Northeastern University

May Institute goes ONLINE!
Computation and statistics for mass spectrometry and proteomics

May 3 – May 14, 2021, Northeastern University, Boston MA
Organizer: Olga Vitek

May 7, 2021

11:00am-2:15pm **Hands-on, Ryan Benz:** Part 1: Introduction to R for beginners

Starting soon

Materials at
https://computationalproteomics.khoury.northeastern.edu/

NIH National Institutes of Health
Chan Zuckerberg Initiative
ASA Boston Chapter

Hands-on Overview

- Day 1 - Friday May 7th
  Introduction to R, RStudio, and R coding fundamentals
  - Introduction to the RStudio IDE
  - First steps with coding: the console, editor & basic syntax
  - R coding essentials: variables, vectors, math operations, conditional expressions
  - Working with tabular data (data frames)

- Day 2 – Saturday May 8th
  Introduction to tidy data, the tidyverse, and dplyr
  - What is tidy data and why is it important?
  - Introduction to the tidyverse and key tidyverse R packages
  - Data manipulation and analysis with dplyr

We Will Cover How To…

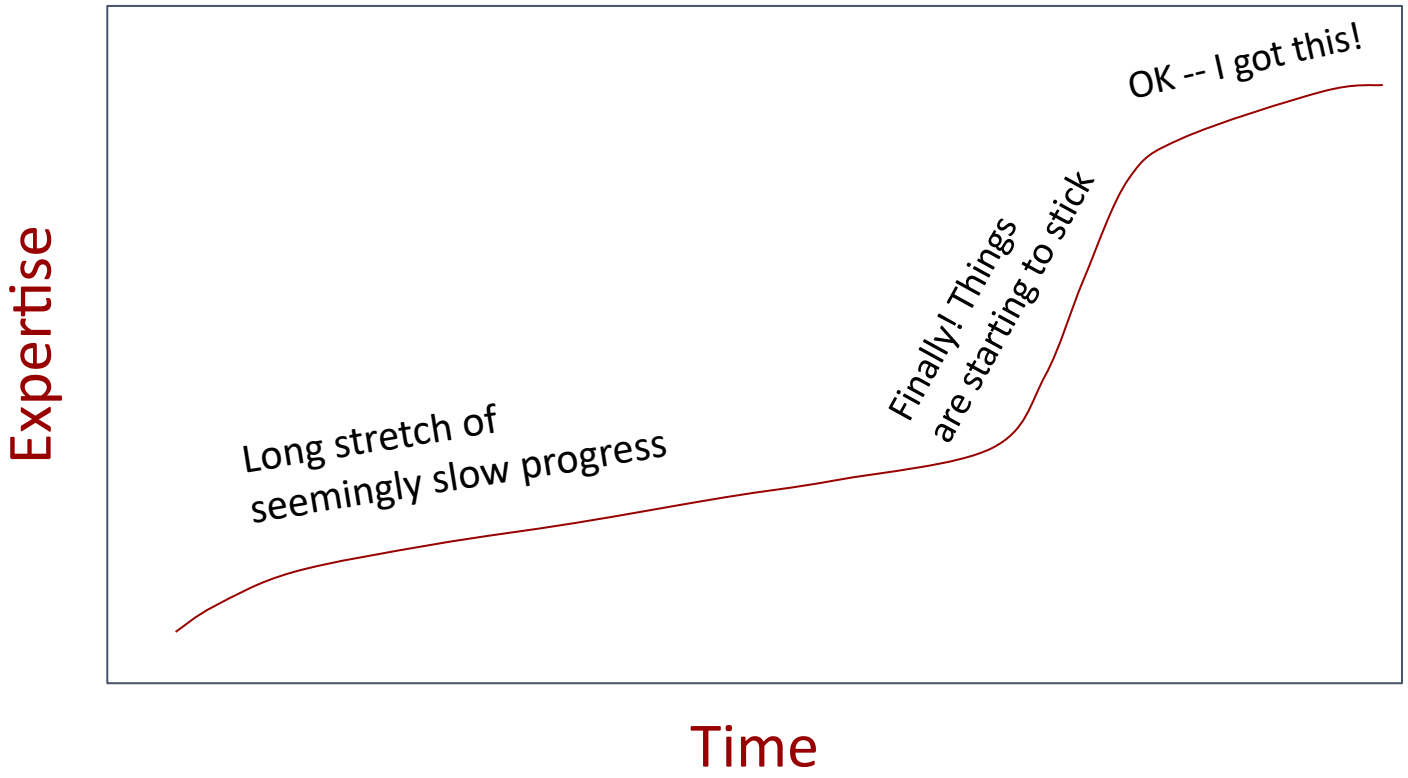
- start-up RStudio, make an RStudio project & write/execute code
- create variables and vectors, and perform calculations with them
- read a formatted text file of data into R
- understand basic properties about data tables
- tell someone what tidy data is and why it’s important
- perform basic data manipulations and operations on data tables

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...

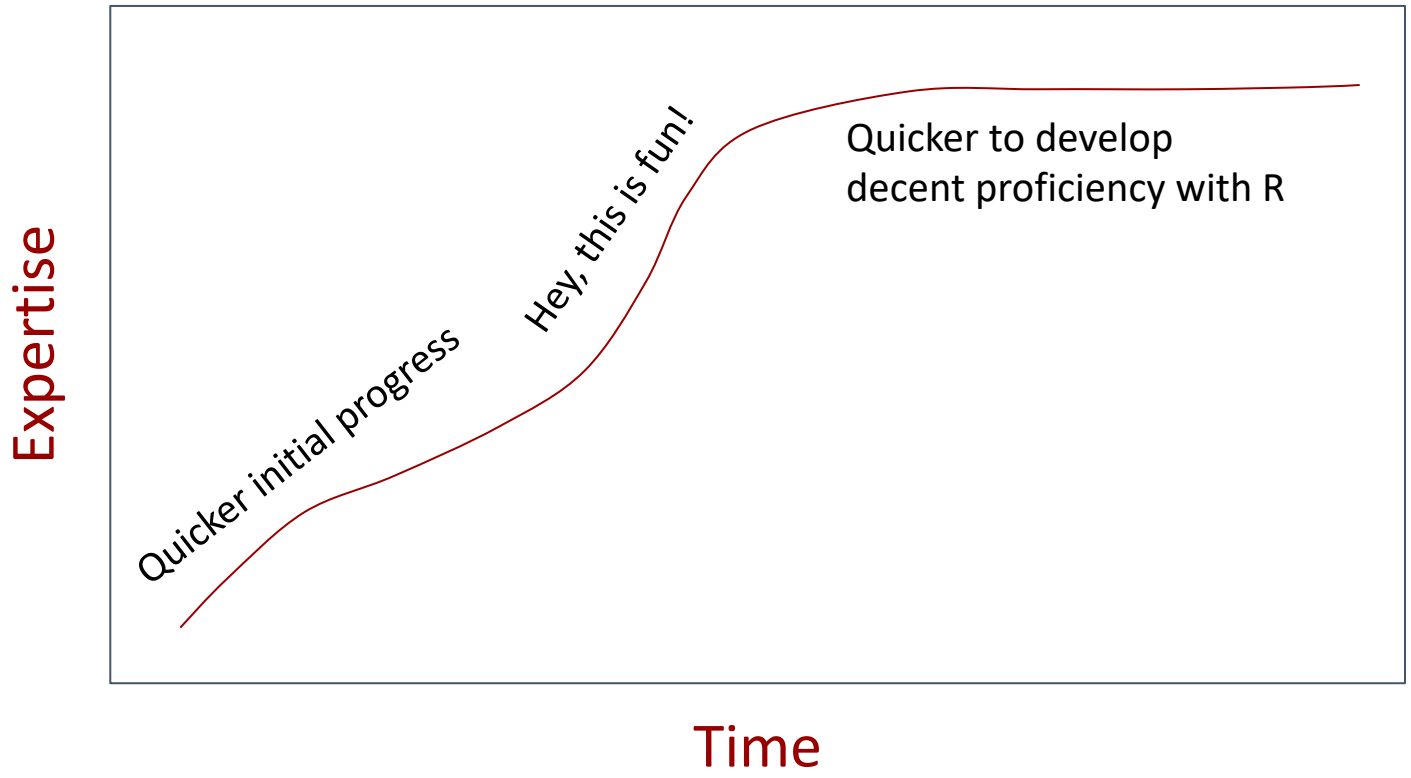


What made R challenging to learn 10 year ago?

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

The R Learning Curve

Today...



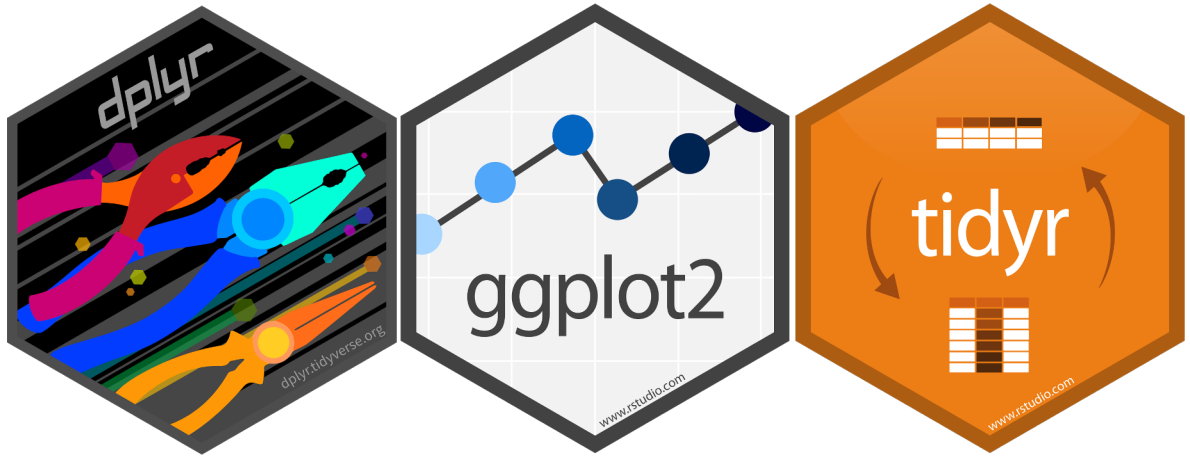
Why is R easier to learn today?

- 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

Logistics

- Lectures and exercises will be in the main Zoom room
- Ask questions any time in the Zoom chat – TAs will be monitoring
- If anyone is completely lost, TAs will create a separate Zoom room for more in-depth help
- Course materials are located here: https://computationalproteomics.khoury.northeastern.edu/
- Also, referenced on GitHub at: https://github.com/ZenBrayn/mayinstitute2021-intro-to-r-for-beginners

Working with RStudio

May Institute 2021: Introduction to R for Beginners

Day 1 – Module 1

Goals of this module

- Learn about RStudio and why it's an important tool for R coding
- Learn how to type and execute R code within RStudio
- Learn about RStudio Projects and how to use them for your own work

Introduction to RStudio

RStudio:
An IDE Where Write & Execute R Code

- IDE = Integrated Development Environment

- RStudio ≠ R
  - R is the programming language & supporting software environment
  - RStudio is a supporting software program that helps you work with R

- You don't need RStudio to use R, but it does make it easier

- RStudio (the IDE) is developed by RStudio (the company)

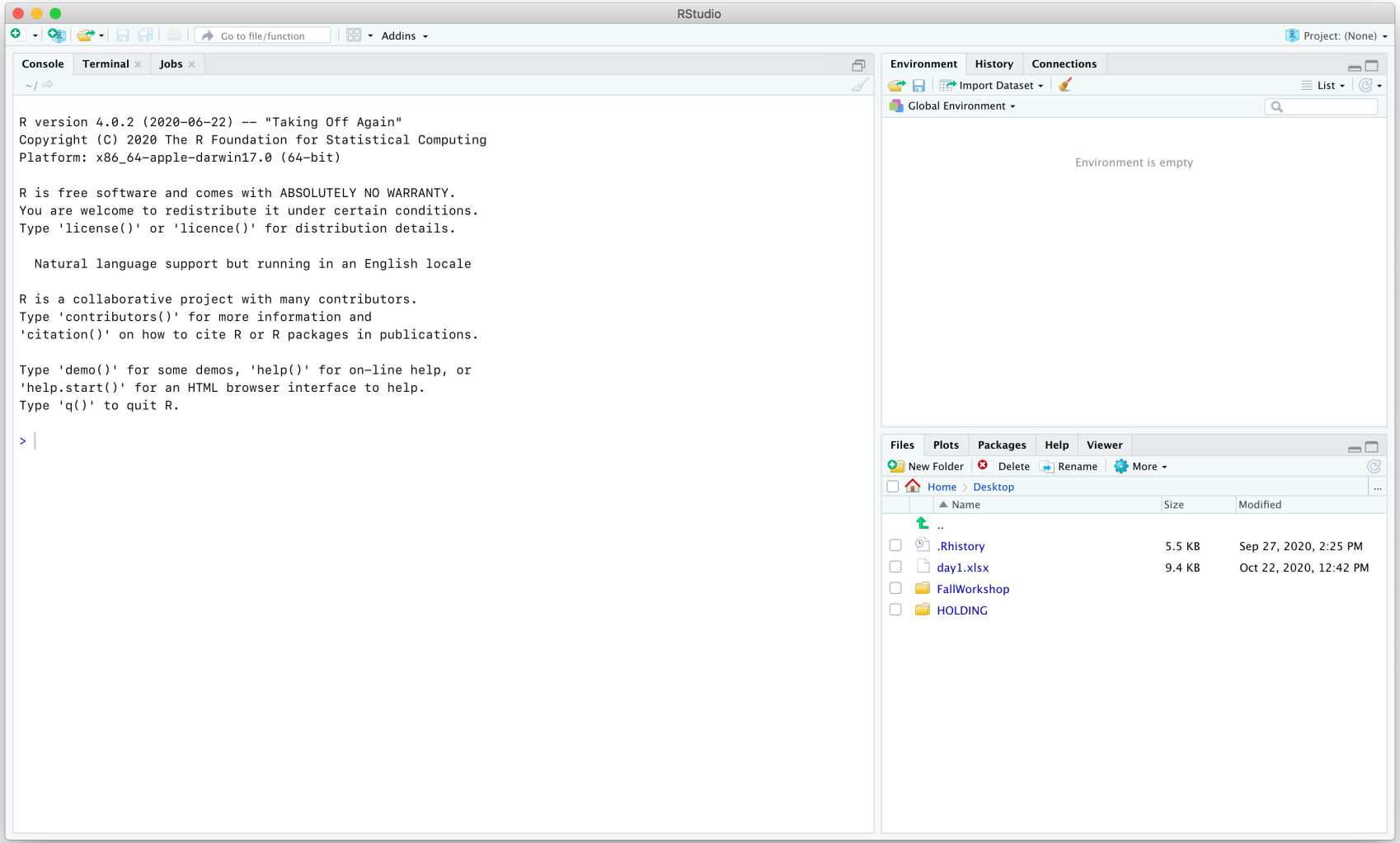
R vs. RStudio vs. RStudio Cloud

- 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

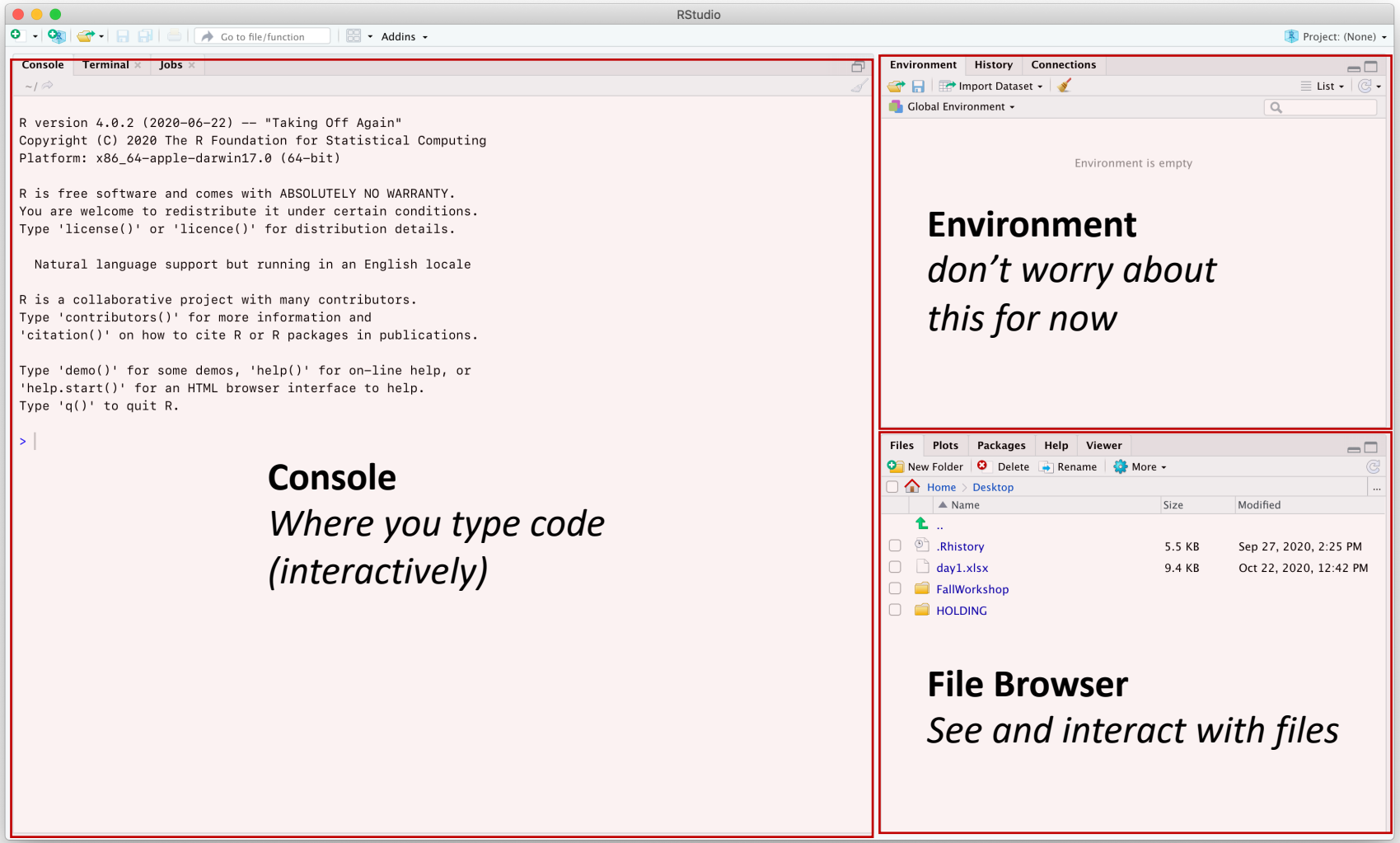
- RStudio is an integrated development environment (IDE) that makes it easier to work with R and R code
- Has free and paid versions
- You install RStudio after you install R – it won't on its own work
- RStudio is like a car's body/interior

- 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

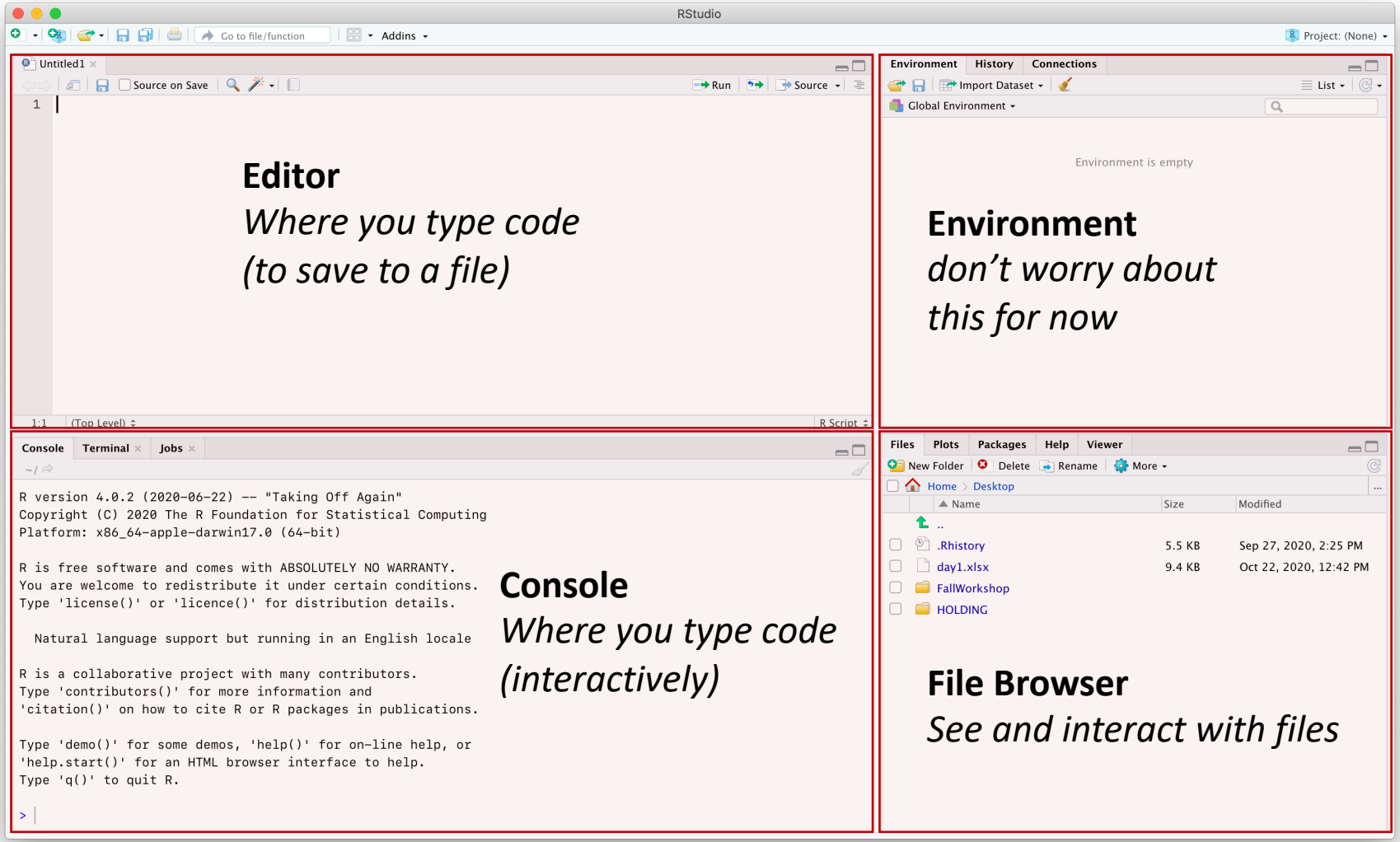
RStudio When you First Open It



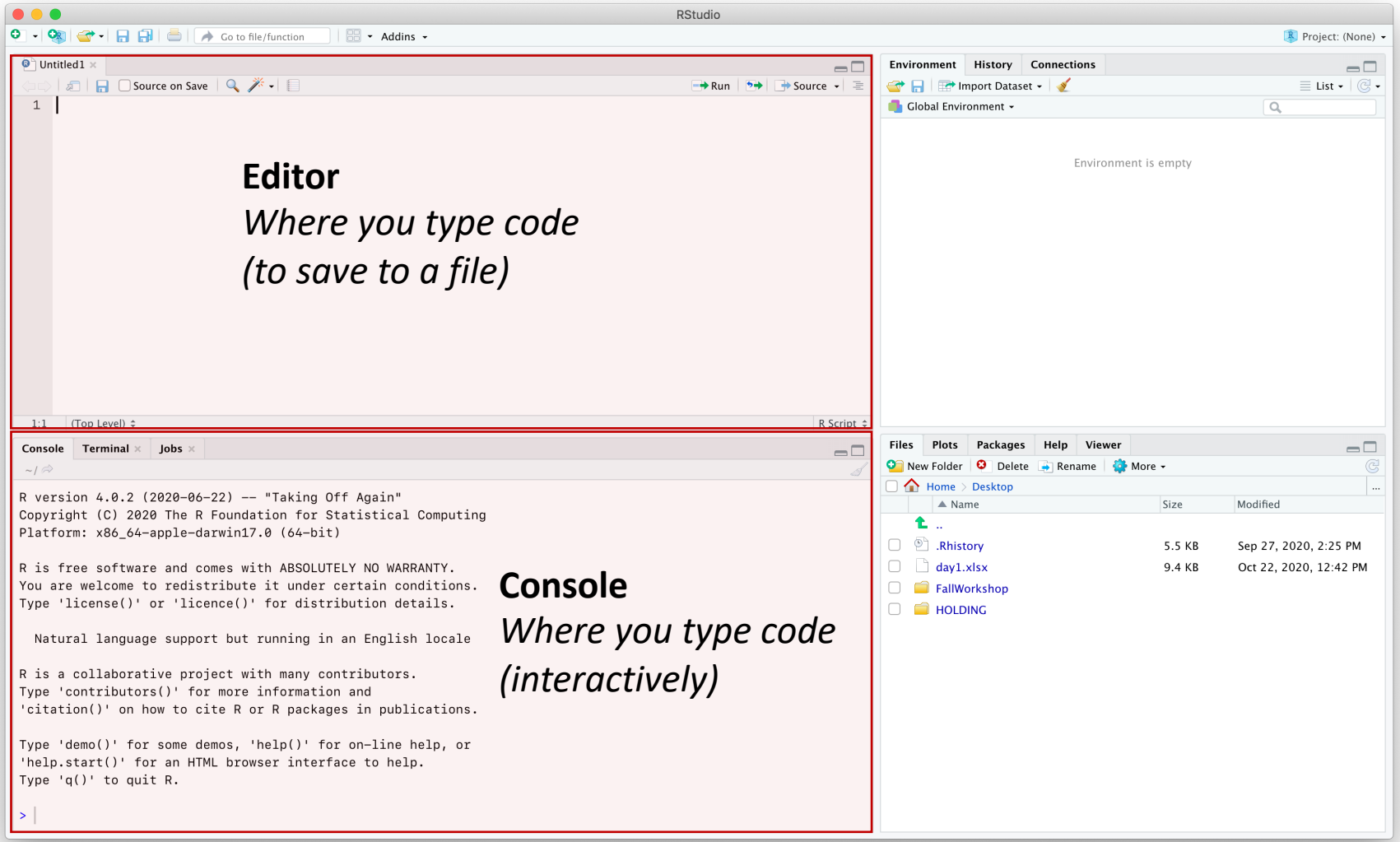
RStudio When you First Open It



File → New File → R Script



First Focus on The Editor and Console



Practice Time – Getting Started with RStudio

1. Open RStudio or RStudio Cloud

2. What's the main panel on the left called?

3. Create a new R script
   File → New File → Rscript

4. What happened? What's the name of the new panel that appeared?

5. Which panel is the Console and which is the Editor? How do you know?

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, might 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

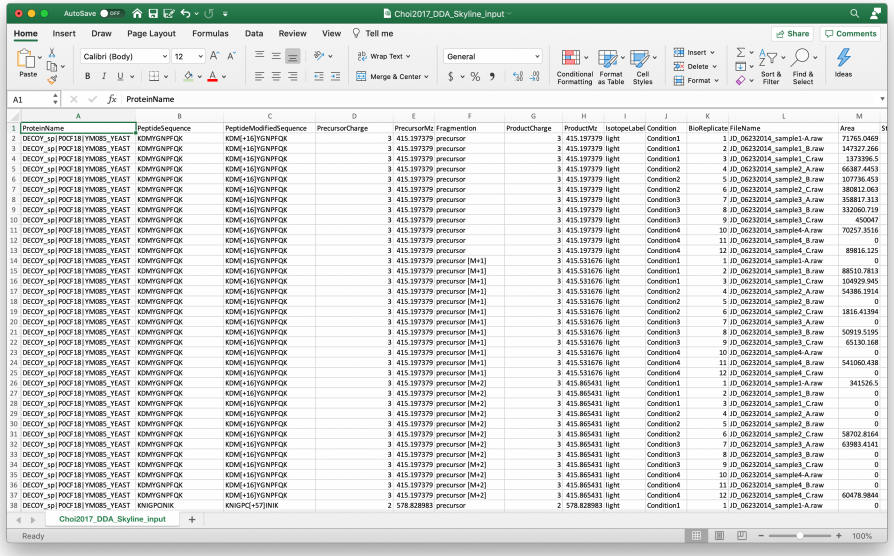
RStudio Can do So Much More!

- Keyboard shortcuts, debugger, code snippets
- Integration with version control systems (e.g. Git)
- Streamlines
  - package development
  - report generation (RMarkdown)
  - web app development (Shiny)
- Continuously updated to support and facilitate data analysis workflows

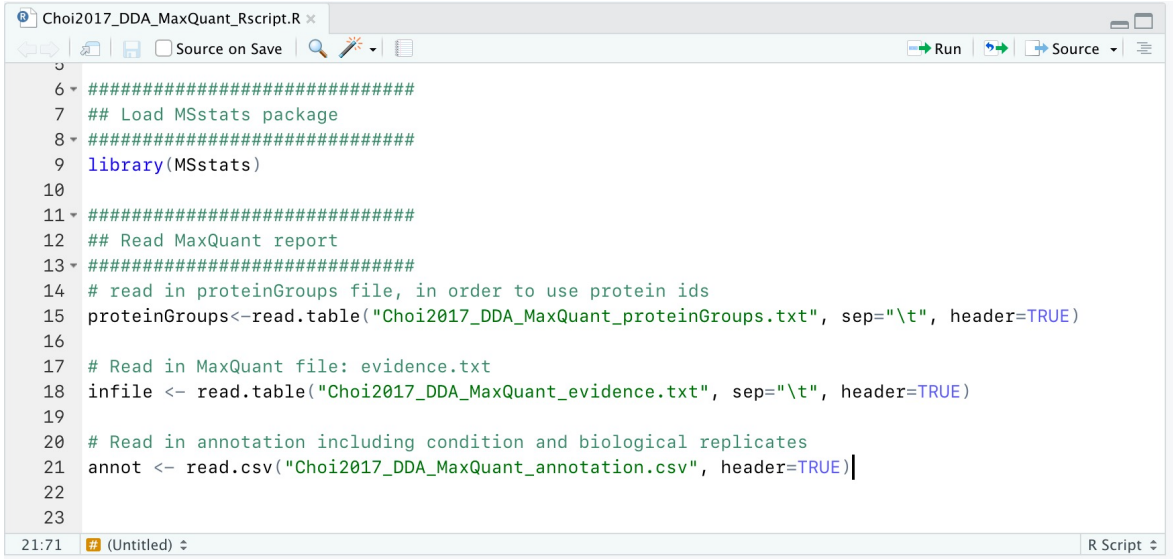
Basic R Coding in RStudio

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!

RStudio Tips

- Console
  - Press the *Up Arrow* to recall previously executed expressions
  - You can use standard text editor keyboard commands for text selection
  - Press ESC to abort an expression

- Editor
  - Lines of code are numbered on the left side
  - You can use standard text editor keyboard commands for text selection
  - CMD/CTRL-RETURN will execute the line of code the cursor is on, and the cursor will jump to the next line after execution

Practice Time – Typing & Executing Code

1. Create a new R script –
   the Console and Editor should both be visible

2. Type and execute several math expression in the Console

3. Type and execute several math expressions in the Editor

4. How are these two modes of interaction similar and different?

RStudio Projects

RStudio Projects Streamline Working with R

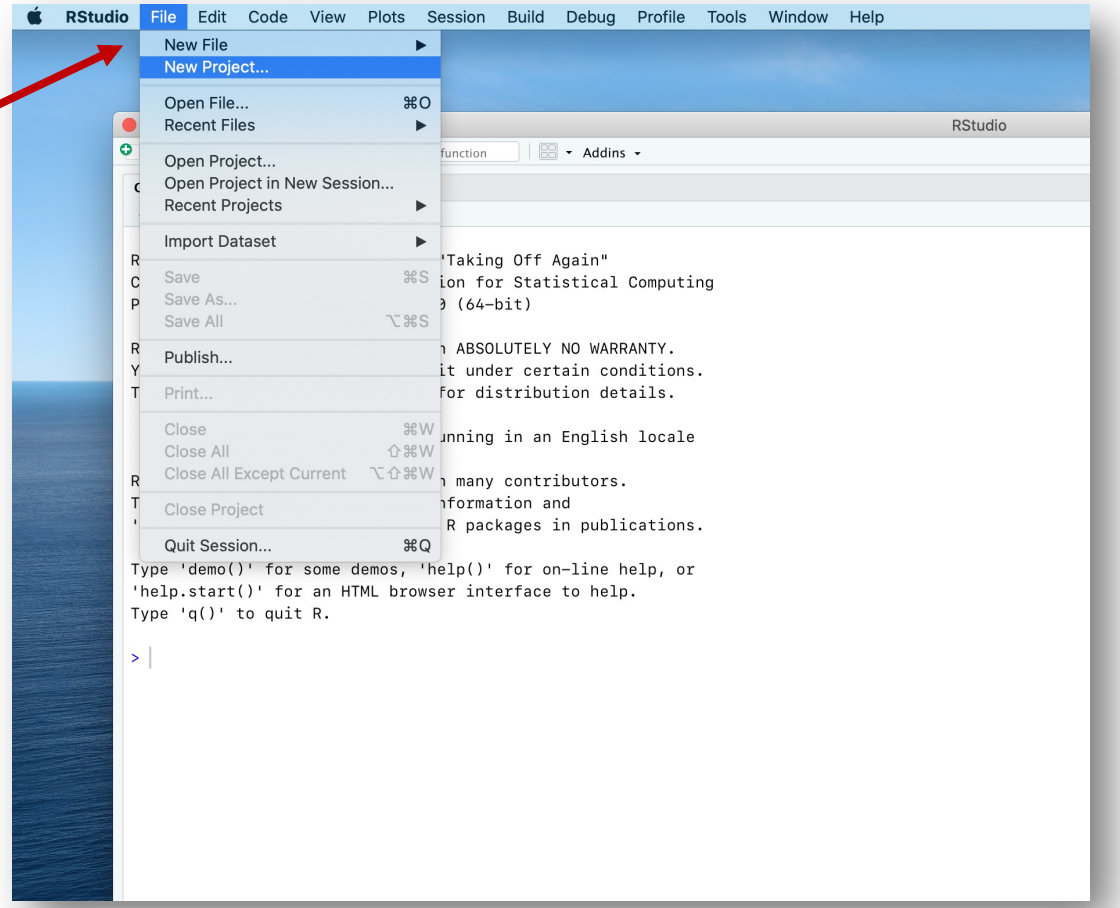
- RStudio project are special directories that organize your work and make it easier to work with R
- Dealing with file paths can be confusing/frustrating/cumbersome
- RStudio projects helps remove some of this friction by automatically setting your working directory to the project directory

Overview of Working with RStudio Projects

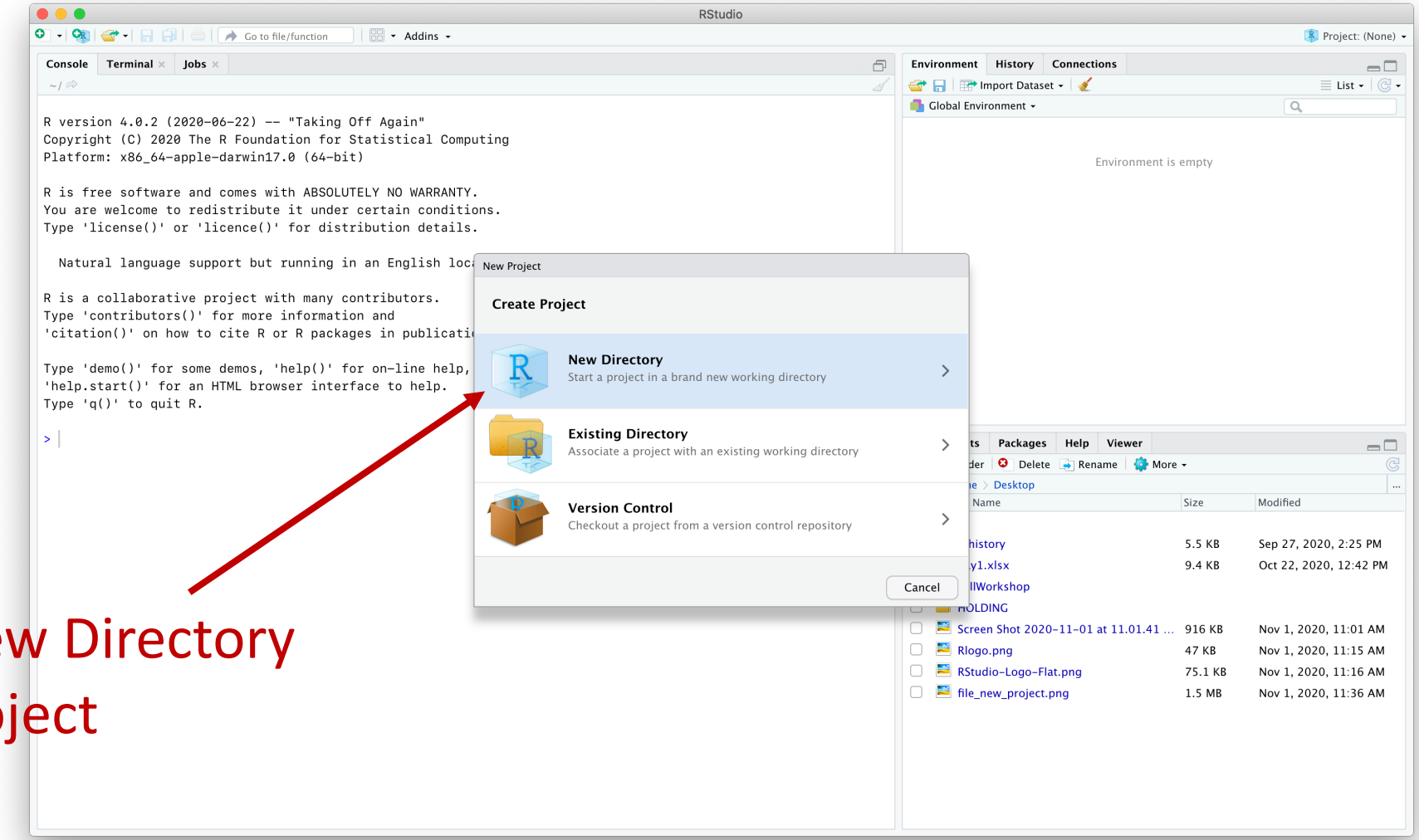
- Place all your project files inside the RStudio project directory
- You can access your files (i.e. data) directly by name (don’t need to specify a full file path)
- You can open your project by double-clicking the .Rproj file (more on this soon)
- If you need to move the project directory somewhere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

Creating an RStudio Project

Start the process from the File menu

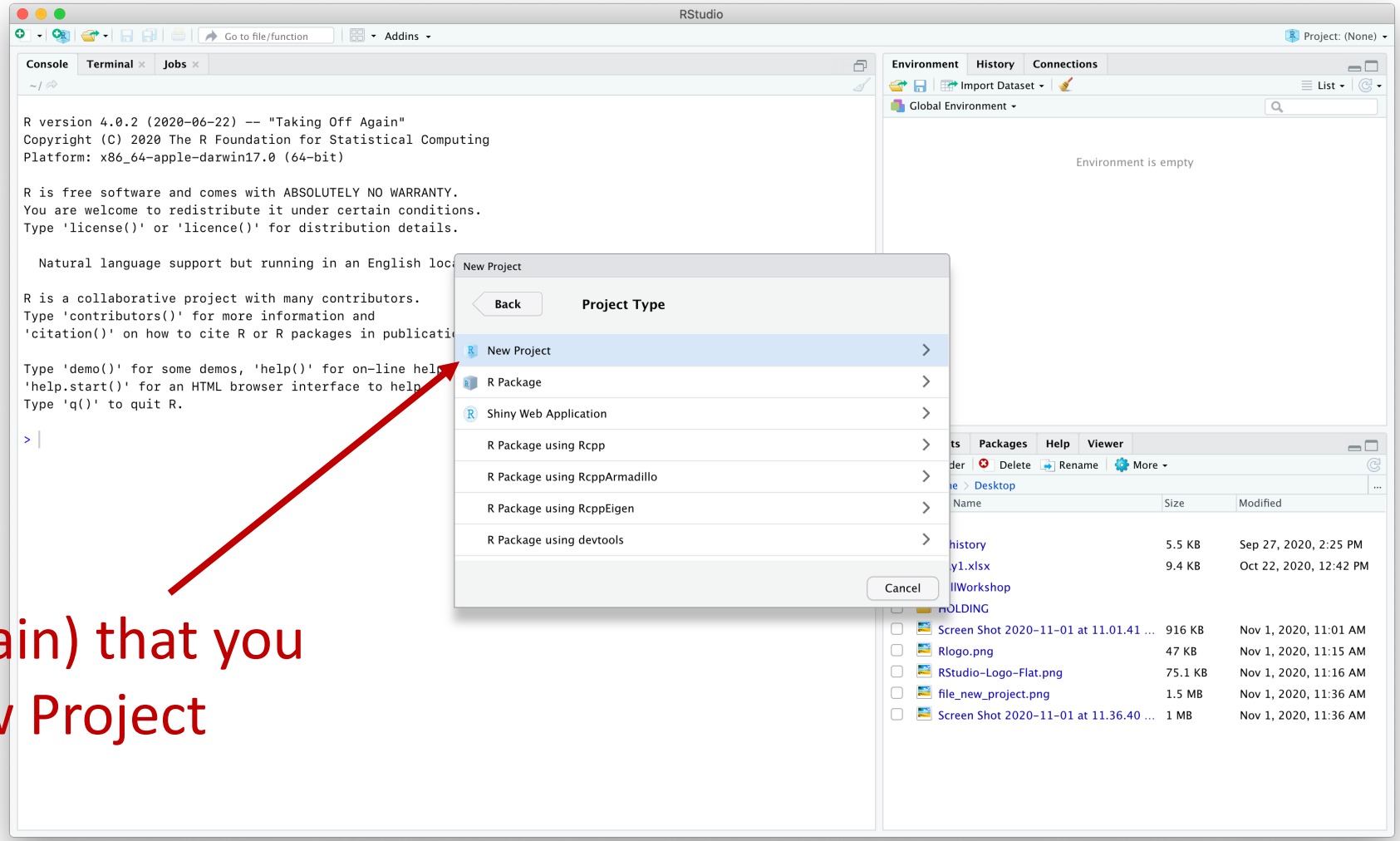


Creating an RStudio Project



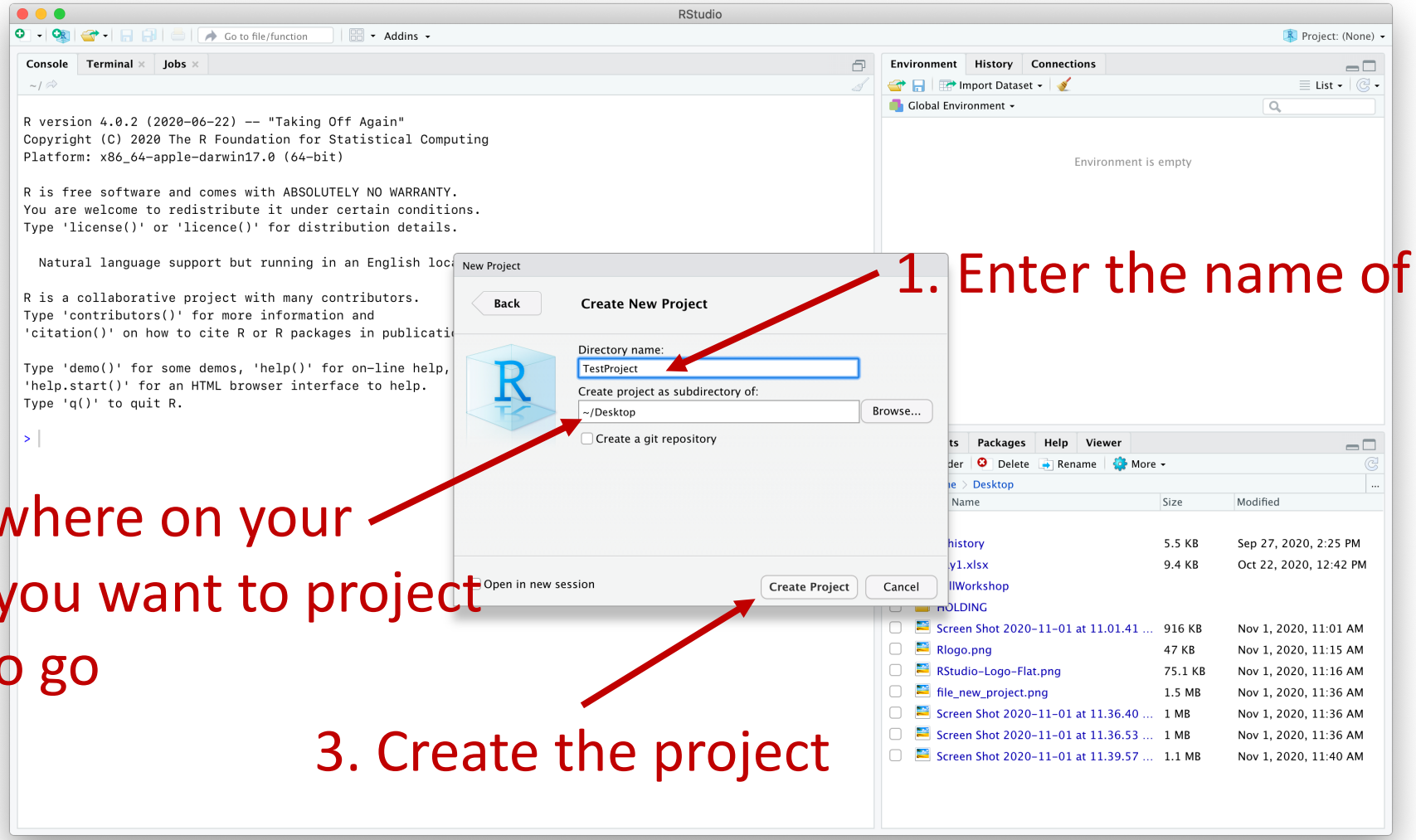
Make a New Directory for the project

Creating an RStudio Project

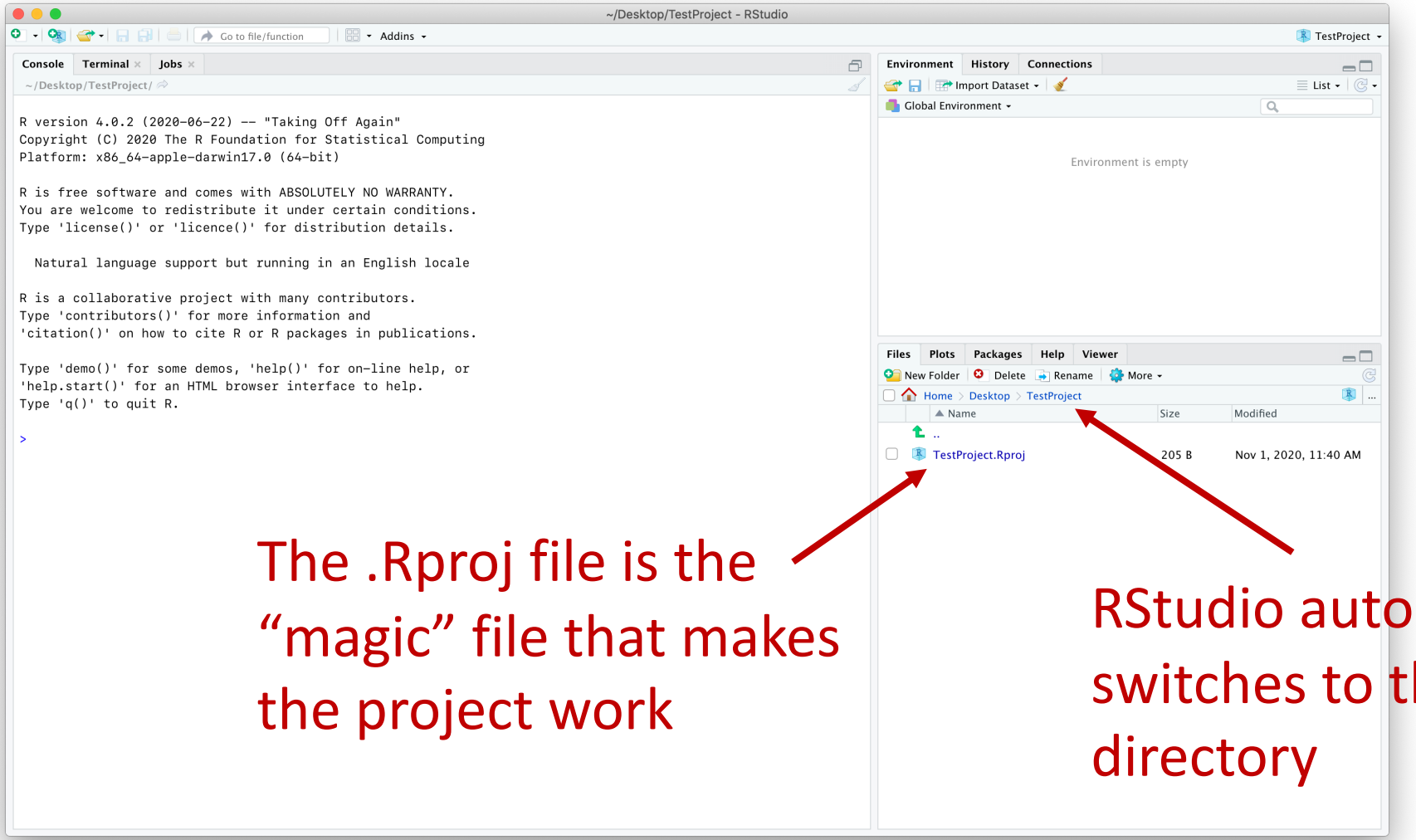


Specify (again) that you want a New Project

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

Practice Time – Working with RStudio Projects

1. Start RStudio (the desktop version)
2. Create a new RStudio Project
    1. “New Directory” → “New Project”
    2. Pay attention to the path 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 (you can copy and paste this code)
```
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

May Institute 2021: Introduction to R for Beginners

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

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 objects & 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 be matched up correctly
- 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

Variables & Variable Assignment

Variables Make Code More Understandable

- Variables allow you to store data using names

```
my_var <- 10
```

- R uses a very odd symbol for variable assignment: `<-`
  - two characters: "less than" followed by "minus"
  - It's good coding practice to put spaces around the `<-`
- Variable benefits
  - Well named variables give context to the data they hold
  - Referring to data through variables is more flexible, makes it easier to update
  - Help reduce bugs and makes code easier to maintain

Naming Variables is Up To You

- You are mostly free to use what ever names you'd like (but see rules below)
- Naming things well can be challenging
  - Too long: hard/cumbersome to type
  - Too short: not enough information or unclear meaning
  - Too general/not relevant: uninformative or misleading
- Variable naming rules
  - can only contain letters, numbers, underscores _ , or periods .
  - No spaces!
  - must start with letter or period
  - cannot start with an underscore, number, or a period followed-up by a number

Variable Naming 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?

Practice Time – Assigning Variables

1. In the Console, type the code that creates a new variable called `exp1_time` with the value `10` assigned to it, and press return
2. What happened after you pressed return? What happens if you now type the name of the variable (only… no assignment) and press return?
3. Create a new variable called `exp2_time` with the value of `20`
4. Add the two variables together and save the result in a new variable called `sum_exp_time`. What's the value of this new variable?
5. Re-assign `exp1_time` to 30. What do you think the value of `sum_exp_time` is now?

Vectors

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 can be named in different ways
- 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

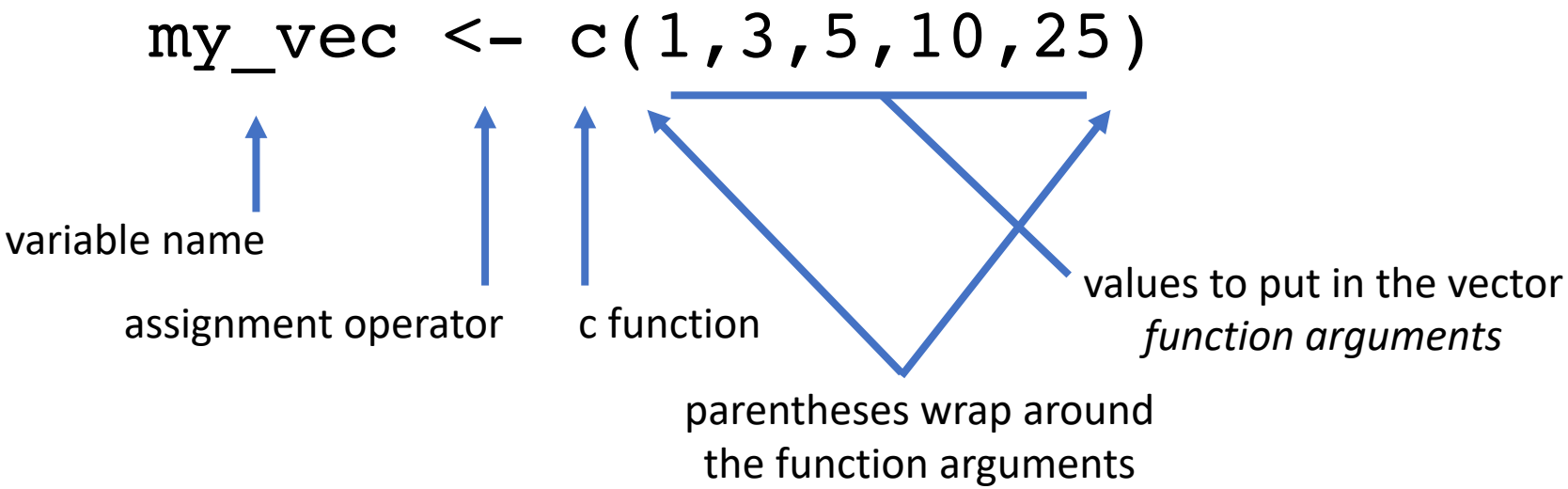
- Vectors hold ordered data, e.g. a set of numbers
- Vectors can be used anytime you have a set of values that you want to keep together
- You could represent a mass spectrum with two vectors:

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

Creating Vectors

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



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 given 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`

Practice Time – Math with Vectors

1. In the Console, create a new vector with the values 1 through 5 and assign it a variable (pick any name you'd like).

2. Run the code examples from the previous slide in the Console and observe the output. Try re-typing the code rather than coping/pasting

3. Compared to the last example, what happens when you run the following code?
```
my_vec + c(1, 2, 3, 4)
```

Functions

Functions are stored 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`
get the unique elements	`unique`
mean	`mean`
standard deviation	`sd`

Examples: Working with Vectors

Code	Output
`my_vec <- c(30, 40, 10, 50, 20, 20)`	*assigns the give vector to the variable* `my_vec`
`length(my_vec)`	`6`
`min(my_vec)`	`10`
`mean(my_vec)`	`28.33333`
`sort(my_vec)`	`10 20 20 30 40 50`
`unique(my_vec)`	`30 40 10 50 20`

Practice Time – Calling Functions

1. In the Console, create a new vector with the values 1 through 5 and assign it a variable (pick any name you'd like).

2. How can you confirm your vector has 5 elements? Use a function to get the answer.

3. Try misspelling the name of the function you used in 2. and executing the expression. What's the error message?

4. Try leaving off the ending) symbol in the code used in 2. What happened? Did you get an error or something else?
   Hint: try pressing the ESC key to "reset" things

Conditional Expression

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 given 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*

Practice Time – Using Conditional Expressions

1. In the Console, run the examples from the previous slide and pay attention to the outputs
2. Why does the following example return multiple values?
   `my_vec > 20`
   What do you think the computer is doing here?
3. What's the difference between
   `my_val == 122` and `my_val = 122`
   Try running these two expressions to understand the differences

Subsetting Vectors

Subsetting Vectors

- It’s often useful to get or exclude specific values from a vector
- The *subset* operator allows 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
  (many other 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`

Practice Time – Subsetting Vectors

1. In the Console, run the examples from the previous slide and pay attention to the outputs
2. How can you predict the number of outputs you expect to get from a subsetting operation?
3. Take a close look at the last example. Can you say in words what the computer is doing?
4. If you were to flip the > sign to < in the last example, can you guess the output? Run the code and see if your guess was correct.

Recap

- Coding syntax are the rules that dictate whether 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
  - vector subsetting

Reading Data Files into R

May Institute 2021: Introduction to R for Beginners

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 and some proprietary MS data formats (e.g. mzML)
- You can 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

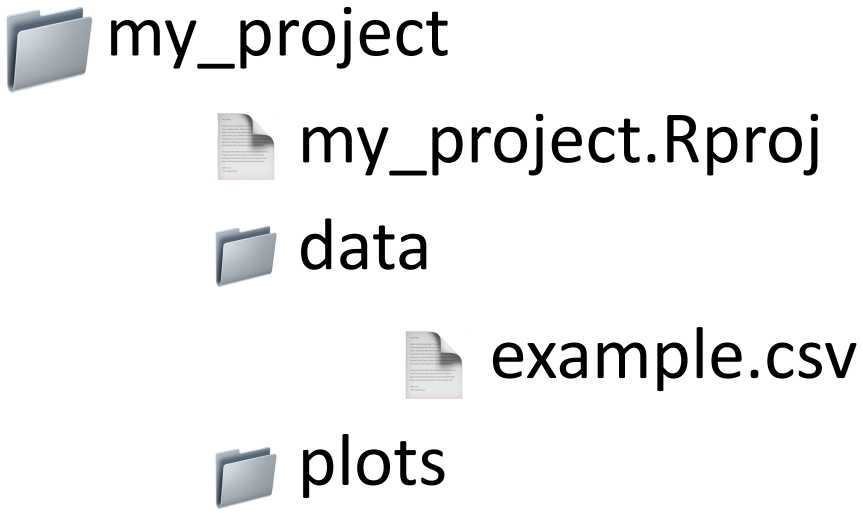
```
# load the tidyverse package
# will also load the readr package
library(tidyverse)

# 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* be the header row
  - lists the names of the columns
  - can contain spaces, but better if not
- The following rows contain 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 usually 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)

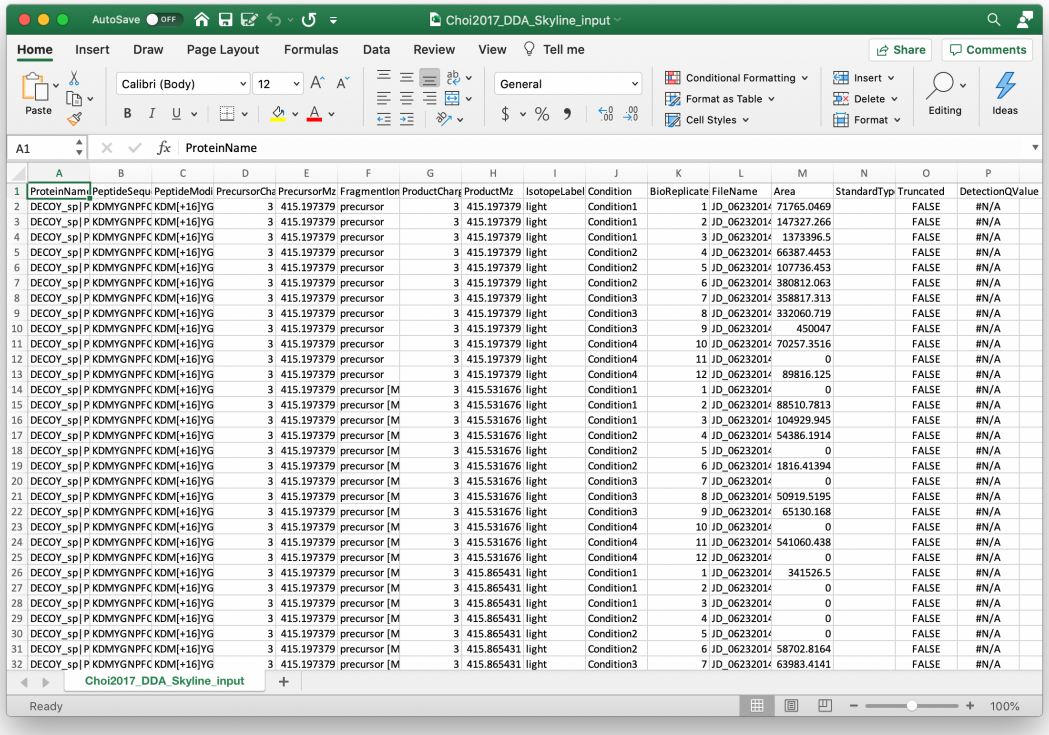
May Institute 2021: Introduction to R for Beginners

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

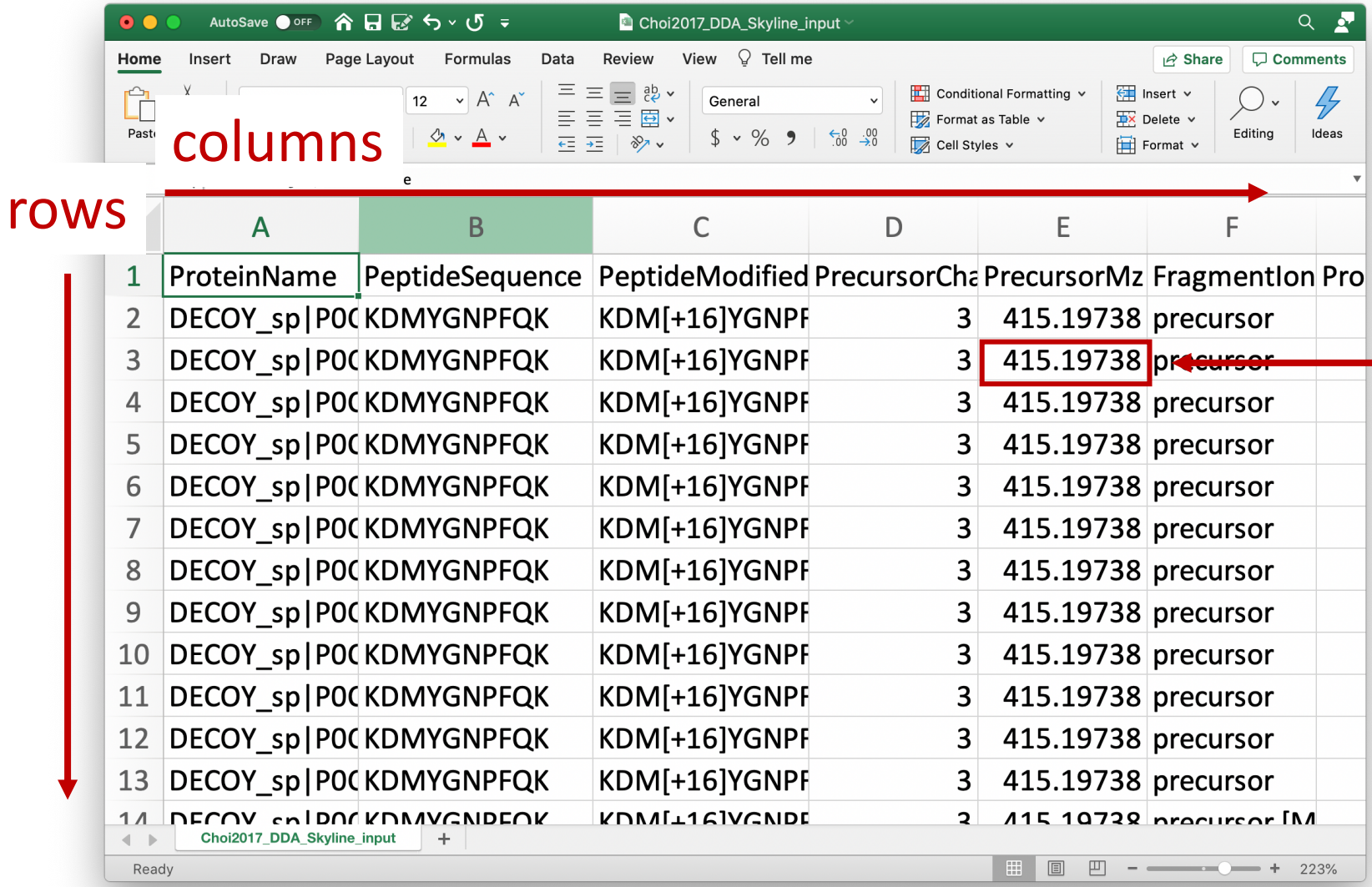
csv files represent tabular data

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

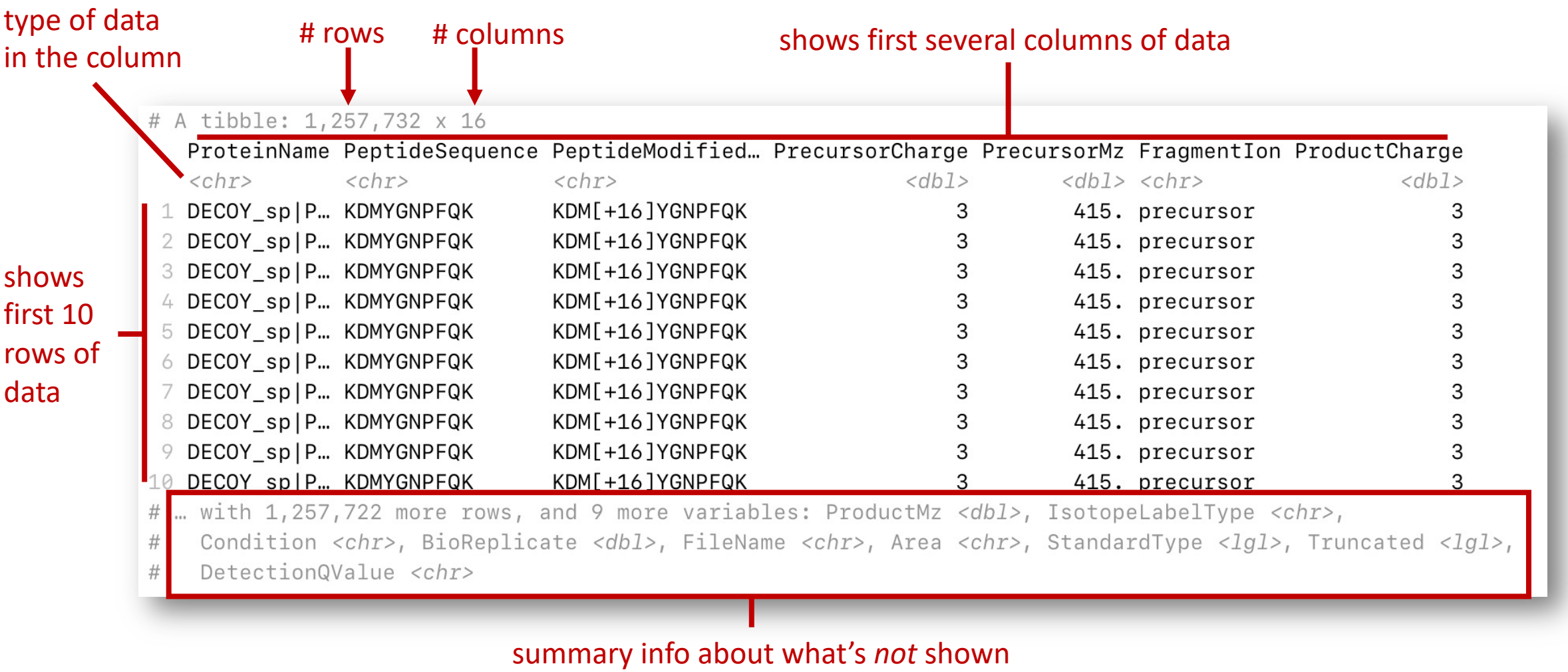


	A	B	C	D	E	F
1	ProteinName	PeptideSequence	PeptideModified	PrecursorCha	PrecursorMz	FragmentIon
2	DECOY_sp|P0C	KDMYGNPFQK	KDM[+16]YGNPF	3	415.19738	precursor
3	DECOY_sp|P0C	KDMYGNPFQK	KDM[+16]YGNPF	3	415.19738	precursor
4	DECOY_sp|P0C	KDMYGNPFQK	KDM[+16]YGNPF	3	415.19738	precursor
5	DECOY_sp|P0C	KDMYGNPFQK	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 influenced 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 Time – Working with Data Frames

1. Use the following code to read the exercise data set:
```
library(tidyverse)
dat <- read_csv("Choi2017_DDA_Skyline_annotation.csv")
```
2. Type the variable name and press return to see a print-out of the data table. Study the output to see what information it provides.
3. How many rows does the data set have? How many columns? What are the variable names? Use functions to answer these questions.
4. Get all of the BioReplicate values from the data table and store them in a new variable. What kind of data structure is this?
   Hint: we reviewed this in Module 2
5. Using code, get the maximum of the BioReplicate values

Subsetting data frames

- Getting data subsets (specific columns or rows) is a very common data manipulation operation
- `$` 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

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

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

- A vector of integers (for each column)
- A vector of column name
  (with quotes around the names)
- A conditional expression
  TRUE keeps a column
  FALSE rejects a column

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 Time – Subsetting Data Frames

1. Use the following code to read the exercise data set:
```
library(tidyverse)
dat <- read_csv("Choi2017_DDA_Skyline_annotation.csv")
```

2. Get the first 5 rows (only) and all of the columns of the data set.

3. Get all the rows and only the last two columns of the data set.
   Try using both integer subsetting and subsetting by name.

4. Get the rows that correspond to Condition4
   Hint: conditional expressions + subsetting

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 investigate, manipulate and transform the data
- Subsetting data frames is an important data operation, and can be accomplished using the $ and [,] operators