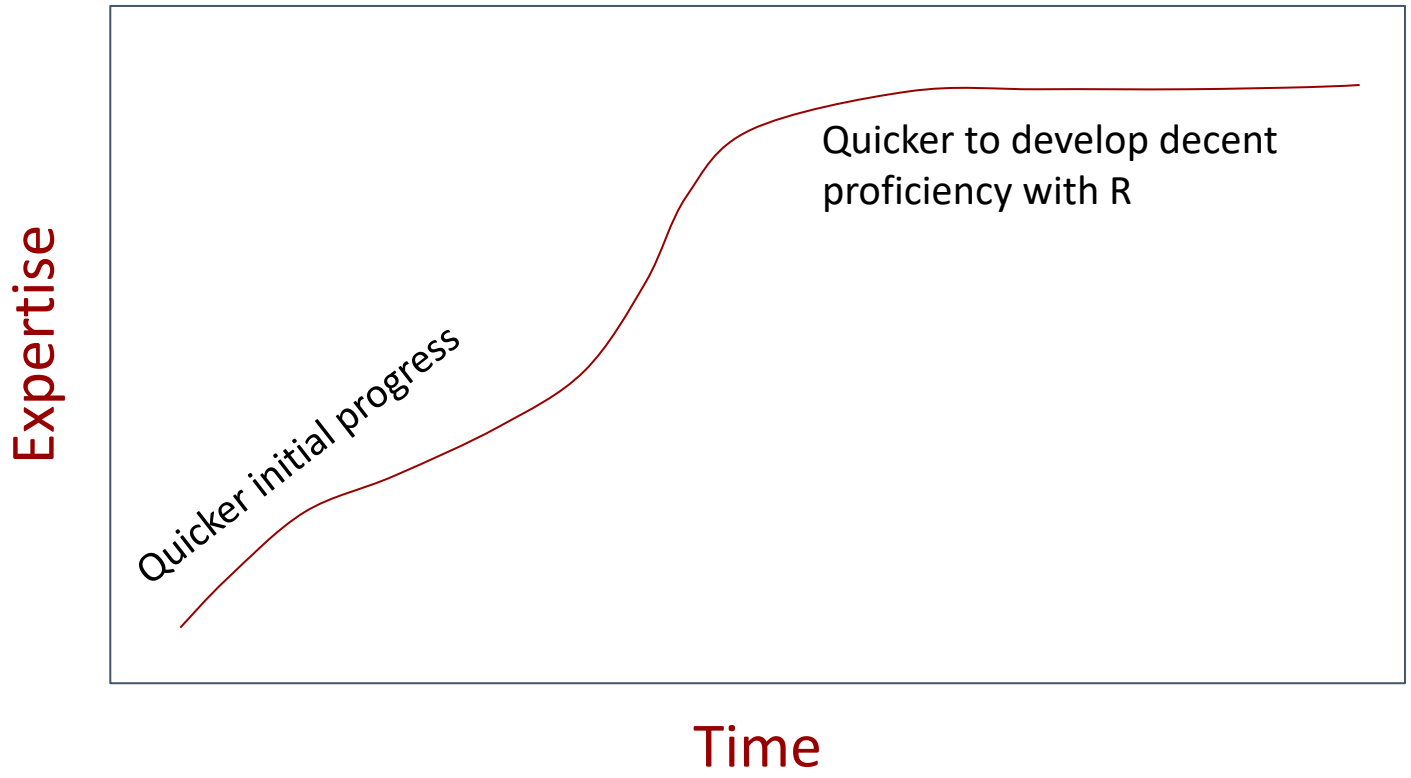


What made R challenging to learn 10 year ago?

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

The R Learning Curve

Today...



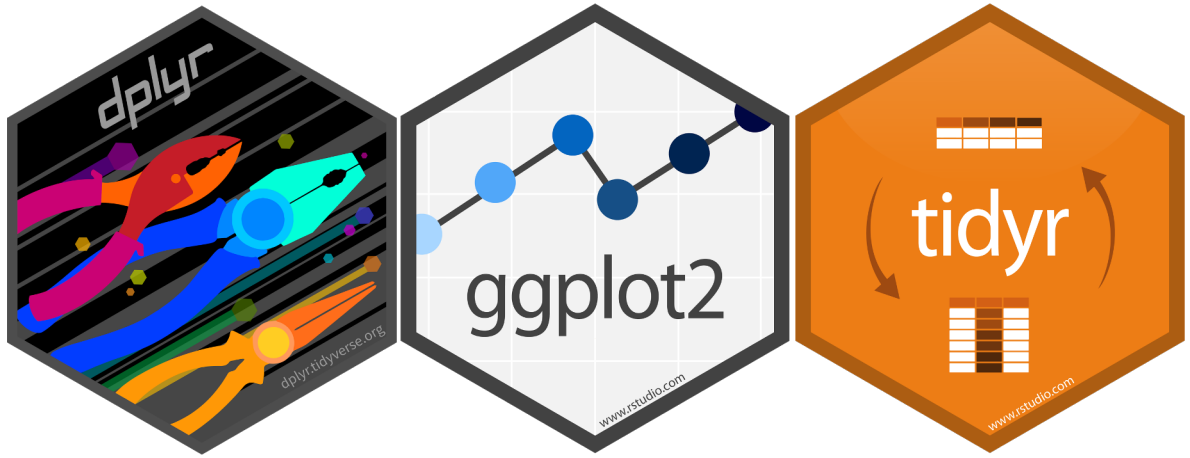
Why is R easier to learn today?

- 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!

The R ecosystem is vast and awesome!



Shiny

Interactive web apps for data

RMarkdown, Blogdown, Bookdown

Reproducible reporting, blogs & books

Day 1 Schedule

Time	Duration	Session Type	Session
12:00 - 12:10 pm	10 min	Lecture	Workshop Introduction
12:10 - 12:25 pm	15 min	Lecture	Intro to RStudio Cloud & RStudio
12:25 - 12:40 pm	15 min	Exercises – Main Room	Working with RStudio Projects
12:40 - 12:55 pm	15 min	Lecture	First Steps with R Code
12:55 - 1:10 pm	15min	Exercises – With TA's	Practice with variables, vectors & conditional expressions
1:10 - 1:20 pm	10 min	Lecture	Reading data files into R
1:20 - 1:30 pm	10 min	BREAK	
1:30 - 1:45 pm	15 min	Lecture	Working with Data Frames
1:45 - 2:00 pm	15 min	Exercises – With TA's	Practice reading data into R, basic data frame operations
2:00 - 2:10 pm	10 min	Lecture	Introduction to Tidy Data & the tidyverse
2:10 - 2:30 pm	20 min	Lecture	Basic Data Manipulation with dplyr
2:30 - 2:45 pm	15 min	Exercises – With TA's	Working with data frames & dplyr
2:45 - 3:15 pm	30 min	Lecture	Day 1 Recap & Questions

Logistics

- Lectures will be in the main Zoom session and broadcast on YouTube
- TA's will be waiting in the break-out Zoom sessions
- You'll practice the exercises in the break-out sessions, ask questions and get help from the TA's (and other students)
- We'll try to migrate to and from the main session with the schedule, but you can stay in the break-out sessions too if you need more help (and monitor lecture on YouTube)
- Course materials are on our workshop GitHub site https://github.com/ZenBrayn/asms_2020_fall_workshop

Working with RStudio & RStudio Projects

ASMS 2020 Fall Workshop

Day 1 – Module 1

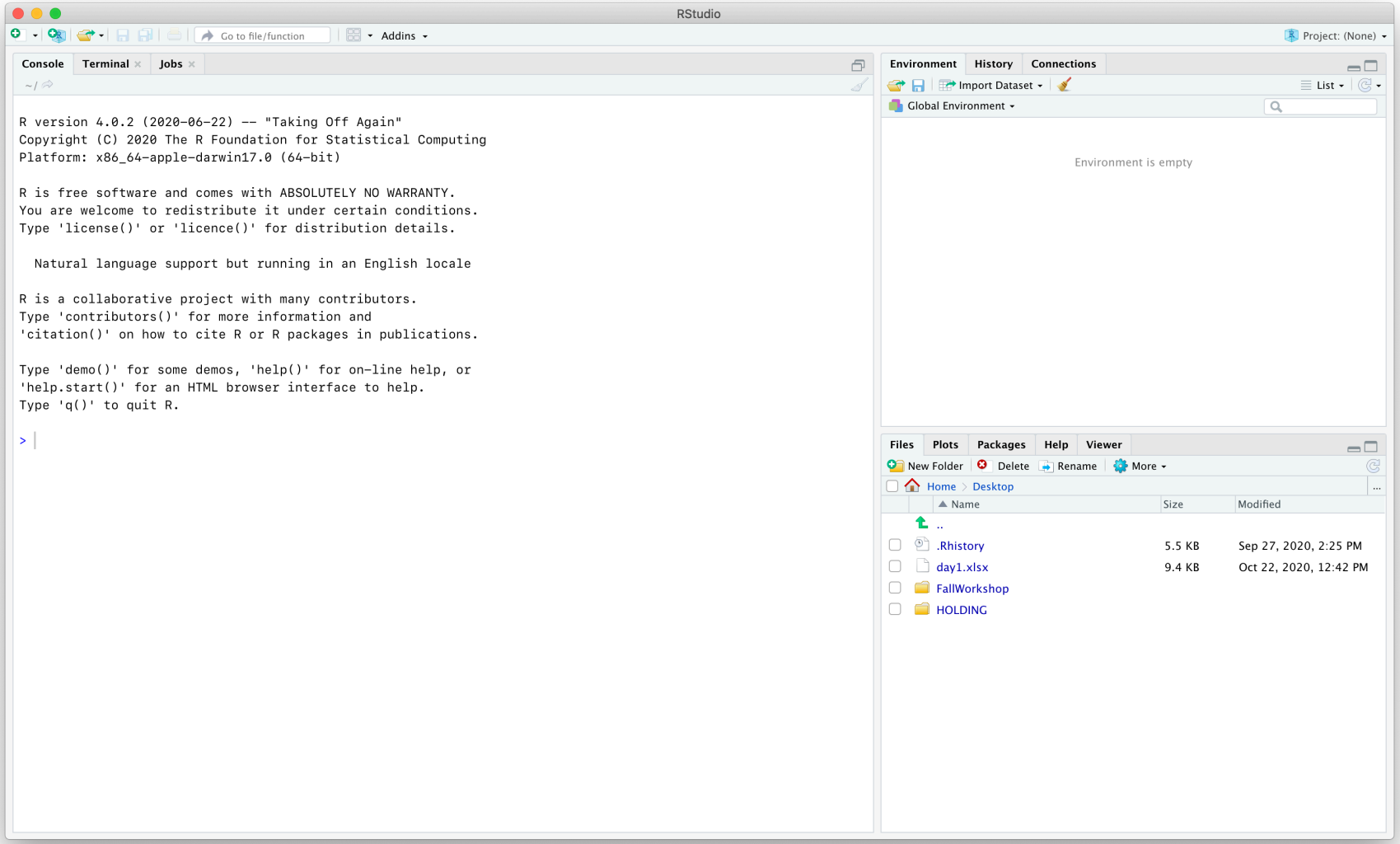
R vs. RStudio vs. RStudio Cloud

- R is a coding language and an underlying system that allows you to execute R code
- R is free (open source)
- You install R first
- R is like a car's engine/transmission

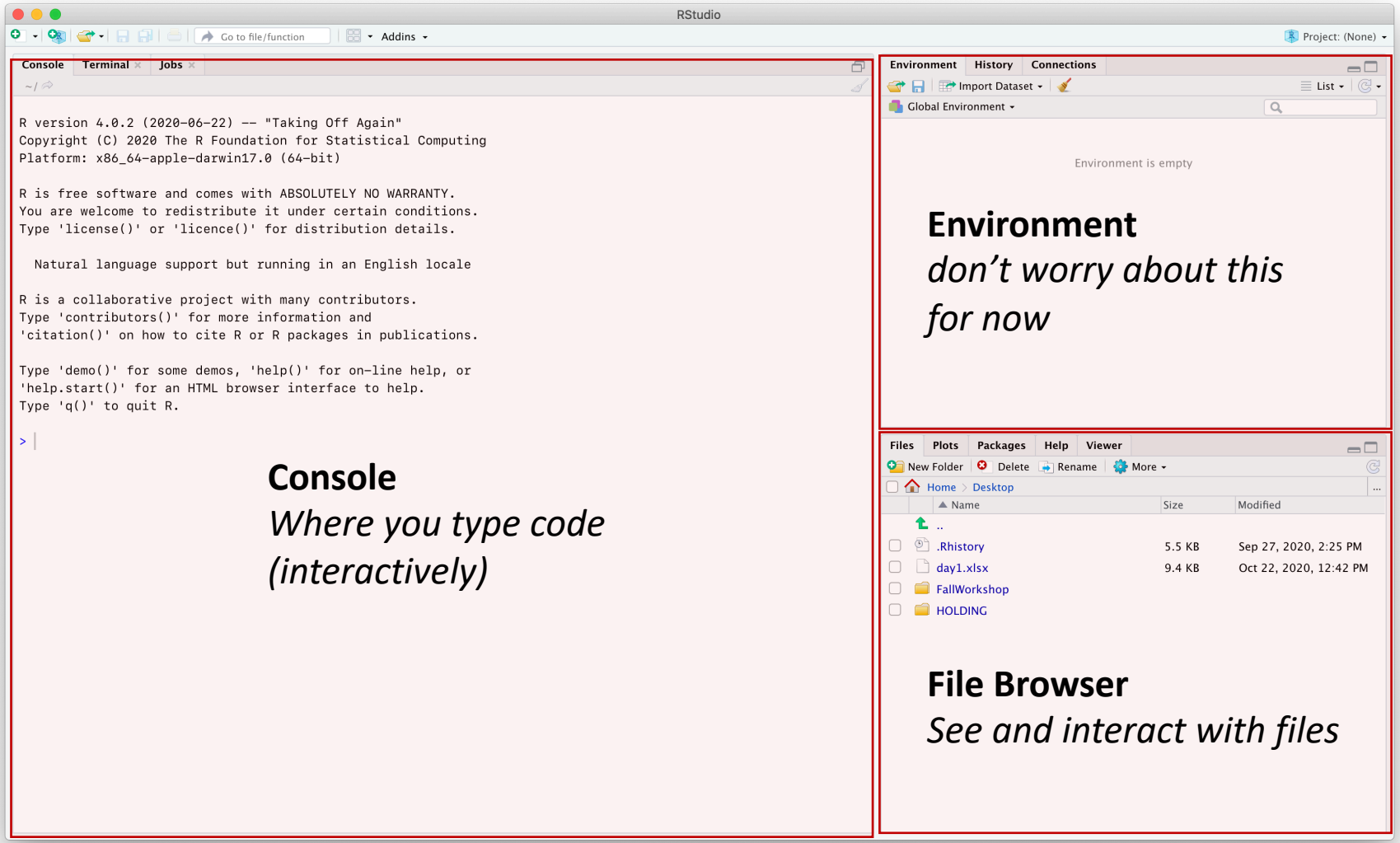
- RStudio is an integrated development environment (IDE) that makes it easier to work with R and R code
- Has free and paid versions
- You install RStudio after you install R – it won't on its own work
- RStudio is like a car's body/interior

- RStudio Cloud is a web-based version of R & RStudio
- Has free and paid versions
- You only need a web browser – don't need to install anything else
- RStudio Cloud is like a complete car

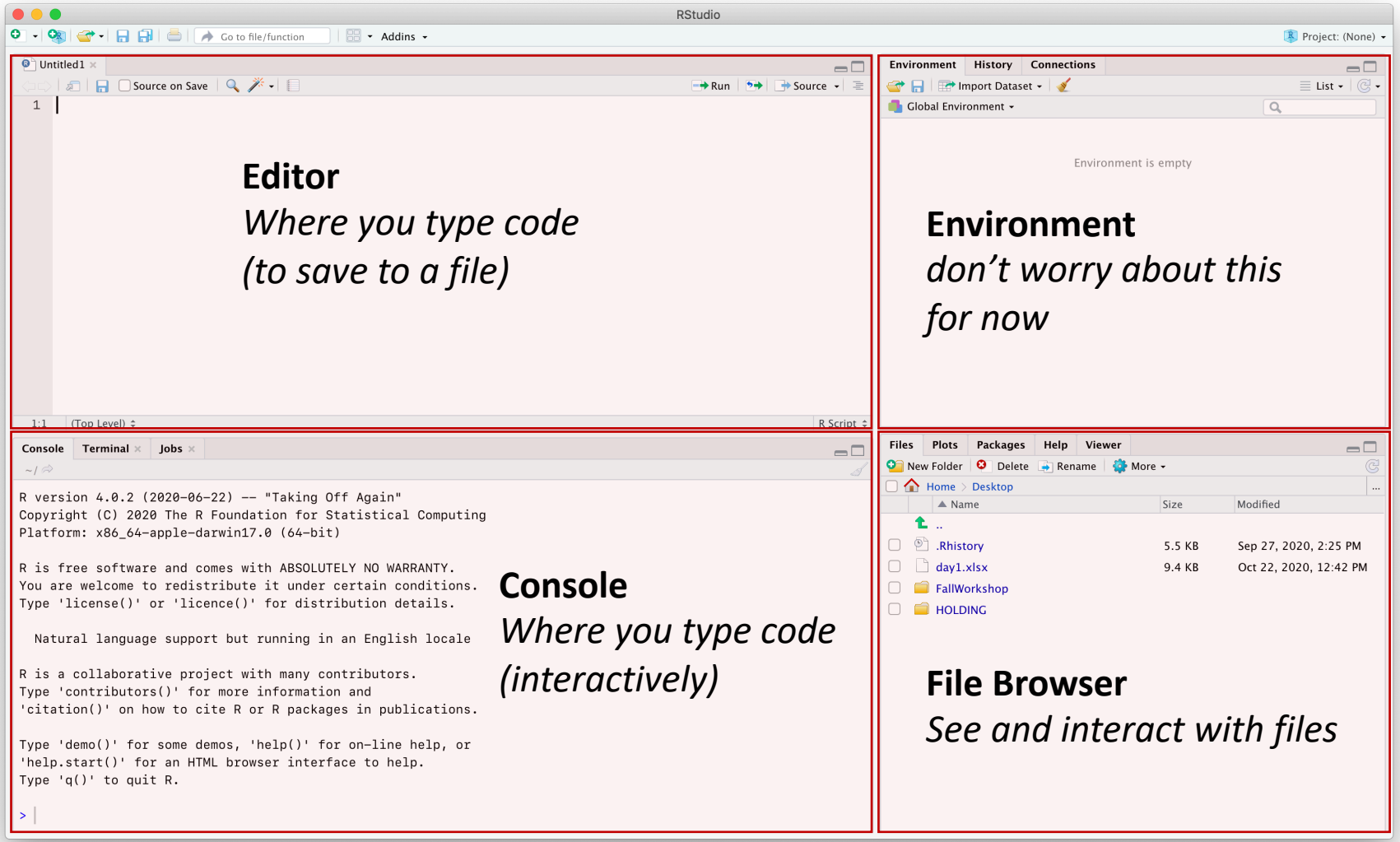
RStudio When you First Open It



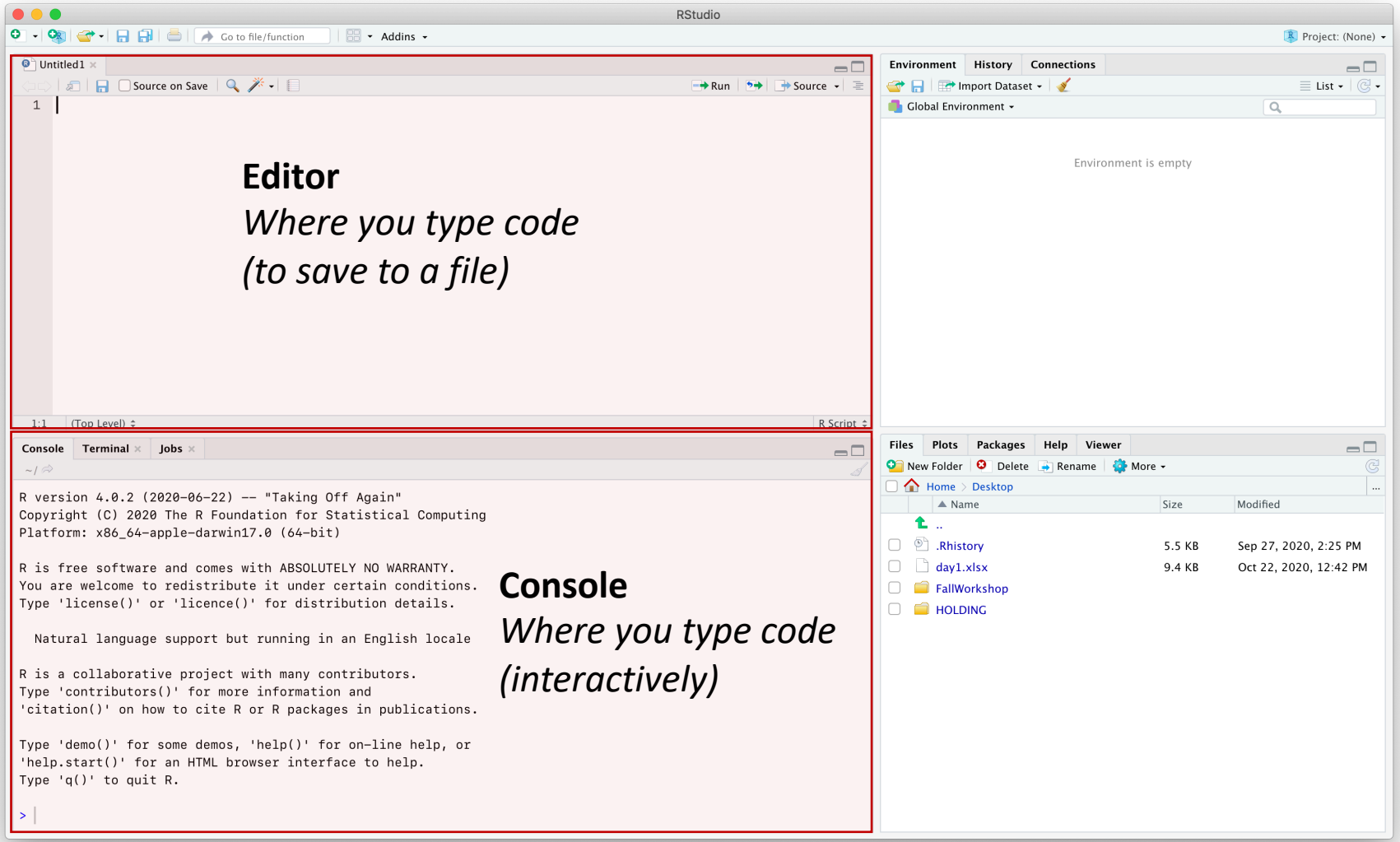
RStudio When you First Open It



File → New File → R Script



First Focus on The Editor and Console



The Console vs. The Editor

The Console

- Where you experiment, test and practice
- Good for quick, one-off coding
- The console is like a digital cocktail napkin or sticky note
- Work you do in the console is transitory, can be thrown away at anytime

The Editor

- Where you write code that you want to keep
- It’s good practice to assume you always want to keep your code
- Code in the editor can be saved to a file (R script)
- In general, you should spend most of your time in the Editor

RStudio Projects Make Working with R Much Easier

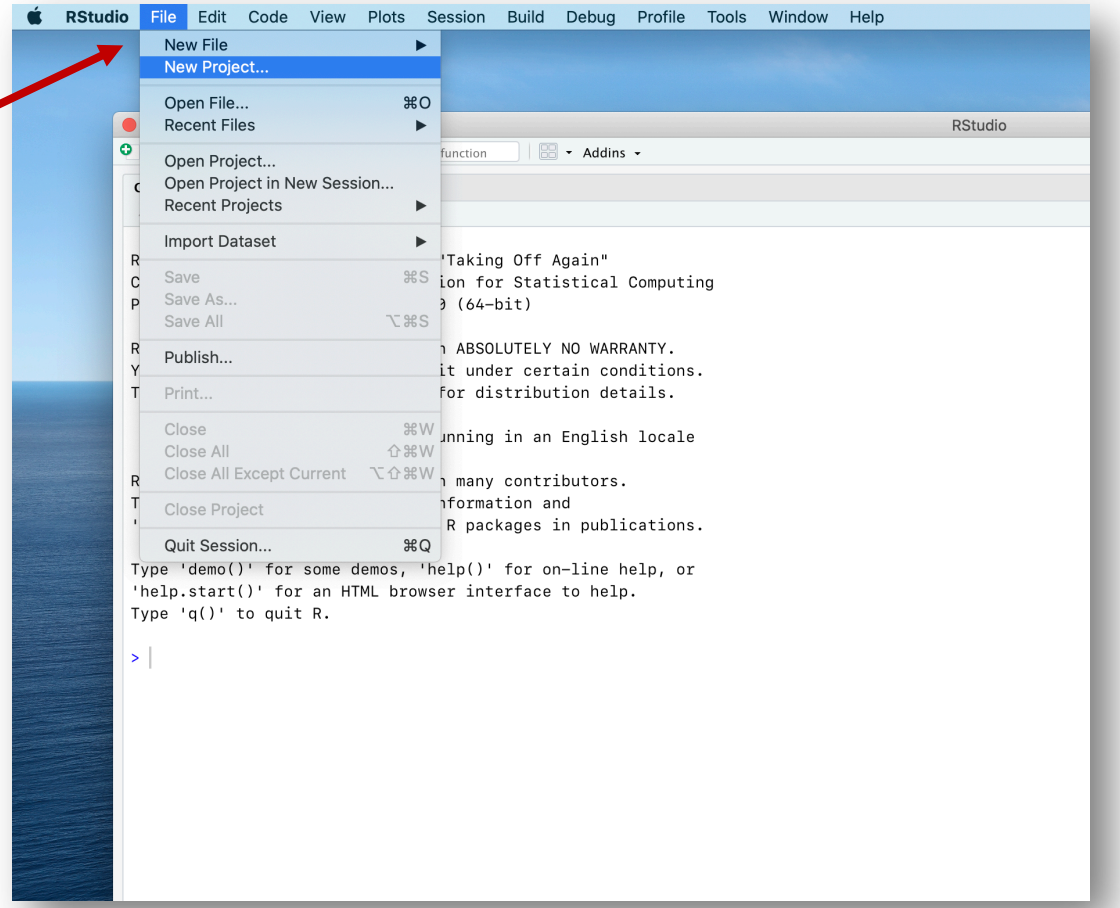
- RStudio project are special directories that organize your work and make it easier to work with R

- Dealing with file paths can be confusing/frustrating/cumbersome; RStudio projects helps remove some of this friction by automatically setting your working directory to the project directory

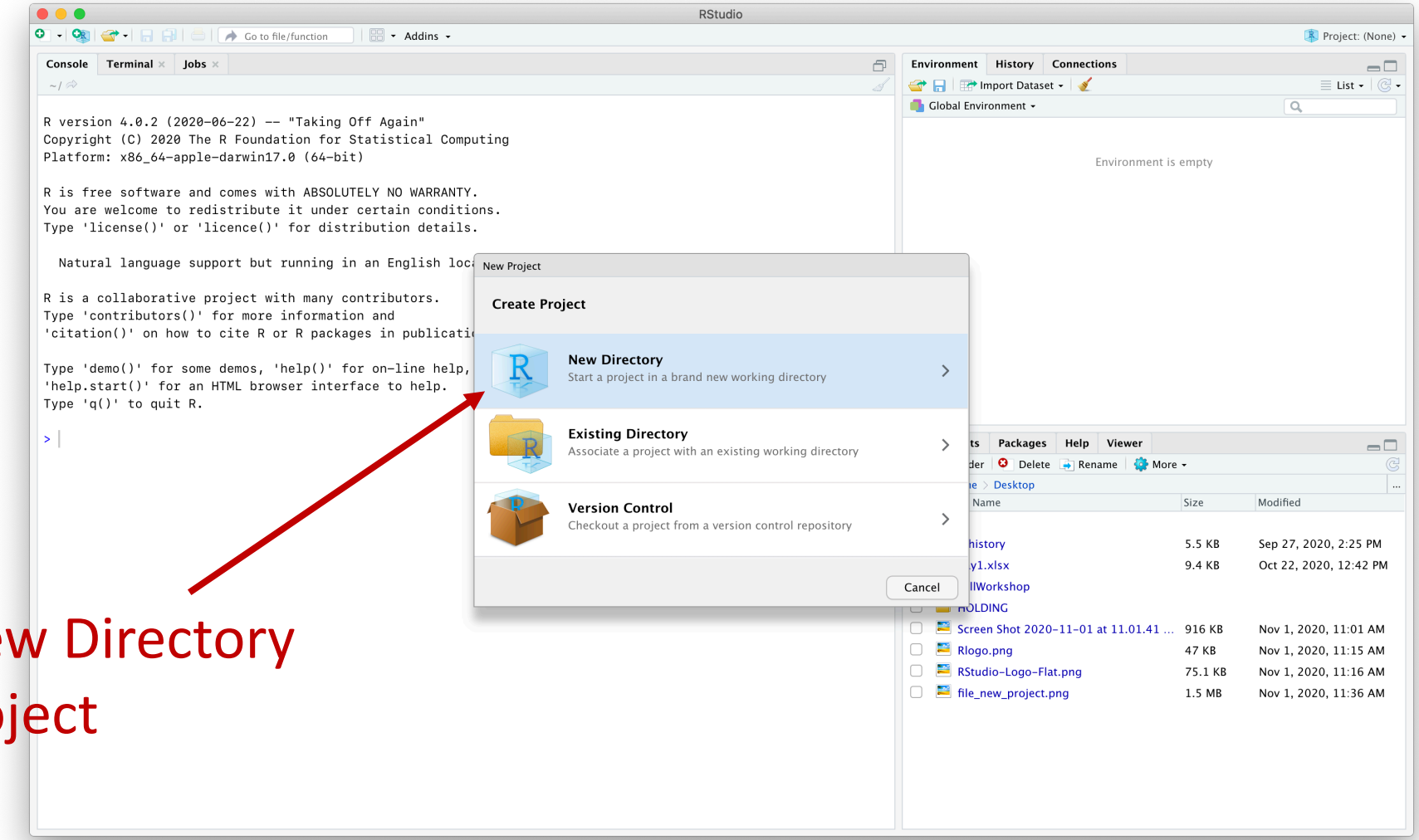
- Main idea
  - Place all of your project files inside the RStudio project directory
  - You can access your files (i.e. data) directly by name (don’t need to specify a full file path)
  - If you need to move the project directory somewhere else, everything should continue to work without modification

Creating an RStudio Project

Start the process from the File menu

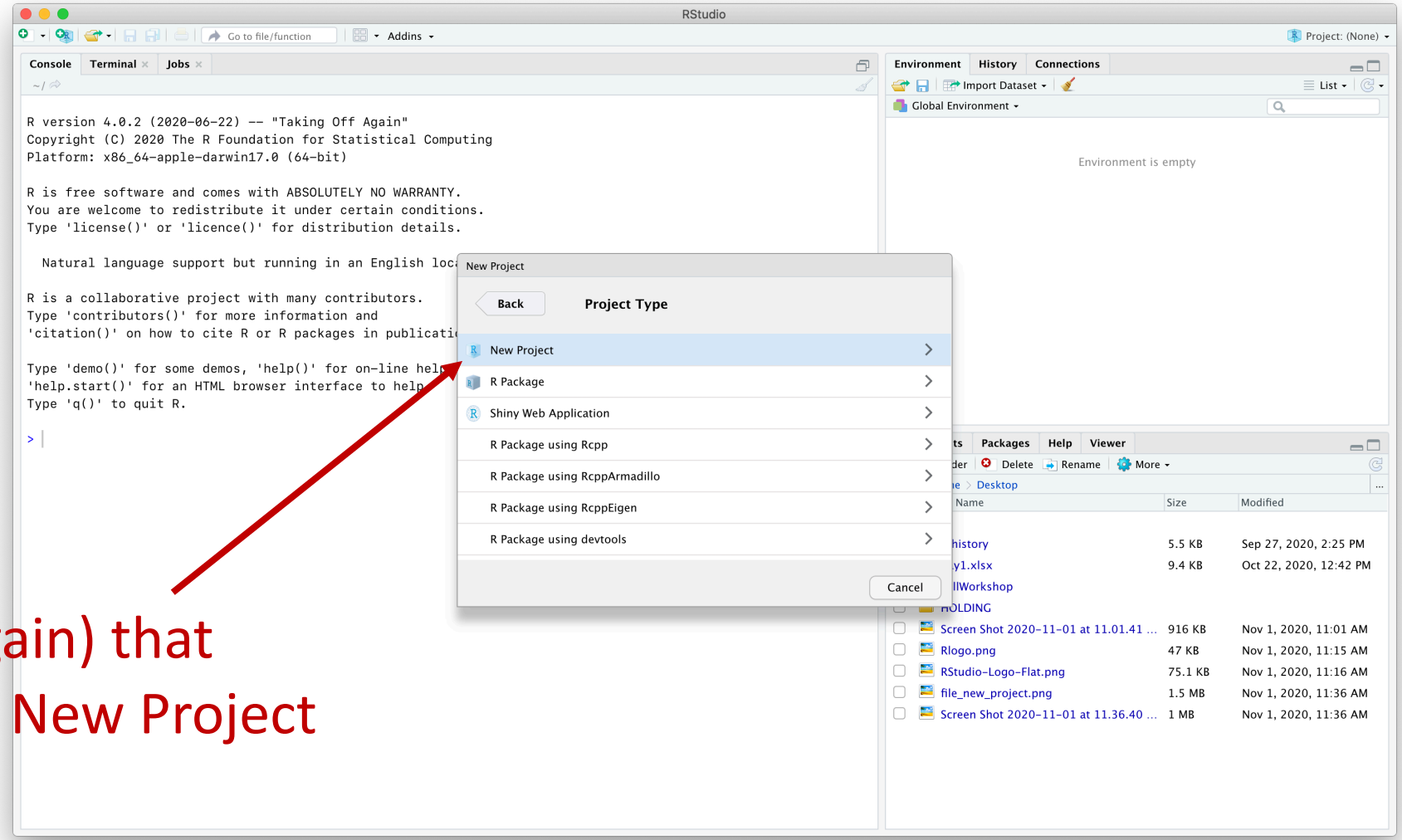


Creating an RStudio Project



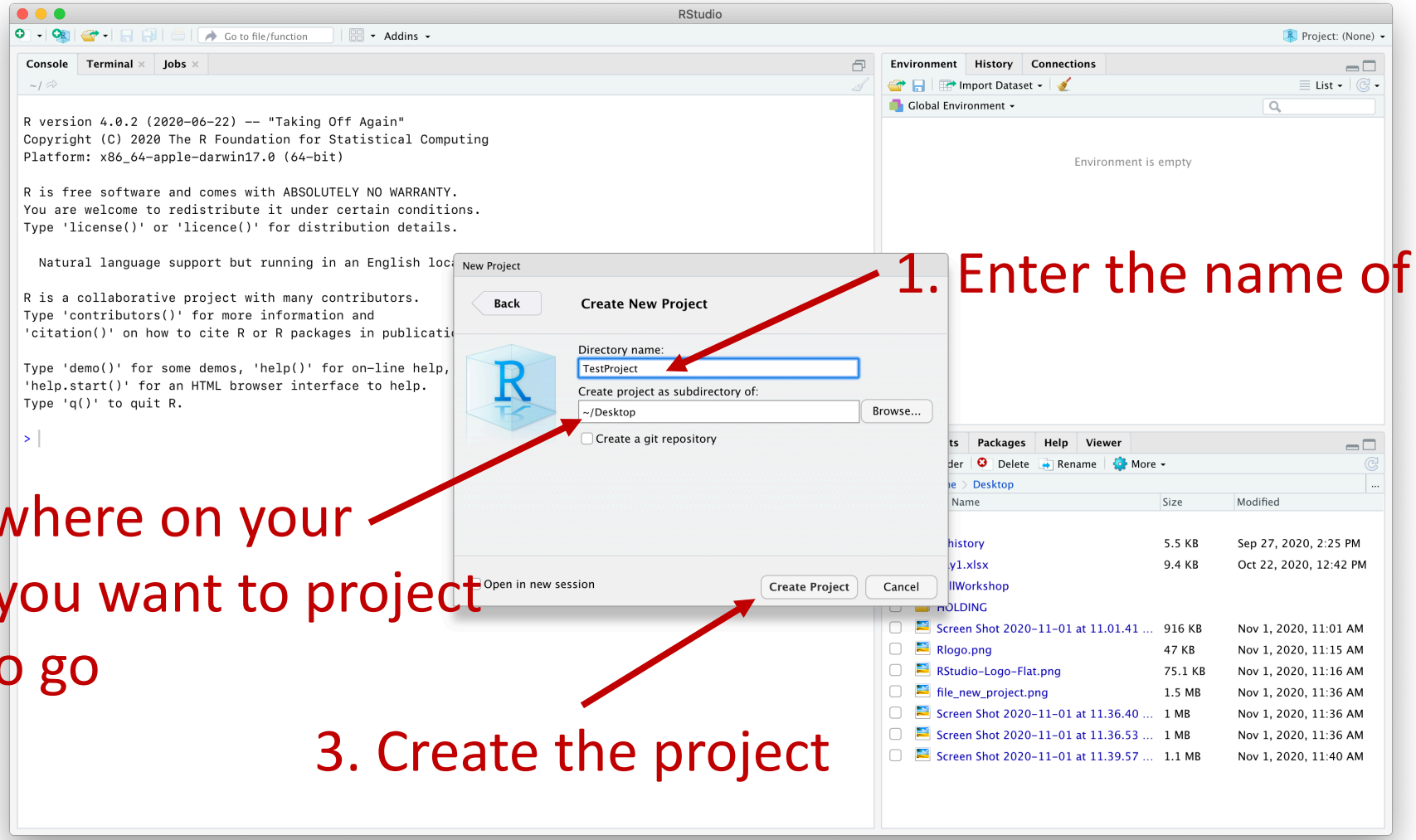
Make a New Directory for the project

Creating an RStudio Project

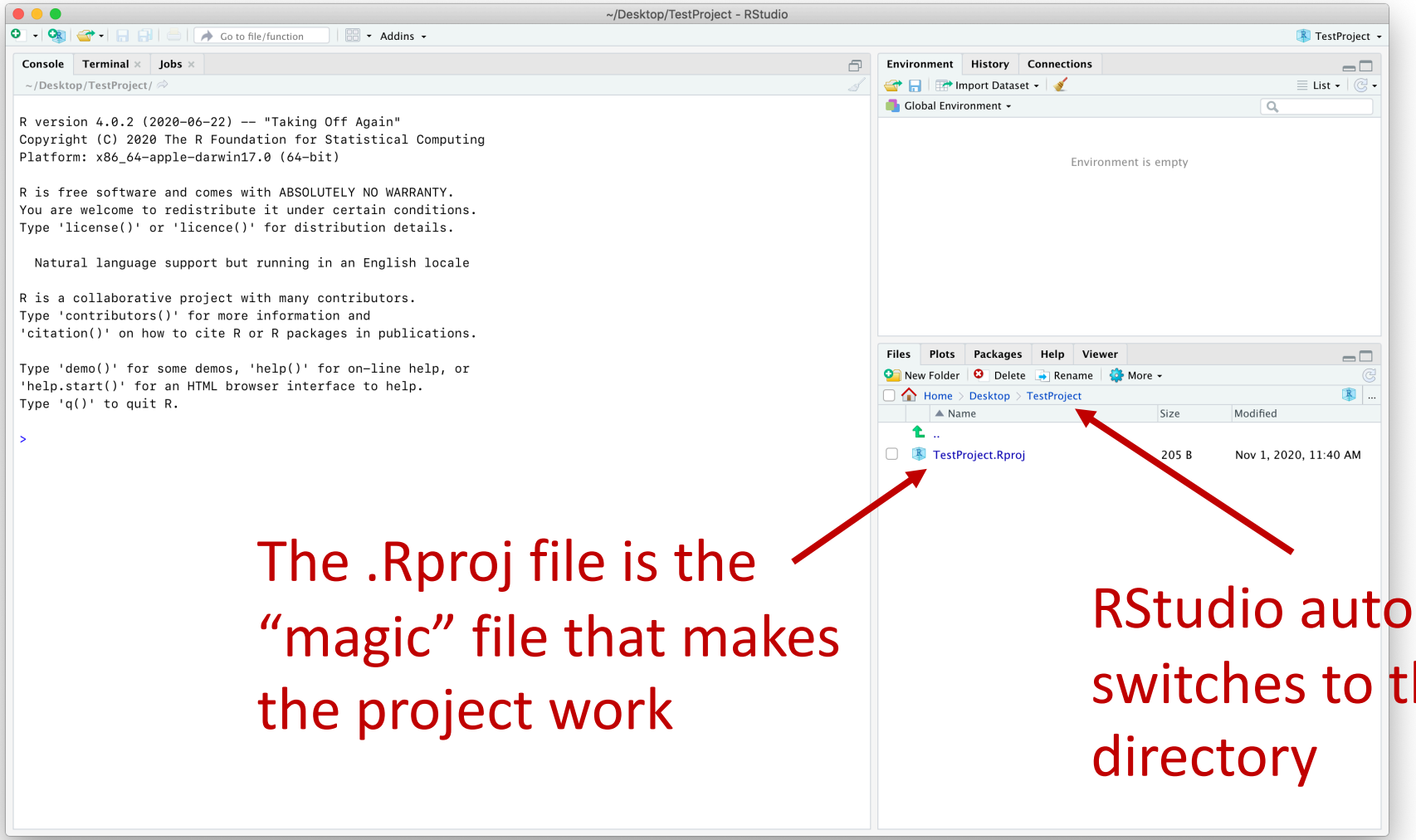


Specific (again) that
you want a New Project

Creating an RStudio Project



Creating an RStudio Project



Working With RStudio Projects

- After you create the project, note where you put it (where the directory lives on your computer)
- You can reference data files in your project directory with a file path *starting from the project directory*
- If the data file is in the main project directory, you can just refer to it my its name (no need to specify a path)
- When you want to come back to your project
  1. Navigate to the project directory in your OS file browser
  2. Double click the .Rproj file
  3. RStudio should launch and set the project working directory

Exercise: Working with RStudio Projects

1. Start RStudio
2. Create a new RStudio Project
    1. “New Directory” → “New Project”
    2. Pay attention paths where the project will be created
3. Locate the project directory using your computer’s file browser and note the .Rproj file
4. Copy `Choi2017_DDA_Skyline_input.csv` into the project directory
5. Read the file into R
   `read.csv(“Choi2017_DDA_Skyline_input.csv”)`
6. Close RStudio (completely quite the application) and relaunch the project by double clicking on the project’s .Rproj file

First Steps with R Coding

ASMS 2020 Fall Workshop

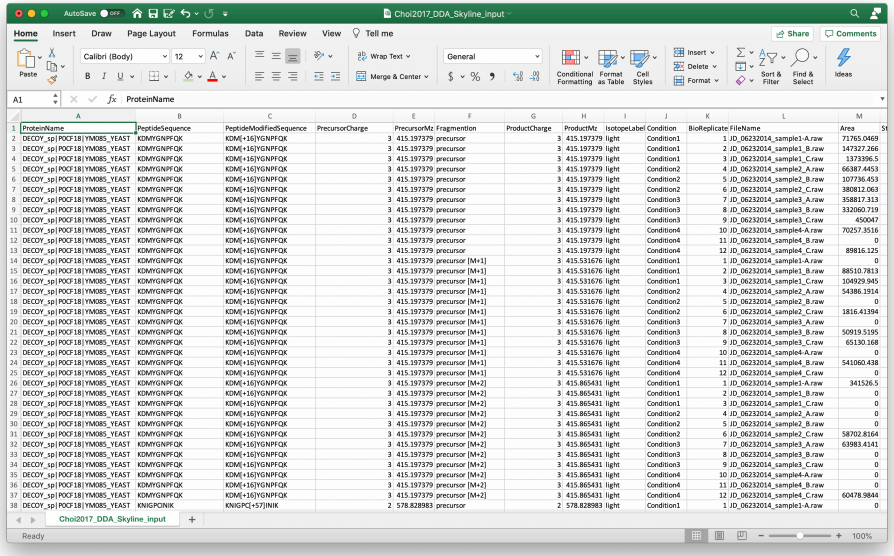
Day 1 – Module 2

Goals of the Module

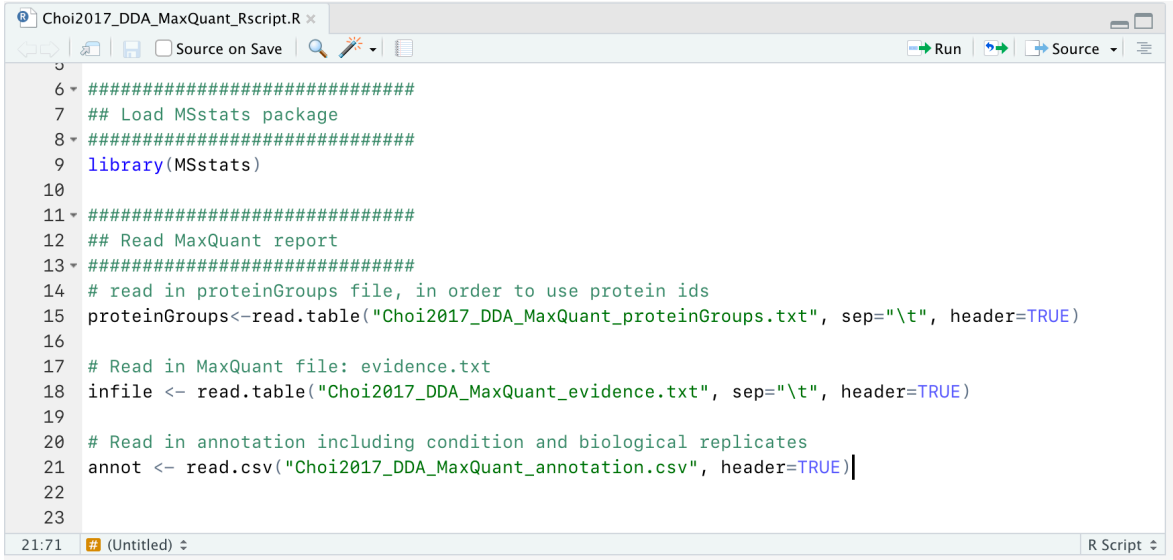
- Understand how to work with R and how you use it to do things
- Understand essential R syntax
- How to work with variables, vectors and conditional expressions
- Overview of some essential R functions and how to learn more

You interact with R by writing code rather than using a graphical user interface

You “do things” in Excel using a GUI
LOTS of pointing and clicking…



You “do things” in R by writing code (text)
LOTS of typing…



Writing code has several important advantages vs. using GUI-based software

- Code explicitly captures what you do
- Code can be easily shared with others
- Code can be reused
- Code enables reproducibility
- Code allows you to do “anything” you want
- Coding can be faster

You can use R like a calculator

- (like any programming language) R supports all standard mathematical operations: +, -, *, /, and lots more
- Base R provides lots of functions for doing standard and more complex mathematical operations
- More advanced and specialized operations are often available through R packages (there are more than 10K packages on CRAN)
- You can write your own functions to anything you want!

live demo

R *syntax* is the set of rules that define how to write correct R code

- You write R code using characters, symbols and numbers
- R syntax defines how you can write correct R code
- Even the smallest deviation from these rules will result in a *syntax error*
- In the beginning, you will make lots of syntax errors, and it will be frustrating – hang in there!
- Even R experts make syntax errors, but as you get more familiar with the language, you'll be able to see and fix problems more quickly

R Syntax & Coding Style Tips

- Computers are *very* picky – even a single misplaced character can cause a syntax error
- R code often follows rules for writing mathematical expressions
  - Put spaces between distinct object & symbols for readability
  - It's common in R code to use new lines to make code more readable
  - Quotes, parentheses, brackets almost always come in pairs and need to match up by their order
- If you make a syntax error, R will let you know with an error message; they are sometimes very cryptic, but can be helpful
- Complex expressions are more prone to syntax errors, try breaking down into several shorter expressions

First steps with R Syntax: Variable Assignment

- Variables allow you to store data using names
  ```
  my_var <- 10
  ```
- R uses a very odd symbol for variable assignment: `<-`
- You can also use the equals sign =, but `<-` is considered best practice
  note: = *is* used for function arguments
- Naming rules
  - can (only) contain: letters, numbers, underscores _ , or periods .
  - No spaces!
  - must start with letter or period (not underscores)
  - cannot start with a number, or a period followed-up by a number

Variable Name Examples

Valid Variable Names

- my_var
- my_var2
- my.var
- .my.var
- gradient_minutes
- min_mz

Invalid Variable Names

- 1my_var
- _my_var
- @ne
- 0ne
- .0ne
- my var

Why are each of these invalid?

live demo

Data structures hold data

- Data structures are like ”bags” that hold data and allow you to work with it in a structured way
- There are many different types of “bags” depending on what you need
- Different programming languages often have similar types of “bags” but often use a variety of different names for them
- Essential R data structures
  - (atomic) **vectors**
  - **data frames** & **tibbles** (modern data frames)
  - lists (won’t cover these here…)

Vectors hold sets of data of the same type

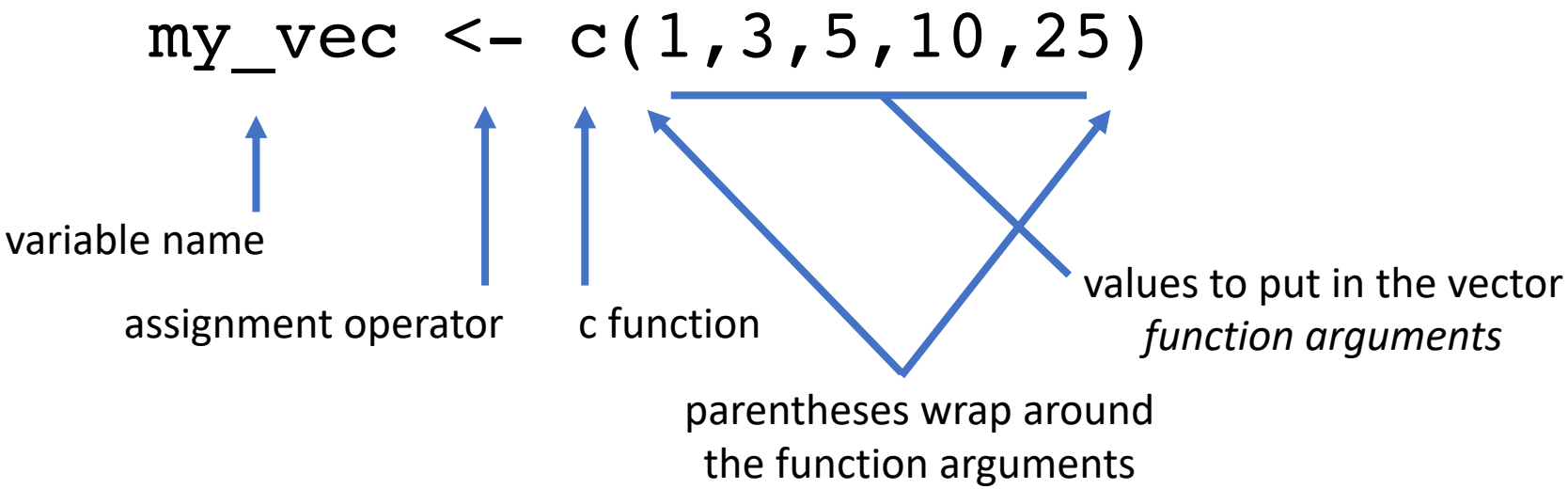
- Vectors hold ordered data, e.g. a set of numbers
- Vectors can be used anytime you have a set of values that you want to keep together
- You could represent a mass spectrum with two vectors:

m/z	intensity
968.4750	1002.1
968.5750	2267.5
968.6750	7998.3
968.7750	2721.0
968.8750	964.3

these vectors have the same number of values, ordered in the same way: m/z, intensity pairs

More R Syntax: Creating Vectors

- Use variable assignment syntax with the `c` function to make a vector



live demo

Performing Math with Vectors

- R makes great use of vectors and *vectorization* to compute on sets of data (i.e. vectors) using concise code
- Two main “patterns” for basic operations (+, -, *, /)
  - *A vector and a single value*
    performs the operation on each element of the vector and the single value
  - *Two vectors of the same length*
    performs the operation element by element
- The output of the expression is a new vector with the same length as the as the original one(s)
- You can operate with vectors of two different lengths… but *don't* do this in general (you might do this in more advanced cases)

Examples: Math with Vectors

Code	Output
`my_vec <- c(30, 40, 10, 50, 20)`	*assigns the give vector to the variable* `my_vec`
`my_vec + 1`	`31 41 11 51 21`
`my_vec * 2`	`60 80 20 100 40`
`my_vec + my_vec`	`60 80 20 100 40`
`my_vec - my_vec`	`0 0 0 0 0`
`my_vec + c(1, 2, 3, 4, 5)`	`31 42 13 54 25`

Functions are canned routines that operate on inputs and give you back something new

- Functions allow you to operate on data to do routine/hard/interesting things
- Functions have a *name* and (optionally) *inputs* (or arguments)
  The *name* comes first, and the inputs are wrapped in ( )
  `func_name(input1, input2, …)`
- Base R provides a ton functions you can use for all kinds of data analysis tasks, and you can get more by installing R packages or writing your own!

Basic Function for Working with Vectors

Operation	Function
Find the length (number of values)	`length`
Find the minimum element	`min`
Find the maximum element	`max`
order the elements	`sort`
mean	`mean`
standard deviation	`sd`

Examples: Working with Vectors

Code	Output
`my_vec <- c(30, 40, 10, 50, 20)`	*assigns the give vector to the variable* `my_vec`
`length(my_vec)`	`5`
`min(my_vec)`	`10`
`mean(my_vec)`	`30`
`sort(my_vec)`	`10 20 30 40 50`
How might you get the median of `my_vec`*?*	`30`

Condition Expressions Allow you To Ask Yes/No Questions about your Data

- Conditional expression use logical operators to “test” your data
  - *less than:* `<`, *less than or equal:* `<=`
  - *greater than:* `>`, *greater than or equal:* `>=`
  - *equal to:* `==` (note the double equals sign), not equal to: `!=`
- Conditional expressions give back `TRUE` or `FALSE` depending on the outcome of the test
- `TRUE` and `FALSE` are *logical* data types in R
  - all caps!
  - `true` or `True` or `tRuE` aren’t valid

Examples: Conditional Expressions

Code	Output
`my_val <- 123`	*assigns the given value to* `my_val`
`my_vec <- c(30, 40, 10, 50, 20)`	*assigns the give vector to the variable* `my_vec`
`my_val > 100`	`TRUE`
`my_val == 122`	`FALSE`
`my_vec > 20`	`TRUE TRUE FALSE TRUE FALSE`
`gt_20 <- my_vec > 20`	*you can save the results of the expression to a variable*

Subsetting Vectors

- It’s often useful to get (or exclude) specific values from a vector
- The *subset* operators allow you to do this: [ ] (square brackets)
- There are a few ways to subset
  - get values by their (integer) position in the vector
  - get values with a logical vector (only keep element with TRUE)
  - get values by name (won’t cover this here..)
- Note: for positional subsetting, R starts with 1
  (believe it or not, many programming languages start with 0)

Examples: Subsetting Vectors

Code	Output
`my_vec <- c(30, 40, 10, 50, 20)`	*assigns the give vector to the variable* `my_vec`
`my_vec[1]`	`30`
`my_vec[5]`	`20`
`my_vec[c(1,3,5)]`	`30 10 20`
`my_vec > 20`	`TRUE TRUE FALSE TRUE FALSE`
`my_vec[my_vec > 20]`	`30 40 50`

Recap

- You interact with R by writing code
- Coding syntax are the rules that dictate what code is valid
- In the beginning, you will likely be making lots of syntax errors, but hang in there – you'll get better with practice
- Coding topics we covered
  - variable assignment
  - vectors and how to work with them
  - conditional expressions

Reading Data Files into R

ASMS 2020 Fall Workshop

Day 1 – Module 3

Goals of the Module

- Learn how to read data files into R
- Understand the basics of file paths & how RStudio Projects help
- Important things to keep in mind when working with .csv files

Reading Data into R

- Before you can work with a data set, you need to get it into R
- R can handle a (large) variety of file formats, including
  - formatted text files (e.g. csv)
  - most standard open formats (e.g. web formats, json, markup formats)
  - (open and some proprietary) binary formats
  - open MS data formats (e.g. mzML)
- In general, R *can't* read proprietary MS vendor formats (but that's changing)
- You read data into R using "reader" functions –
  available in base R and R packages
- Formatted text files, e.g. csv files, are the easiest to work with
  (we'll focus on those here)

Reading Formatted Text Files

- R has several built-in function for reading text files, including
  - `read.csv` – comma separated files
  - `read.delim` – tab separated files
  - `read.fwf` – fixed width files
- However, we're going to use the reader functions provided by the `readr` package (part of the `tidyverse`)
  - `read_csv` – comma separated files
  - `read_tsv` – tab separated files
  - `read_fwf` – fixed width files

Example: Reading a csv File

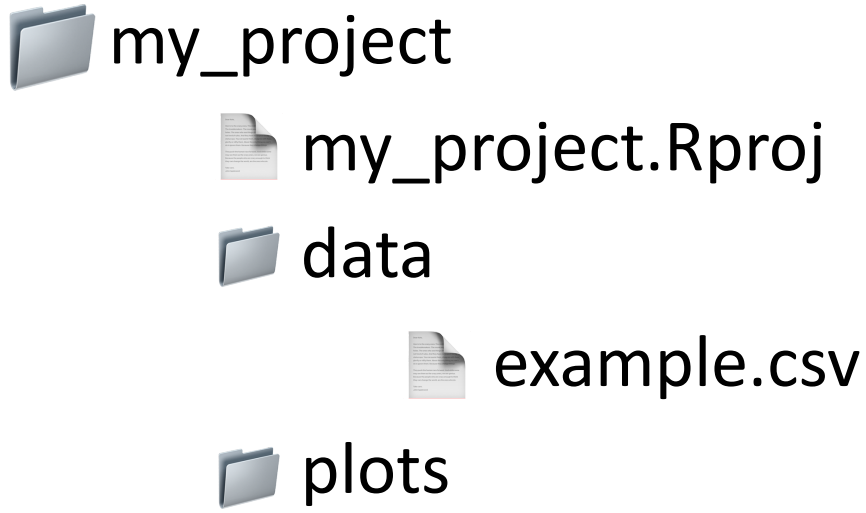
```
# load the tidyverse package
# will also load the readr package
library(tidyverse)

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

If your data file is in a sub directory…

Directory Structure



R Code

```
read_csv("data/example.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 (I say MUST) be the header row
  - lists the names of the columns
  - can contain spaces, but better if not
- Every other row contains the data, each value separated by a comma
- 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

Notes about reading data files into R

- You'll almost always want to store the output as a variable, e.g.

```
dat <- read_csv("example.csv")
```

- For table-like file formats, the data will be read as a *data frame*
- We'll learn how to work with data frames (tibbles) in the next module
- If a data file is large (~100's MBs to GBs)
  - it might take some time to read
  - you'll want to have sufficient RAM
  - (very roughly) R may start to have issues with files larger than several GBs

Recap

- Getting data into R is a fundamental part of the analysis process
- R can read lots of different types of data files "out of the box", and can read even more using contributed R packages
- Formatted text files are some of the easiest to work with
- The `read_csv` function from the `readr` package can be used to read .csv files

Working with Data Frames (tibbles)

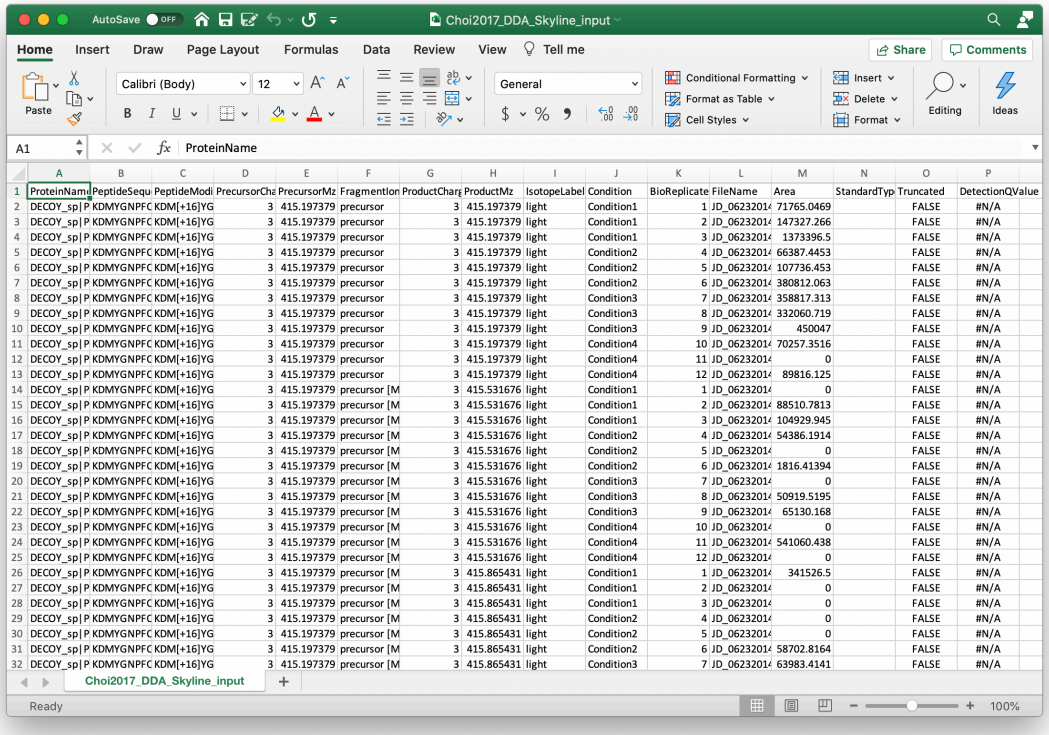
ASMS 2020 Fall Workshop

Day 1 – Module 4

Goals of the Module

- Understand what a data frame is
- Learn how to review and work with data frames
- Learn how to perform basic operations on data frames, including getting subsets

Tabular data is everywhere…



Excel uses a (flexible) table format to hold data

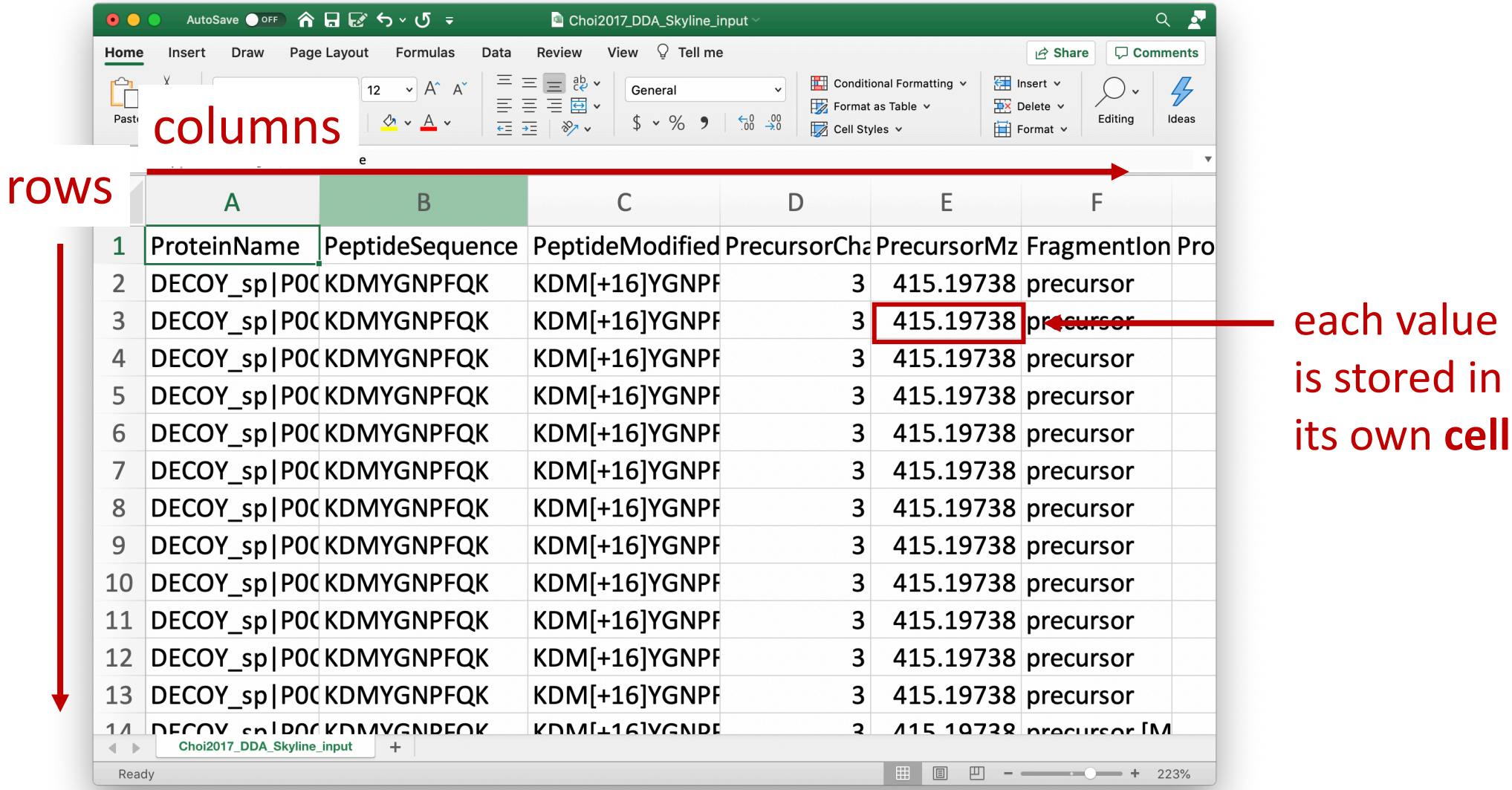
csv files represent tabular data

tables are often used to present data

year	artist	track	time	date.entered	wk1	wk2	wk3
2000	2 Pac	Baby Don't Cry	4:22	2000-02-26	87	82	72
2000	2Ge+her	The Hardest Part Of ...	3:15	2000-09-02	91	87	92
2000	3 Doors Down	Kryptonite	3:53	2000-04-08	81	70	68
2000	98^0	Give Me Just One Nig...	3:24	2000-08-19	51	39	34
2000	A*Teens	Dancing Queen	3:44	2000-07-08	97	97	96
2000	Aaliyah	I Don't Wanna	4:15	2000-01-29	84	62	51
2000	Aaliyah	Try Again	4:03	2000-03-18	59	53	38
2000	Adams, Yolanda	Open My Heart	5:30	2000-08-26	76	76	74

https://vita.had.co.nz/papers/tidy-data.pdf

Tables arrange data in rows, columns, & cells

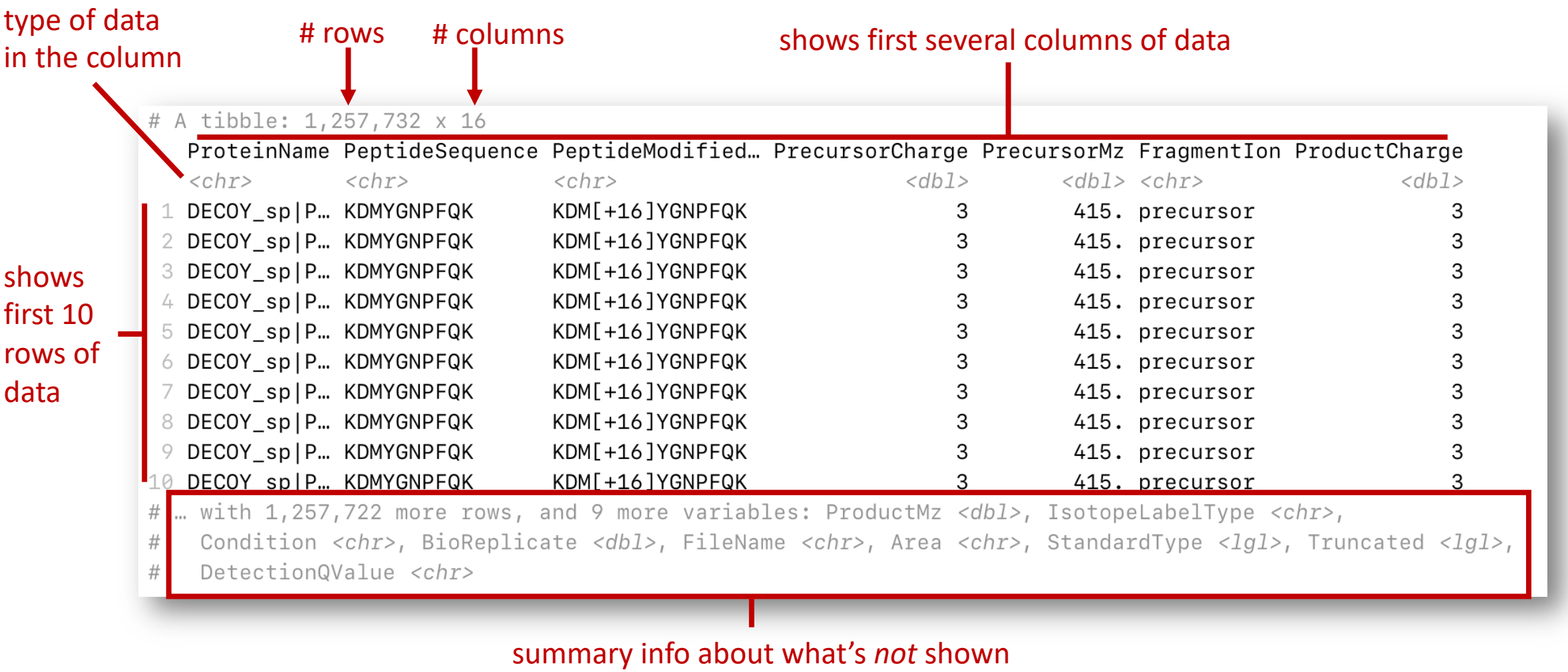


	A	B	C	D	E	F
1	ProteinName	PeptideSequence	PeptideModified	PrecursorCha	PrecursorMz	FragmentIon
2	DECOY_sp|P0C	KDMYGNPFQK	KDM[+16]YGNPF	3	415.19738	precursor
3	DECOY_sp|P0C	KDMYGNPFQK	KDM[+16]YGNPF	3	415.19738	precursor
4	DECOY_sp|P0C	KDMYGNPFQK	KDM[+16]YGNPF	3	415.19738	precursor
5	DECOY_sp|P0C	KDMYGNPFQK	KDM[+16]YGNPF	3	415.19738	precursor
6	DECOY_sp|P0C	KDMYGNPFQK	KDM[+16]YGNPF	3	415.19738	precursor
7	DECOY_sp|P0C	KDMYGNPFQK	KDM[+16]YGNPF	3	415.19738	precursor
8	DECOY_sp|P0C	KDMYGNPFQK	KDM[+16]YGNPF	3	415.19738	precursor
9	DECOY_sp|P0C	KDMYGNPFQK	KDM[+16]YGNPF	3	415.19738	precursor
10	DECOY_sp|P0C	KDMYGNPFQK	KDM[+16]YGNPF	3	415.19738	precursor
11	DECOY_sp|P0C	KDMYGNPFQK	KDM[+16]YGNPF	3	415.19738	precursor
12	DECOY_sp|P0C	KDMYGNPFQK	KDM[+16]YGNPF	3	415.19738	precursor
13	DECOY_sp|P0C	KDMYGNPFQK	KDM[+16]YGNPF	3	415.19738	precursor

R stores tabular data in Data Frames

- The data.frame is one of the most important data structures in R
  - Lots (most?) of data you encounter can be represented as a table
  - Most of the `read_*` functions produce a data frame
  - R's design and on-going development is informed by data tables and ways of making it easier to work with them
- We'll be working with a "modern reimagining" of the data frame, known as the `tibble` (used in the `tidyverse`)
- "data frame" = a `data.frame` or a `tibble`

Default view of a tibble




```
# A tibble: 1,257,732 x 16
   ProteinName PeptideSequence PeptideModified… PrecursorCharge PrecursorMz FragmentIon ProductCharge
   <chr>       <chr>           <chr>                      <dbl>       <dbl> <chr>               <dbl>
 1 DECOY_sp|P… KDMYGNPFQK      KDM[+16]YGNPFQK                3        415. precursor               3
 2 DECOY_sp|P… KDMYGNPFQK      KDM[+16]YGNPFQK                3        415. precursor               3
 3 DECOY_sp|P… KDMYGNPFQK      KDM[+16]YGNPFQK                3        415. precursor               3
 4 DECOY_sp|P… KDMYGNPFQK      KDM[+16]YGNPFQK                3        415. precursor               3
 5 DECOY_sp|P… KDMYGNPFQK      KDM[+16]YGNPFQK                3        415. precursor               3
 6 DECOY_sp|P… KDMYGNPFQK      KDM[+16]YGNPFQK                3        415. precursor               3
 7 DECOY_sp|P… KDMYGNPFQK      KDM[+16]YGNPFQK                3        415. precursor               3
 8 DECOY_sp|P… KDMYGNPFQK      KDM[+16]YGNPFQK                3        415. precursor               3
 9 DECOY_sp|P… KDMYGNPFQK      KDM[+16]YGNPFQK                3        415. precursor               3
10 DECOY_sp|P… KDMYGNPFQK      KDM[+16]YGNPFQK                3        415. precursor               3
# … with 1,257,722 more rows, and 9 more variables: ProductMz <dbl>, IsotopeLabelType <chr>,
#   Condition <chr>, BioReplicate <dbl>, FileName <chr>, Area <chr>, StandardType <lgl>, Truncated <lgl>,
#   DetectionQValue <chr>
```

Basic functions for working with data frames

Operation	Function
Find the number of rows, columns	`nrow, ncol`
Find the number of rows, columns (alternative)	`dim`
Get the names of the columns	`names`
View the data table (from within Rstudio)	`View`
Access data from a particular column	*`my_df$col_name`*

Practice with data frames

```
# Load the tidyverse package (so we can use read_csv)
library(tidyverse)

# Read in the csv file
dat <- read_csv("iPRG2015-Skyline/Choi2017_DDA_Skyline_annotation.csv")

# Type the name of the variable and press return to see a summary
dat

# Get the number of rows and columns
nrow(dat)
ncol(dat)
dim(dat)

# Get the names of the columns
names(dat)

# Get the unique values of the Condition column
unique(dat$Condition)
```

Subsetting data frames

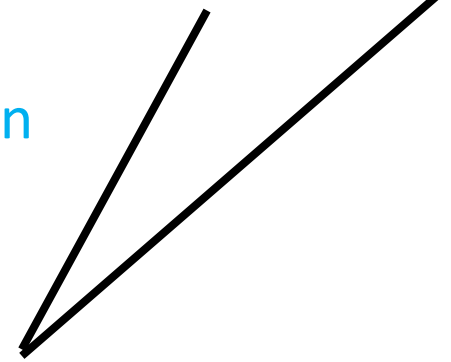
- Getting data subsets (specific columns or rows) is very common operation in any data analysis
- `$` can be used to subset to a single column
  `my_df$column_name`
- Square brackets, `[ ]` , can be used to subset rows & columns
  `my_df[row_selection, col_selection]`

Subsetting with [ ]

```
my_df[row_selection, col_selection]
```

- A vector of integers (for each row)
- A conditional expression
  TRUE keeps a row
  FALSE rejects a row

- A vector of integers (for each column)
- A vector of column names
- A conditional expression
  TRUE keeps a column
  FALSE rejects a column



Tip: use the colon operator to get a range of integers

`1:5` is the same as `c(1,2,3,4,5)`

If you don't specify either a row or column selection (i.e. a blank space) you get every row or column

`my_df[,]` is the same as `my_df`

Practice subsetting data frames

```
# Load the tidyverse package (so we can use read_csv)
library(tidyverse)

# Read in the csv file
dat <- read_csv("iPRG2015-Skyline/Choi2017_DDA_Skyline_annotation.csv")

# Get the first 10 rows of the data frame
dat[c(1,2,3,4,5,6,7,8,9,10),]
dat[1:10,]

# Get the last two columns
dat[, c(2,3)]
dat[, c("Condition", "BioReplicate")

# Get the rows for Condition4 (only)
rows_sel <- dat$Condition == "Condition4"
dat[rows_sel,]
# or put it all in one expression
dat[dat$Condition == "Condition4",]
```

Recap

- Data frames are used to represent tabular data in R
- Data frames are one of the most important and widely used data structures in R
- The `read_csv` function reads data in as a data frame (tibble)
- There are LOTS of functions that operate on data frames to do investigate, manipulate and transform the data
- Subsetting data frames is an important data operation, and can be accomplished using square brackets, [ ]

Tidy data & the tidyverse

ASMS 2020 Fall Workshop

Day 1 – Module 5

Goals for this module

- Understand what tidy data is, and how it differs from messy data
- Learn how to recognize messy data and why it’s hard to work with
- Learn about the tidyverse and why it’s a great ecosystem for working with data in R

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~~ un~~happy 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

Example Tidy Data Table

Run	Condition	BioReplicate
JD_06232014_sample1-A.raw	Condition1	1
JD_06232014_sample1_B.raw	Condition1	1
JD_06232014_sample1_C.raw	Condition1	1
JD_06232014_sample2_A.raw	Condition2	2
JD_06232014_sample2_B.raw	Condition2	2
JD_06232014_sample2_C.raw	Condition2	2
JD_06232014_sample3_A.raw	Condition3	3
JD_06232014_sample3_B.raw	Condition3	3
JD_06232014_sample3_C.raw	Condition3	3
JD_06232014_sample4-A.raw	Condition4	4
JD_06232014_sample4_B.raw	Condition4	4
JD_06232014_sample4_C.raw	Condition4	4

What is this table about?

What does each row represent?

What is being measured
(what are the columns)?

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

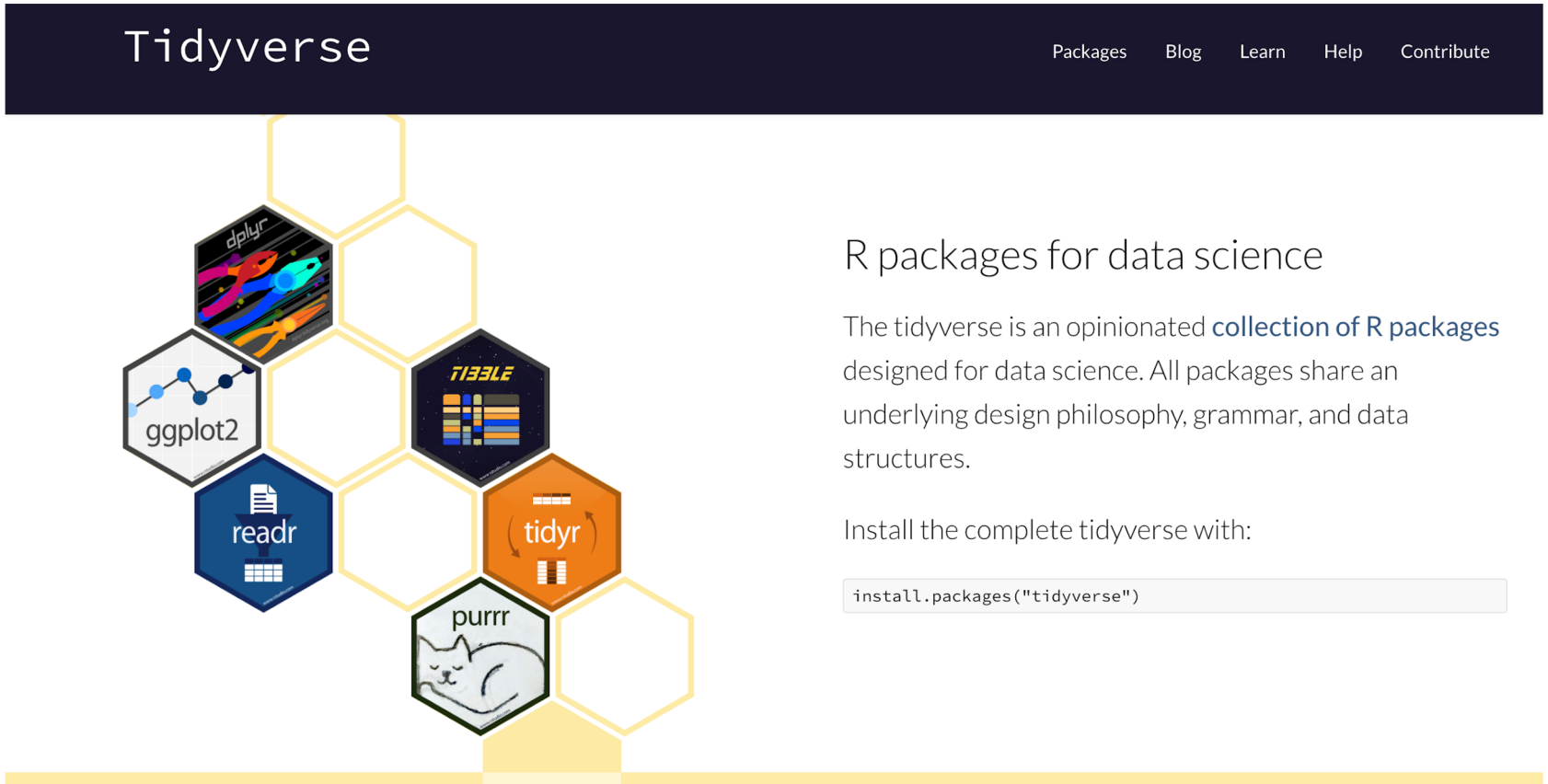
Be on the lookout for signs of messy data!

- Overall structure of the data is unclear or inconsistent
- Multiple pieces of information are stored in a single cell, e.g. `Male_age16, Female_age42`
- A variable is spread across multiple columns
- An observation is spread across multiple rows

As you get more experience recognizing messy data, you can better communicate to others (i.e. collaborators) how to produce well structured data in order to make your job easier as a data analyst!

The Tidyverse

An opinionated collection of R packages for data science



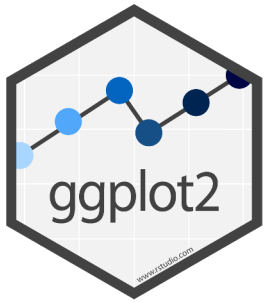
https://www.tidyverse.org

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

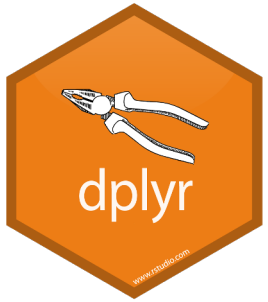
That's the tidyverse!

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...

Recap

- Tidy data is well structured data that's ready for analysis
- In a tidy data table
  - each row is an observation
  - each column is a variable
  - each (individual) piece of data goes in its own cell
- Messy data is everywhere – as you get more proficient with R, you'll learn how to better deal with messy data
- The tidyverse leverages the well-structured nature of tidy data to provide awesome tools for working with data

Basic Data Manipulation with dplyr

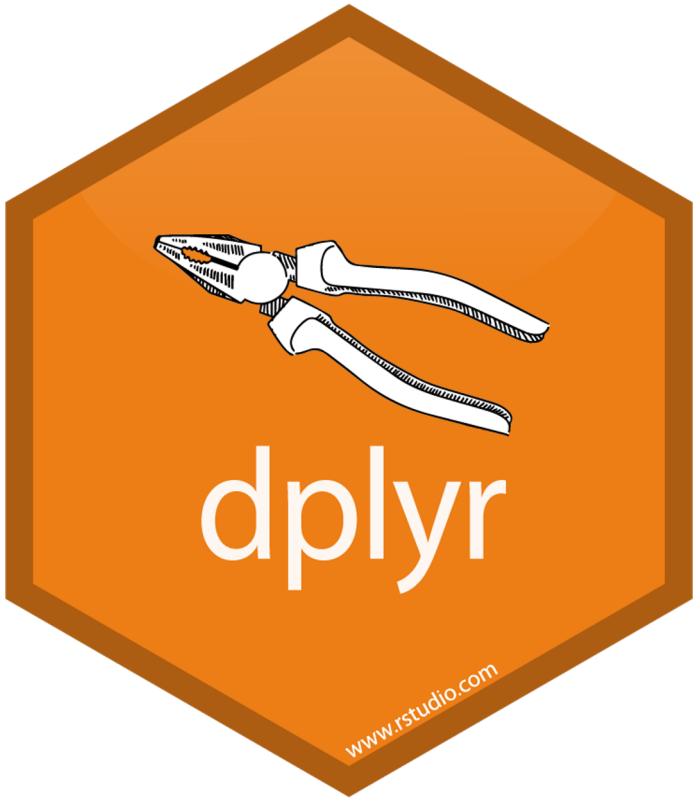
ASMS 2020 Fall Workshop

Day 1 – Module 6

Goals of the Module

- Understand fundamental operations that underlie most common data manipulation challenges
- Learn about the `dplyr` R package and how it helps you perform these data manipulation operations
- Learn about the pipe `%>%` operator
- Learn how to build data manipulation and transformation pipelines with `dplyr`

`dplyr` is a tidyverse R package for data manipulation

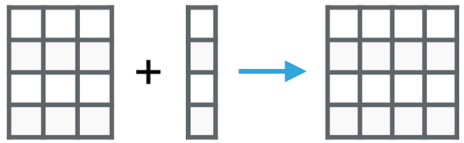
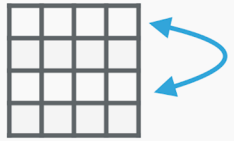
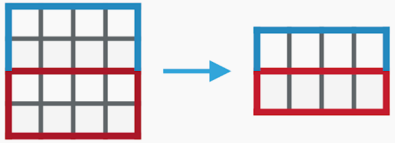


- Defines the fundamental operations that encompass most data analysis tasks
- Assumes you already have tidy data
- Uses a pipeline coding structure to perform complex operations a straight-forward, natural way

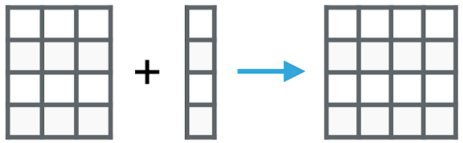
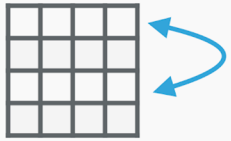
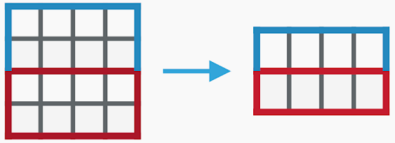
`dplyr` defines data manipulation *verbs*

- 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`

Using dplyr verbs (functions)

- The first argument of a dplyr function is *always* the data frame that you want to operate on, e.g.

  `mutate(my_df, …)` – *adds a column to my_df*

  `select(my_df, …)` – *subsets to specific columns of my_df*

- The … above are additional arguments that further define what the operation is going to do

- dplyr verbs always give back a data frame

  `my_df2 <- mutate(my_df, …)`

  my_df2 is a data frame that has a new column

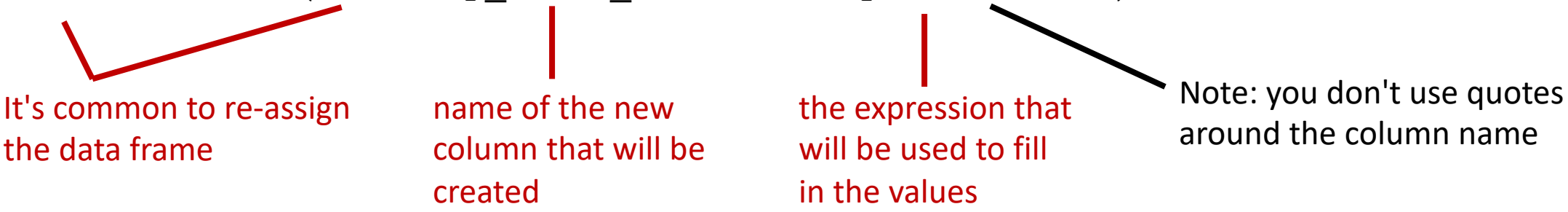
dplyr verb examples: mutate

- You use mutate when you want to add a new column to your data frame, often based upon existing columns

```
# load the tidyverse package
library(tidyverse)

# load the data
dat <- read_csv("iPRG2015-Skyline/Choi2017_DDA_Skyline_annotation.csv")

# add a new column that is 10 times the value of the BioReplicate column
dat <- mutate(dat, rep_times_ten = BioReplicate * 10)
```



dplyr verb examples: select

- You use select when you to to get specific columns, or get rid of ones you don't want

```
# load the tidyverse package
library(tidyverse)

# load the data
dat <- read_csv("iPRG2015-Skyline/Choi2017_DDA_Skyline_annotation.csv")

# get only the Condition and BioReplicate columns
dat <- select(dat, c("Condition", "BioReplicate"))
```


give a vector of column name
that you want to choose

dplyr verb examples: filter

- You use filter when you to to get specific rows, very often specified using a conditional expression

```
# load the tidyverse package
library(tidyverse)

# load the data
dat <- read_csv("iPRG2015-Skyline/Choi2017_DDA_Skyline_annotation.csv")

# get only the rows for Condition4
dat <- filter(dat, Condition == "Condition4")
```

this is a conditional expression
that tests values in the
Condition column

dplyr verb examples: arrange

- You use arrange when you want to re-order the rows of a data frame

```
# load the tidyverse package
library(tidyverse)

# load the data
dat <- read_csv("iPRG2015-Skyline/Choi2017_DDA_Skyline_annotation.csv")

# sort the rows by Condition
dat <- arrange(dat, Condition)

# use desc to reverse the sorting
dat <- arrange(dat, desc(Condition))
```

orders the rows based on the
(alphabetical) sorting of Condition

dplyr verb examples: group_by + summarize

- These two verbs are usually used together, when you want to group by a particular column and compute summaries for each group

```
# load the tidyverse package
library(tidyverse)

# load the data
dat <- read_csv("iPRG2015-Skyline/Choi2017_DDA_Skyline_annotation.csv")

# count the number of runs per condition
dat_grouped <- group_by(dat, Condition)
summarize(dat_grouped, n_runs = length(Run))
```

makes 4 groups, one for each Condition

for each individual group…

compute a new column…

that has the number of Run values

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 individual characters
- Yes -- it looks strange, think of %>% as a pipe, funneling the input data into the function to do some work

The pipe operator: %>%

Template

tidy data frame %>% dplyr verb *produces a* modified data frame

---

tidy data frame %>% `mutate` *produces a* modified data frame with a new column

tidy data frame %>% `select` *produces a* modified data frame with selected columns

tidy data frame %>% `filter` *produces a* modified data frame with filtered rows

Practice with pipes

R Code without pipes

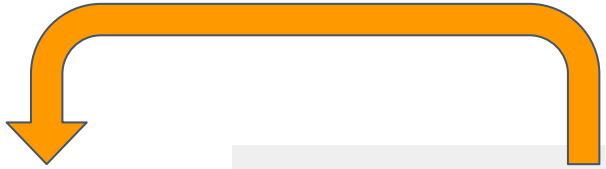
```
filter(dat, Condition == "Condition4")
```

R Code *with* pipes

```
dat %>% filter(Condition == "Condition4")
```

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

Practice with pipes



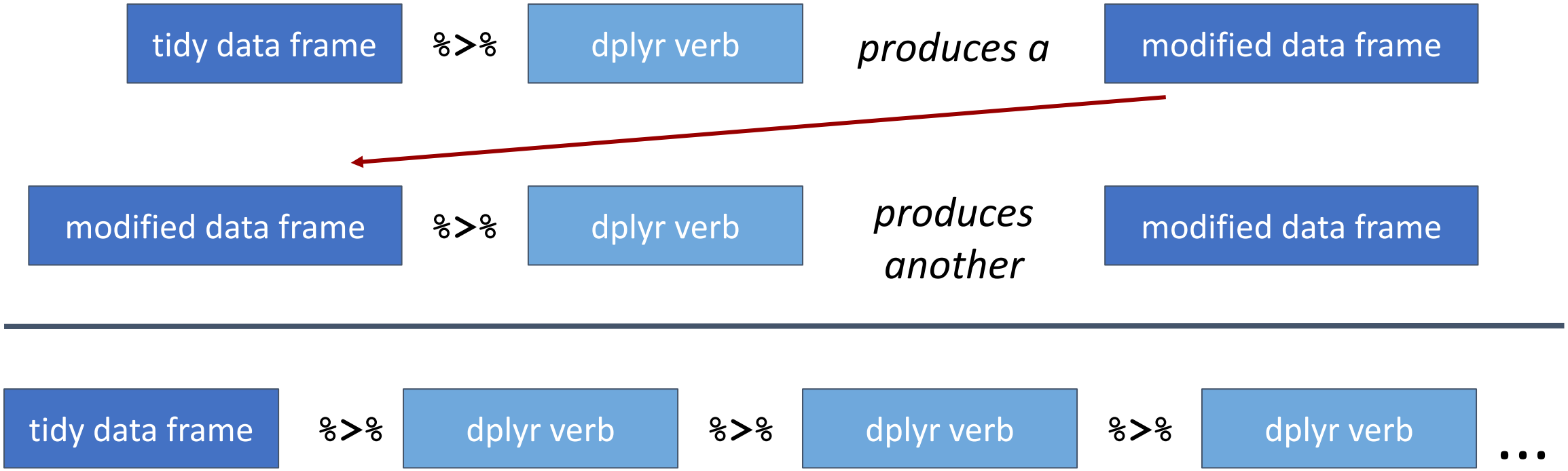
```
filter(dat, Condition == "Condition4")
```

```
dat %>% filter(Condition == "Condition4")
```

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

dplyr pipeline example

Do the following:

1. only get BioReplicate values of 2 or higher, then…
2. count the number of Run values per Condition, then…
3. order the results in descending Condition order

```
# load the tidyverse package
library(tidyverse)

# load the data
dat <- read_csv("iPRG2015-Skyline/Choi2017_DDA_Skyline_annotation.csv")

# accomplish the above steps in a dplyr pipeline
dat %>%
  filter(BioReplicate >= 2) %>%
  group_by(Condition) %>%
  summarize(n_runs = length(Run)) %>%
  arrange(desc(Condition))
```

"only get BioReplicate values of 2 or higher"

"count the number of Run values per Condition"

"order the results in descending Condition order"

dplyr pipelines can (and often are) placed on multiple lines (make sure %>% is at the end of a line)

Recap

- 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

Day 1 Recap & Next Steps

ASMS 2020 Fall Workshop

Day 1 – Wrap-up

Today we learned

- What is R & RStudio
- Basic R Syntax:
  - variable assignment
  - working with functions
  - subsetting
  - conditional expressions
- How to work with vectors and data frames
- How to read data into R
- What tidy data is
- How to use dplyr to operate on data frames

Are you now able to…

- start-up RStudio and make an RStudio project?
- Create variables and vectors of values, and work with them?
- 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?

Resources

Websites

- RStudio: https://rstudio.com/
- The tidyverse: https://www.tidyverse.org/

Books

- Hands on Programming with R: https://rstudio-education.github.io/hopr/
- R for Data Science: https://r4ds.had.co.nz/

People/Orgs to Follow

- RStudio
- Hadley Wickham (@hadleywickham)
- Jenny Bryan (@JennyBryan)
- Julia Silge (@juliasilge)
- Mara Averick (@dataandme)
- David Robinson (@drob)
- Twitter hash-tags: #rstats, #DataScience

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!)

Thank you to the instructors and to the teaching assistants!

Ryan Benz
Meena Choi
Niyati Chopra
Miguel Cosenza
Matthias Fahrner
Amanda Figueroa-Navedo
Melanie Foell
Omkar Reddy Gojala
Dan Guo
Shubhanshu Gupta
Ting Huang
Maanasa Kaza
Smit Anish Kiri
Devon Kohler
Sai Srikanth Lakkimsetty
Danielle LaMay
Ajeya Makanahalli Kempegowda
Yogesh Nizzer
Harish Ramani
Ruthvik Ravindra
Abdul Rehman
Sai Divya Sangeetha Bhagavatula
Siddarth Sathyanarayanan
Gopalika Shama
Rishabh Rajesh Shanbhag
Sagar Singh
Mateusz Staniak
Sara Taheri
Anuska Tak
Derrie Susan Varghese
Amrutha Vempati

Video of the presentation: https://www.youtube.com/channel/UCnbUMFlIRLaY7fwfSintWuQ/

Thank you!

Question time…