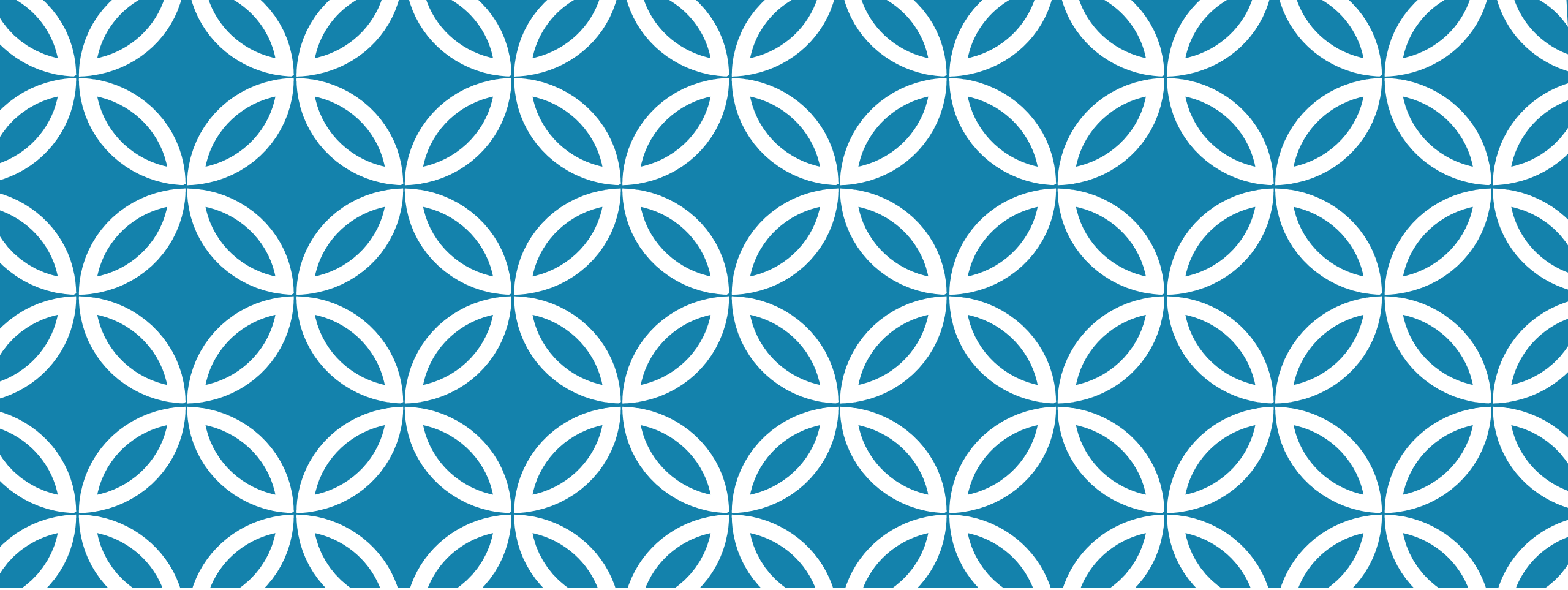


How to Truly "Excel" at Data Analysis and Visualization: An Introduction to the R Programming Language

Daniel Herman
Patrick Mathias
Joseph Rudolf

December 13, 2020

Course Introduction

Course Goals

1. Gain familiarity with the R programming language and RStudio software
2. Learn some fundamentals of coding using Tidyverse tools on a clinical data set
3. Develop workflows for reproducible data analysis using RMarkdown

December 13 2020	**Session**	**Instructor**
8:30 am - 8:50 am	Introduction to the Course	Daniel Herman Patrick Mathias Joseph Rudolf
9:00 am - 9:45 am	Introduction to R and RStudio	Joseph Rudolf
10:00 am - 10:45 am	Reproducible Reporting	Patrick Mathias
11:00 am - 11:00 am	Data Visualization	Joseph Rudolf
12:30 pm - 1:15 pm	Data Slicing	Patrick Mathias
1:30 pm - 2:15 pm	Data Transformation	Daniel Herman
2:30 pm - 3:15 pm	Statistical Analysis	Daniel Herman

Who are we?

Daniel Herman

Assistant Professor of Pathology and Laboratory Medicine

University of Pennsylvania Perelman School of Medicine

Director, Endocrinology Laboratory

Hospital of the University of Pennsylvania

Patrick Mathias

Assistant Professor, Department of Laboratory Medicine

University of Washington School of Medicine

Associate Medical Director, Laboratory Medicine Informatics

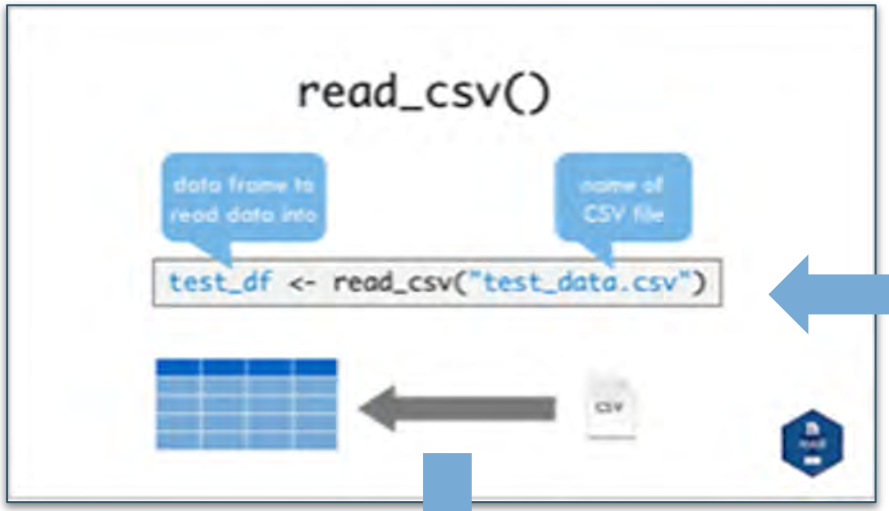
Joseph Rudolf

Assistant Professor, Department of Pathology, University of Utah Medical School

Medical Director, Automated Core Laboratory, ARUP Laboratories

Workshop Workflow

Sessions

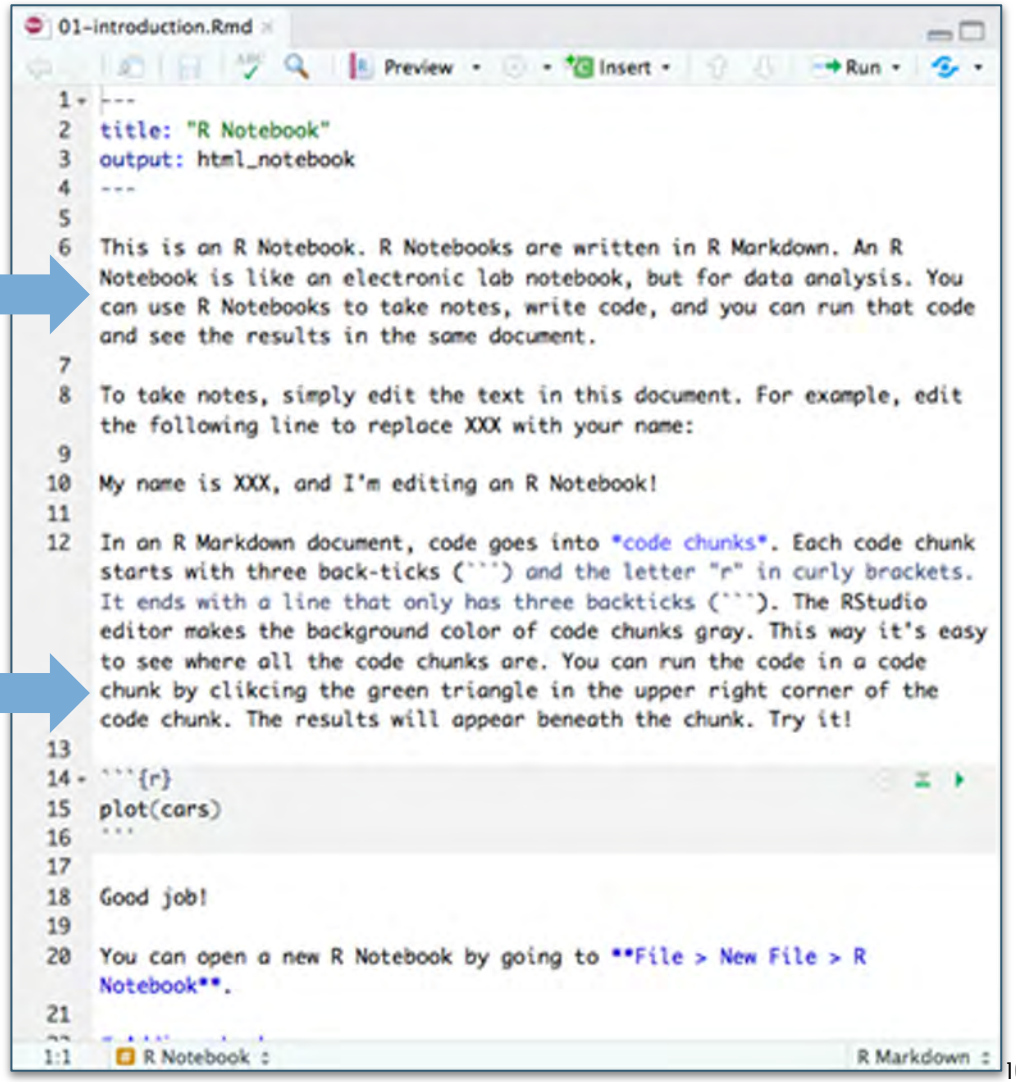


Your Turn

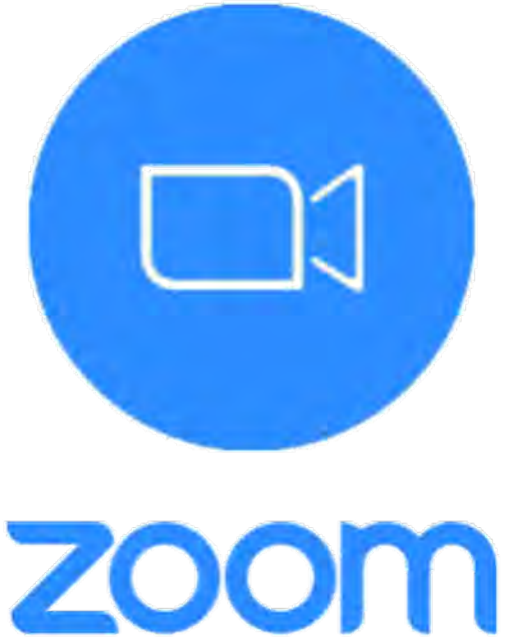
Introduce yourself to your neighbors

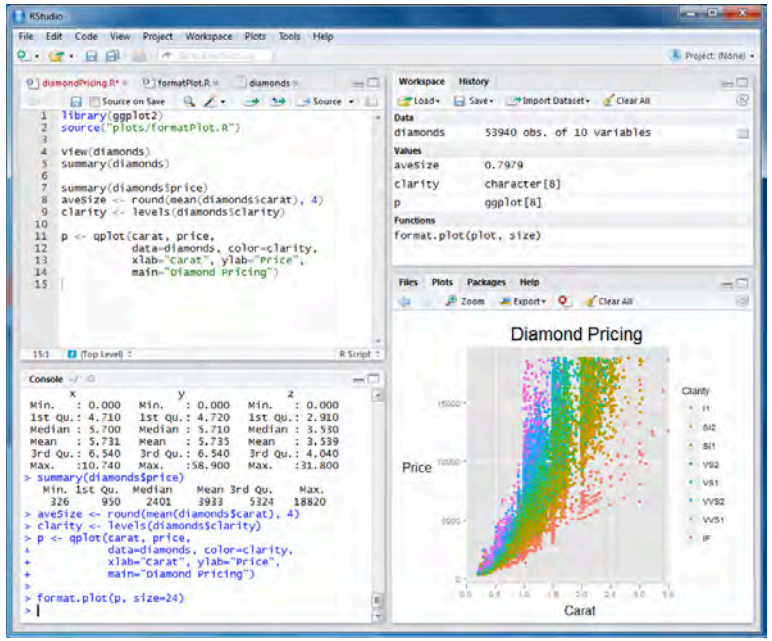
- Who are you?
- Where are you from?
- What do you do with data?
- Have you ever used R?

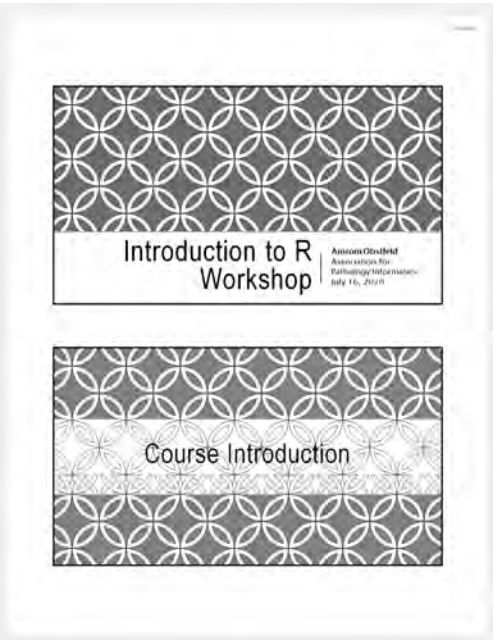
3:00



Configuring Your Setup

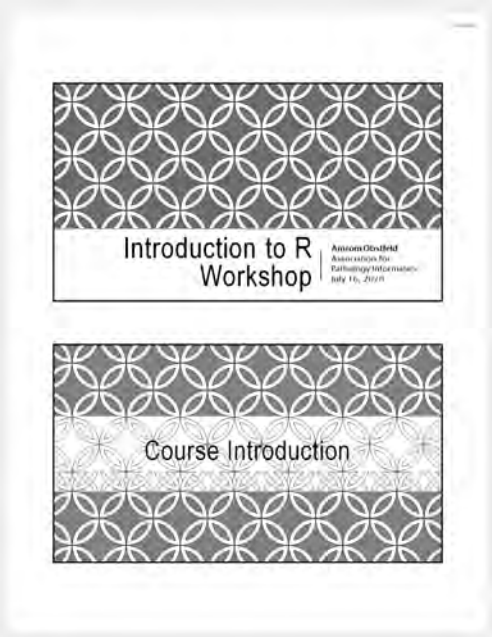




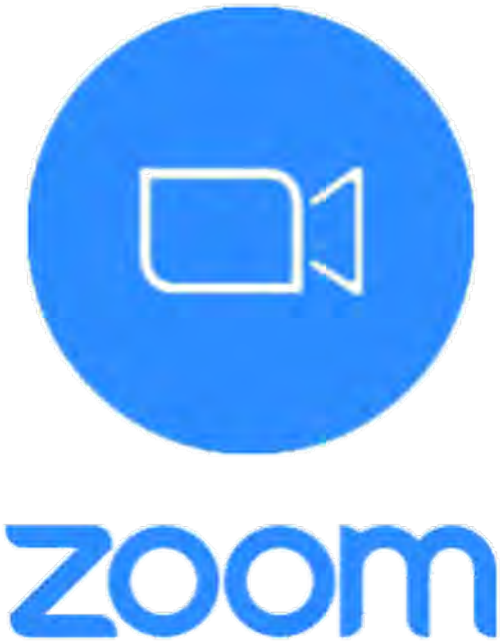


Workshop Coursebook

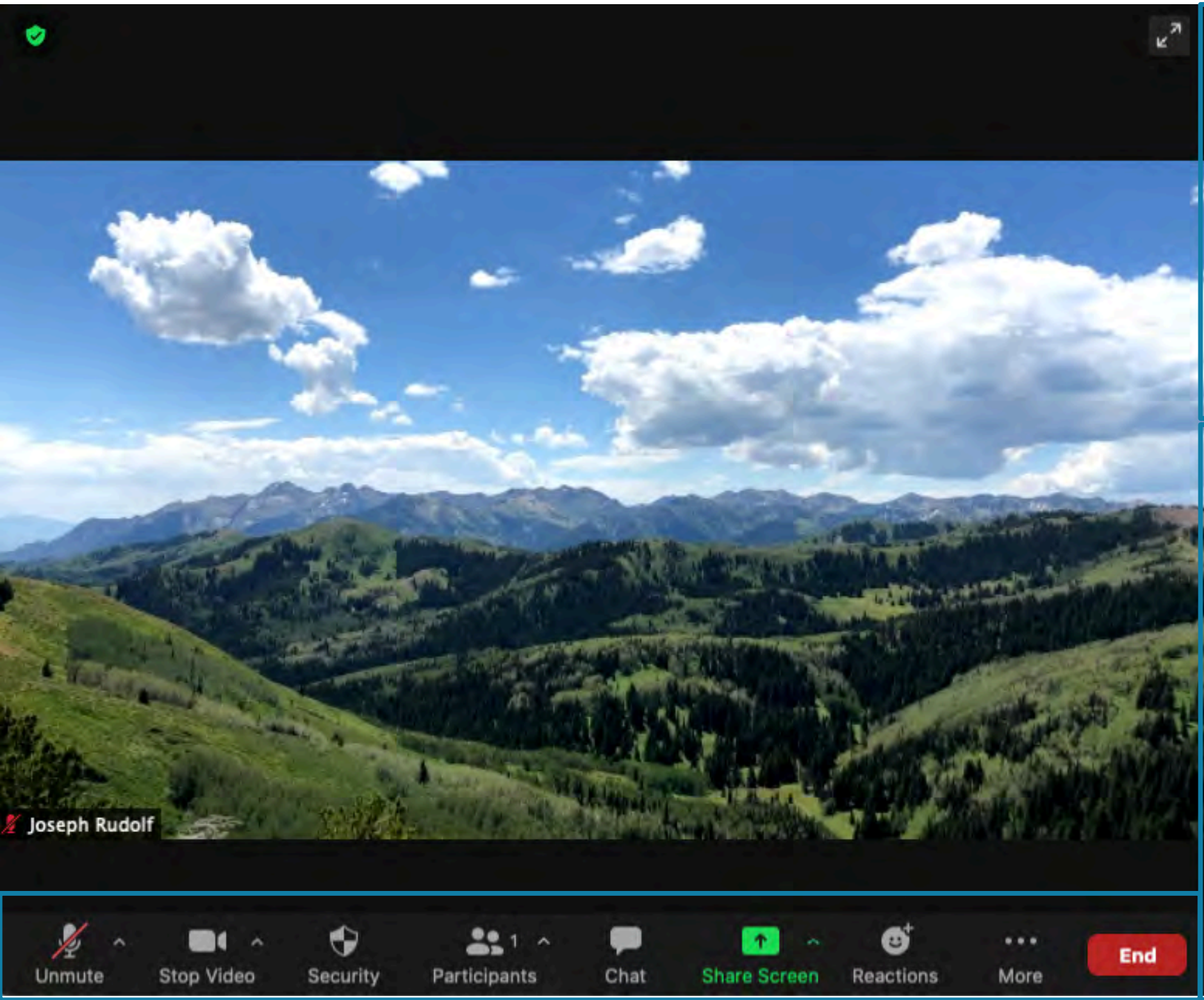
- Print out of all slides
- Appendix
  - Cheat Sheets
  - Useful resources



Using Zoom



- Participants muted
- Chat window
- Non-verbal feedback



Participants (1)

JR Joseph Rudolf (Host, me)

yes no go slower go faster more clear all

Invite Mute All More

Chat

To: Everyone File

Type message here...

Getting Help

- During presentation – Enter your question in the chat, an instructor will respond to the group our direct message you as appropriate.

- Between sessions – Feel free to unmute yourself and converse with instructors.

Who are you?

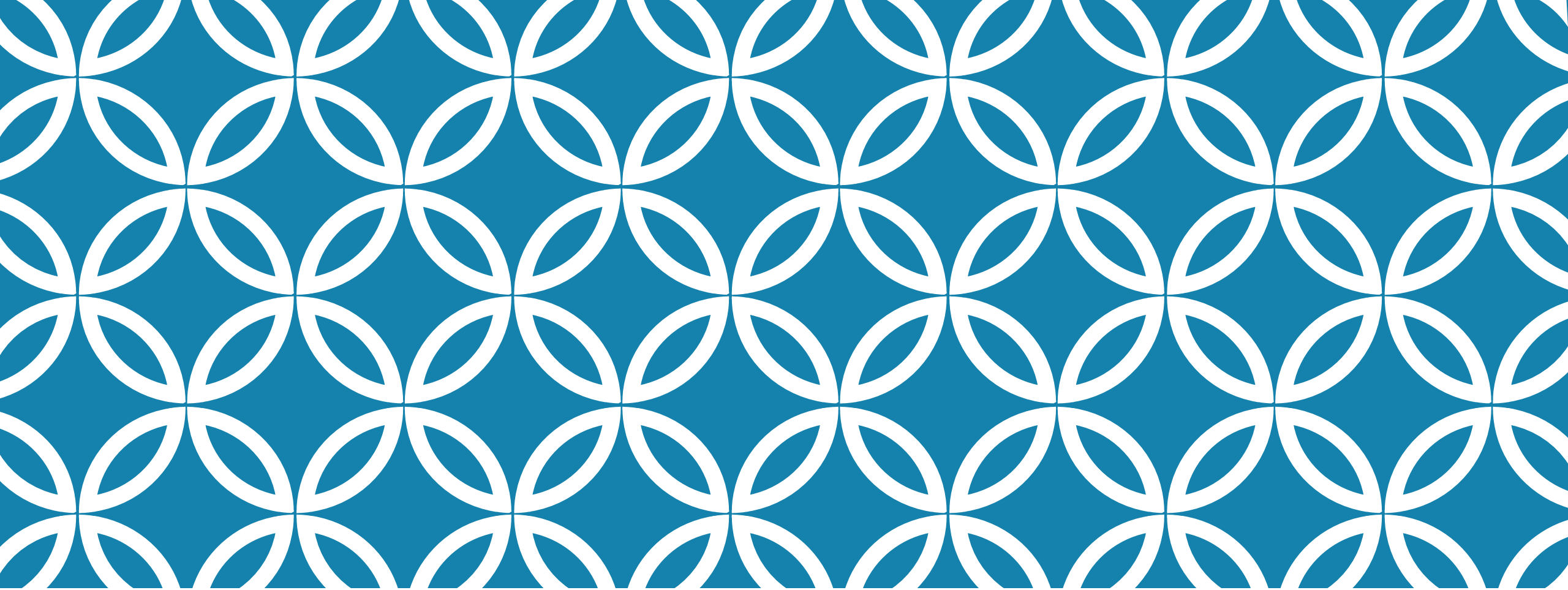
Your Turn

Where are you today?

Type your name and location in the chat.

Pre-Course Survey Results

Time for the Course!!



Introduction to R and R Studio

Session 1

Joseph Rudolf

December 13, 2020

Lesson Goals

1. Get oriented to R and RStudio
2. Learn some fundamentals of coding

Lesson Objectives

1. Log in and tour RStudio Cloud
2. Execute code at the console
3. Define and use functions
4. Define and create objects in the environment
5. Load data into R and interact with a dataframe

Getting Oriented to R

What is R?

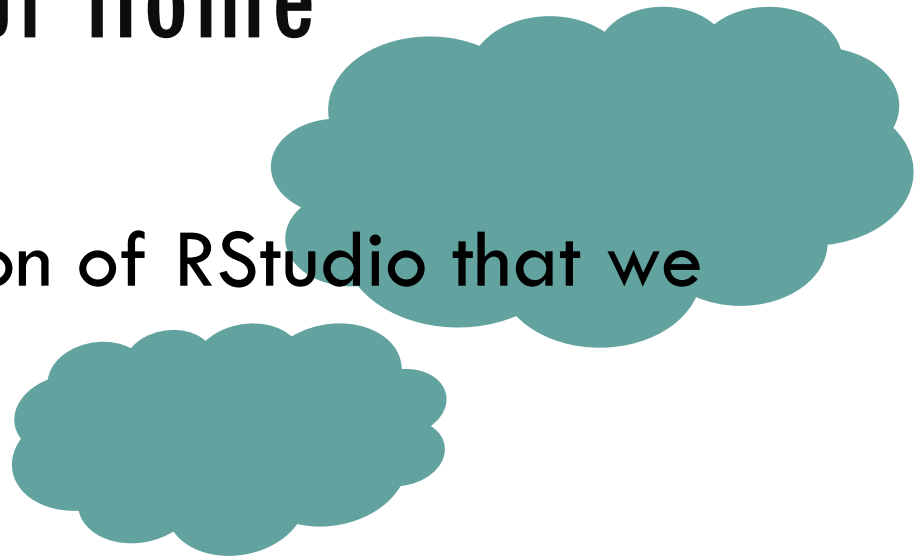
- R is a statistical programming language.
- Using R you can load, analyze, and visualize data.
- R also provides an environment in which we can conduct reproducible data analysis.
  - Documented
  - Revisable
  - Shareable

RStudio: The Portal to R

- RStudio is an integrated development environment (IDE)

- Using RStudio we can interact with the R programming language to:
  - Write and execute code interactively
  - View data
  - Debug and fix errors
  - Author our code

RStudio: In the Cloud… In Your Home



- RStudio Cloud: An online hosted version of RStudio that we will use for these course sessions

- RStudio Desktop: A locally installed version of RStudio that you will use when you get home to continue your learning

Note: Use Rstudio Cloud only for this course. Do not upload protected health information to the cloud!

Your Turn

Navigate to: **https://tinyurl.com/r-aacc-2020**

Enter your log in credentials

Join Space

Make a copy of the Core Exercises for yourself

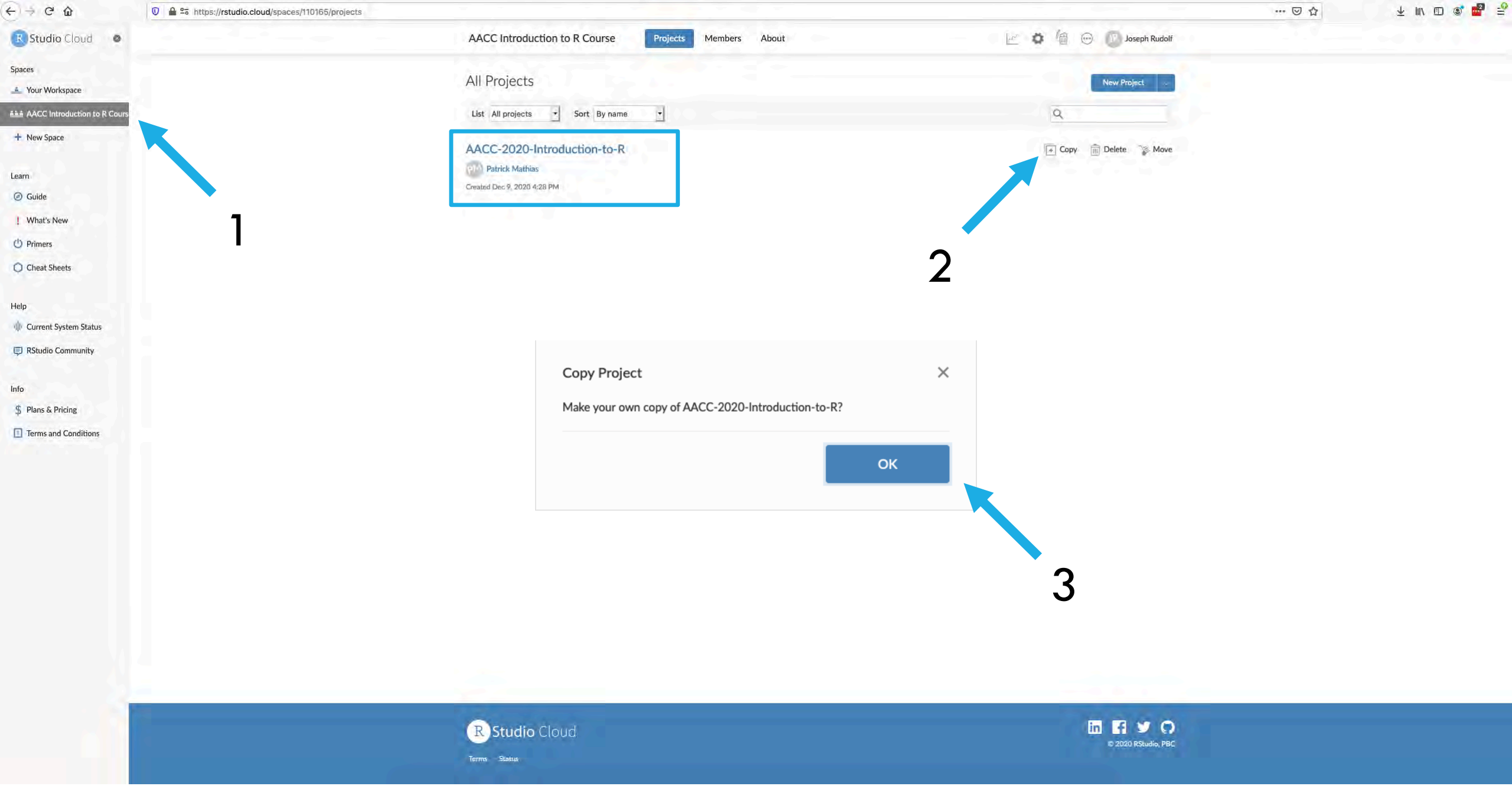
Join Space?

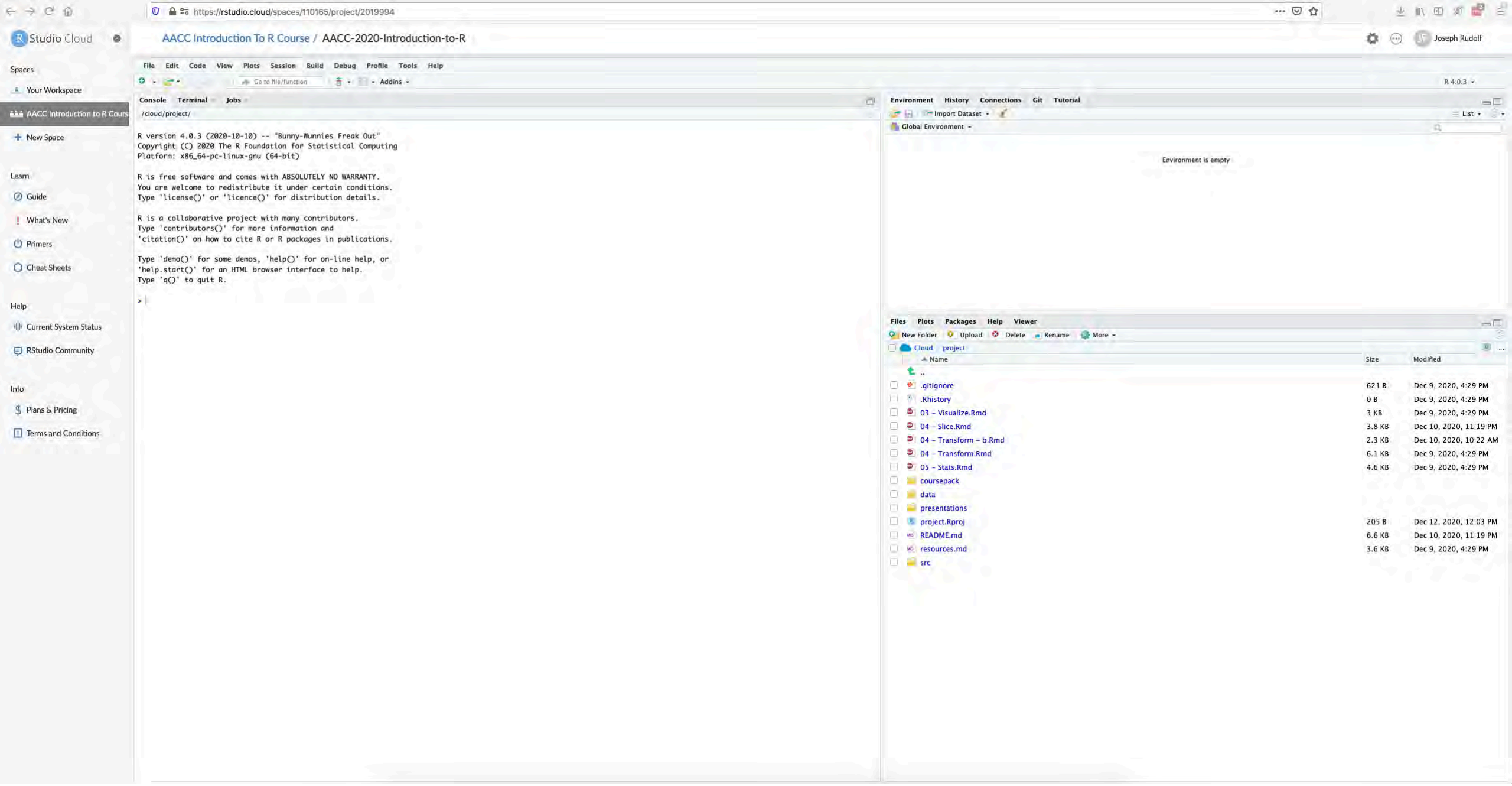
Joining a space gives you access to it and to its contents.

Once you join, admins will be able to see your email address.

Would you like to join this space?

Join Space | Cancel


https://rstudio.cloud/spaces/110165/projects
Studio Cloud
Spaces
Your Workspace
AACC Introduction to R Cours
New Space
Learn
Guide
What's New
Primers
Cheat Sheets
Help
Current System Status
RStudio Community
Info
Plans & Pricing
Terms and Conditions
AACC Introduction to R Course
Projects
Members
About
Joseph Rudolf
All Projects
New Project
List
All projects
Sort
By name
AACC-2020-Introduction-to-R
Patrick Mathias
Created Dec 9, 2020 4:28 PM
Copy
Delete
Move
1
2
Copy Project
Make your own copy of AACC-2020-Introduction-to-R?
OK
3
Studio Cloud
Terms
Status
© 2020 RStudio, PBC


https://rstudio.cloud/spaces/110165/project/2019994
Studio Cloud
AACC Introduction To R Course / AACC-2020-Introduction-to-R
Joseph Rudolf
Spaces
Your Workspace
AACC Introduction to R Cours
New Space
Learn
Guide
What's New
Primers
Cheat Sheets
Help
Current System Status
RStudio Community
Info
Plans & Pricing
Terms and Conditions
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
R 4.0.3
Console Terminal Jobs
/cloud/project/
R version 4.0.3 (2020-10-10) -- "Bunny-Wunnies Freak Out"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
>
Environment History Connections Git Tutorial
Import Dataset
List
Global Environment
Environment is empty
Files Plots Packages Help Viewer
New Folder Upload Delete Rename More
Cloud project
Name Size Modified
..
.gitignore 621 B Dec 9, 2020, 4:29 PM
.Rhistory 0 B Dec 9, 2020, 4:29 PM
03 – Visualize.Rmd 3 KB Dec 9, 2020, 4:29 PM
04 – Slice.Rmd 3.8 KB Dec 10, 2020, 11:19 PM
04 – Transform – b.Rmd 2.3 KB Dec 10, 2020, 10:22 AM
04 – Transform.Rmd 6.1 KB Dec 9, 2020, 4:29 PM
05 – Stats.Rmd 4.6 KB Dec 9, 2020, 4:29 PM
coursepack
data
presentations
project.Rproj 205 B Dec 12, 2020, 12:03 PM
README.md 6.6 KB Dec 10, 2020, 11:19 PM
resources.md 3.6 KB Dec 9, 2020, 4:29 PM
src

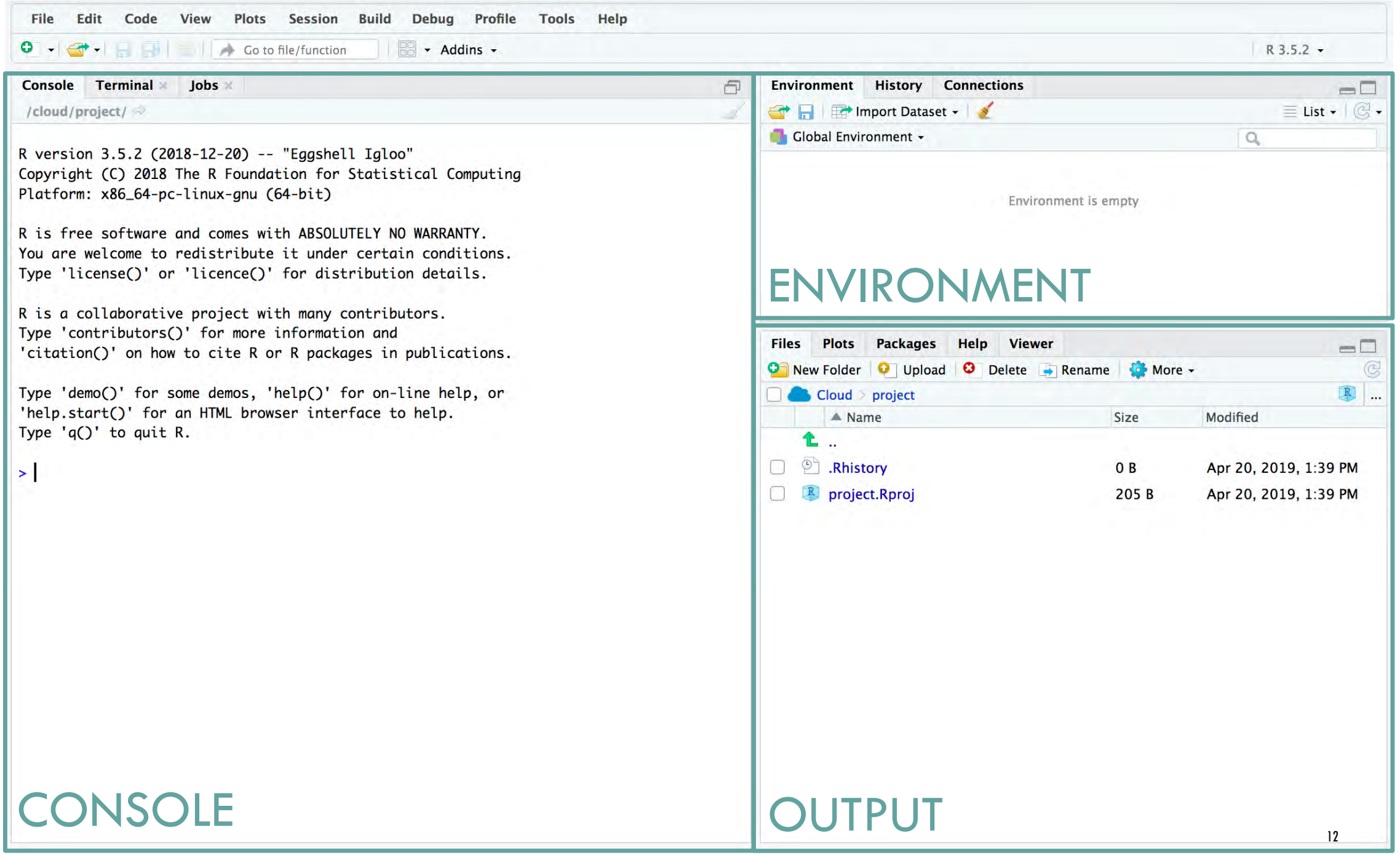
Your Turn

Navigate to: **https://tinyurl.com/r-aacc-2020**

Enter your log in credentials

Join Space

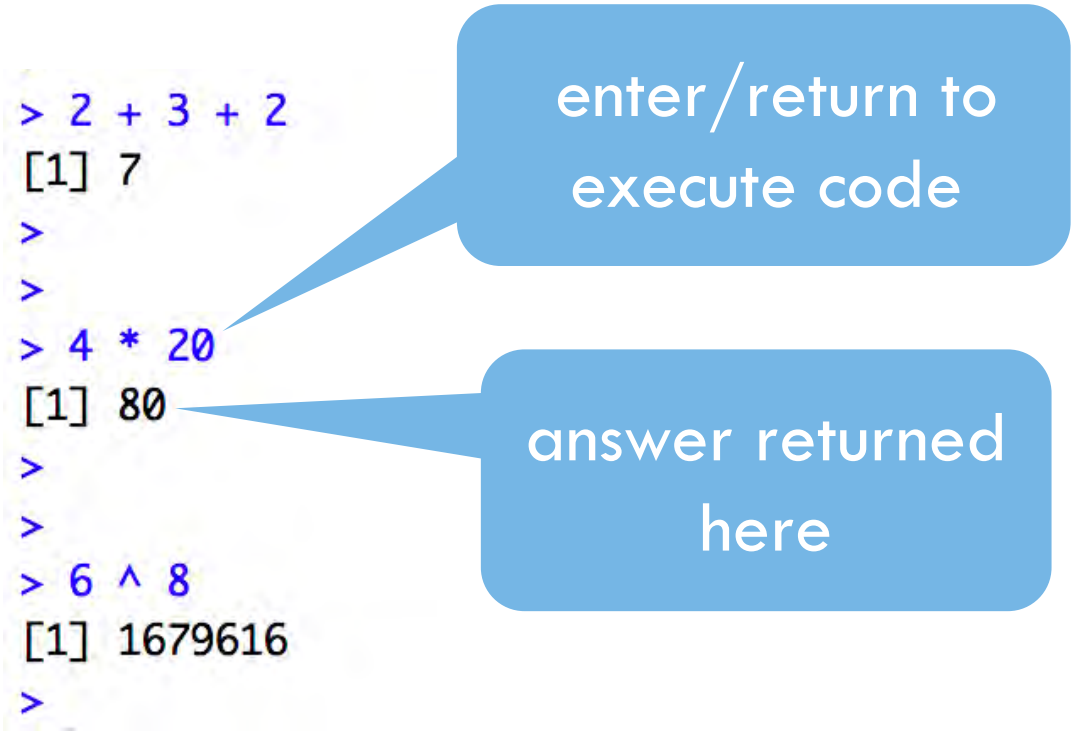
Make a copy of the Core Exercises for yourself


File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
R 3.5.2
Console Terminal Jobs
/cloud/project/
R version 3.5.2 (2018-12-20) -- "Eggshell Igloo"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
>
CONSOLE
Environment History Connections
Import Dataset
List
Global Environment
Environment is empty
ENVIRONMENT
Files Plots Packages Help Viewer
New Folder
Upload
Delete
Rename
More
Cloud
project
Name
Size
Modified
..
.Rhistory
0 B
Apr 20, 2019, 1:39 PM
project.Rproj
205 B
Apr 20, 2019, 1:39 PM
OUTPUT

The Basics of Coding

The Basics of Coding: Calculation

- R is a calculator!



Your Turn 1

Place your cursor at the console and click to enter the console.

Complete the following calculation:

-Take the integer 1974

-Subtract 12

-Multiply by 29

What did you get?

- A four digit number? A five digit number?

```
> 1974 - 12 * 29
[1] 1626
>
>
> (1974 - 12) * 29
[1] 56898
```

- Order of operations matters!

The Basics of Coding: Functions

- Code that extends our reach beyond the basic operators

```
> abs(-77)
[1] 77
>
```

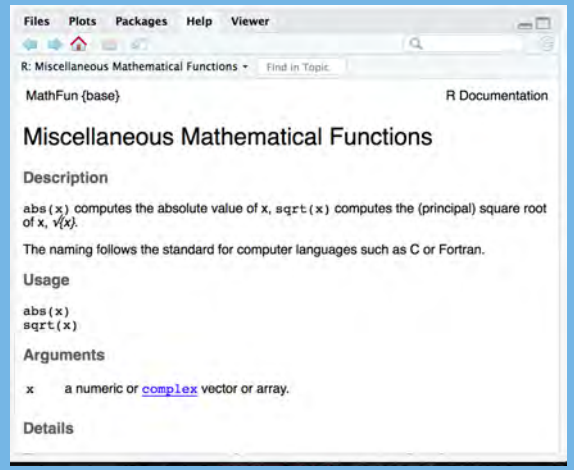
- What if I don't know what a function does?

```
>
> ?abs()
>
```

function (does stuff)

argument (input)

```
abs( -77 )
```



When you need more help

- The Internet (Stack Overflow: https://stackoverflow.com/)
- Work Aids (RStudio Cheat Sheets: https://www.rstudio.com/resources/cheatsheets/)
- A Good Book (R for Data Science: http://r4ds.had.co.nz/)

Putting Functions to Work

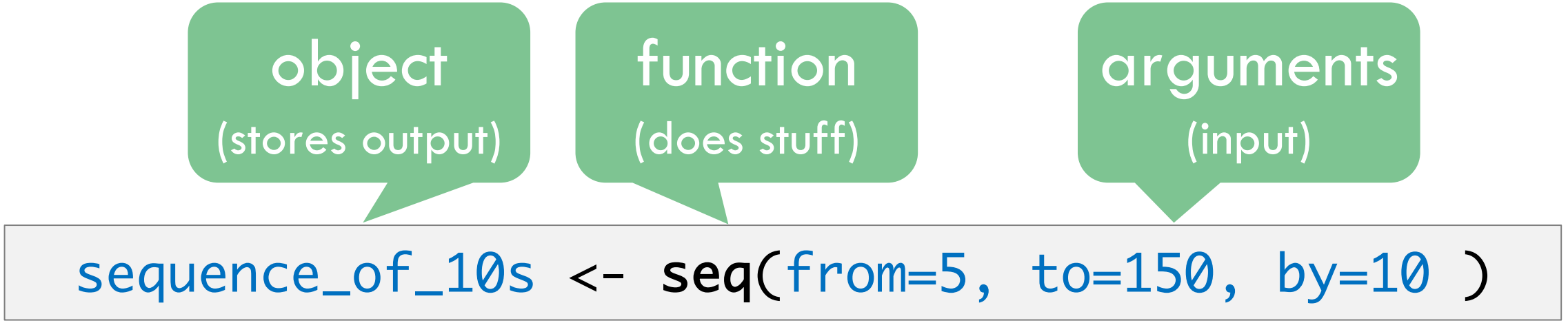
- We can use functions to do more than simple math, we can make things!

- We can create a series of integers (a vector) using the seq() function

```
>
> seq(from=5, to=150, by=10)
 [1]   5  15  25  35  45  55  65  75  85  95 105 115 125 135 145
```

The Basics of Coding: Objects

- Objects are the container for your output



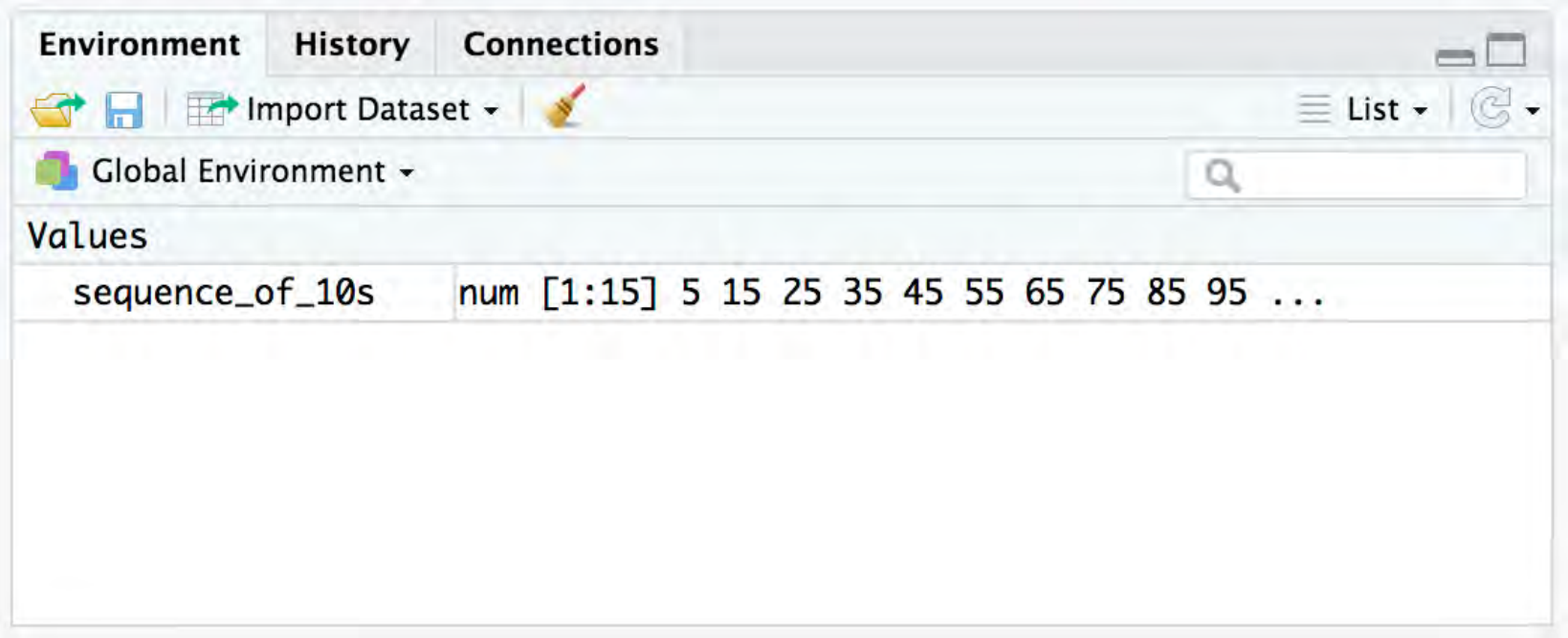
Checking the Contents of an Object

- Entering the object name at the console allows us to output the contents of an object.

```
>
> sequence_of_10s
 [1]   5  15  25  35  45  55  65  75  85  95 105 115 125 135 145
```

Checking the contents of an object

- The environment tab shows us the objects we have created.



Bending objects to your will

- Once we have created an object we can start to interact with it.
- This includes passing our objects to other functions… Whoa!

```
>
> min(sequence_of_10s)
[1] 5
>
> max(sequence_of_10s)
[1] 145
>
```

Your Turn 2

Generate a sequence, store it to an object, and ply your object

Type the following code to create a sequence from 0 to 500 in increments of 25 called sequence_of_25s:

```
sequence_of_25s <- seq(from=0, to=500, by=25)
```

Calculate the median value of this series using the `median()` function

The Basics of Coding: Packages

- A package is a collection of functions.
- Packages extend the capabilities of the base R programming language.
- The **tidyverse** includes functions for reading data into the R environment, cleaning and manipulating data, and plotting our results.

Installing and Loading Packages

- Installing a package

function (does stuff)

arguments (input)

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

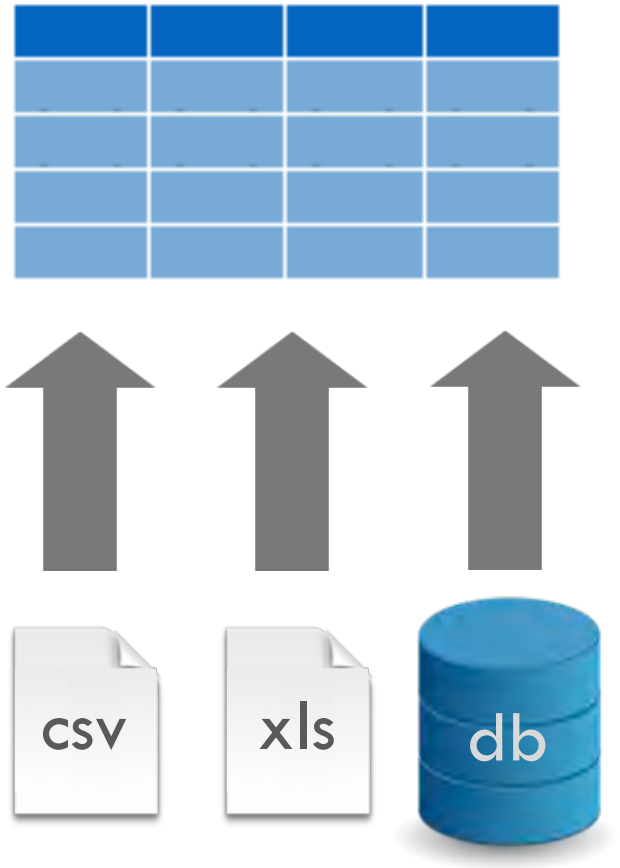
- Loading into your environment

```
library(tidyverse)
```

Importing Data and Working with Dataframes (aka Useful Data)

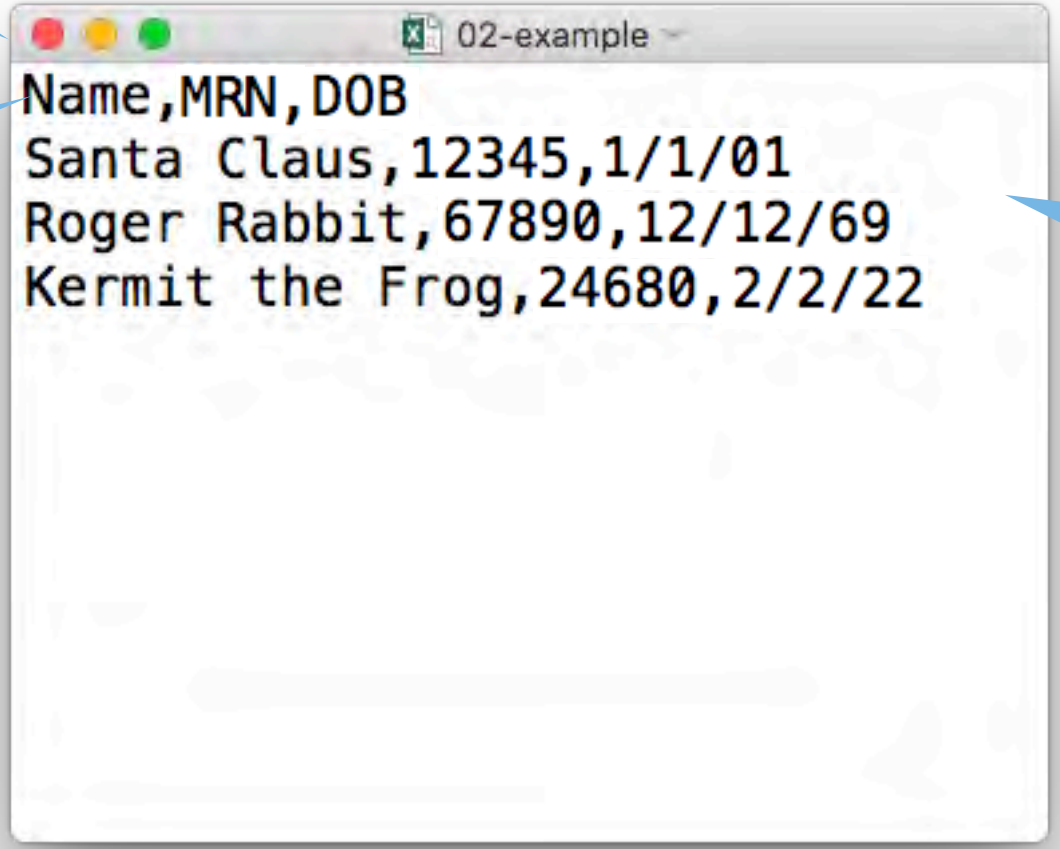
Dataframes: Beyond the Vector

- Dataframe is the term for a table
- Dataframes are composed:
  Columns (Variables)
  Rows (Observations)
- Dataframes are objects and can be acted on like other objectsç



plain text ("flat") file

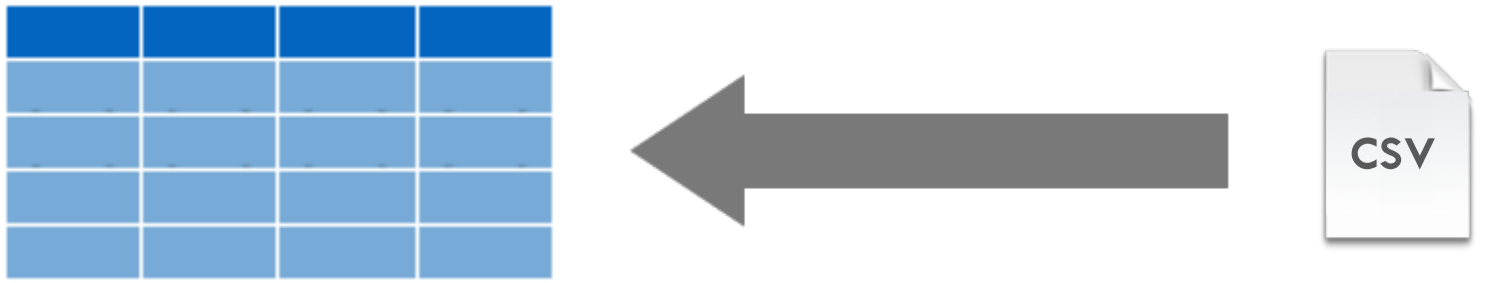
header row



rectangular structure

Loading Data to Create a Dataframe

```
data_frame <- read_csv("file_name")
```



Your Turn 3

Configure environment and load the Covid Testing CSV:

Load the tidyverse library using `library(tidyverse)`

Use the **`read_csv()`** function to load the data

-File_name argument: `"data/covid_testing.csv"`

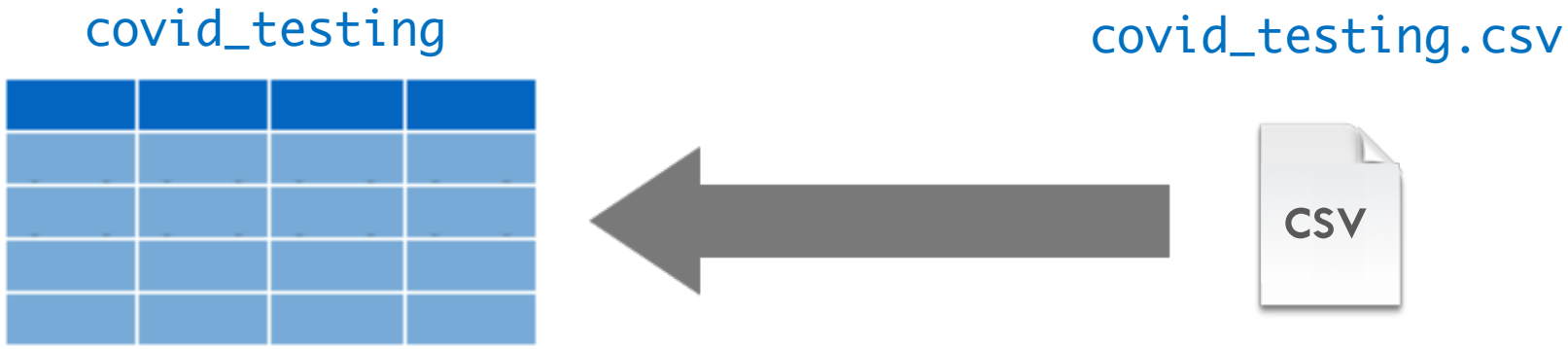
-Object name: `covid_testing`

read_csv()

data frame to read data into

name of CSV file

```
covid_testing <- read_csv("data/covid_testing.csv")
```



What's in a name?

- Capitalization matters

`covid_testing` ≠ `Covid_testing` ≠ `COVID_TESTING`

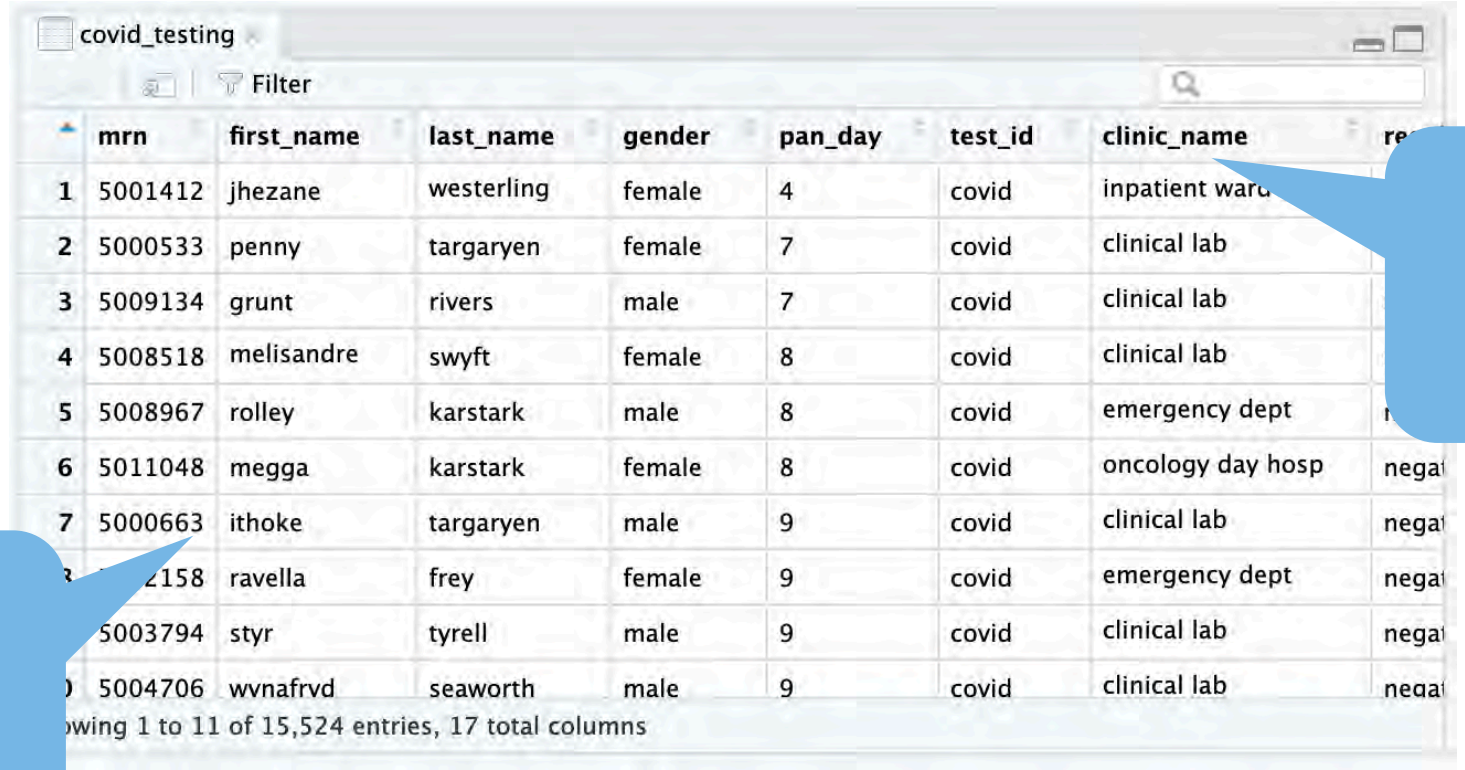
- Strive for names that are concise and meaningful (not easy!)

Bad	Still not great	Good
`p`	`name`	`patient_name`

Viewing the Contents of a Dataframe

single click to explore the data

Viewing the Contents of a Dataframe


covid_testing

Filter

	mrn	first_name	last_name	gender	pan_day	test_id	clinic_name	re
1	5001412	jhezane	westerling	female	4	covid	inpatient ward	
2	5000533	penny	targaryen	female	7	covid	clinical lab	
3	5009134	grunt	rivers	male	7	covid	clinical lab	
4	5008518	melisandre	swyft	female	8	covid	clinical lab	
5	5008967	rolley	karstark	male	8	covid	emergency dept	
6	5011048	megga	karstark	female	8	covid	oncology day hosp	negat
7	5000663	ithoke	targaryen	male	9	covid	clinical lab	negat
8	[illegible]2158	ravella	frey	female	9	covid	emergency dept	negat
9	5003794	styr	tyrell	male	9	covid	clinical lab	negat
10	5004706	wvnafrvd	seaworth	male	9	covid	clinical lab	negat

[illegible]owing 1 to 11 of 15,524 entries, 17 total columns

17 Attributes (Columns)

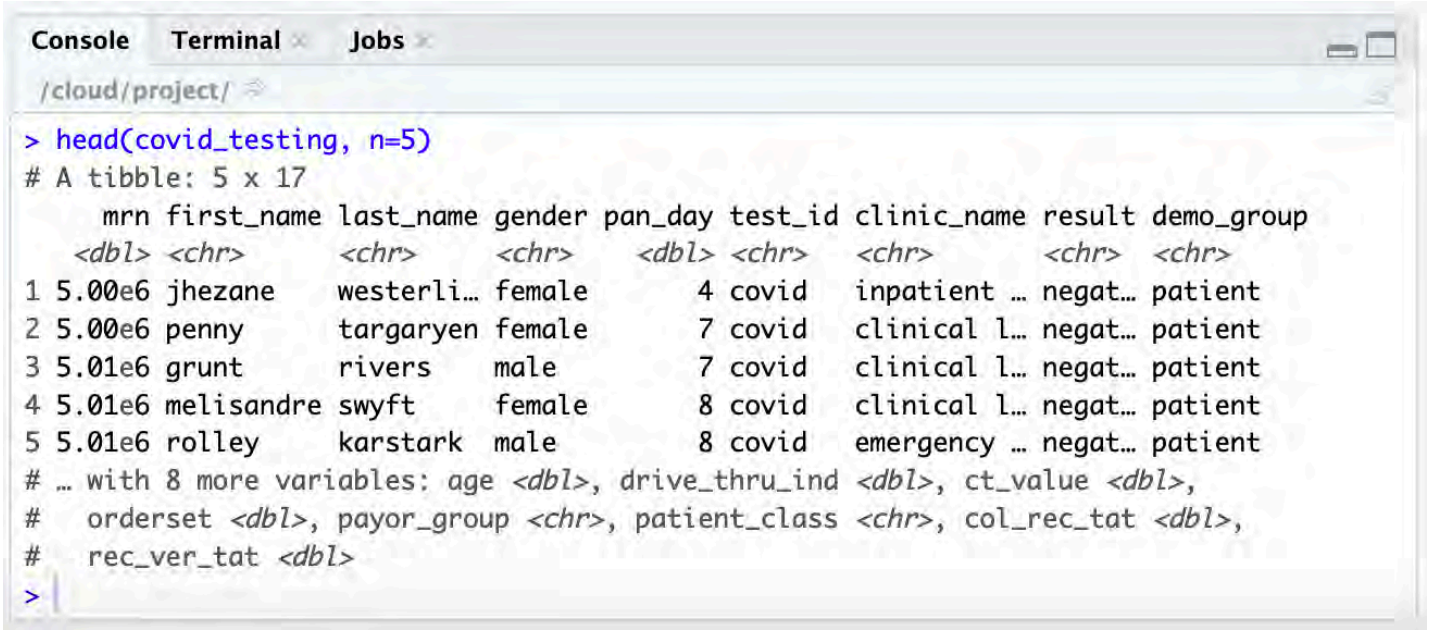
15,524 Observations (Rows)

Working with Dataframes at the Console

- The head() function is helpful for displaying a snippet of your dataframe

```
head(object_name, n=number of rows to view)
```

Sample of Data in Your Object



Your Turn 4

Understanding Object Contents

Use the `tail()` function to view the **last 10 rows** of our object `covid_testing`.

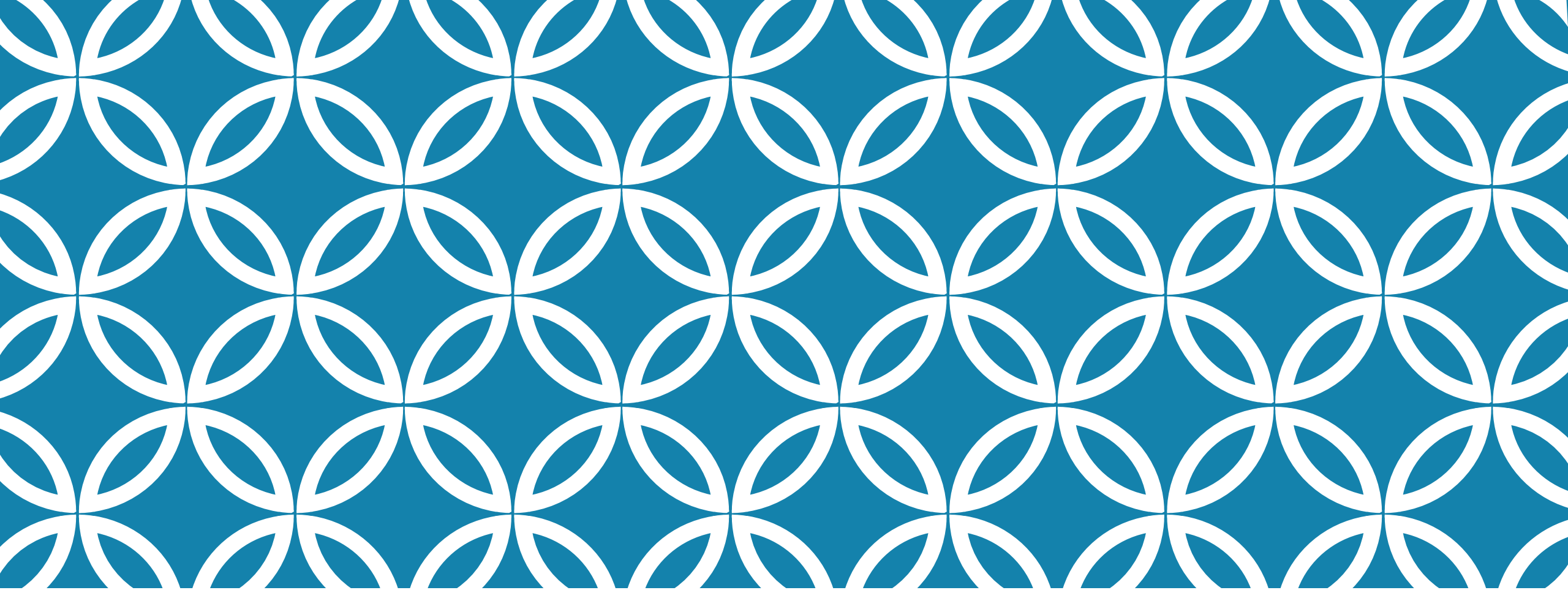
-What is the ratio of female to male patients in this subset of data?

Lesson Goals

1. Get oriented to R and RStudio
2. Learn some fundamentals of coding

Lesson Objectives

1. Log in and tour RStudio Cloud
2. Execute code at the console
3. Define and use functions
4. Define and create objects in the environment
5. Load data into R and interact with a dataframe



Reproducible Reporting

Patrick Mathias

December 13, 2020

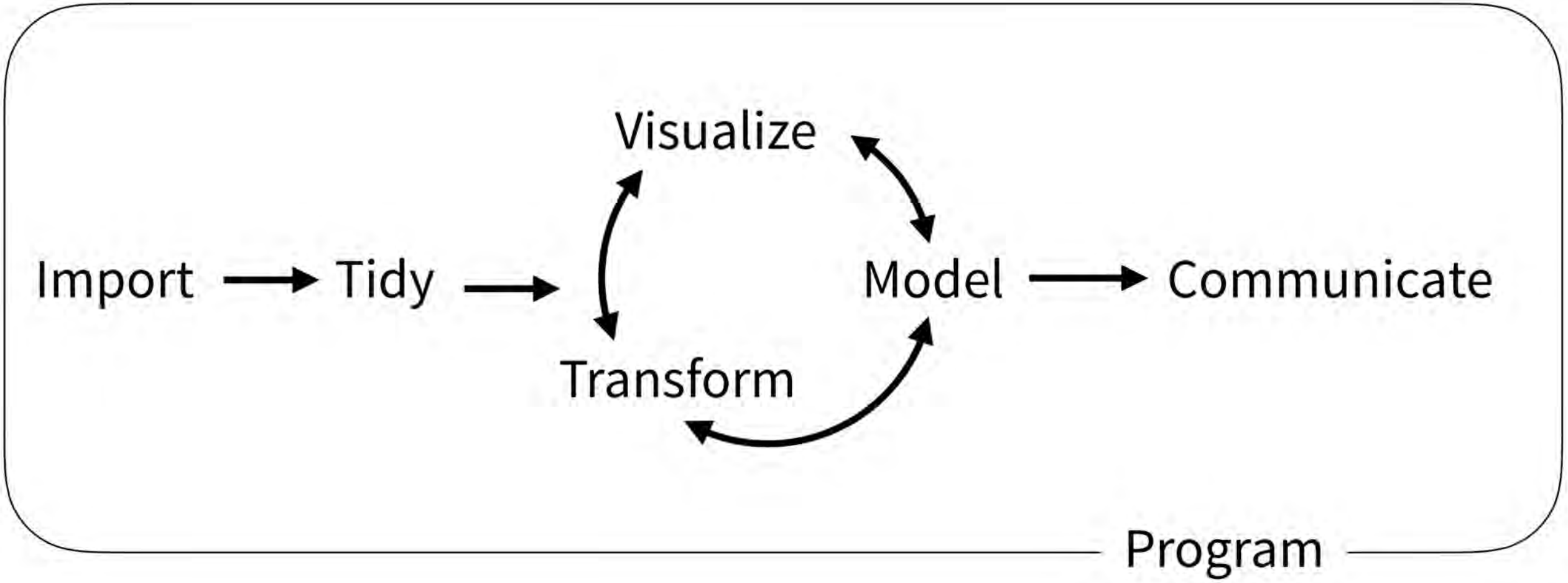
Goals

1. Understand why reproducible reporting is important
2. Learn to work within R Markdown for reproducible reports

Objectives

1. Create an R Markdown document and generate different types of output files
2. Practice modifying each component of a R Markdown file

Typical Data Science Pipeline



From *R for Data Science* (https://r4ds.had.co.nz/introduction.html)

Your Turn #1

1. List 3 benefits of doing data analysis using spreadsheet software like Microsoft Excel
2. List 3 drawbacks/problems with doing data analysis using spreadsheet software like Microsoft Excel

Why is reproducibility important?

Replication vs Reproduction

- Replication: other people collect new data
  - Scientific gold standard
  - Difficult and time-consuming

- Reproduction: other people analyze the same data
  - Does not by itself validate the analysis …
  - Has been proposed as a minimal standard

Point-and-Click Is Not Reproducible

- Interactive tools do not record user actions
- Manual documentation is error-prone
- Manual analyses cannot be repeated on new data sets or shared with collaborators

Computer code can precisely document each step of the analysis

Why YOU Should Do Data Analysis Reproducibly

"Can we redo the analysis with this month's data?"

"Why do the data in Table 1 not seem to agree with Figure 2?"

"Why did I decide to omit these six samples?"

YOUR CLOSEST COLLABORATOR IS **YOU** FROM 6 MONTHS AGO
(BUT YOU DON'T ANSWER E-MAILS)

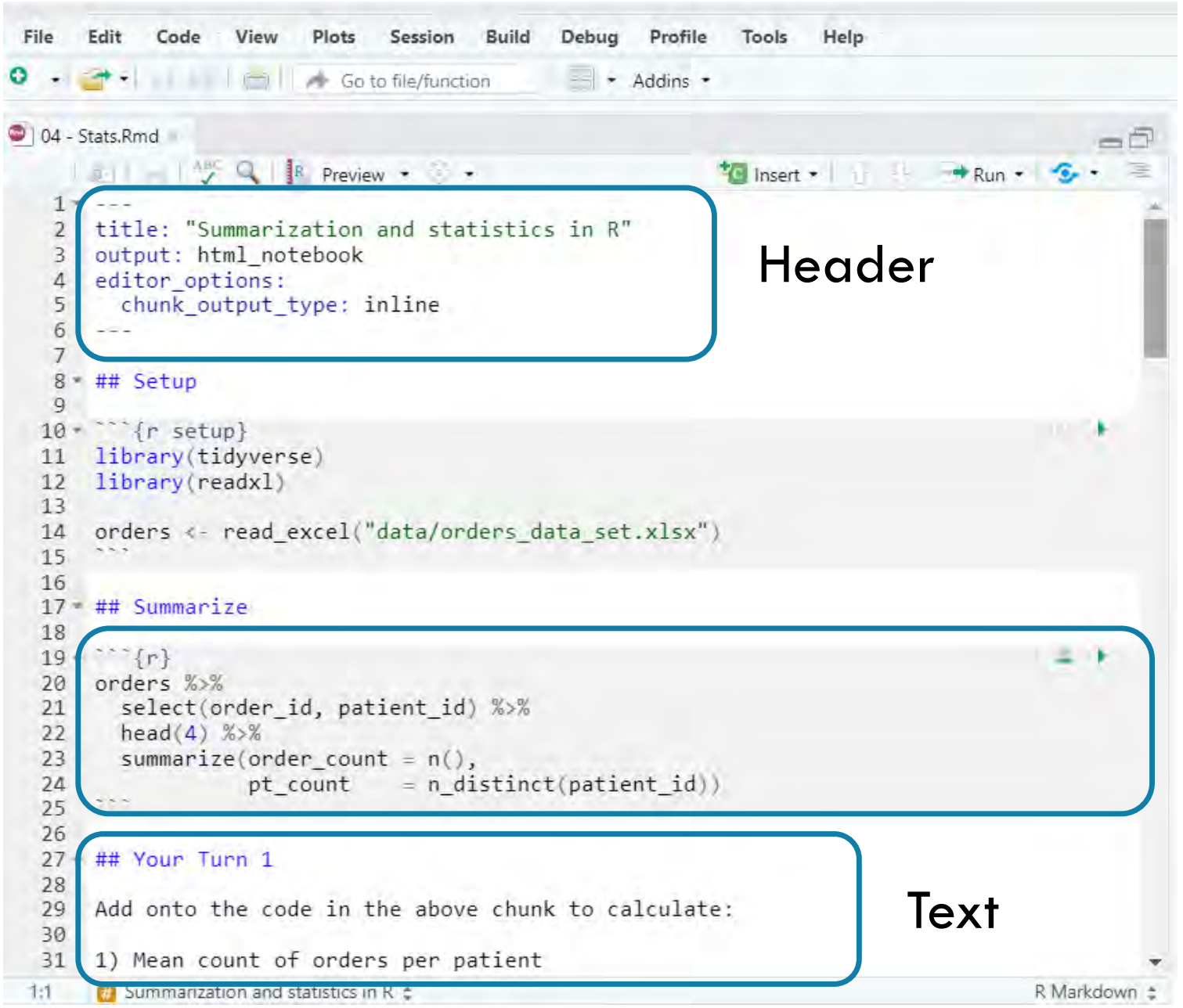
Using R for Reproducibility

Programming in R (or another language) allows one to reproduce analysis steps exactly or perform same analysis on new data

Better practice is to create documentation about analysis to accompany and explain code

Best practice is include documentation and code in one place

Using R Markdown to Support Reproducibility

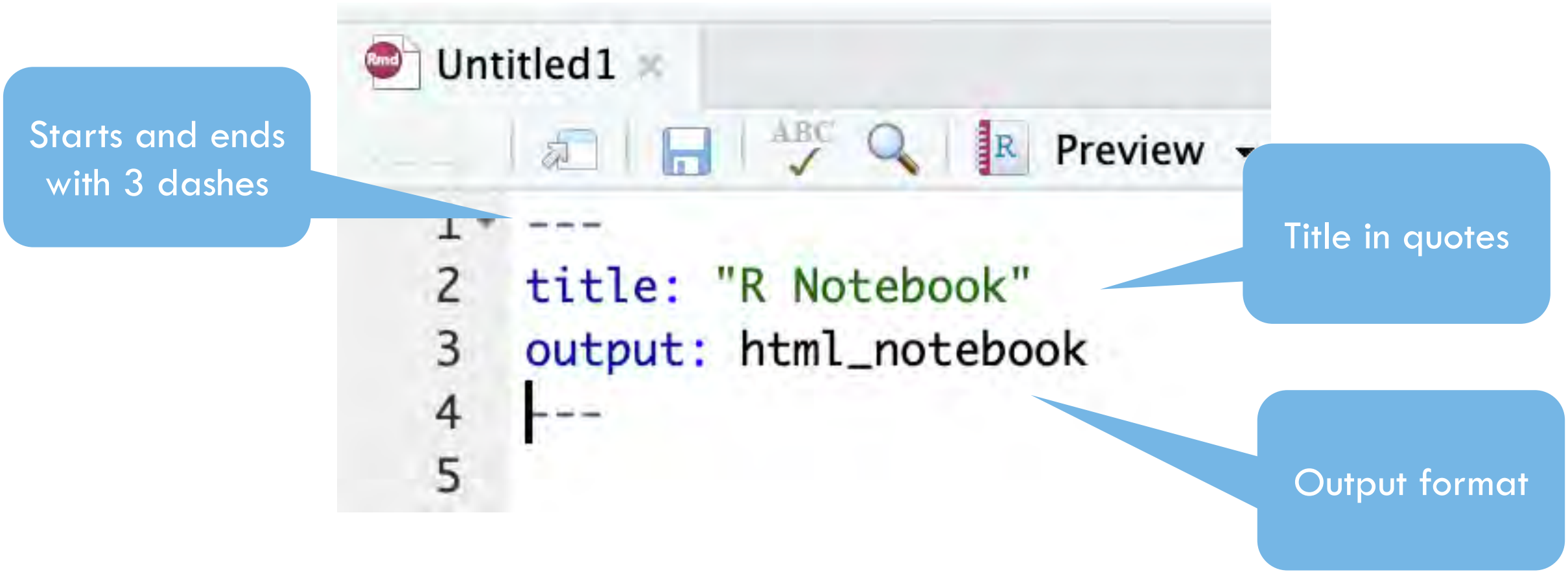


Header

Code chunk

Text

Header


Untitled1
Preview
---
title: "R Notebook"
output: html_notebook

Starts and ends with 3 dashes
Title in quotes
Output format

Text

```
5
6  This is an [R Markdown](http://rmarkdown.rstudio.com) Notebook. When you execute
   code within the notebook, the results appear beneath the code.
7
8  Try executing this chunk by clicking the *Run* button within the chunk or by
   placing your cursor inside it and pressing *Cmd+Shift+Enter*.
9
```

Include hyperlinks with [text](link) format

1 asterisk for *italics* (*italics*)

2 asterisks for **bold** (**bold**)

Hyphens (-bullet 1) for bullet points

1 asterisk for *italics* (*italics*)

R Markdown Cheat Sheet

learn more at rmarkdown.rstudio.com

rmarkdown 0.2.50 Updated: 8/14

RStudio

1. Workflow

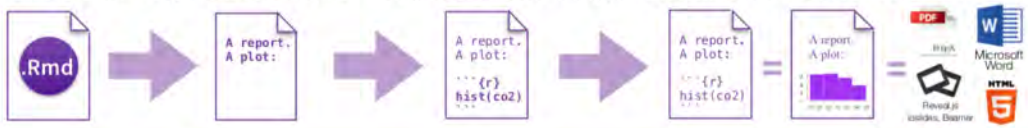
R Markdown is a format for writing reproducible, dynamic reports with R. Use it to embed R code and results into slideshows, pdfs, html documents, Word files and more. To make a report:

i. Open - Open a file that uses the .Rmd extension.

ii. Write - Write content with the easy to use R Markdown syntax

iii. Embed - Embed R code that creates output to include in the report

iv. Render - Replace R code with its output and transform the report into a slideshow, pdf, html or ms Word file.



2. Open File

Start by saving a text file with the extension .Rmd, or open an RStudio Rmd template

3. Markdown

Next, write your report in plain text. Use markdown syntax to describe how to format text in the final report.

syntax

```
Plain text
End a line with two spaces to start a new paragraph.
*italics* and _italics_
**bold** and __bold__
superscript^2^
~~strikethrough~~
[link](www.rstudio.com)

# Header 1

## Header 2

### Header 3

#### Header 4

##### Header 5

###### Header 6

endash: --
emdash: ---
ellipsis: ...
inline equation: $A = \pi*r^{2}$
image: ![](path/to/smallorb.png)

horizontal rule (or slide break):
```

becomes

Plain text
End a line with two spaces to start a new paragraph.
italics and *italics*
bold and **bold**
superscript^2
~~strikethrough~~
link

Header 1
Header 2
Header 3
Header 4
Header 5
Header 6

endash: –
emdash: —
ellipsis: ...
inline equation: $A = \pi * r^2$
image:

output: beamer_presentation ..... beamer slideshow (pdf)

output: ioslides_presentation ..... ioslides slideshow (html)

RStudio® is a trademark of RStudio, Inc. • CC BY RStudio • info@rstudio.com • 844-448-1212 • rstudio.com

syntax

```
Plain text
End a line with two spaces to start a new paragraph.
*italics* and _italics_
**bold** and __bold__
```

becomes

Plain text
End a line with two spaces to start a new paragraph.
italics and *italics*
bold and **bold**

horizontal rule (or slide break):

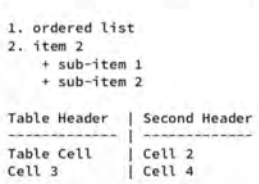
```
***

> block quote

* unordered list
* item 2
    + sub-item 1
    + sub-item 2

1. ordered list
2. item 2
    + sub-item 1
    + sub-item 2

Table Header  | Second Header
------------- | -------------
Table Cell    | Cell 2
Cell 3        | Cell 4
```

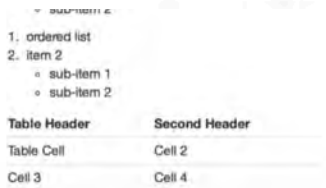
> block quote

- unordered list
- item 2
  - sub-item 1
  - sub-item 2

1. ordered list
2. item 2
   - sub-item 1
   - sub-item 2

Table Header	Second Header
Table Cell	Cell 2
Cell 3	Cell 4





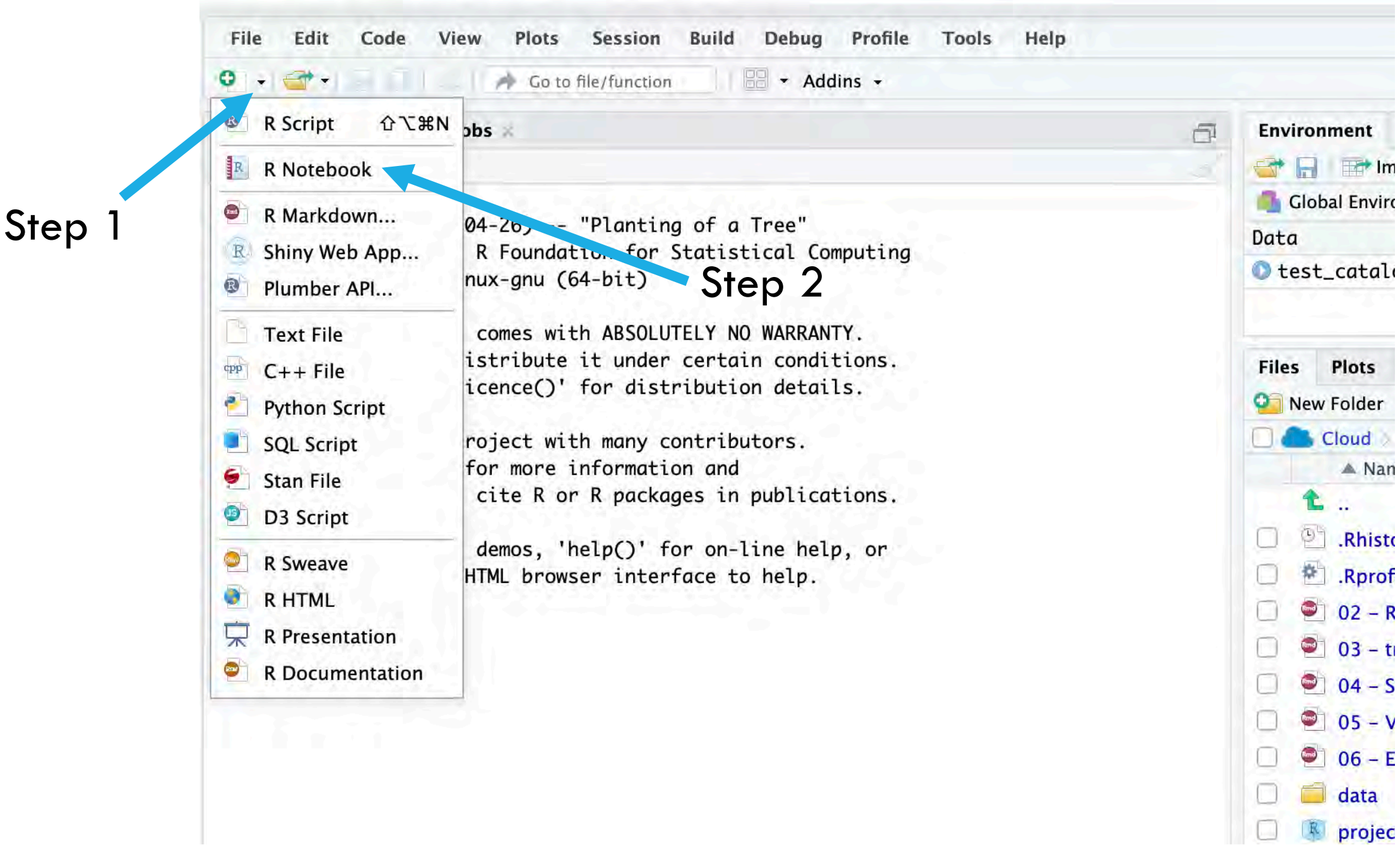
https://www.rstudio.com/wp-content/uploads/2015/02/rmarkdown-cheatsheet.pdf

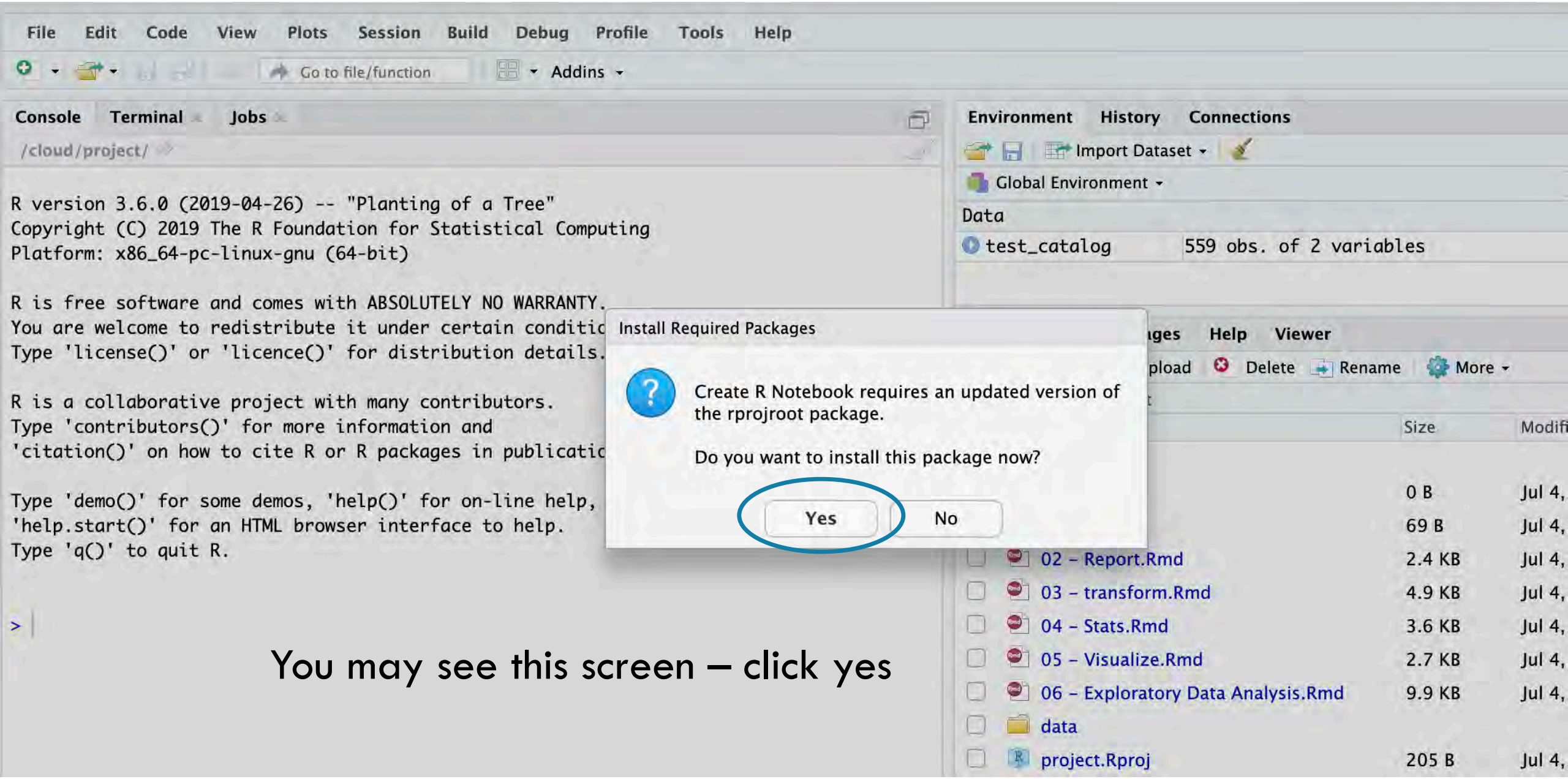
Your Turn #2

1. Open up an R Notebook per guidance on next few slides.
2. Change the title of your R Notebook to “My First R Notebook” by modifying the header.
3. Add your name as the author by adding another line to the header:

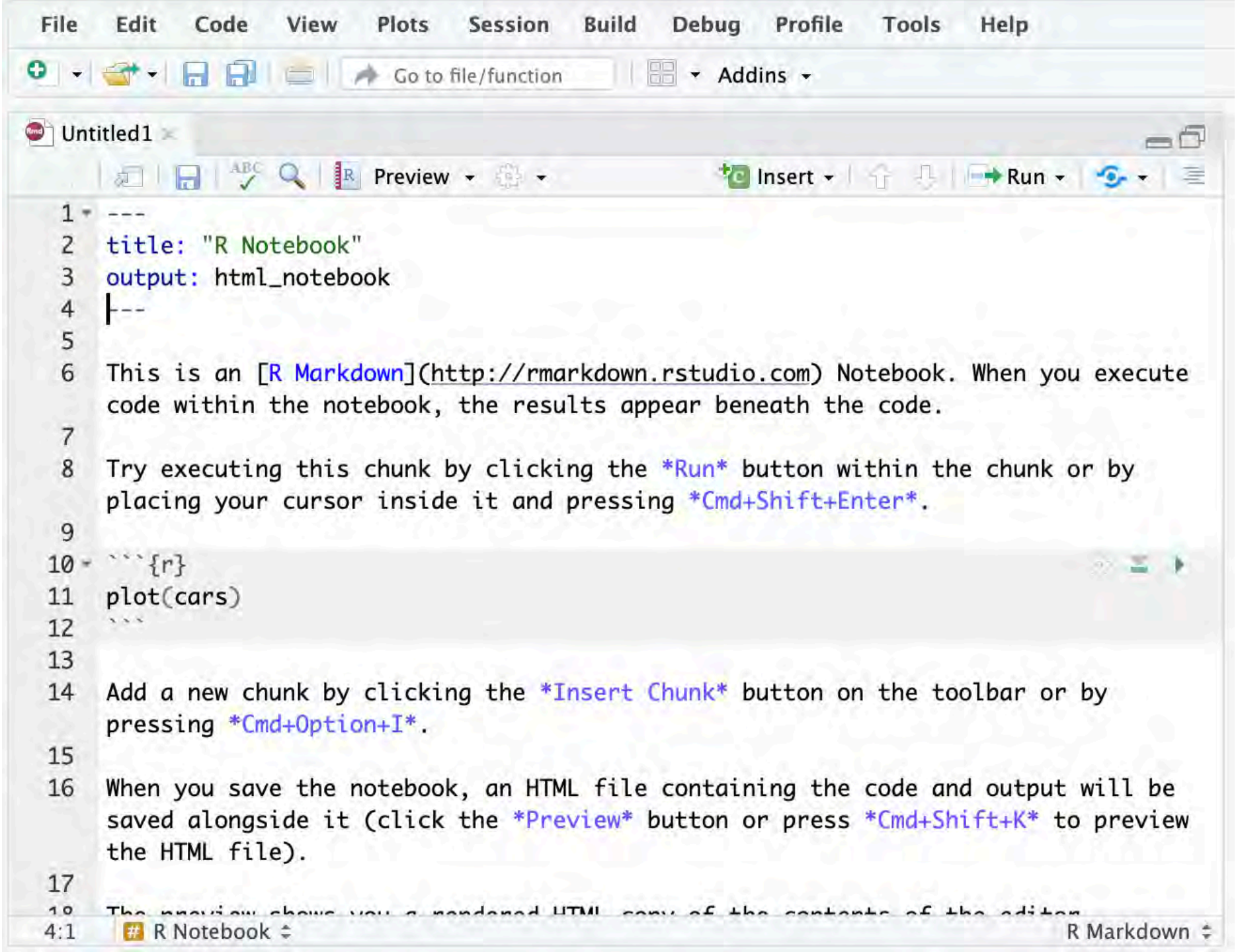
   author: “Your Name”

3. Add a second level heading (##) at the end of the notebook called “My Calculation”


Step 1
Step 2
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
R Script ⇧⌥⌘N
R Notebook
R Markdown...
Shiny Web App...
Plumber API...
Text File
C++ File
Python Script
SQL Script
Stan File
D3 Script
R Sweave
R HTML
R Presentation
R Documentation
Environment
Global Enviro
Data
test_catal
Files Plots
New Folder
Cloud
.Rhist
.Rprof
02 – R
03 – t
04 – S
05 – V
06 – E
data
projec



You may see this screen – click yes


File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Untitled1
Preview
Insert
Run
---
title: "R Notebook"
output: html_notebook

This is an [R Markdown](http://rmarkdown.rstudio.com) Notebook. When you execute code within the notebook, the results appear beneath the code.
Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor inside it and pressing *Cmd+Shift+Enter*.
```{r}
plot(cars)
```
Add a new chunk by clicking the *Insert Chunk* button on the toolbar or by pressing *Cmd+Option+I*.
When you save the notebook, an HTML file containing the code and output will be saved alongside it (click the *Preview* button or press *Cmd+Shift+K* to preview the HTML file).
4:1
R Notebook
R Markdown

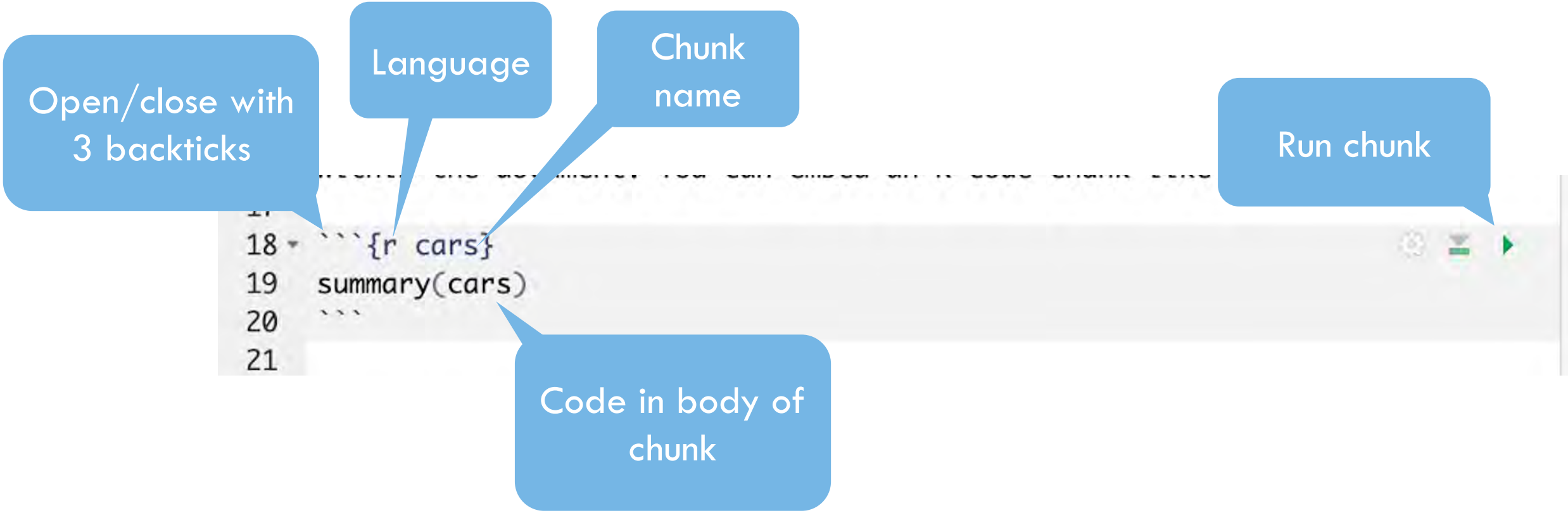
Your Turn #2

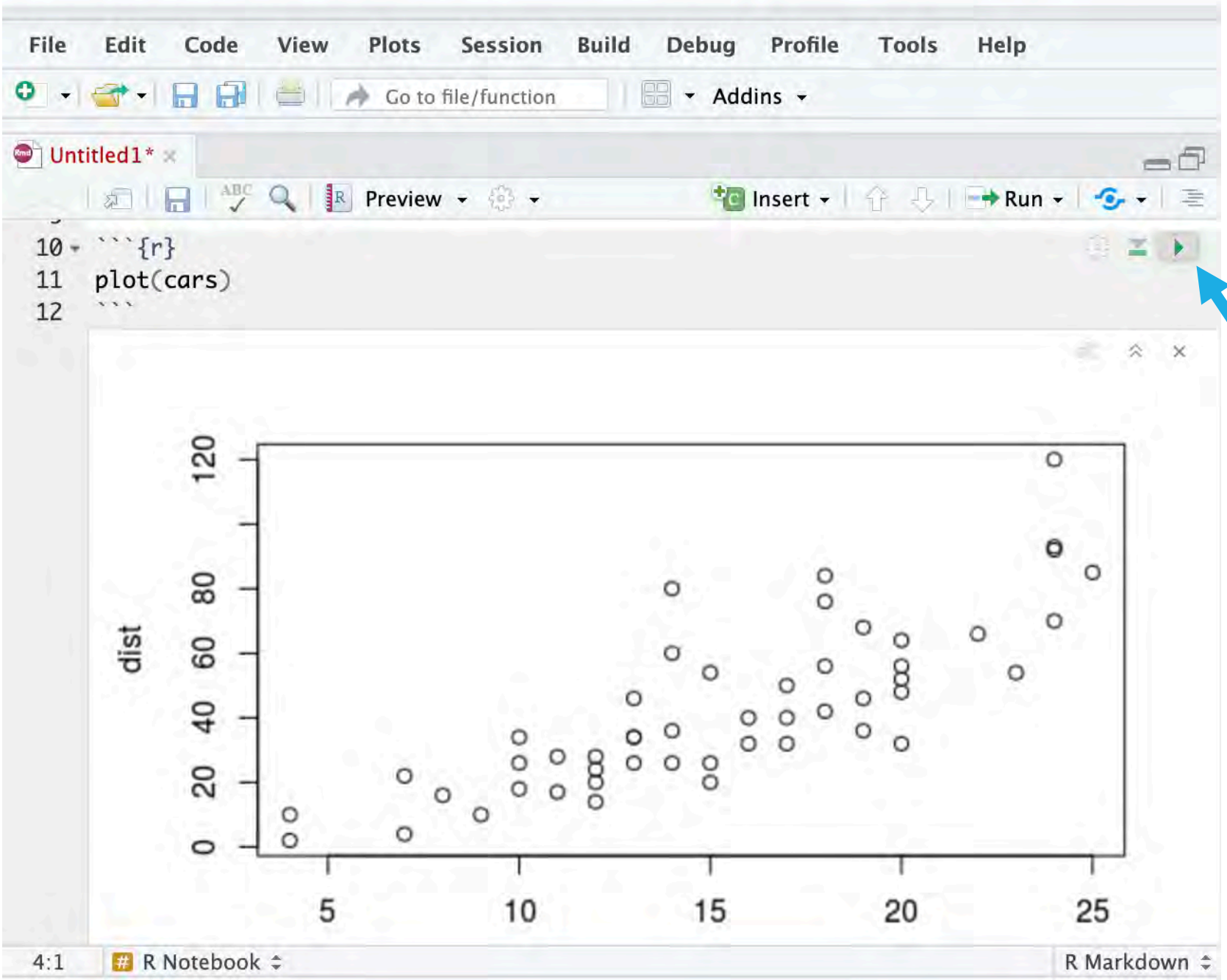
1. Open up an R Notebook per guidance on next few slides.
2. Change the title of your R Notebook to “My First R Notebook” by modifying the header.
3. Add your name as the author by adding another line to the header:

   Author: “Your Name”

3. Add a second level heading (##) at the end of the notebook called “My Calculation”

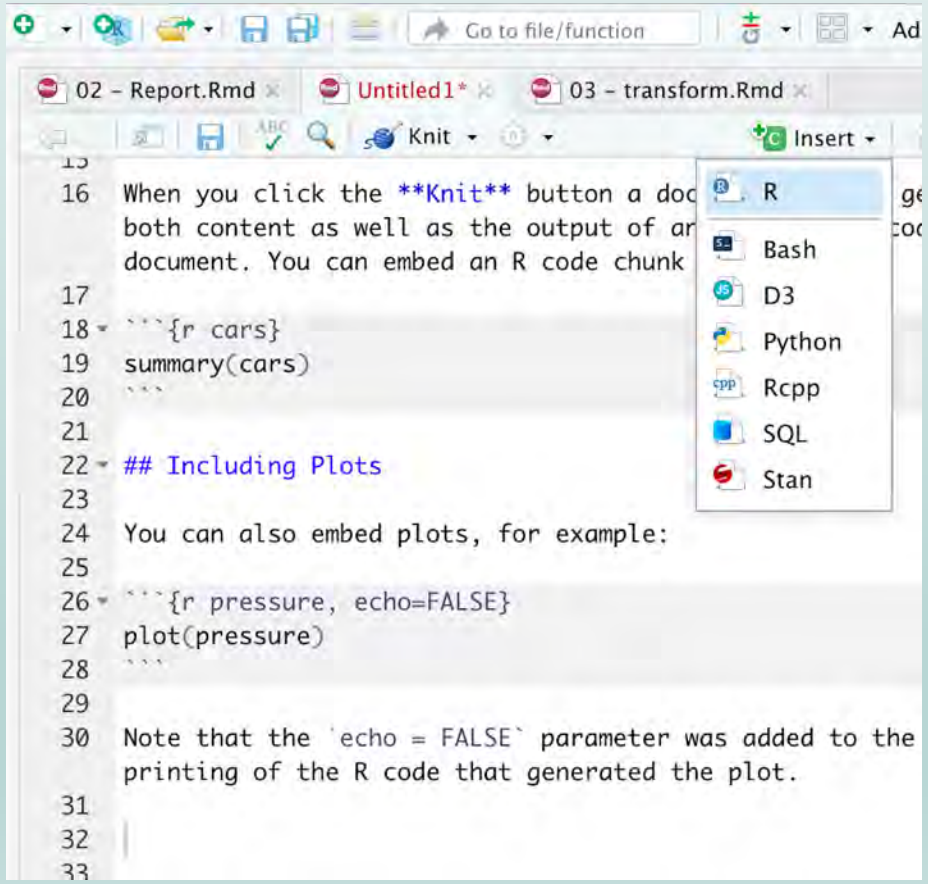
Code chunks


Open/close with 3 backticks
Language
Chunk name
Run chunk
18 ```{r cars}
19 summary(cars)
20 ```
21
Code in body of chunk



Run code chunk

Your Turn #3

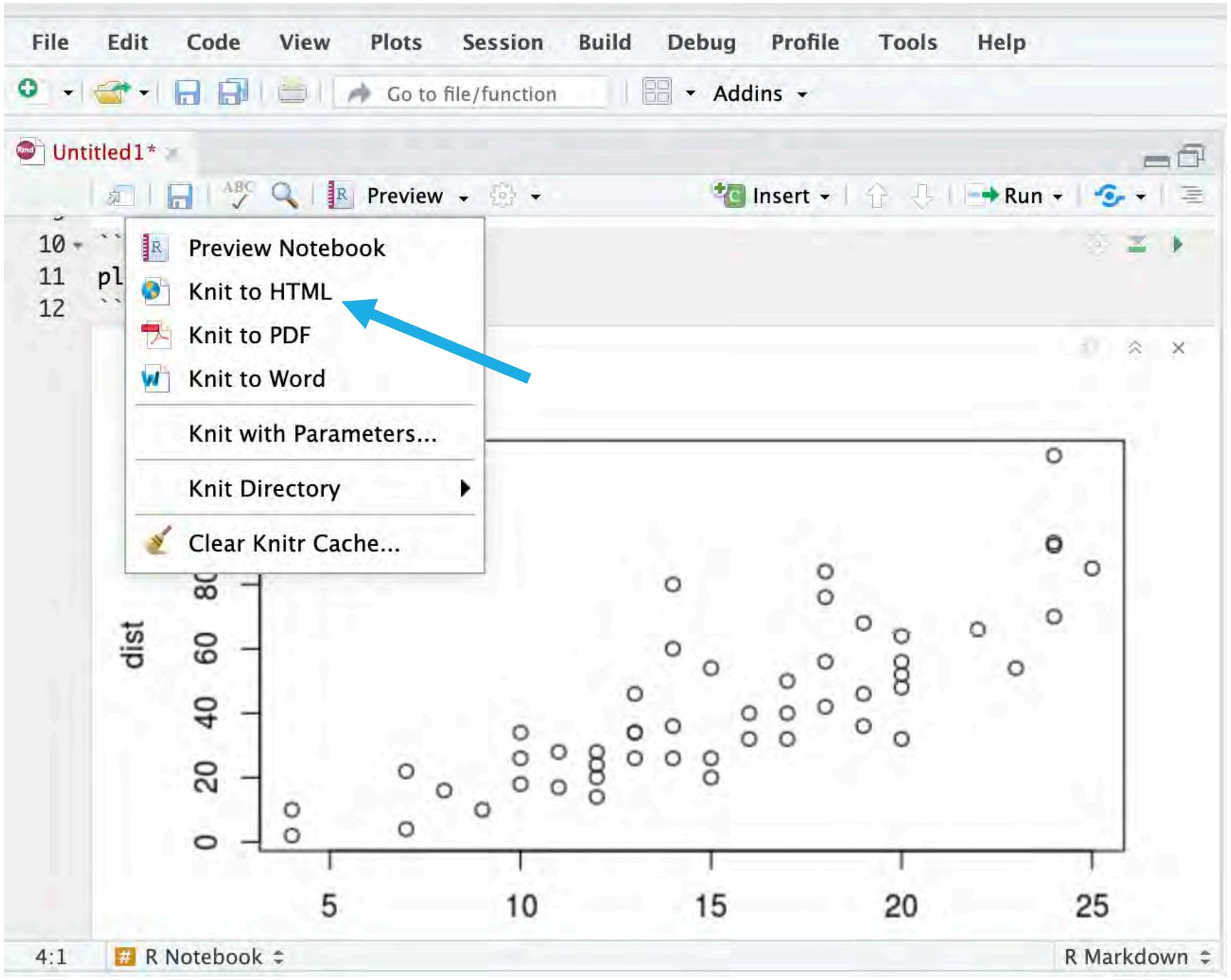


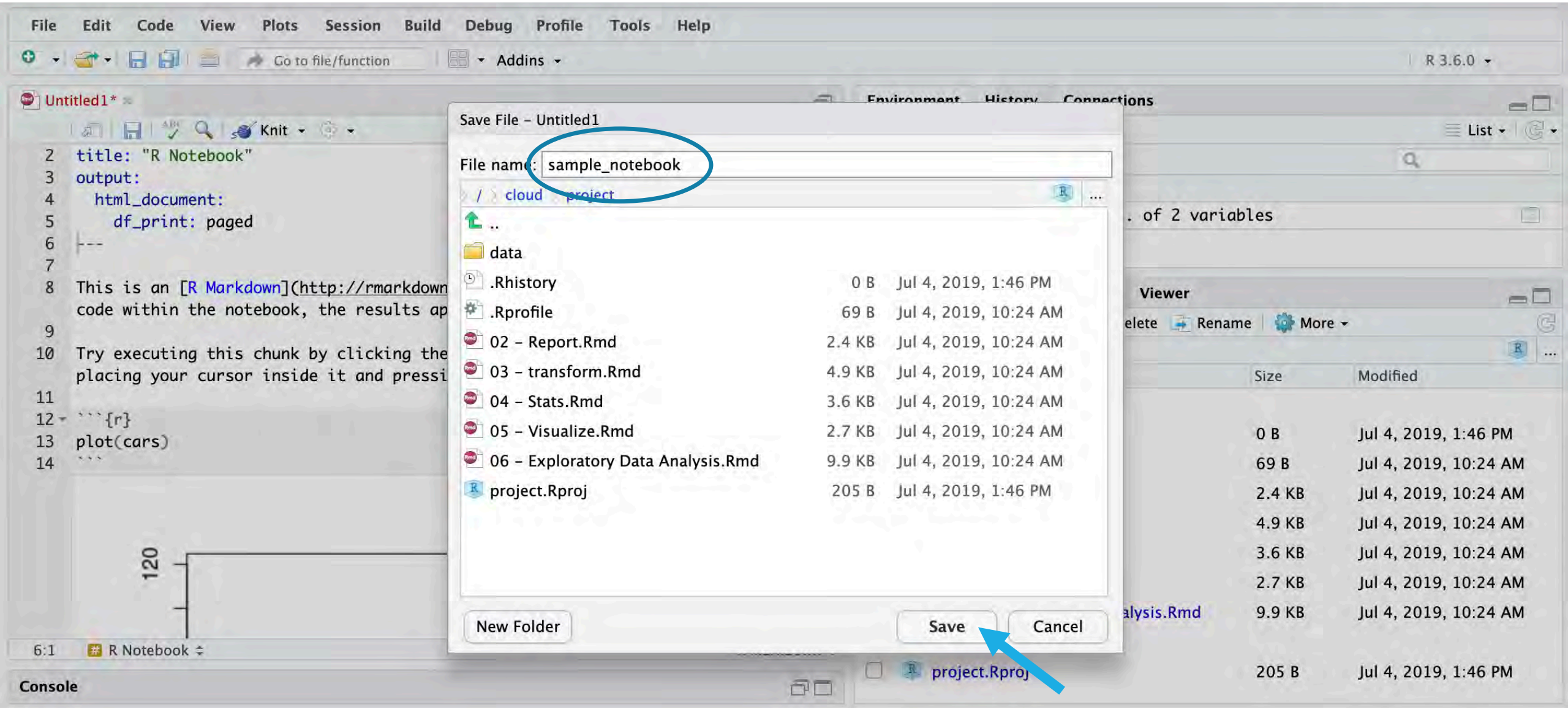
1. Under your new "My Calcuation" heading, insert a code chunk into white space using Insert button on top right of code window

2. Type the following in the grey box of your new code chunk:

```
mean(c(10, 20, 30))
```

3. Execute code chunk by pressing Run button on top right of code chunk


File
Edit
Code
View
Plots
Session
Build
Debug
Profile
Tools
Help
Go to file/function
Addins
Untitled1*
Preview
Insert
Run
10
11
12
pl
Preview Notebook
Knit to HTML
Knit to PDF
Knit to Word
Knit with Parameters...
Knit Directory
Clear Knitr Cache...
dist
0
20
40
60
5
10
15
20
25
4:1
R Notebook
R Markdown

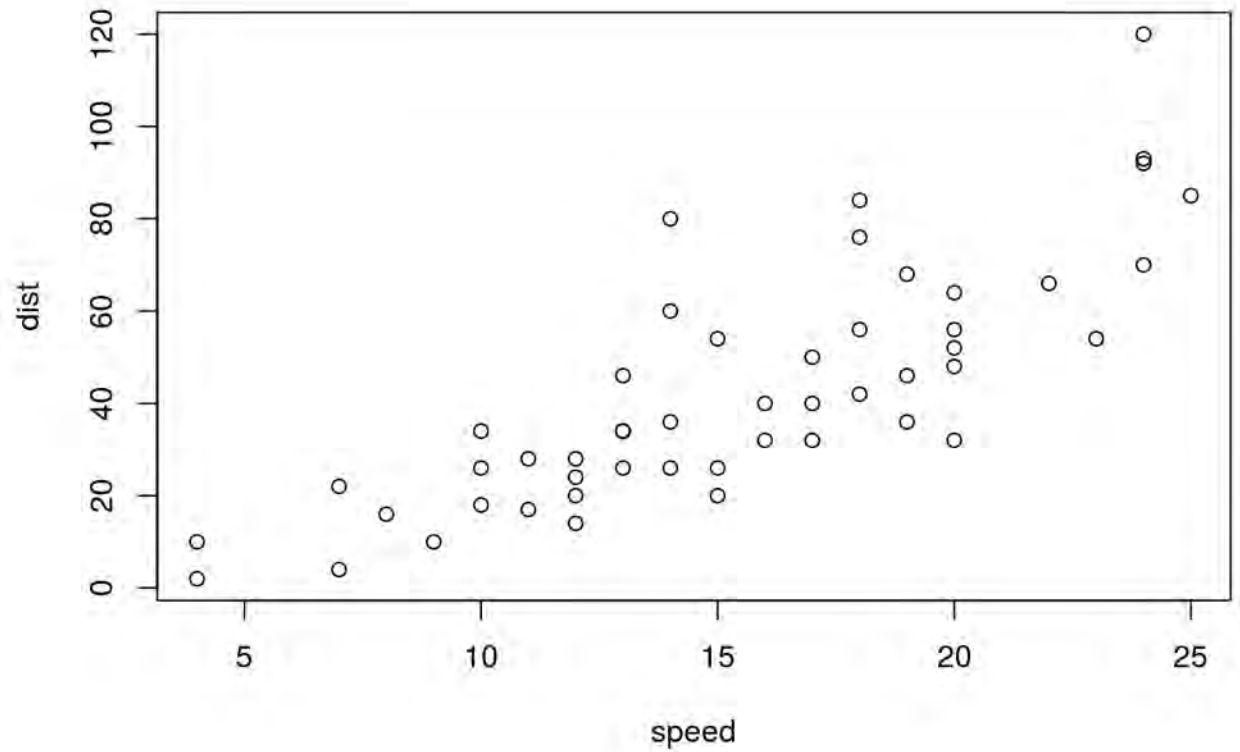

File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
R 3.6.0
Untitled1*
Knit
2 title: "R Notebook"
3 output:
4 html_document:
5 df_print: paged
6 ---
8 This is an [R Markdown](http://rmarkdown
code within the notebook, the results ap
10 Try executing this chunk by clicking the
placing your cursor inside it and pressi
12 ```{r}
13 plot(cars)
14 ```
120
6:1 R Notebook
Console
Save File – Untitled1
File name: sample_notebook
/ cloud project
..
data
.Rhistory 0 B Jul 4, 2019, 1:46 PM
.Rprofile 69 B Jul 4, 2019, 10:24 AM
02 – Report.Rmd 2.4 KB Jul 4, 2019, 10:24 AM
03 – transform.Rmd 4.9 KB Jul 4, 2019, 10:24 AM
04 – Stats.Rmd 3.6 KB Jul 4, 2019, 10:24 AM
05 – Visualize.Rmd 2.7 KB Jul 4, 2019, 10:24 AM
06 – Exploratory Data Analysis.Rmd 9.9 KB Jul 4, 2019, 10:24 AM
project.Rproj 205 B Jul 4, 2019, 1:46 PM
New Folder
Save
Cancel
Environment History Connections
List
. of 2 variables
Viewer
elete Rename More
Size Modified
0 B Jul 4, 2019, 1:46 PM
69 B Jul 4, 2019, 10:24 AM
2.4 KB Jul 4, 2019, 10:24 AM
4.9 KB Jul 4, 2019, 10:24 AM
3.6 KB Jul 4, 2019, 10:24 AM
2.7 KB Jul 4, 2019, 10:24 AM
alysis.Rmd 9.9 KB Jul 4, 2019, 10:24 AM
project.Rproj 205 B Jul 4, 2019, 1:46 PM

R Notebook

This is an R Markdown Notebook. When you execute code within the notebook, the results appear beneath the code.

Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor inside it and pressing *Cmd+Shift+Enter*.

```
plot(cars)
```

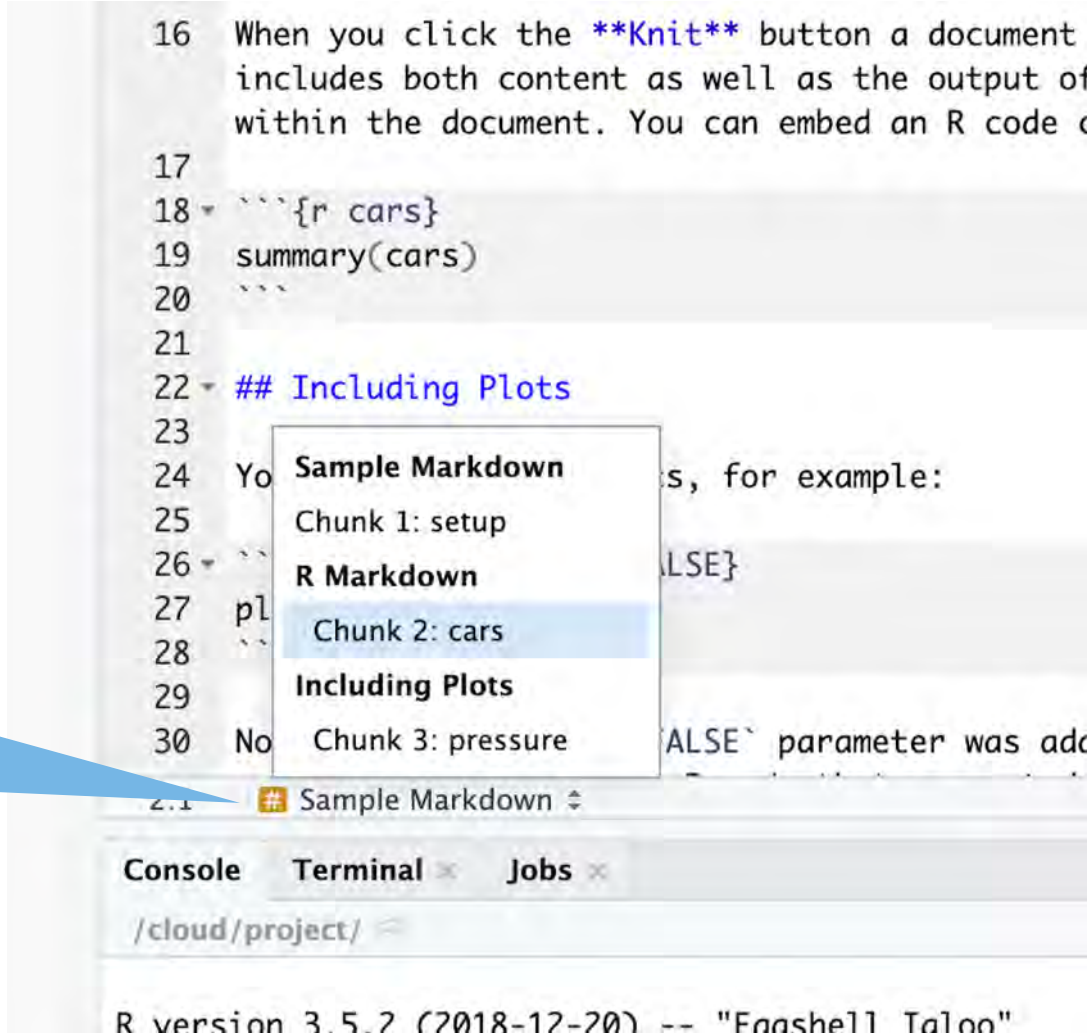


Add a new chunk by clicking the *Insert Chunk* button on the toolbar or by pressing *Cmd+Option+I*.

When you save the notebook, an HTML file containing the code and output will be saved alongside it (click the *Preview* button or press

What Else?

Why name Code Chunks?



Jump between chunks

“Setup” Chunk & Chunk Options

chunk name
(optional)

“chunk option”
don’t show code in rendered document

````
6 ▾ ```{r setup, include=FALSE}
7   library(tidyverse)
8   library(lubridate)
9   ```
````

for dealing
with dates

Chunk options

option	default	effect
eval	TRUE	Whether to evaluate the code and include its results
echo	TRUE	Whether to display code along with its results
warning	TRUE	Whether to display warnings
error	FALSE	Whether to display errors
message	TRUE	Whether to display messages
tidy	FALSE	Whether to reformat code in a tidy way when displaying it
results	"markup"	"markup", "asis", "hold", or "hide"
cache	FALSE	Whether to cache results for future renders
comment	"##"	Comment character to preface results with
fig.width	7	Width in inches for plots created in chunk
fig.height	7	Height in inches for plots created in chunk

For more details visit yihui.name/knitr/

Keyboard Shortcuts

Insert a code chunk into white space within your open R Markdown document using:

- Windows: CTRL+ALT+i
- Mac: COMMAND+OPTION+I

Execute using shortcuts:

- Windows: CTRL+SHIFT+ENTER
- Mac: COMMMAND+SHIFT+ENTER

Multiple Output Formats are Available

Pandoc universal document converter can create multiple document types:

- html
- pdf
- docx

Can also create presentations and dashboards

- Including Powerpoint (most recent version of RStudio)

Creating a pdf report

R & RStudio require additional packages to create a nicely formatted pdf report

Behind the scenes, R will use a markup language called LaTeX to turn your markdown into the pdf

Install the tinytex package with the following commands:

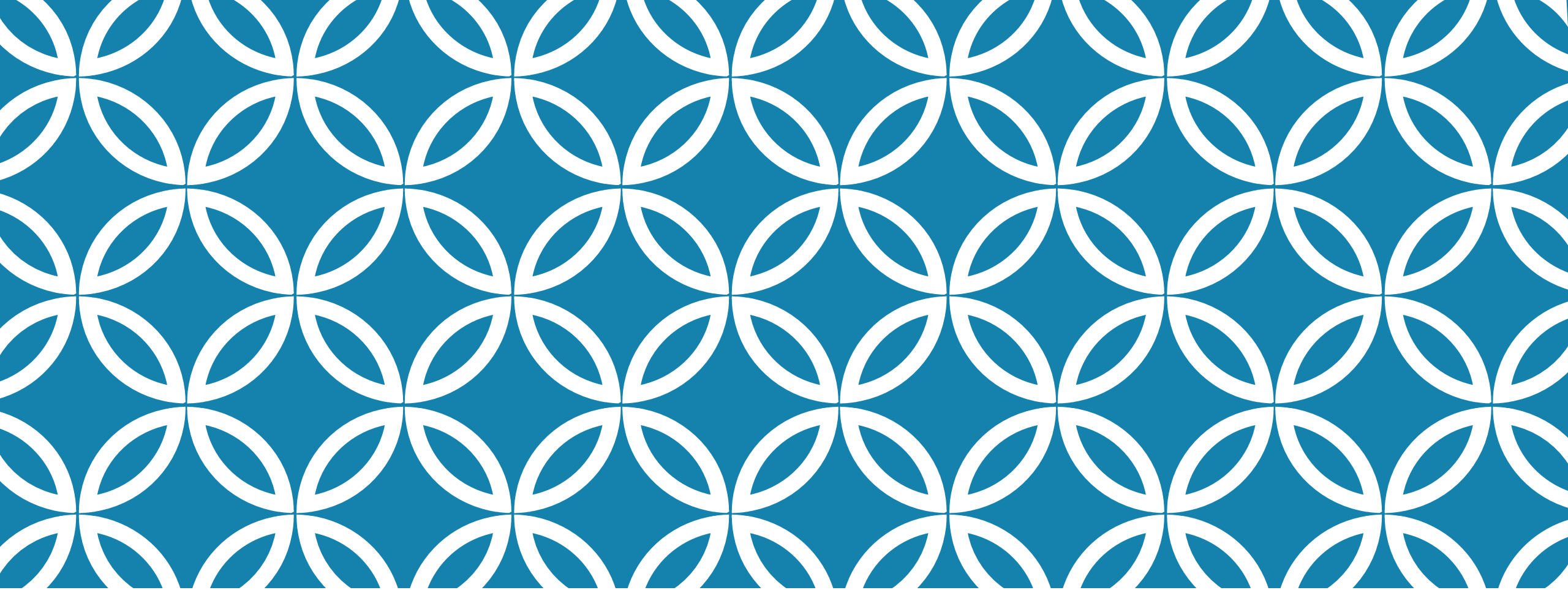
- install.packages('tinytex')
- tinytex::install_tinytex()

Goals

1. Understand why reproducible reporting is important
2. Learn to work within R Markdown for reproducible reports

Objectives

1. Create an R Markdown document and generate different types of output files
2. Practice modifying each component of a R Markdown file



Data Visualization

Session 3

Joseph Rudolf

December 13, 2020

Visualization Lecture Courtesy of Stephan Kadauke

Assistant Professor of Clinical Pathology and Laboratory Medicine

University of Pennsylvania Perelman School of Medicine

Assistant Director of the Cell and Gene Therapy Laboratory

Children's Hospital of Philadelphia

Goals

1. Appreciate the importance of visualization for understanding data
2. Learn how to use ggplot2 to visualize data

Objectives

1. Create a basic visualization using a simple template
2. Define “aesthetic mapping” and explain how aesthetic mappings relate variables of a data frame to features of graphic markings on a plot
3. Write the code to specify a type of plot and fine tune its appearance using “geom” functions
4. Explain how to add layers to a ggplot object to create complex and highly customized visualizations

covid_testing

covid_testing

Filter

	mrn	first_name	last_name	gender	pan_day	test_id	clinic_name	result
1	5001412	jhezane	westerling	female	4	covid	inpatient ward a	negative
2	5000533	penny	targaryen	female	7	covid	clinical lab	negative
3	5009134	grunt	rivers	male	7	covid	clinical lab	negative
4	5008518	melisandre	swyft	female	8	covid	clinical lab	negative
5	5008967	rolley	karstark	male	8	covid	emergency dept	negative
6	5011048	megga	karstark	female	8	covid	oncology day hosp	negative
7	5000663	ithoke	targaryen	male	9	covid	clinical lab	negative
8	5002158	ravella	frey	female	9	covid	emergency dept	negative
9	5003794	styr	tyrell	male	9	covid	clinical lab	negative
10	5004706	wynafryd	seaworth	male	9	covid	clinical lab	negative
11	5008115	patrek	frey	male	9	covid	clinical lab	negative
12	5009309	maege	sand	female	9	covid	medical center	negative
13	5008943	myria	rivers	female	9	covid	picu	negative

Showing 1 to 14 of 15,524 entries, 17 total columns

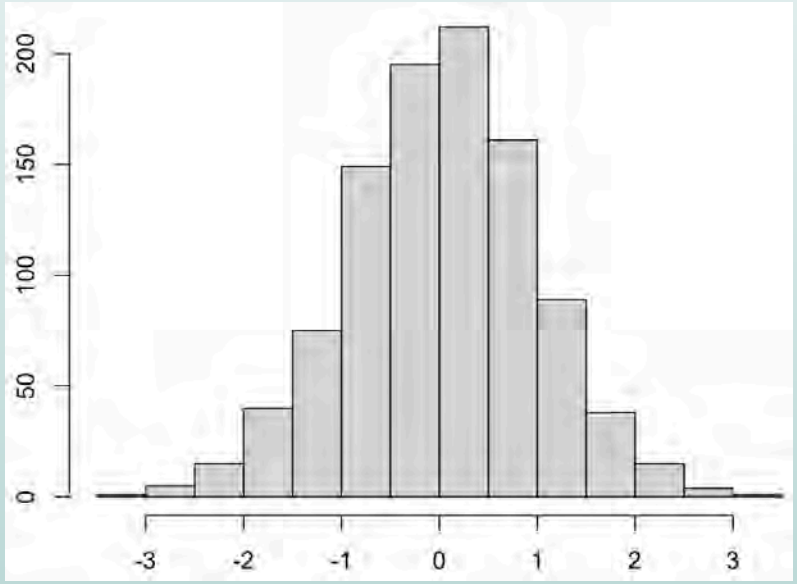
Your Turn 1

Consider the `covid_testing` data frame.

What do you think the plot would look like in which:

- the x-axis represents `pan_day` (day of the pandemic), and
- the y-axis represents the number of tests that were performed on that day?

Your Turn 2



What is the name of this kind of plot?

Type the answer into the chat!

Your Turn 3

Type the following code in the RStudio console to make a graph.

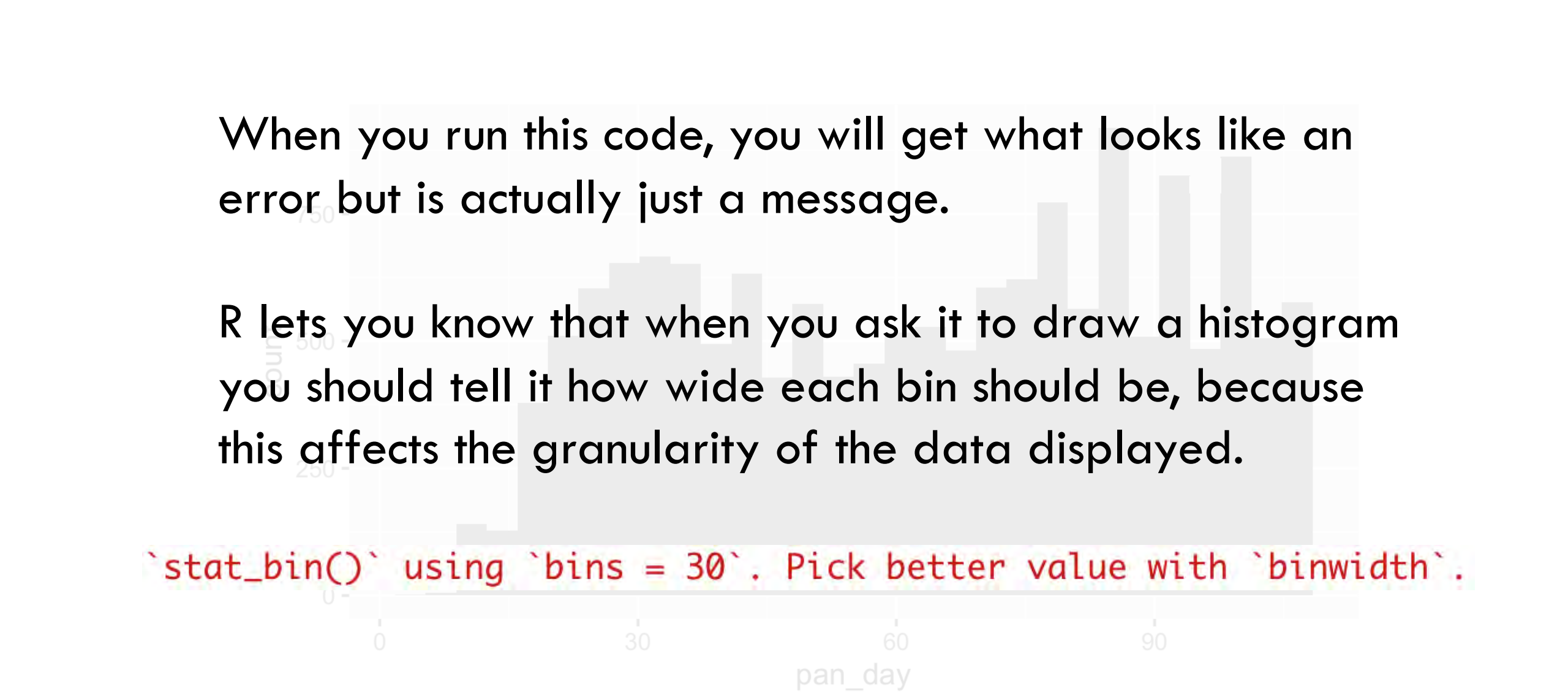
Pay attention to the spelling, capitalization, and parentheses!

```
ggplot(data = covid_testing) +
  geom_histogram(mapping = aes(x = pan_day))
```

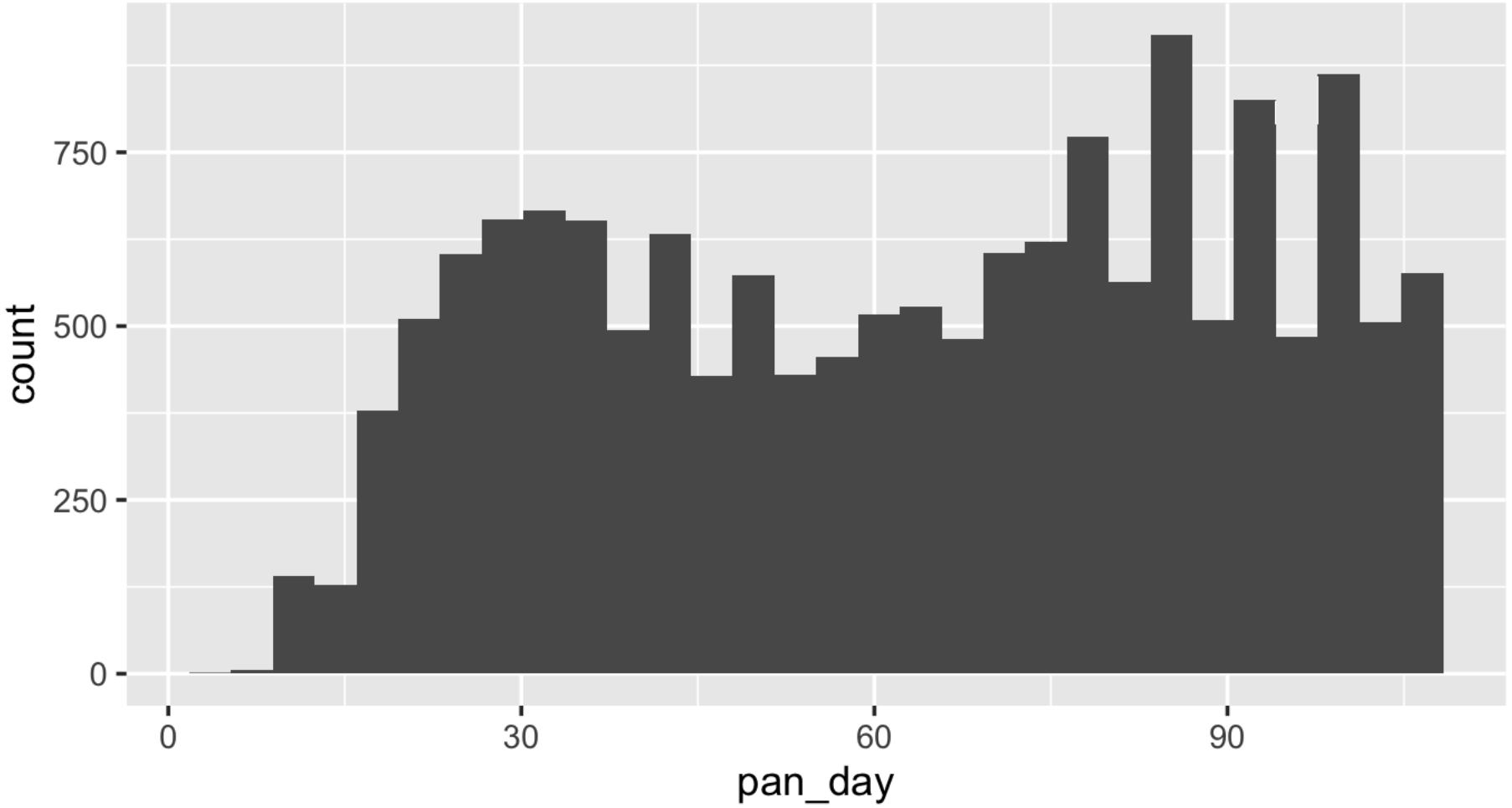
When you run this code, you will get what looks like an error but is actually just a message.

R lets you know that when you ask it to draw a histogram you should tell it how wide each bin should be, because this affects the granularity of the data displayed.

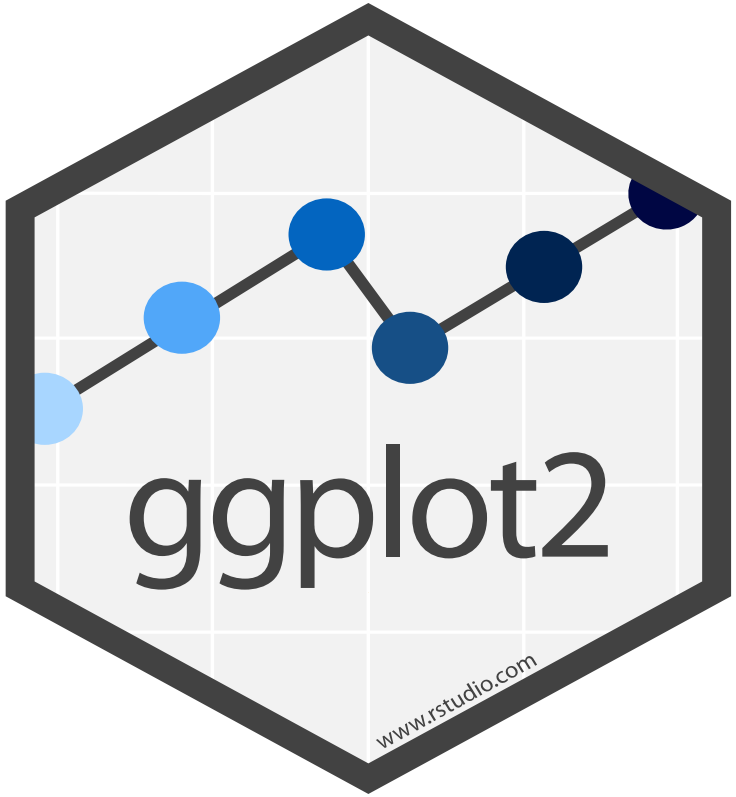
```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
ggplot(data = covid_testing) +
  geom_histogram(mapping = aes(x = pan_day))
```




```
ggplot(data = covid_testing) +
  geom_histogram(mapping = aes(x = pan_day))
```


ggplot2
www.rstudio.com

ggplot()

Always start with ggplot()

data frame

+ sign before new line

```
ggplot(data = covid_testing) +
  geom_histogram(mapping = aes(x = pan_day))
```

type of plot

mappings inside aes() function

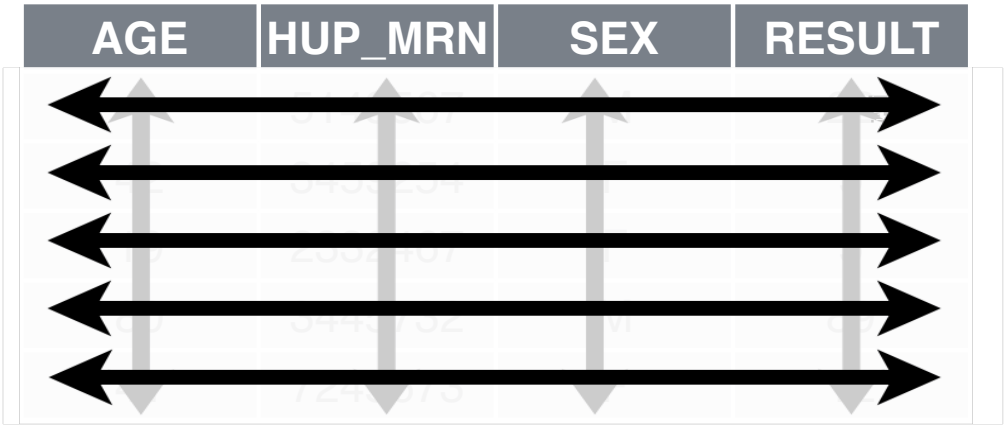
x axis mapping

To make **any** kind of graph:

1. Choose a "tidy" **data** frame

```
ggplot(data = data_frame) +
  geom_function(mapping = aes(mappings))
```

1. Pick a "Tidy" Data Frame



A data set is **tidy** if:

1. Each **variable** is in its own **column**
2. Each **observation** is in its own **row**
3. Each **value** is in its own **cell**

Wickham H. **Tidy Data.** *J Stat Soft* 2014.

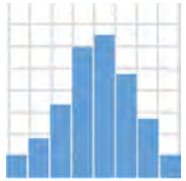
To make **any** kind of graph:

1. Choose a "tidy" **data** frame

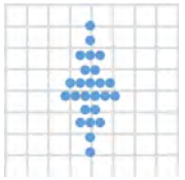
```
ggplot(data = data_frame) +
  geom_function(mapping = aes(mappings))
```

2. Pick a "**geom**" function

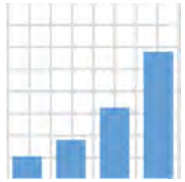
2. Choose a "Geom" Function



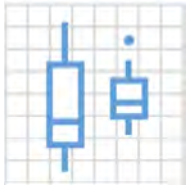
`geom_histogram()`



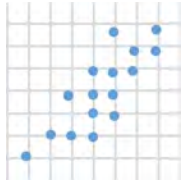
`geom_dotplot()`



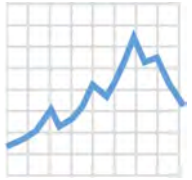
`geom_bar()`



`geom_boxplot()`



`geom_point()`

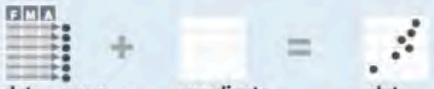


`geom_line()`

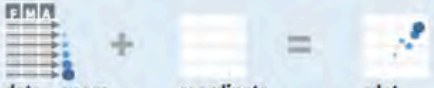
Data Visualization with ggplot2 : : CHEAT SHEET

Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same components: a **data** set, a **coordinate system**, and geoms—visual marks that represent data points.



To display values, map variables in the data to visual properties of the geom (**aesthetics**) like **size**, **color**, and **x** and **y** locations.



Complete the template below to build a graph.

required

```
ggplot (data = <DATA>) +
<GEOM_FUNCTION> (mapping = aes( <MAPPINGS> ),
 stat = <STAT> , position = <POSITION> ) +
<COORDINATE_FUNCTION> +
<FACET_FUNCTION> +
<SCALE_FUNCTION> +
<THEME_FUNCTION>
```

Not required, sensible defaults supplied

ggplot(data = mpg, **aes**(x = cty, y = hwy)) Begins a plot that you finish by adding layers to. Add one geom function per layer.

aesthetic mappings | data | geom

```
qplot(x = cty, y = hwy, data = mpg, geom = "point")
```

Geoms

Use a geom function to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

GRAPHICAL PRIMITIVES

```
a <- ggplot(economics, aes(date, unemploy))
b <- ggplot(seals, aes(x = long, y = lat))
```

a + geom_blank()
(Useful for expanding limits)

b + geom_curve(aes(yend = lat + 1, xend=long+1,curvature=z)) - x, xend, y, yend, alpha, angle, color, curvature, linetype, size

a + geom_path(lineend="butt", linejoin="round", linemitre=1)
x, y, alpha, color, group, linetype, size

a + geom_polygon(aes(group = group))
x, y, alpha, color, fill, group, linetype, size

b + geom_rect(aes(xmin = long, ymin=lat, xmax= long + 1, ymax = lat + 1)) - xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

a + geom_ribbon(aes(ymin=unemploy - 900, ymax=unemploy + 900)) - x, ymax, ymin, alpha, color, fill, group, linetype, size

LINE SEGMENTS

common aesthetics: x, y, alpha, color, linetype, size

b + geom_abline(aes(intercept=0, slope=1))
b + geom_hline(aes(yintercept = lat))
b + geom_vline(aes(xintercept = long))

b + geom_segment(aes(yend=lat+1, xend=long+1))
b + geom_spoke(aes(angle = 1:1155, radius = 1))

ONE VARIABLE continuous

```
c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)
```

c + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size

c + geom_density(kernel = "gaussian")
x, y, alpha, color, fill, group, linetype, size, weight

c + geom_dotplot()
x, y, alpha, color, fill

TWO VARIABLES

continuous x , continuous y

```
e <- ggplot(mpg, aes(cty, hwy))
```

e + geom_label(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE) x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

e + geom_jitter(height = 2, width = 2)
x, y, alpha, color, fill, shape, size

e + geom_point(), x, y, alpha, color, fill, shape, size, stroke

e + geom_quantile(), x, y, alpha, color, group, linetype, size, weight

e + geom_rug(sides = "bl"), x, y, alpha, color, linetype, size

e + geom_smooth(method = lm), x, y, alpha, color, fill, group, linetype, size, weight

e + geom_text(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE), x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

discrete x , continuous y

```
f <- ggplot(mpg, aes(class, hwy))
```

f + geom_col(), x, y, alpha, color, fill, group, linetype, size

f + geom_boxplot(), x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight

f + geom_dotplot(binaxis = "y", stackdir = "center"), x, y, alpha, color, fill, group

f + geom_violin(scale = "area"), x, y, alpha, color, fill, group, linetype, size, weight

continuous bivariate distribution

```
h <- ggplot(diamonds, aes(carat, price))
```

h + geom_bin2d(binwidth = c(0.25, 500))
x, y, alpha, color, fill, linetype, size, weight

h + geom_density2d()
x, y, alpha, colour, group, linetype, size

h + geom_hex()
x, y, alpha, colour, fill, size

continuous function

```
i <- ggplot(economics, aes(date, unemploy))
```

i + geom_area()
x, y, alpha, color, fill, linetype, size

i + geom_line()
x, y, alpha, color, group, linetype, size

i + geom_step(direction = "hv")
x, y, alpha, color, group, linetype, size

visualizing error

```
df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
j <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))
```

j + geom_crossbar(fatten = 2)
x, y, ymax, ymin, alpha, color, fill, group, linetype, size

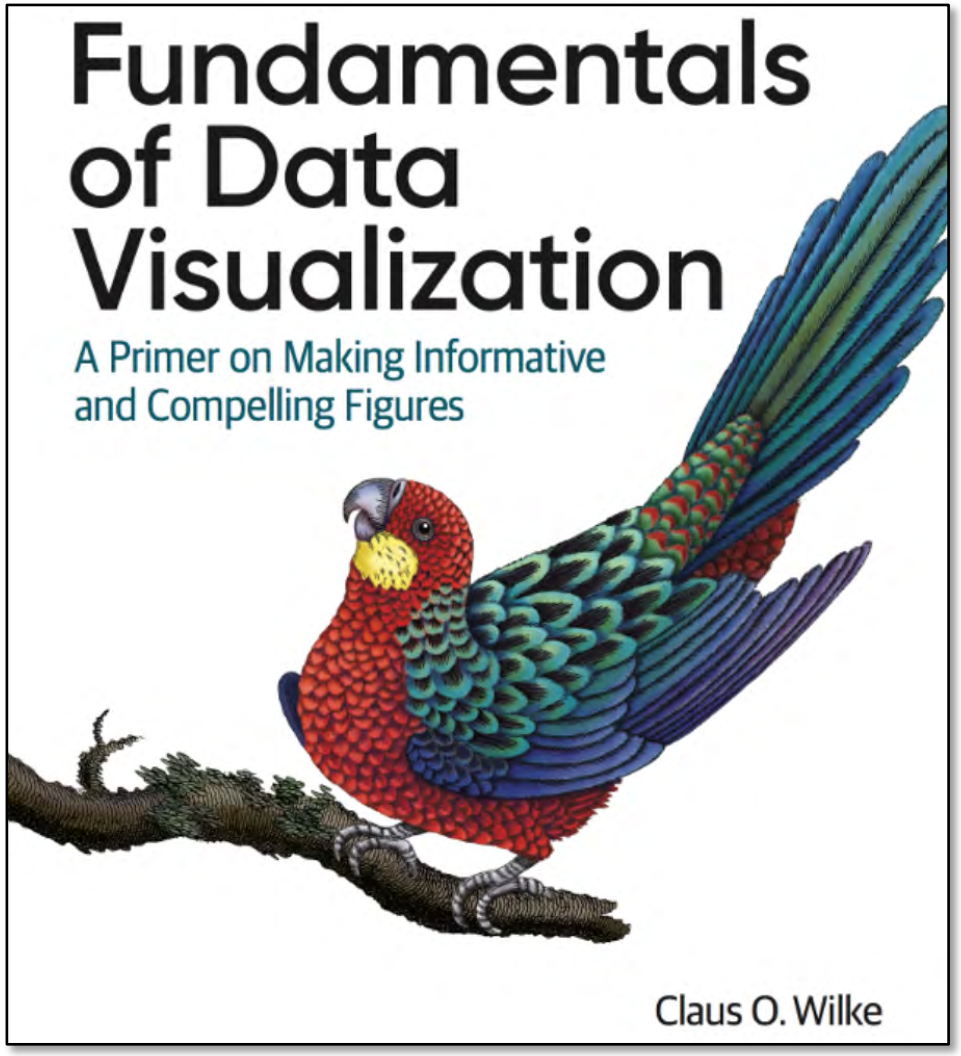
j + geom_errorbar(), x, ymax, ymin, alpha, color, group, linetype, size, width (also **geom_errorbarh()**)

j + geom_linerange()
x, ymin, ymax, alpha, color, group, linetype, size

j + geom_pointrange()
x, y, ymin, ymax, alpha, color, fill, group, linetype, shape, size

maps

```
data <- data.frame(murder = USArrests$Murder
```



https://serialmentor.com/dataviz

To make **any** kind of graph:

1. Choose a "tidy" **data** frame

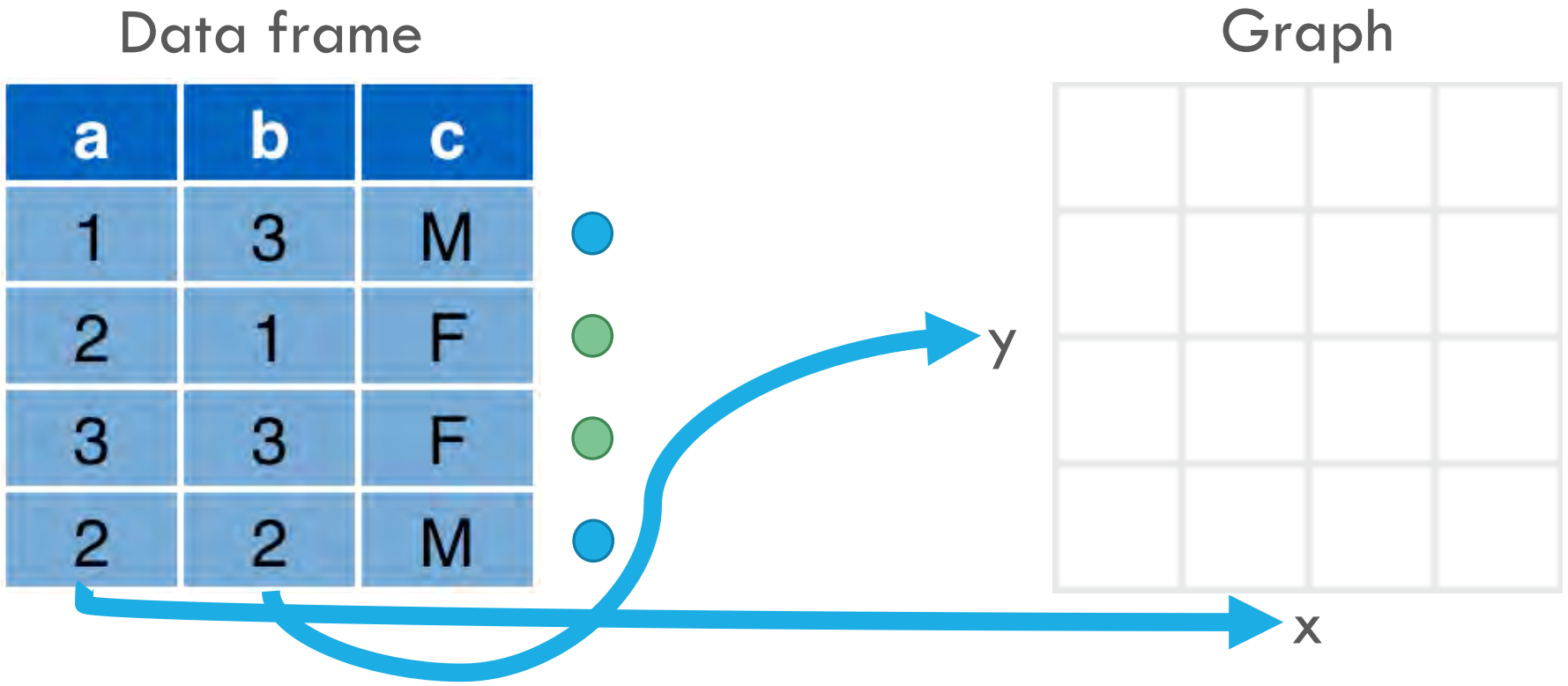
```
ggplot(data = data_frame) +
  geom_function(mapping = aes(mappings))
```

2. Pick a "**geom**" function

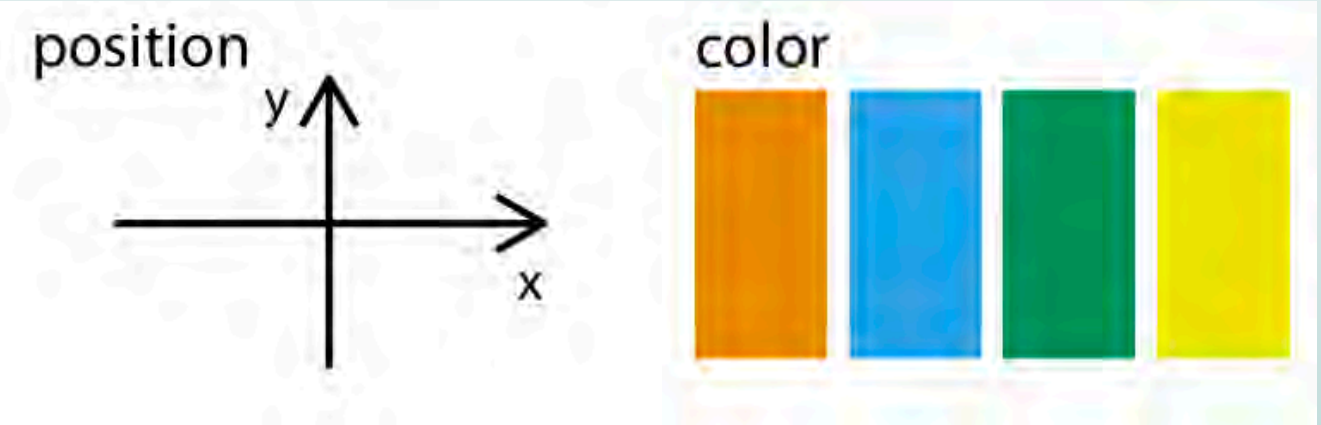
3. Write aesthetic **mappings**

3. Write Aesthetic Mappings

```
aes(x = a, y = b, color = c)
```



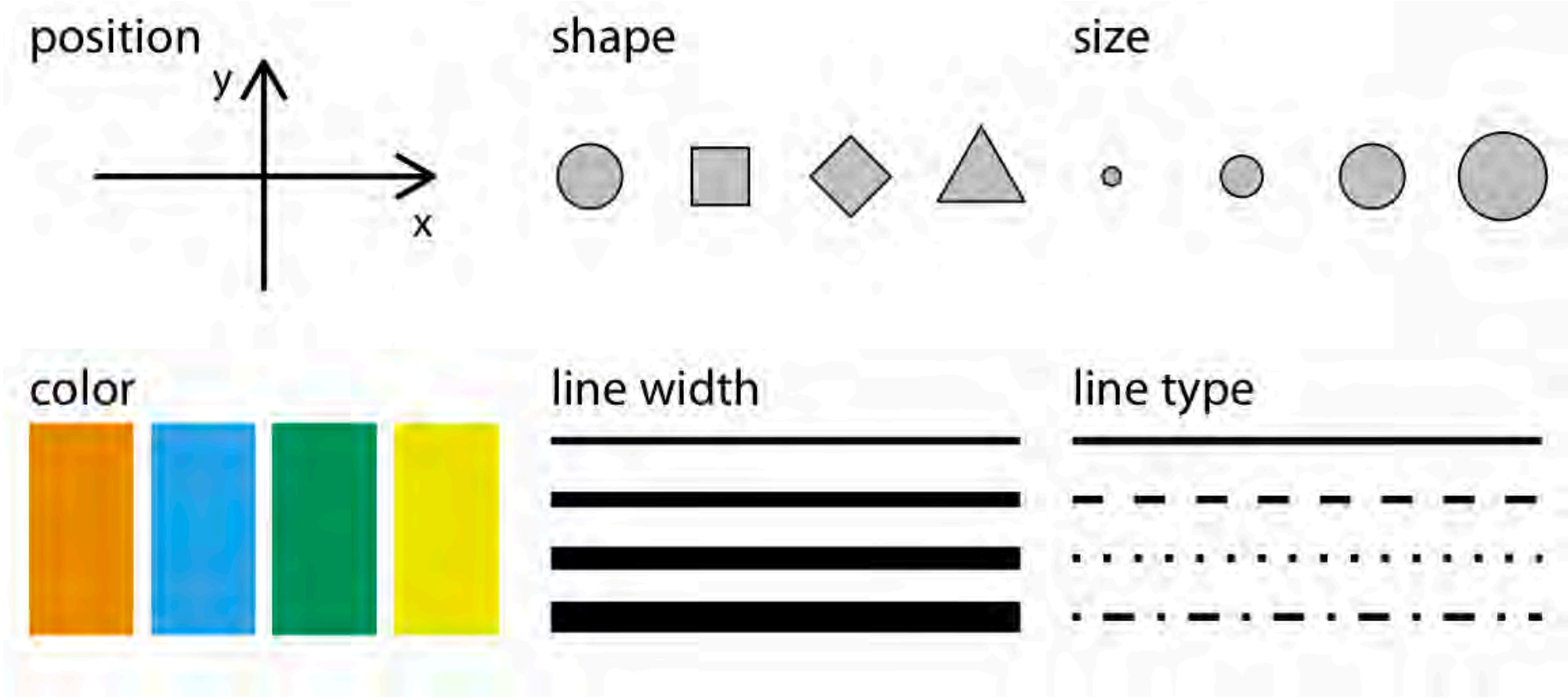
Your Turn 4



In addition to x/y position and color, what other aesthetics can you think of?

Type your answers in the chat!

Aesthetics



From Fundamentals of Data Visualization, by Claus Wilke, licensed under CC-BY-NC-ND

To make **any** kind of graph:

1. Choose a "tidy" **data** frame

```
ggplot(data = data_frame) +
  geom_function(mapping = aes(mappings))
```

2. Pick a "**geom**" function

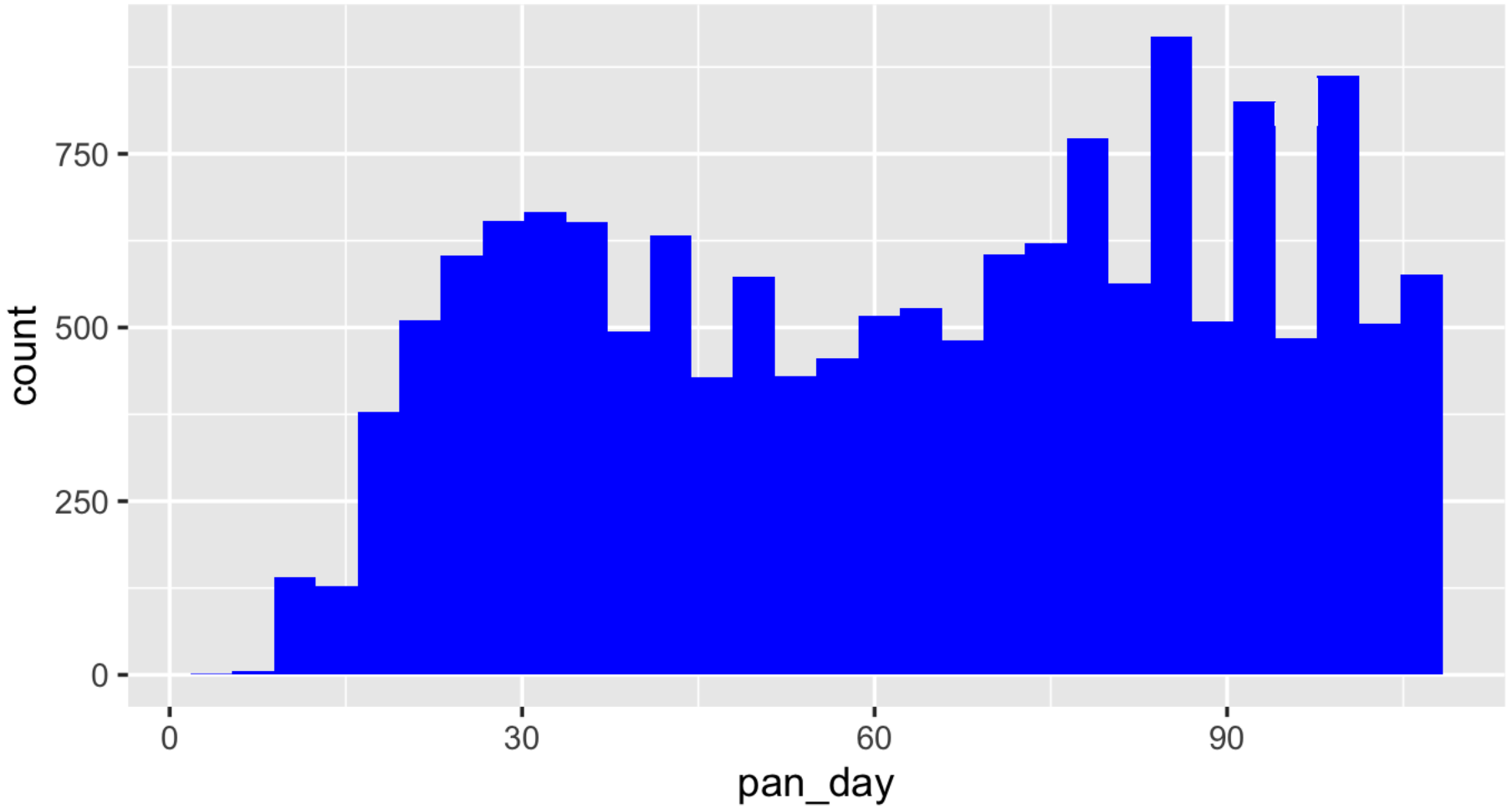
3. Write aesthetic **mappings**

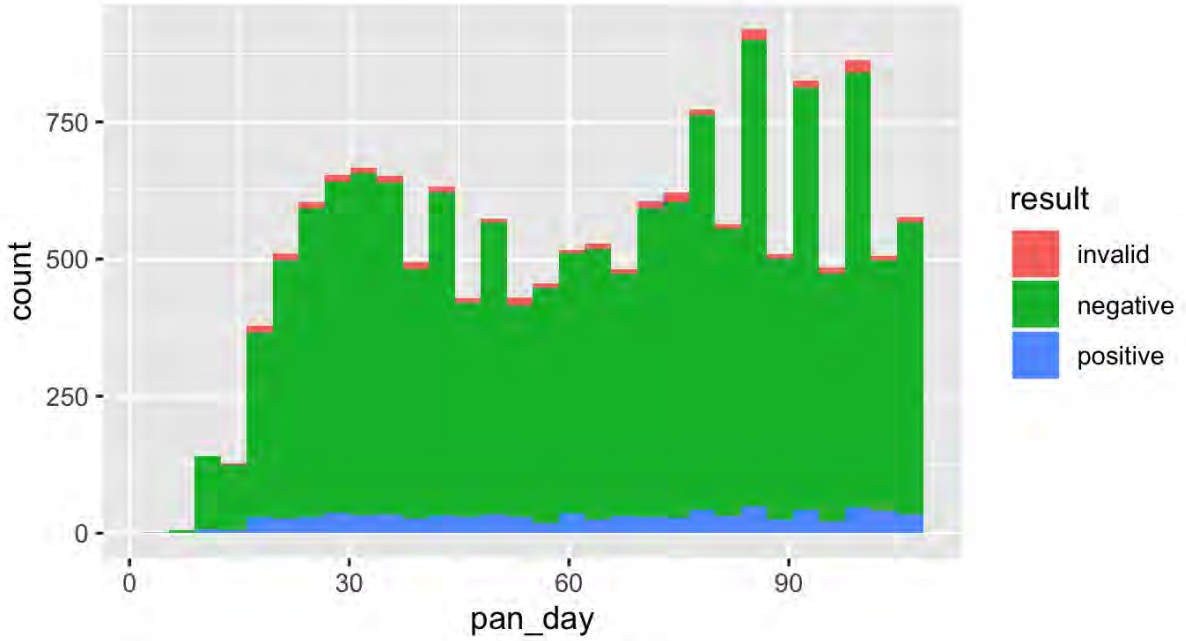
Your Turn 5

Open `03-Visualize.Rmd`. Work through the exercises of the section titled "Your Turn 5."

Setting vs Mapping Aesthetics

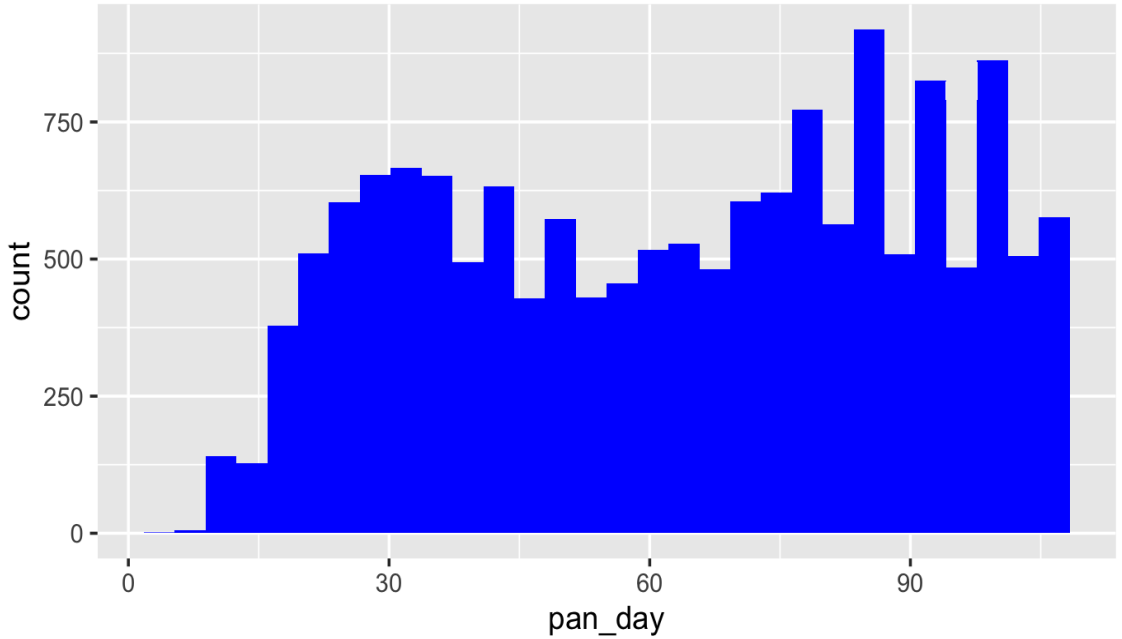
How would you make this plot?





Inside of aes():
map an aesthetic
to a variable

```
ggplot(data = covid_testing) +
  geom_histogram(mapping = aes(x = pan_day, fill = result))
```

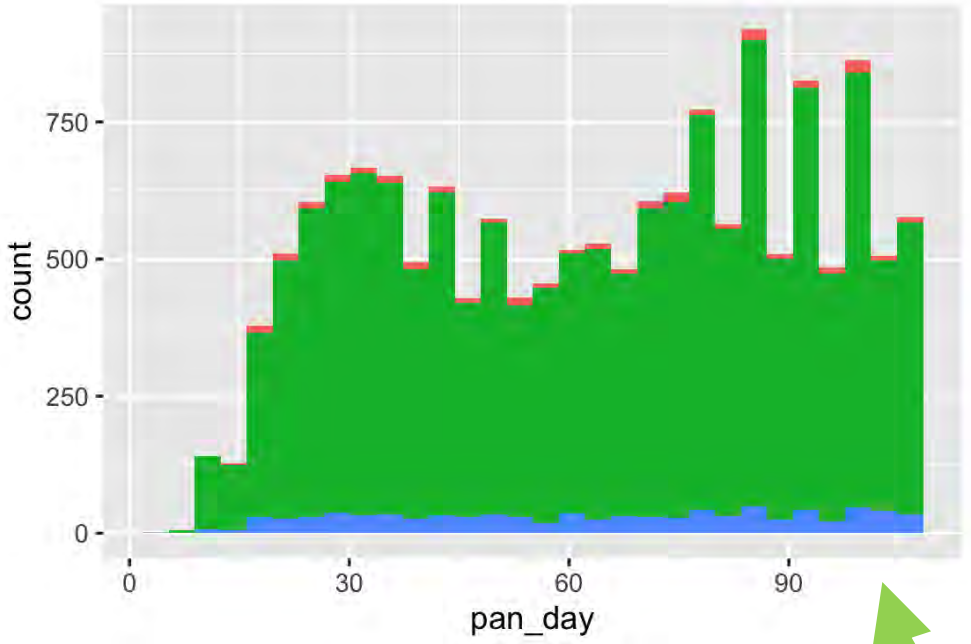


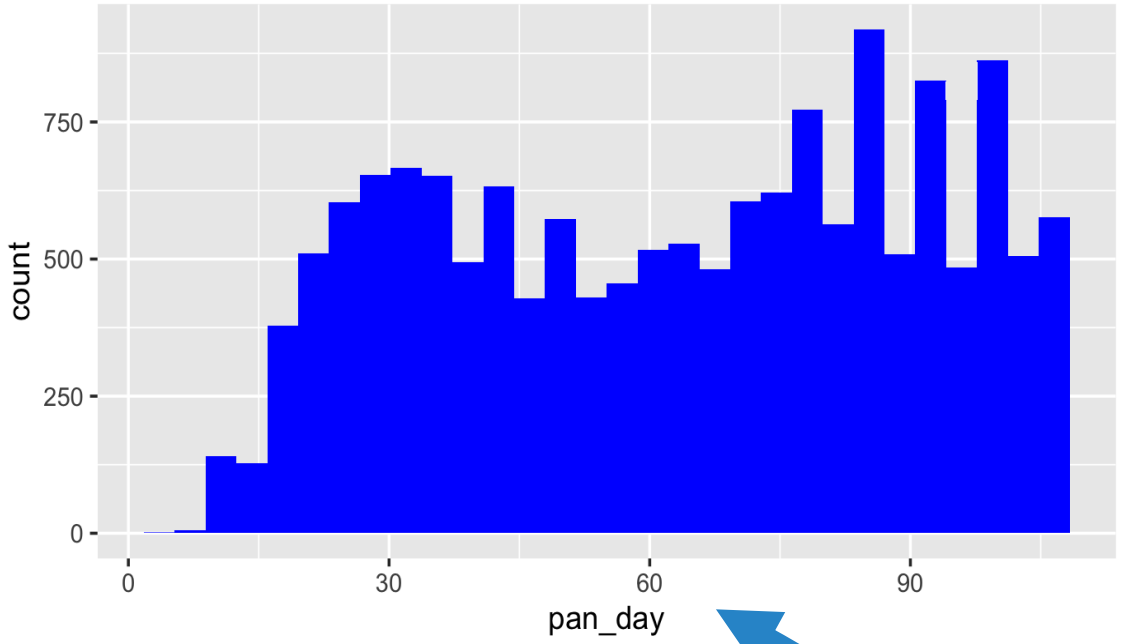
Outside of **aes()**:

set an aesthetic to a **value**

color name in "quotes"

```
ggplot(data = covid_testing) +
  geom_histogram(mapping = aes(x = pan_day), fill = "blue")
```



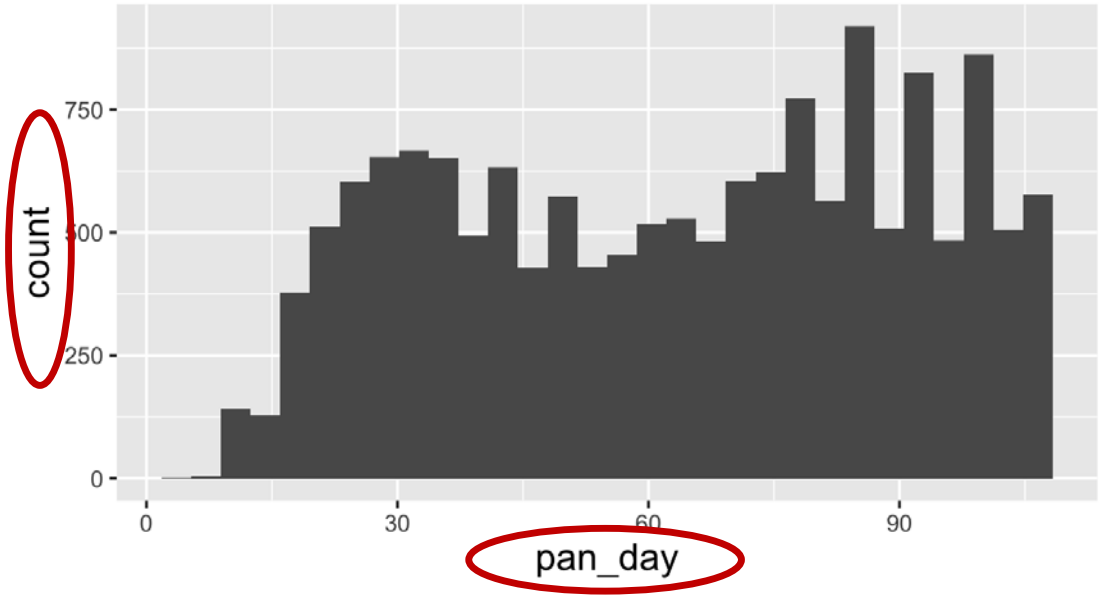


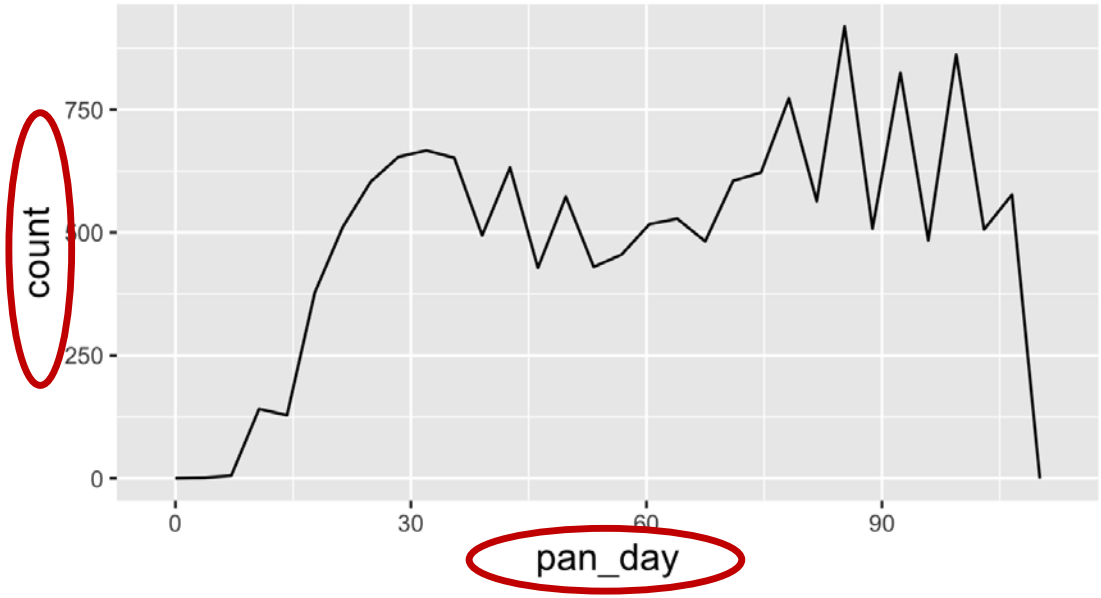

```
ggplot(data = covid_testing) +
  geom_histogram(mapping = aes(x = pan_day, fill = result))
```

```
ggplot(data = covid_testing) +
  geom_histogram(mapping = aes(x = pan_day), fill = "blue")
```

Geom Functions

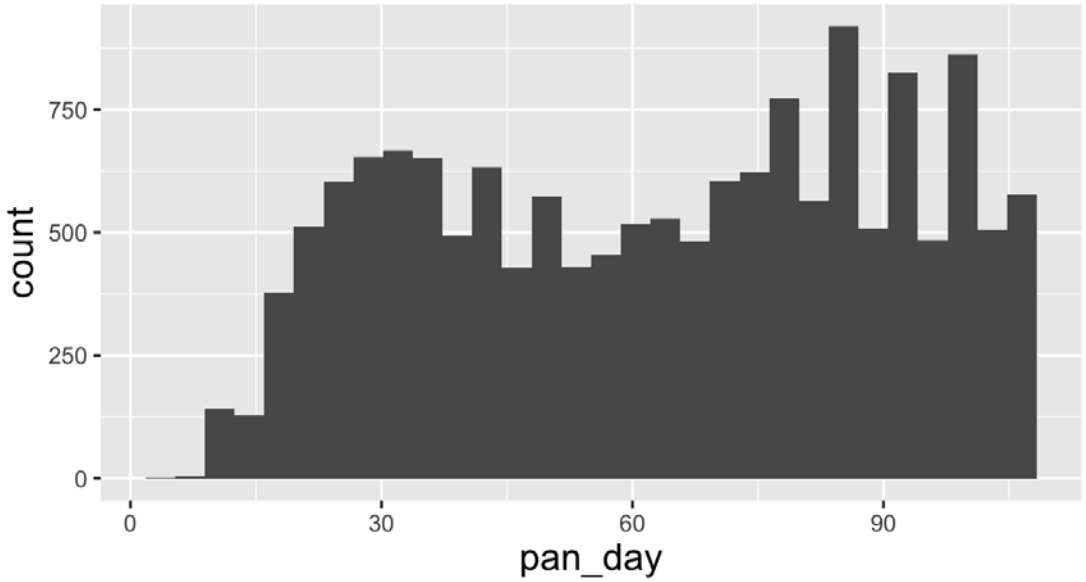
How are these plots similar?

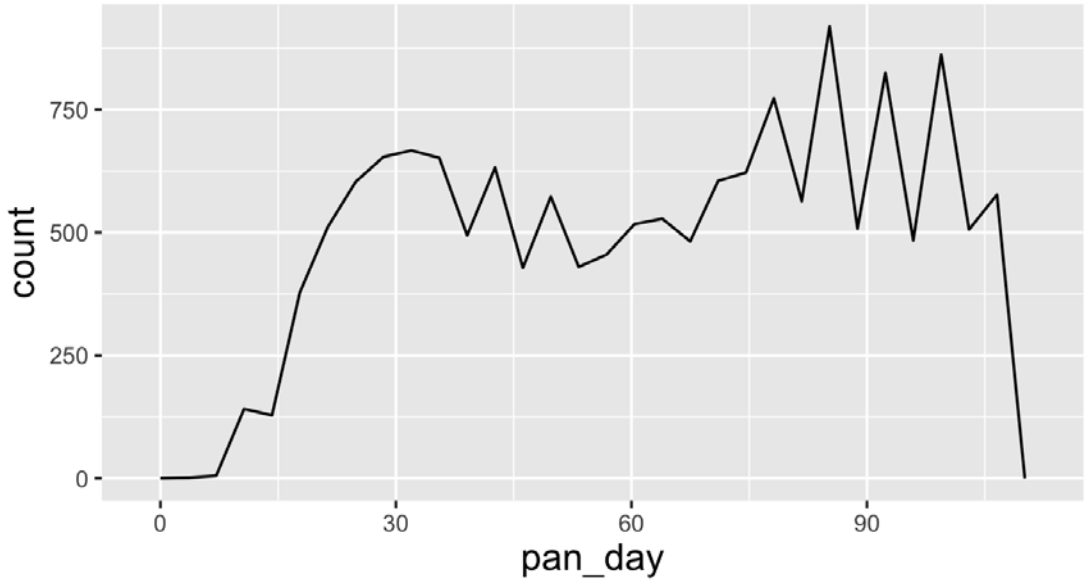




Same: x axis, y axis, data

How are these plots different?



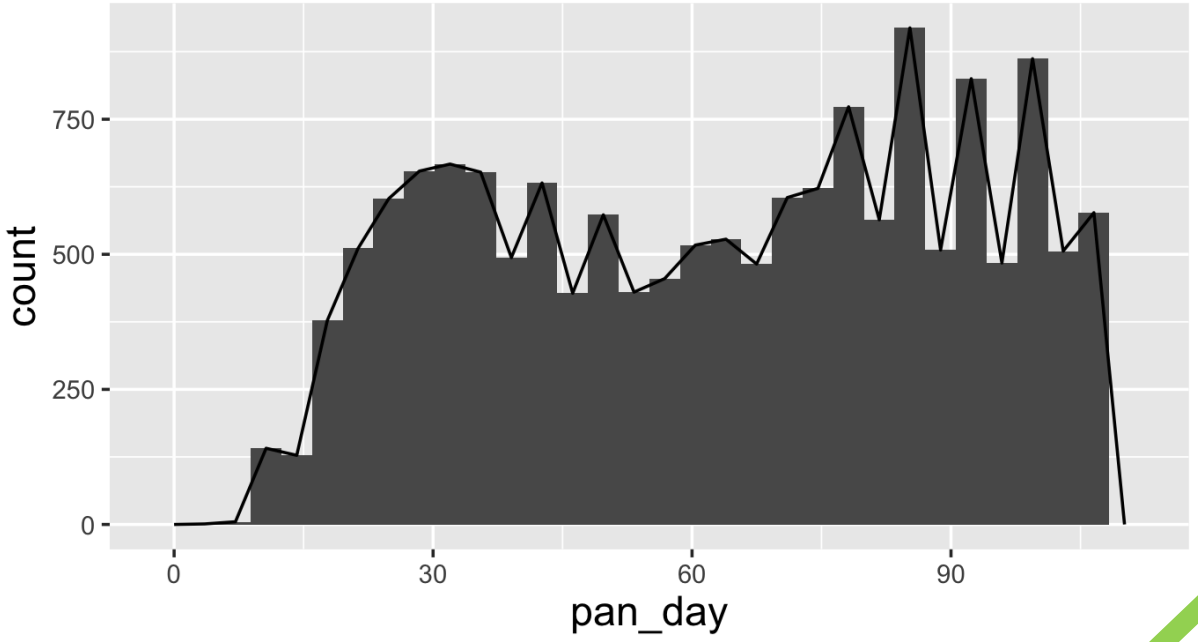


Different **geometric object** ("geom") used to represent the data

Your Turn 6

Open `03-Visualize.Rmd`. Work through the exercises of the section titled "Your Turn 6."

Global vs Local Settings



Inside of geom_ function: apply locally to only the current layer

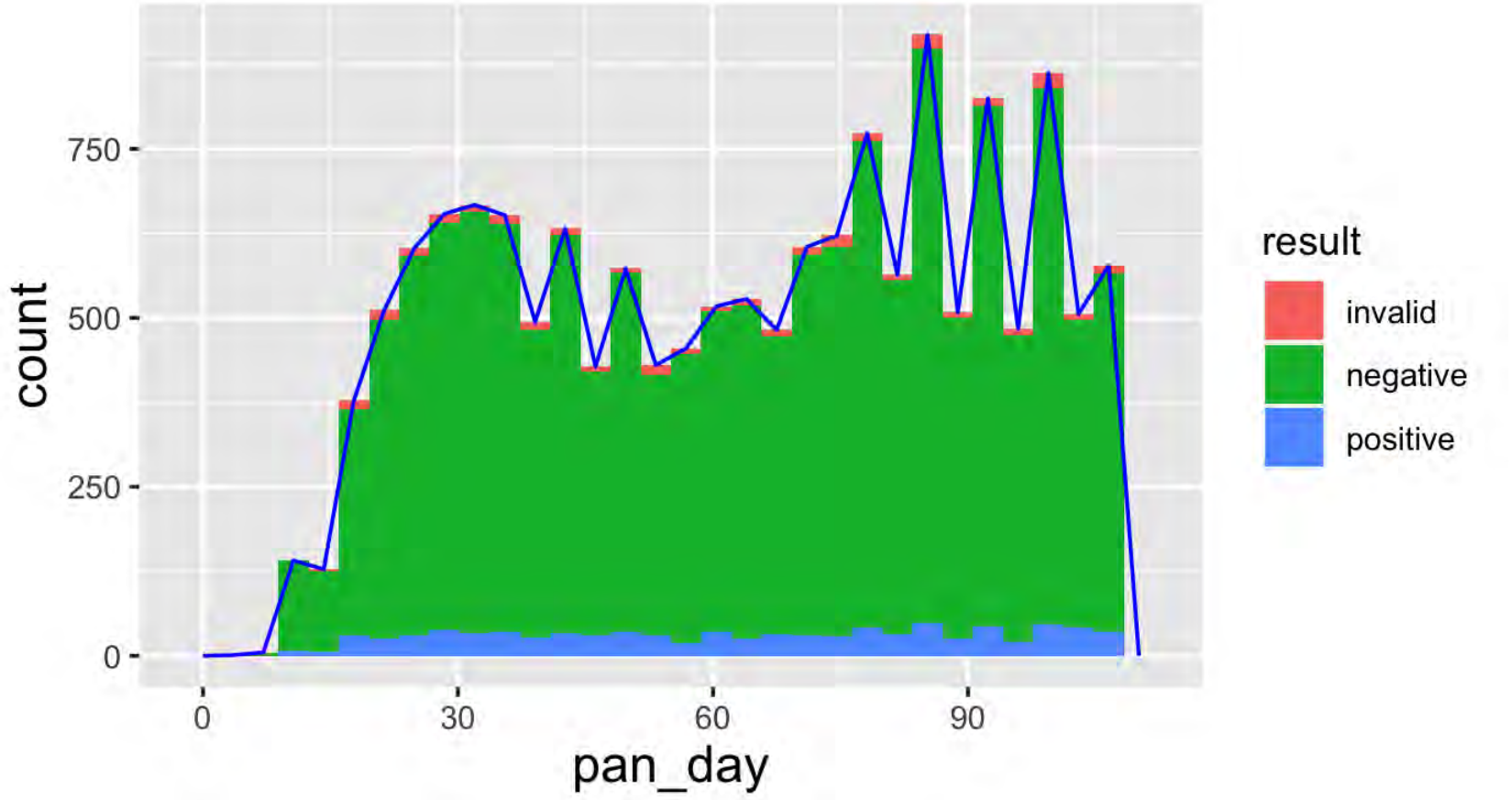
Inside of ggplot(): apply globally to every layer

```
ggplot(data = covid_testing) +
  geom_histogram(mapping = aes(x = pan_day)) +
  geom_freqpoly(mapping = aes(x = pan_day))
```

```
ggplot(data = covid_testing, mapping = aes(x = pan_day)) +
  geom_histogram() +
  geom_freqpoly()
```

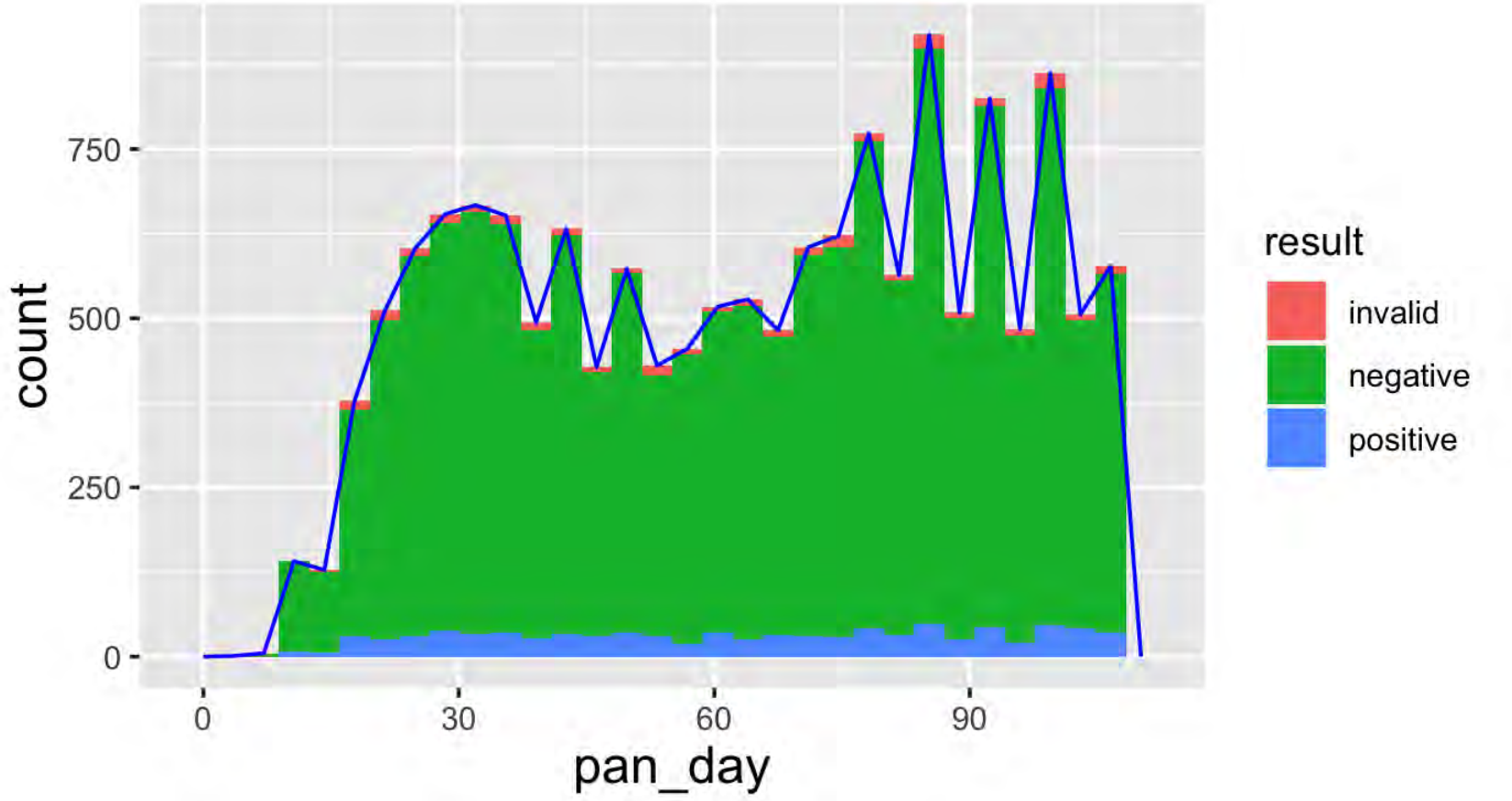
empty ()

How would you make this plot?



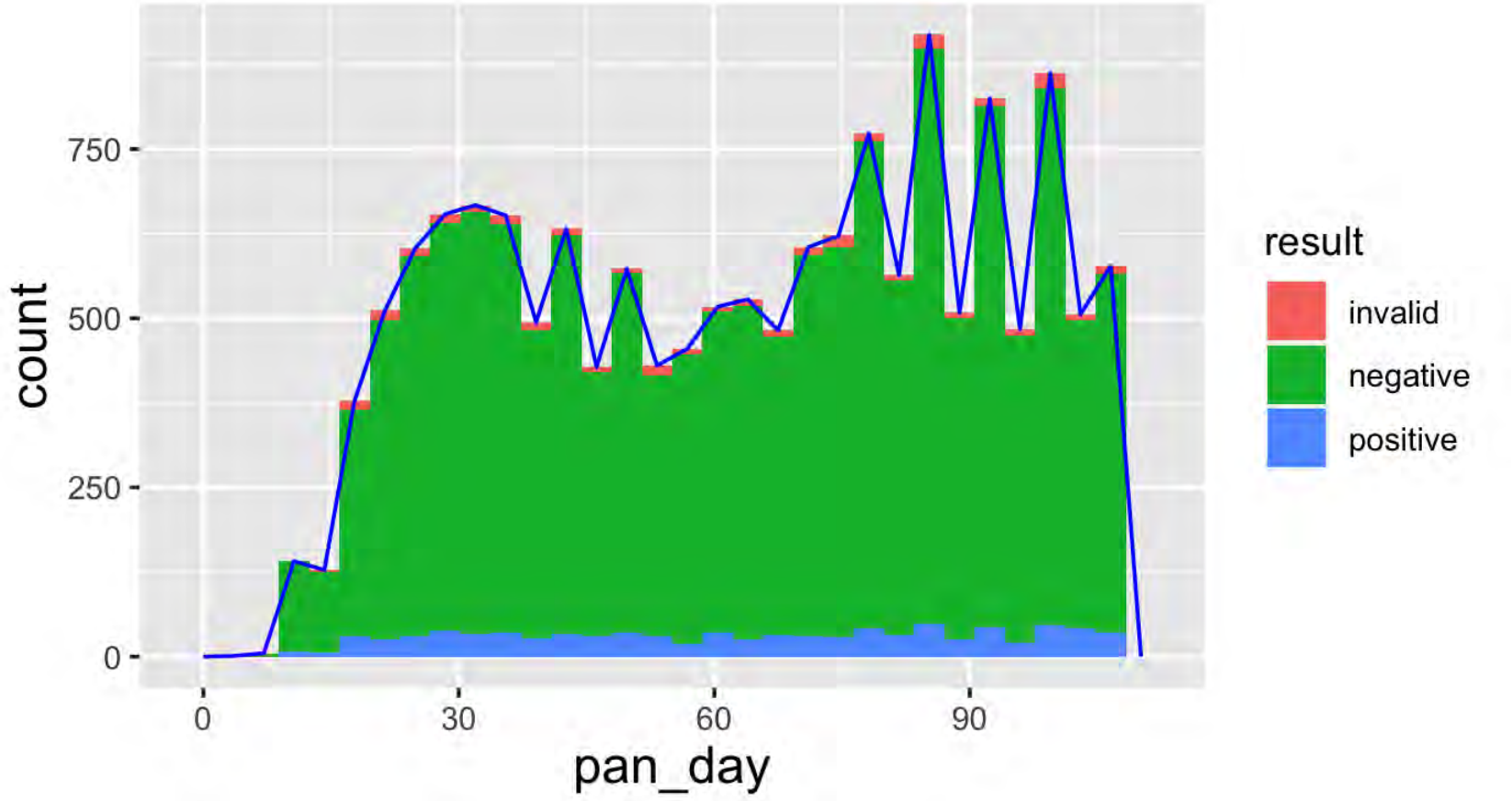

```
ggplot(
```

How would you make this plot?



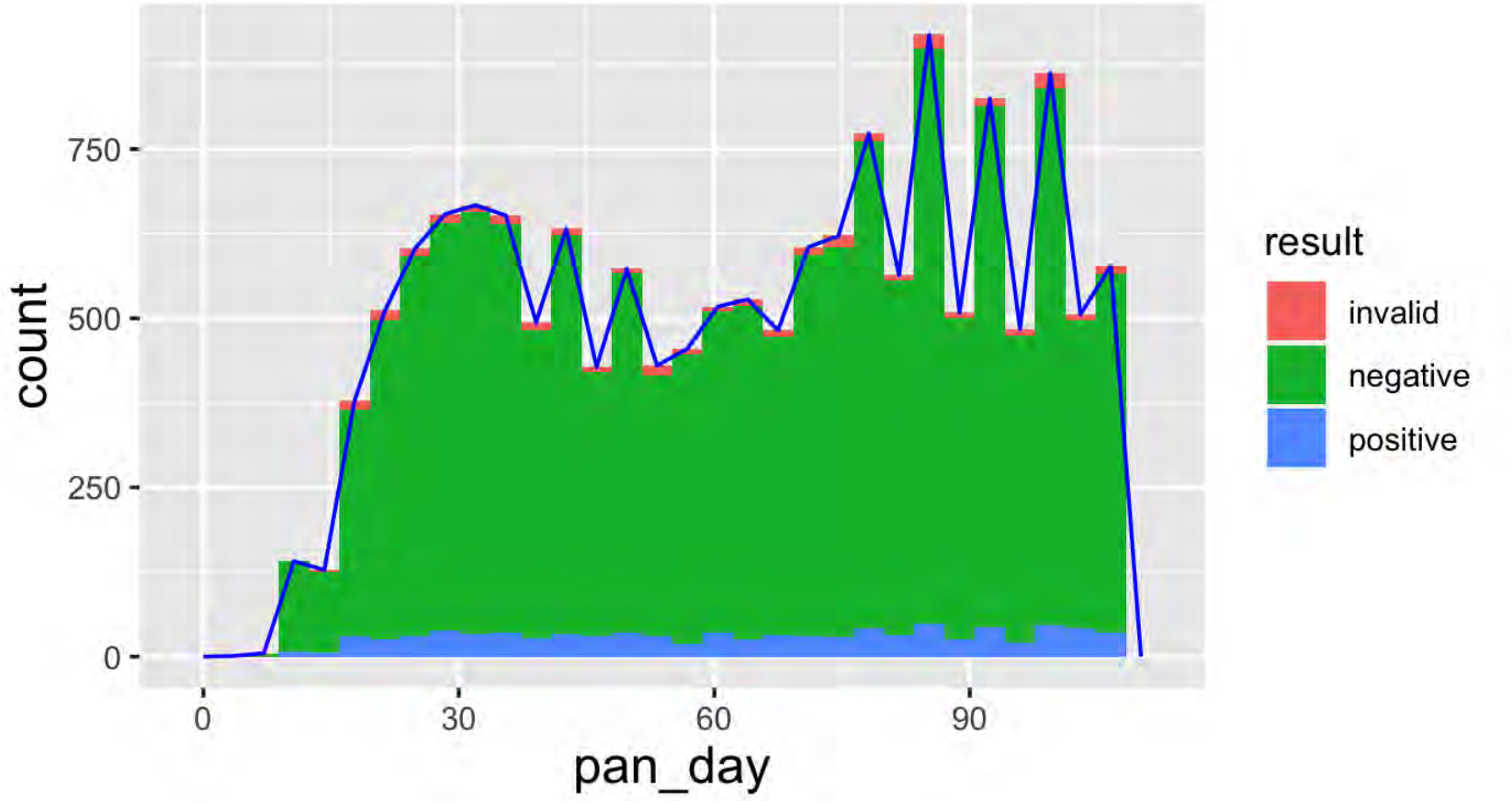

```
ggplot(data = covid_testing
```

How would you make this plot?



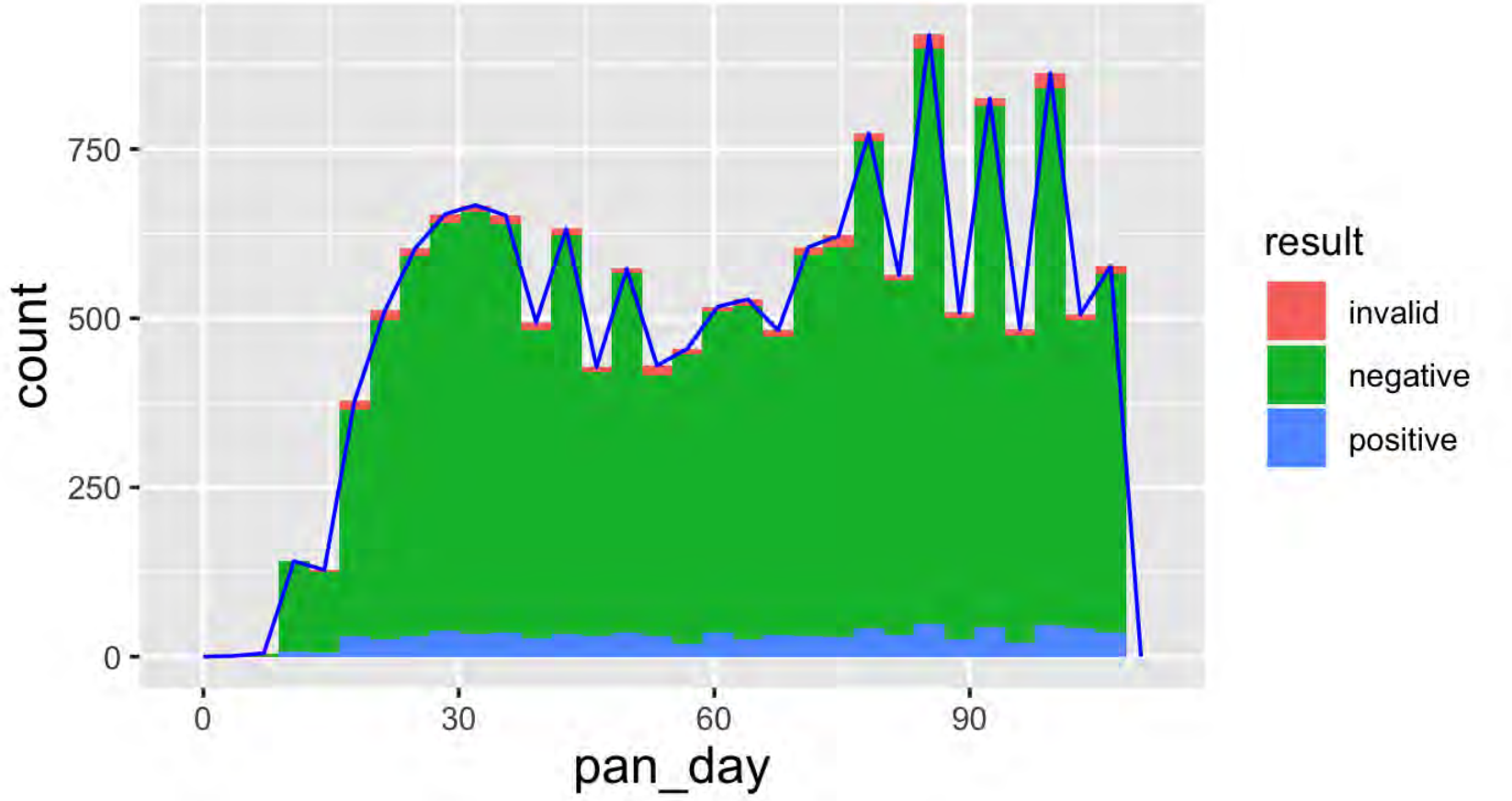

```
ggplot(data = covid_testing, mapping = aes(x = pan_day)) +
  geom_histogram(
```

How would you make this plot?




```
ggplot(data = covid_testing, mapping = aes(x = pan_day)) +
  geom_histogram(mapping = aes(fill = result)) +
  geom_freqpoly(
```

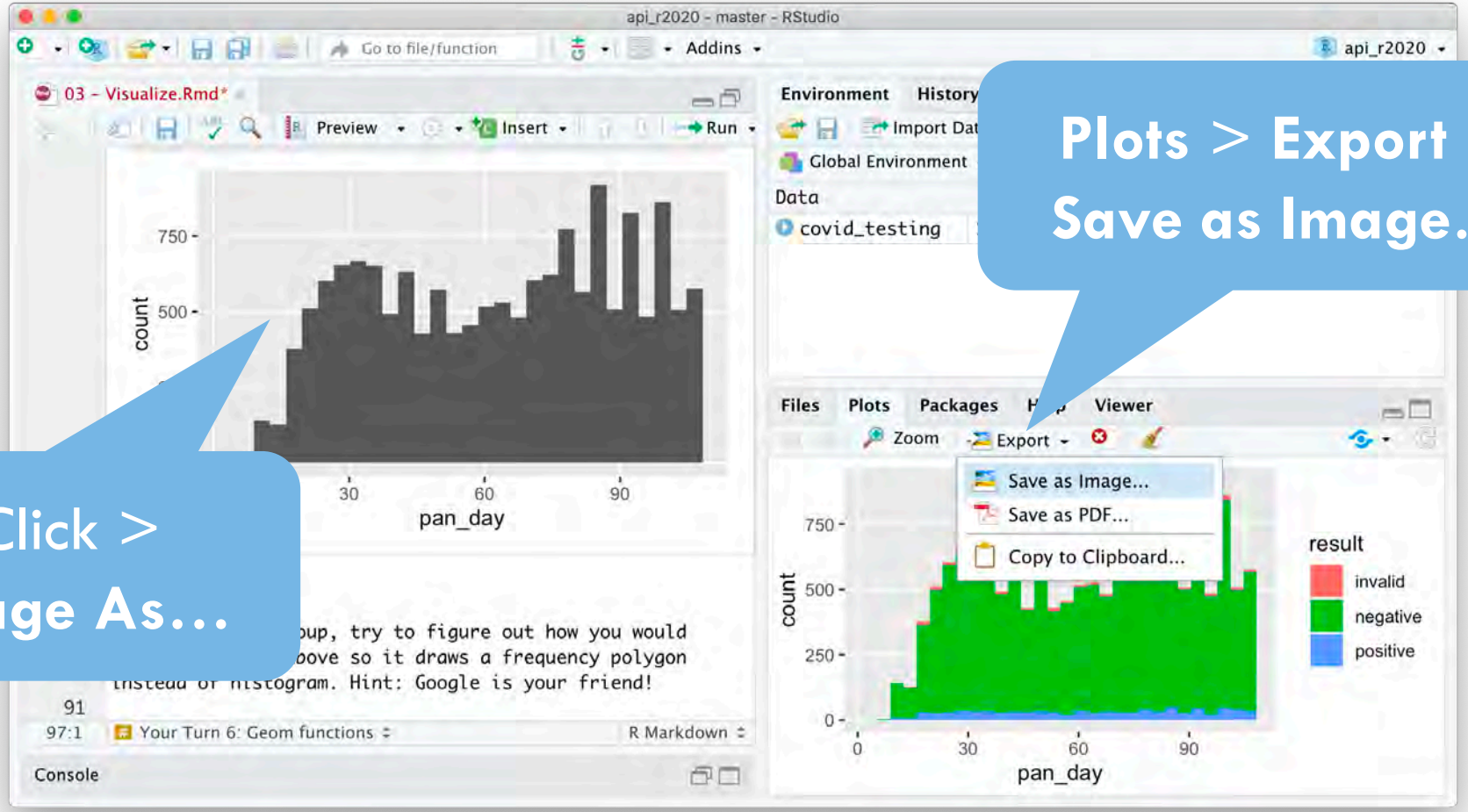
How would you make this plot?




```
ggplot(data = covid_testing, mapping = aes(x = pan_day)) +
  geom_histogram(mapping = aes(fill = result)) +
  geom_freqpoly(color = "blue")
```

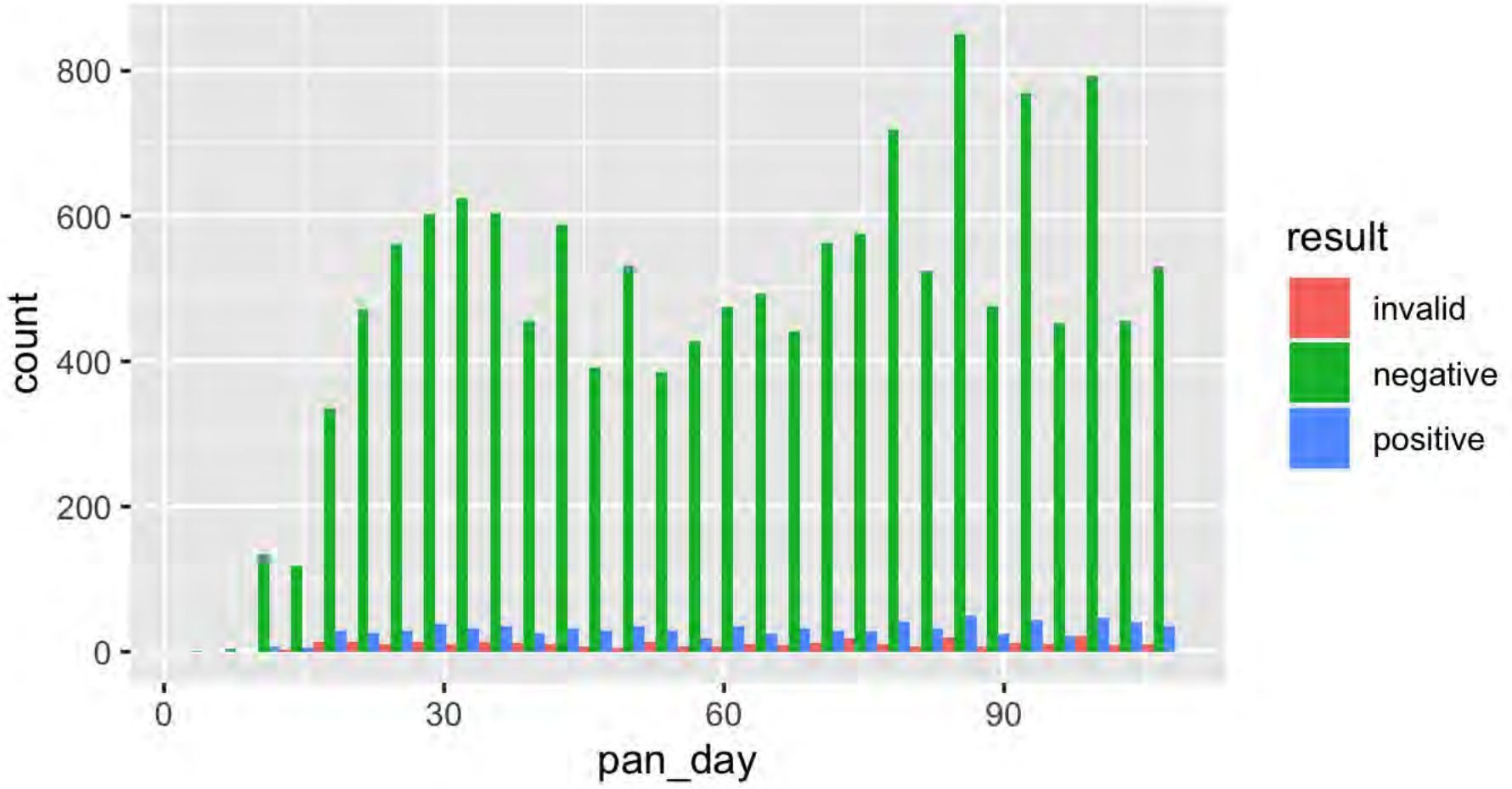
What Else?

Manually saving plots



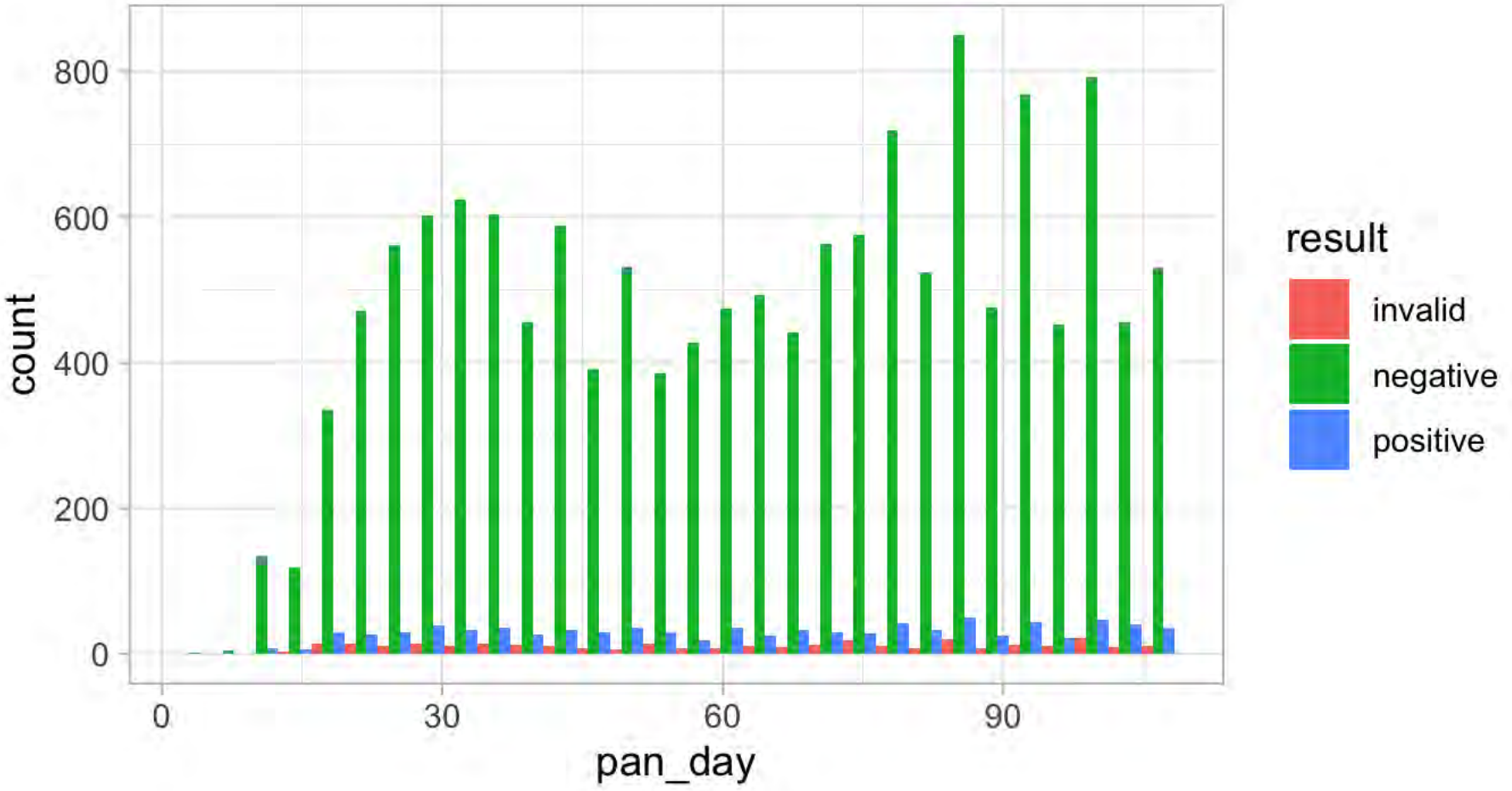
Position adjustments

How overlapping objects are arranged



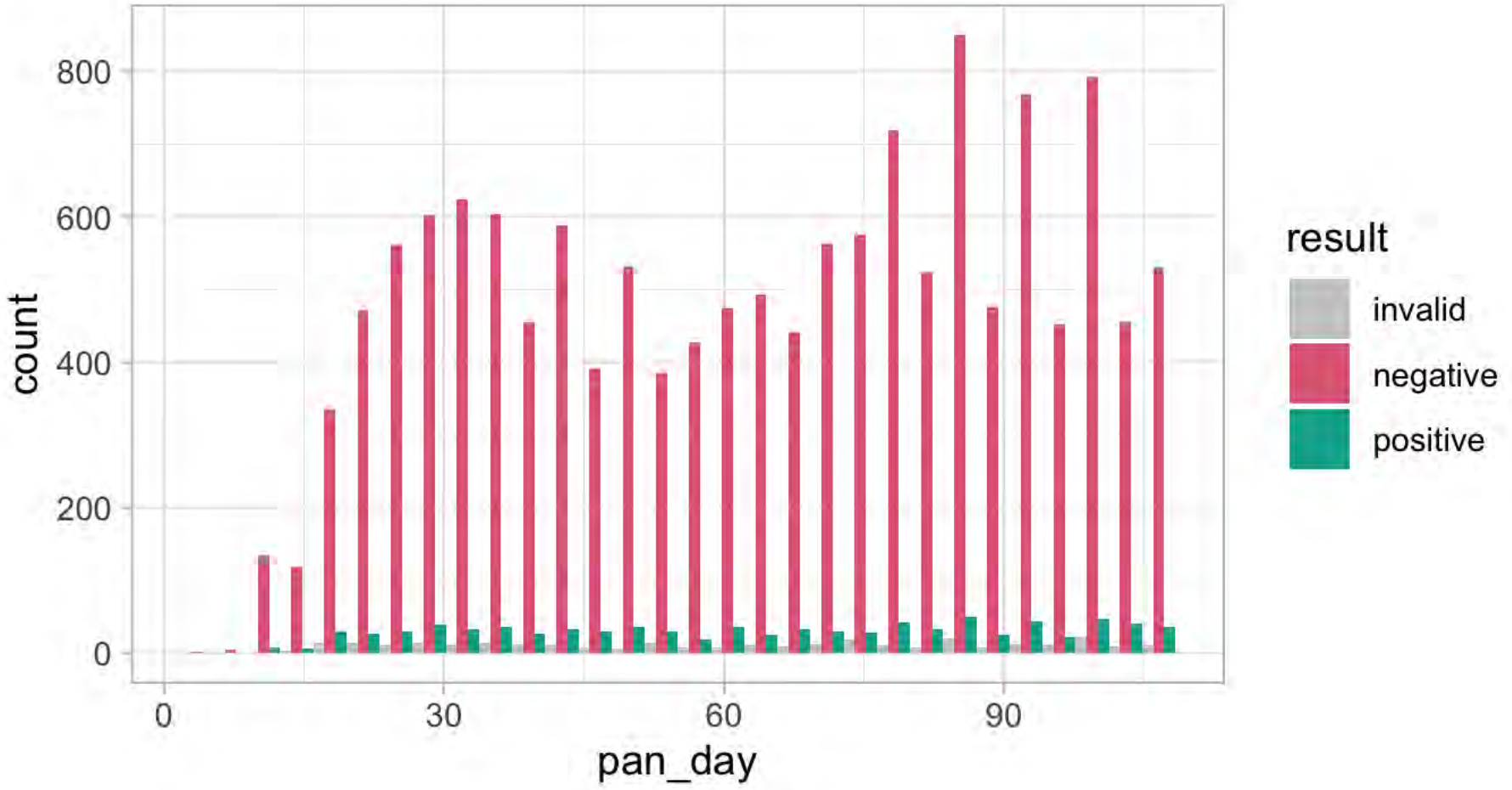
Themes

Visual appearance of non-data elements



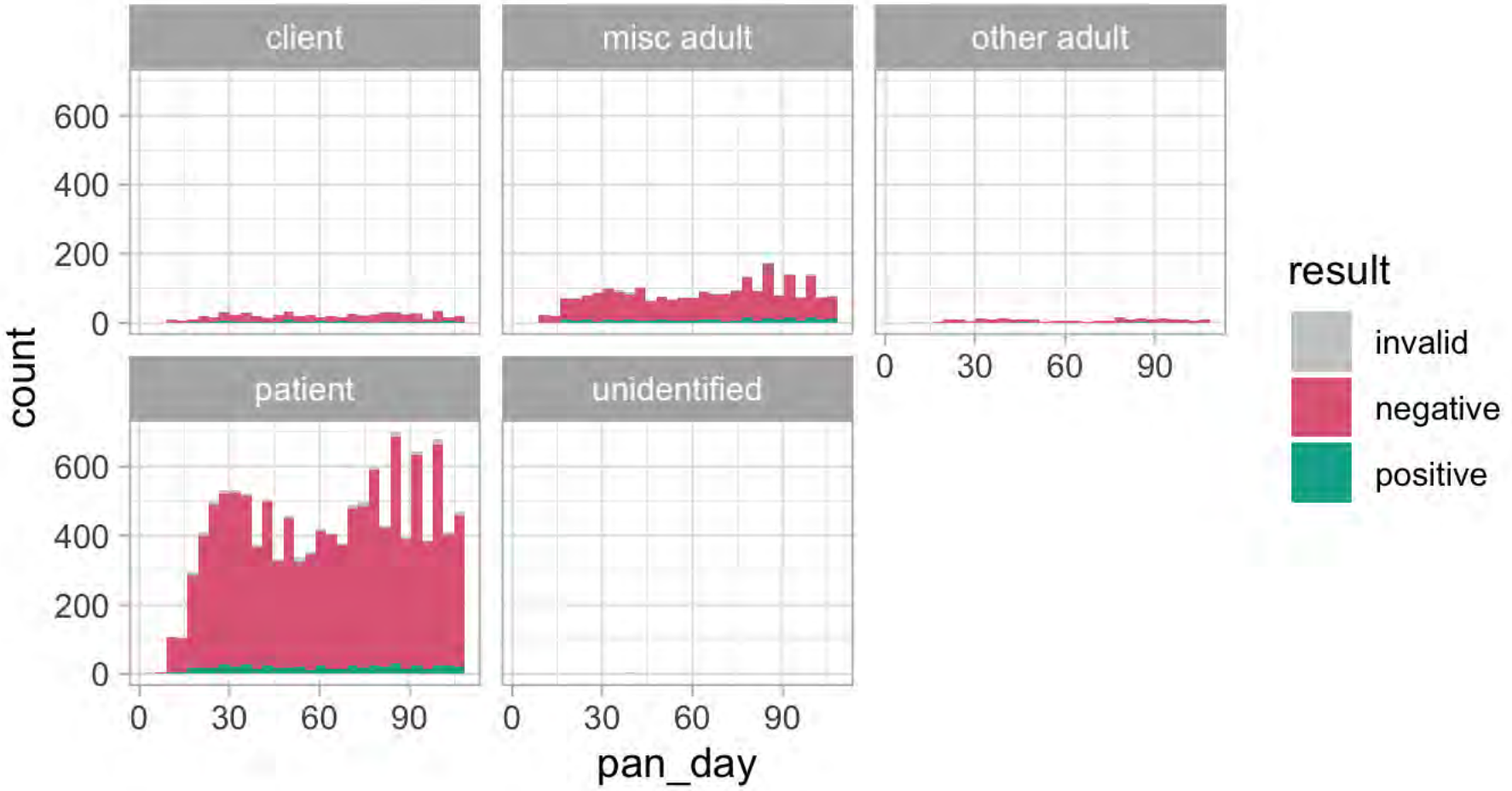
Scales

Customize color scales and other mappings

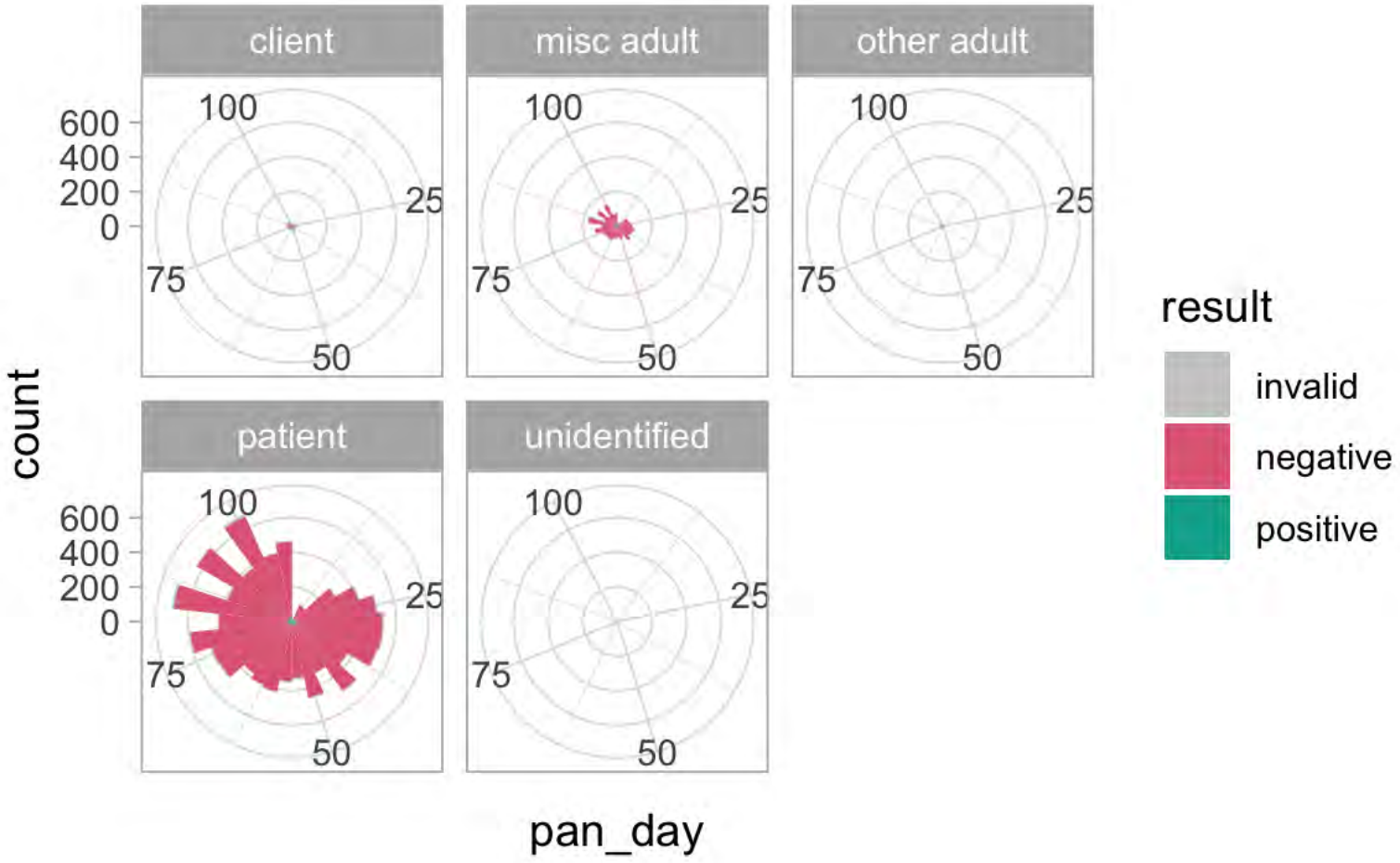


Facets

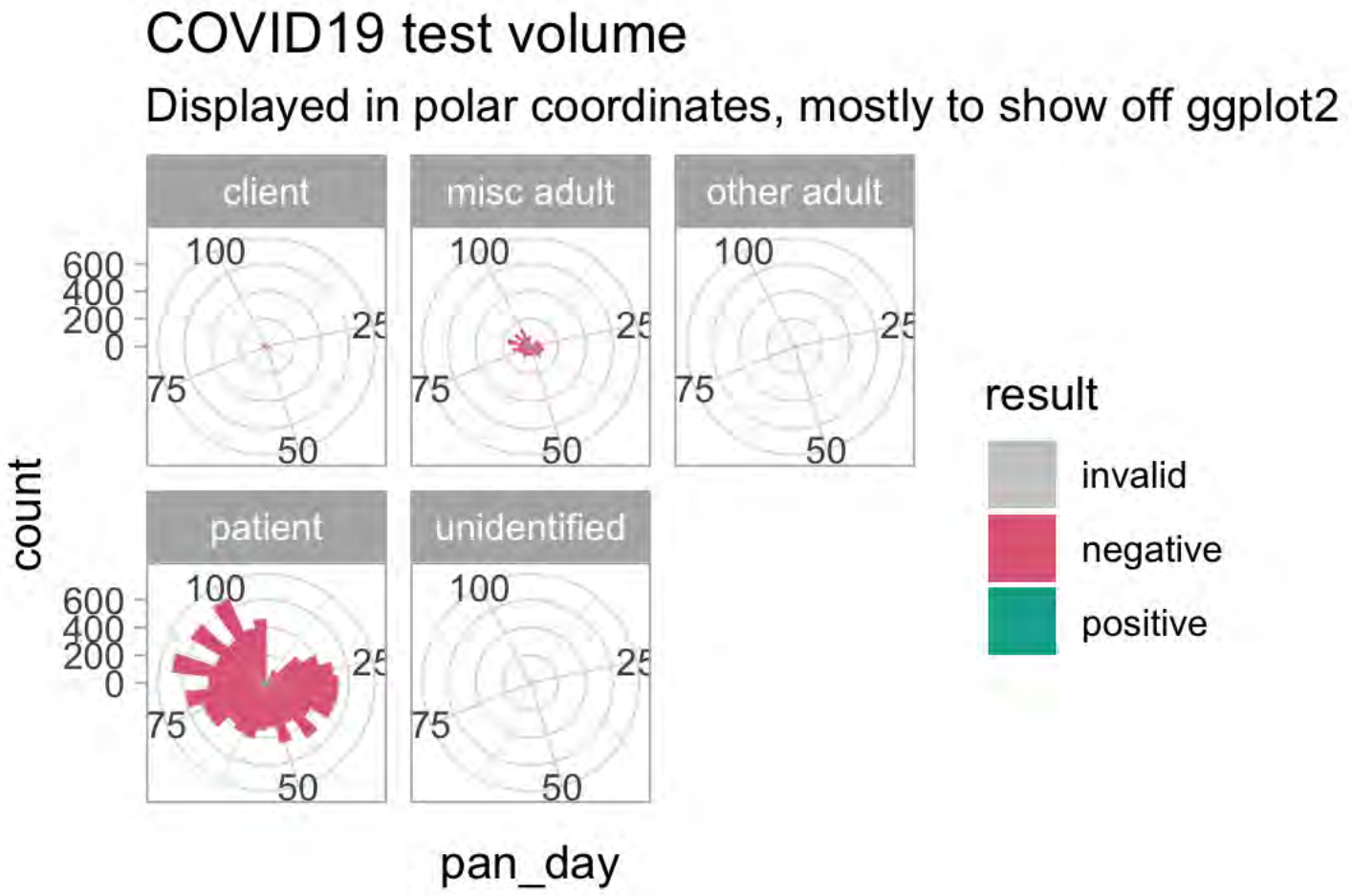
Subplots that display subsets of the data



Coordinate systems



Titles and captions



```
ggplot(data = data_frame) +
  geom_function(mapping = aes(mappings)) +
  theme_function +
  scale_function +
  facet_function +
  coordinate_function +
  ...
```

Required

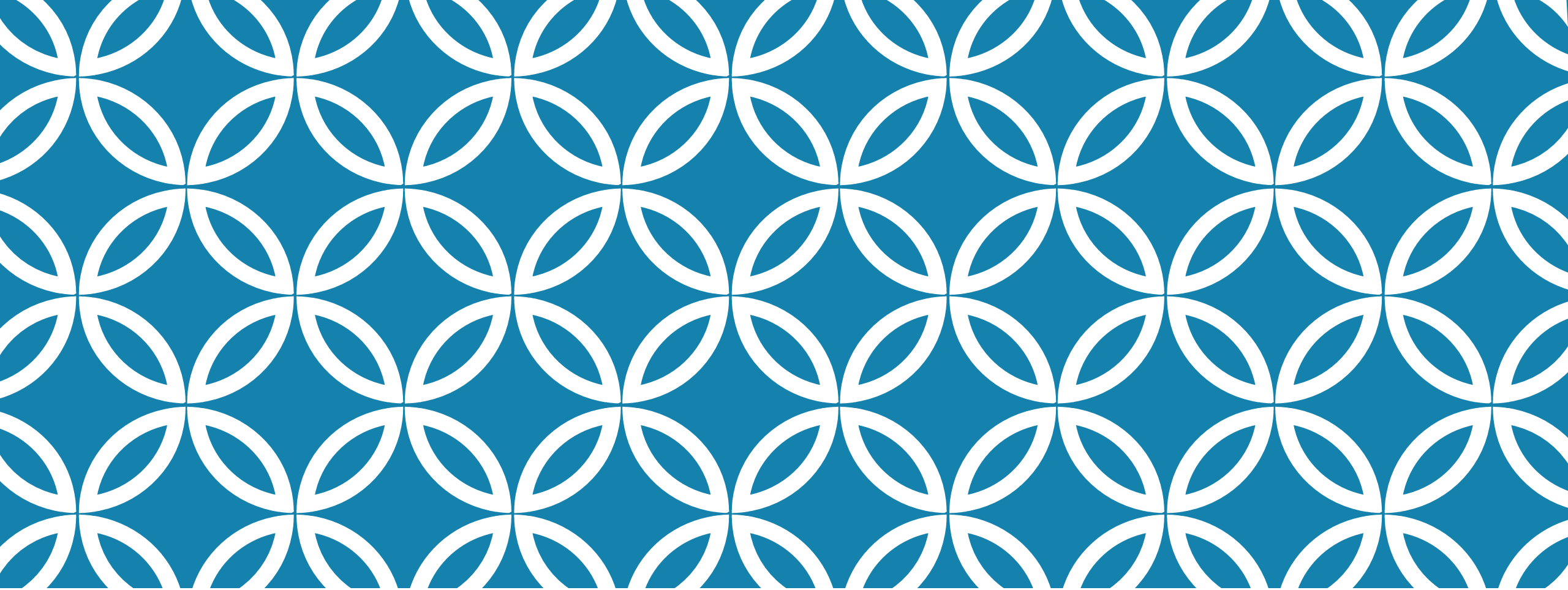
Optional

Goals

1. Appreciate the importance of visualization for understanding data
2. Learn how to use ggplot2 to visualize data

Objectives

1. Create a basic visualization using a simple **template**
2. Define "**aesthetic mapping**" and explain how aesthetic mappings relate variables of a data frame to features of graphic markings on a plot
3. Write the code to specify a type of plot and fine tune its appearance using "**geom**" functions
4. Explain how to add **layers** to a ggplot object to create complex and highly customized visualizations



Data Transformation: Slicing Your Data

Patrick Mathias

December 13, 2020

Presentation adapted from…

Amrom Obstfeld

Assistant Professor of Clinical Pathology and Laboratory Medicine

University of Pennsylvania Perelman School of Medicine

Director of Hematology and Coagulation Laboratories

Children's Hospital of Philadelphia

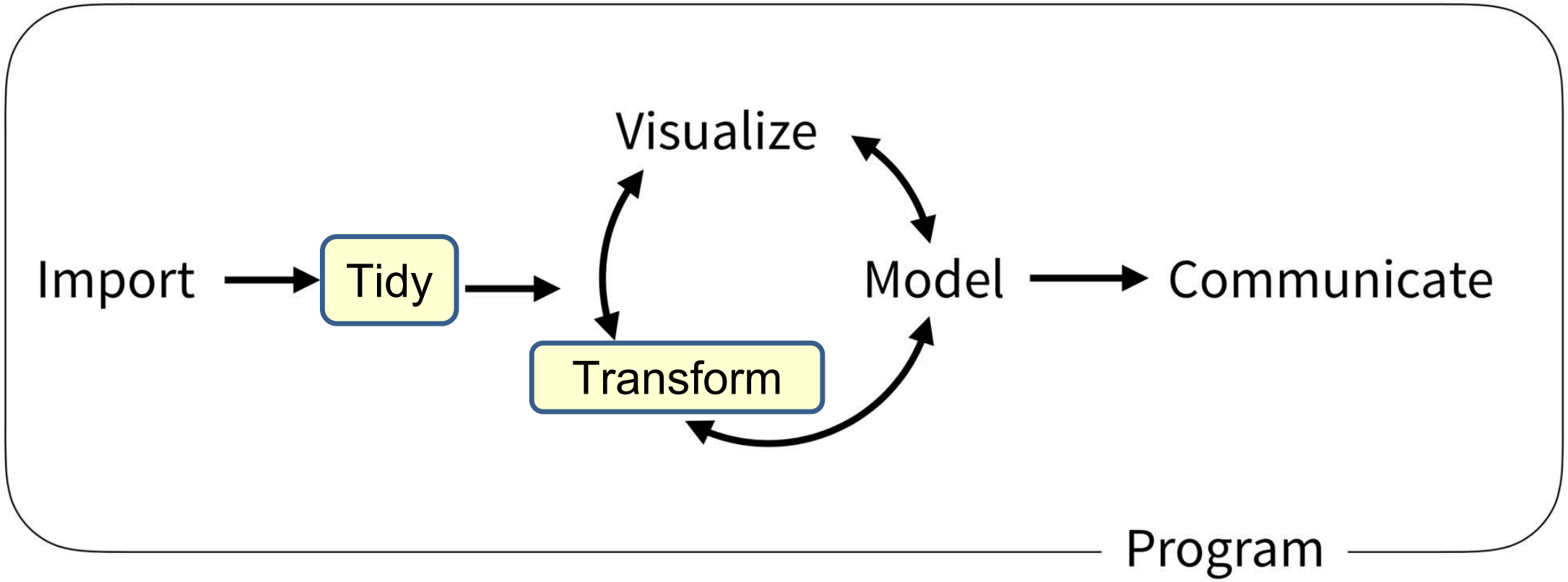
Goal

1. Learn how to use dplyr to transform data frames

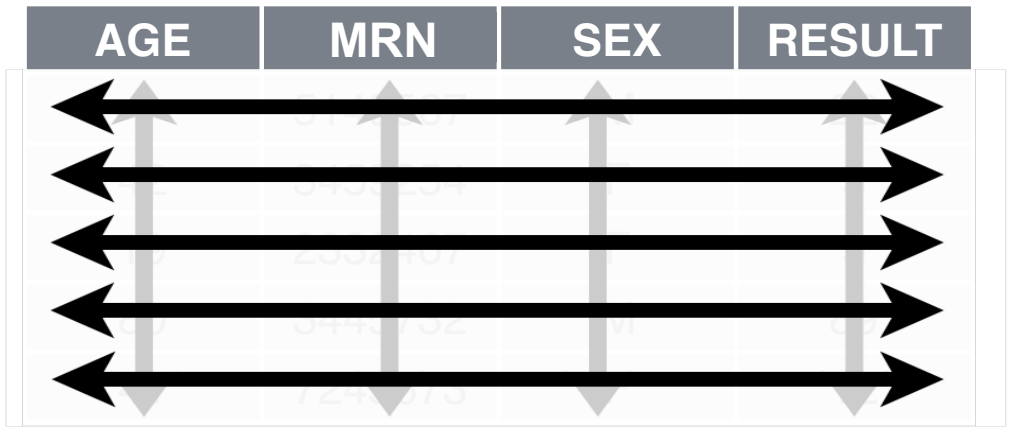
Objectives

1. List the major forms of data transformation implemented in dplyr
2. Use code templates with dplyr functions to tidy a raw data set

Typical Data Science Pipeline



What is a "Tidy" Data Frame



A data set is **tidy** if:

1. Each **variable** is in its own **column**
2. Each **observation** is in its own **row**
3. Each **value** is in its own **cell**

Your Turn 1

Open "**04-Transform.Rmd**"

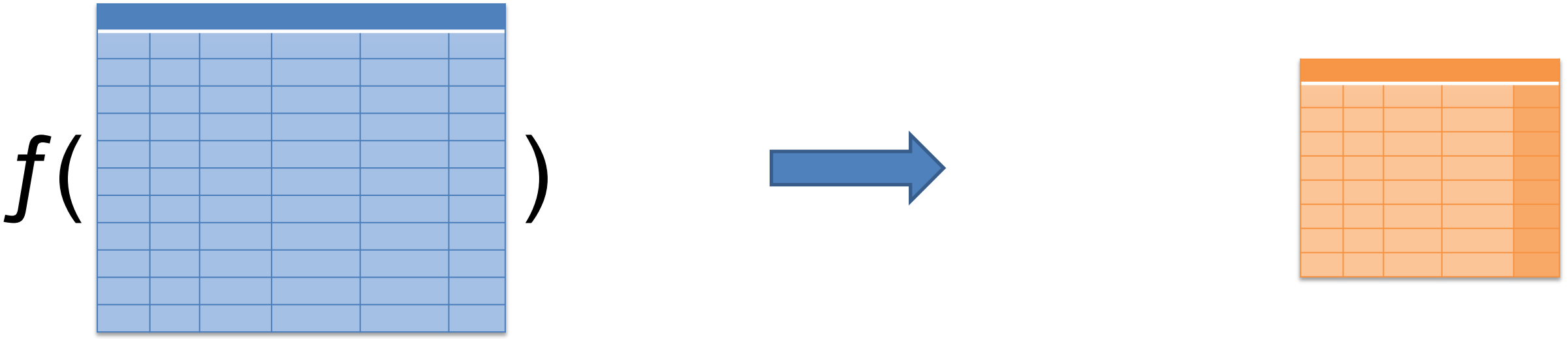
Run the setup chunk

Transform Data with

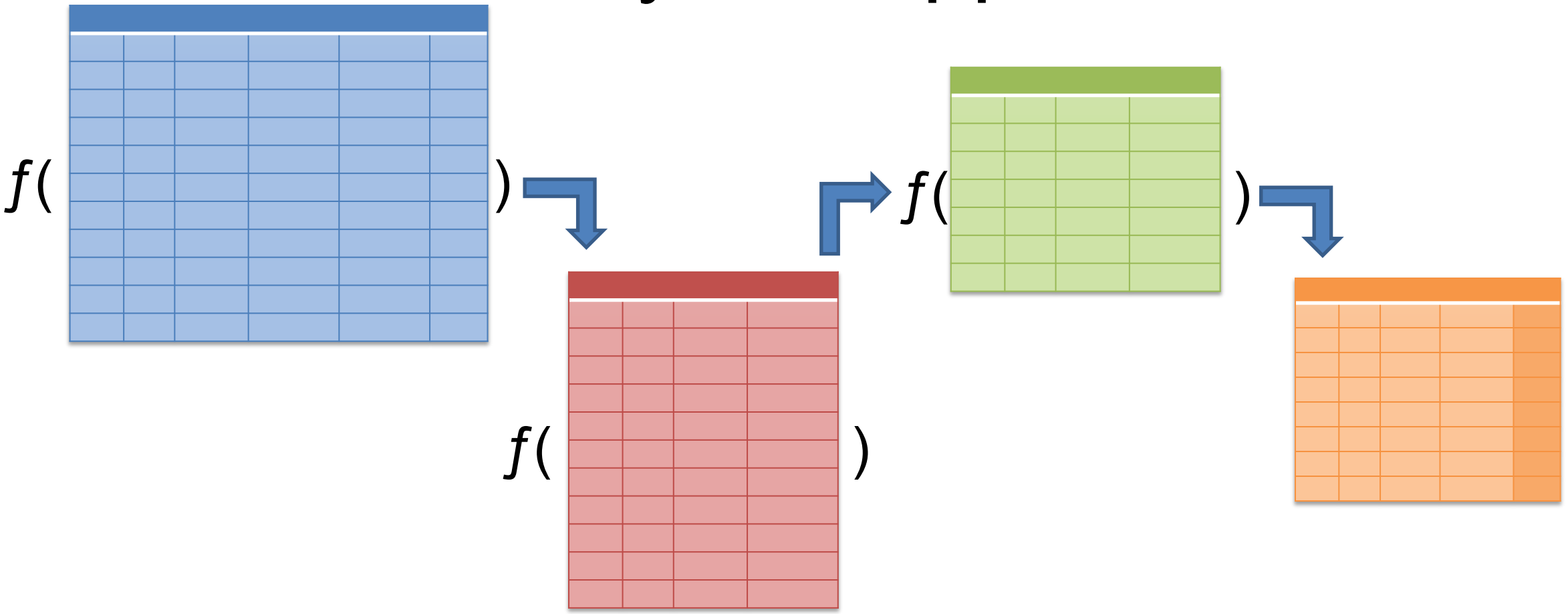
dplyr

dplyr implements a *grammar* for transforming tabular data.

Analytical Approach



Analytical Approach



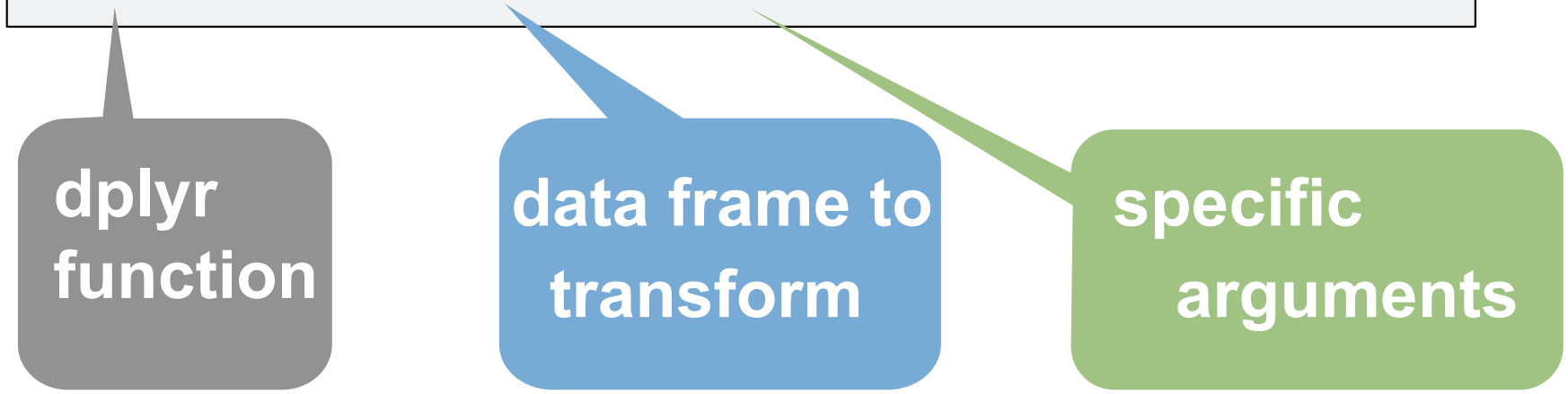
dplyr: a grammar for transforming data

1	**Choose** columns.	**select()**
2	**Extract** rows.	**filter()**
3	**Derive** new columns.	**mutate()**
4	**Change** the unit of analysis.	**summarize()**

Common syntax

Each function takes a data frame as its first argument and returns a data frame as its output.

```
function(data, ...)
```

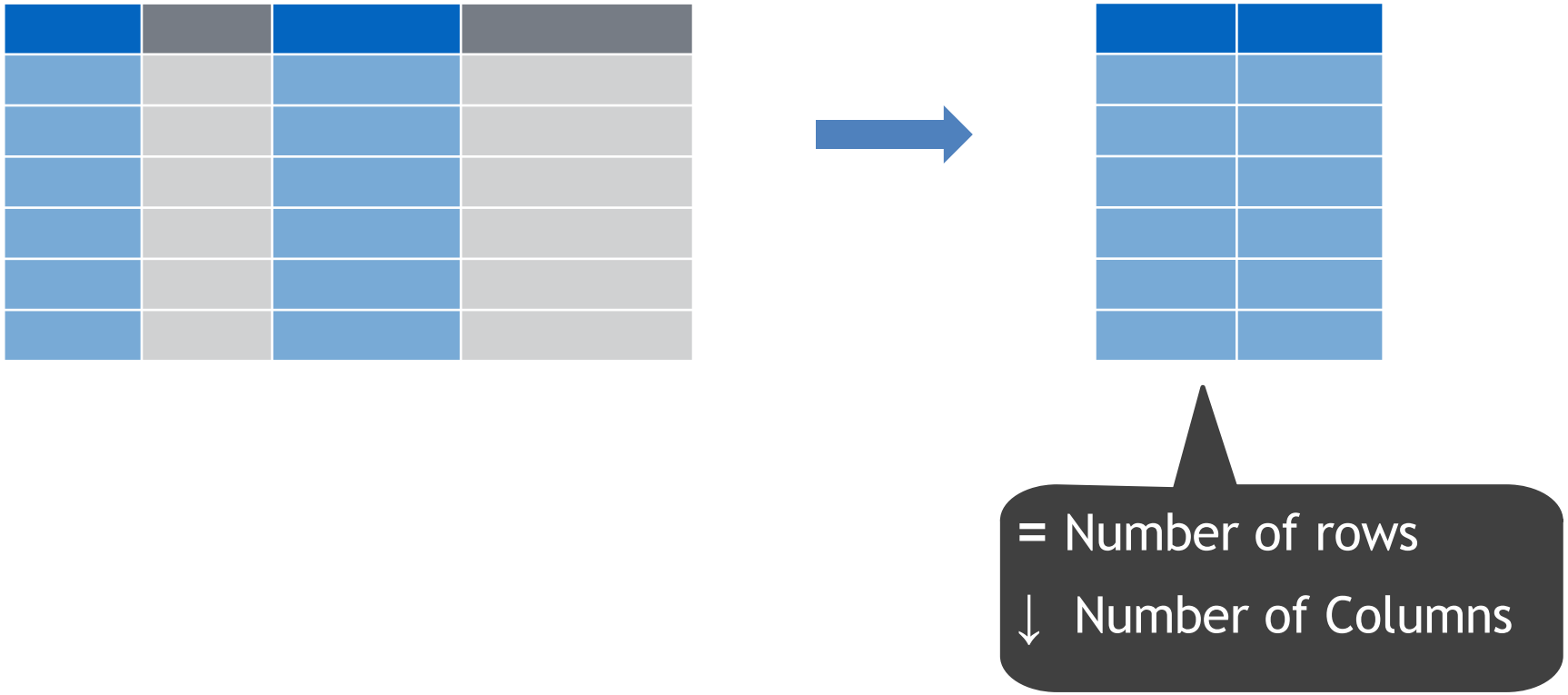


Pulling specific columns out of your data frame

select()

select()

Extract columns from a data frame



select()

Extract columns from a data frame

```
select(covid_testing, mrn, last_name)
```

dplyr function

data frame to transform

name(s) of columns to extract
(or a select helper)

select()

Extract columns from a data frame **by name**

```
select(covid_testing, mrn, last_name)
```

covid_testing

mrn	first_name	last_name	gender
5000876	sarella	stark	female
5006017	alester	stark	male
5001412	jhezane	westerling	female
5000533	penny	targaryen	female
...			

mrn	last_name
5000876	stark
5006017	stark
5001412	westerling
5000533	targaryen

select()

Extract columns from a data frame **by name**

```
select(covid_testing, -mrn, -last_name)
```

covid_testing

mrn	first_name	last_name	gender
5000876	sarella	stark	female
5006017	alester	stark	male
5001412	jhezane	westerling	female
5000533	penny	targaryen	female
...			

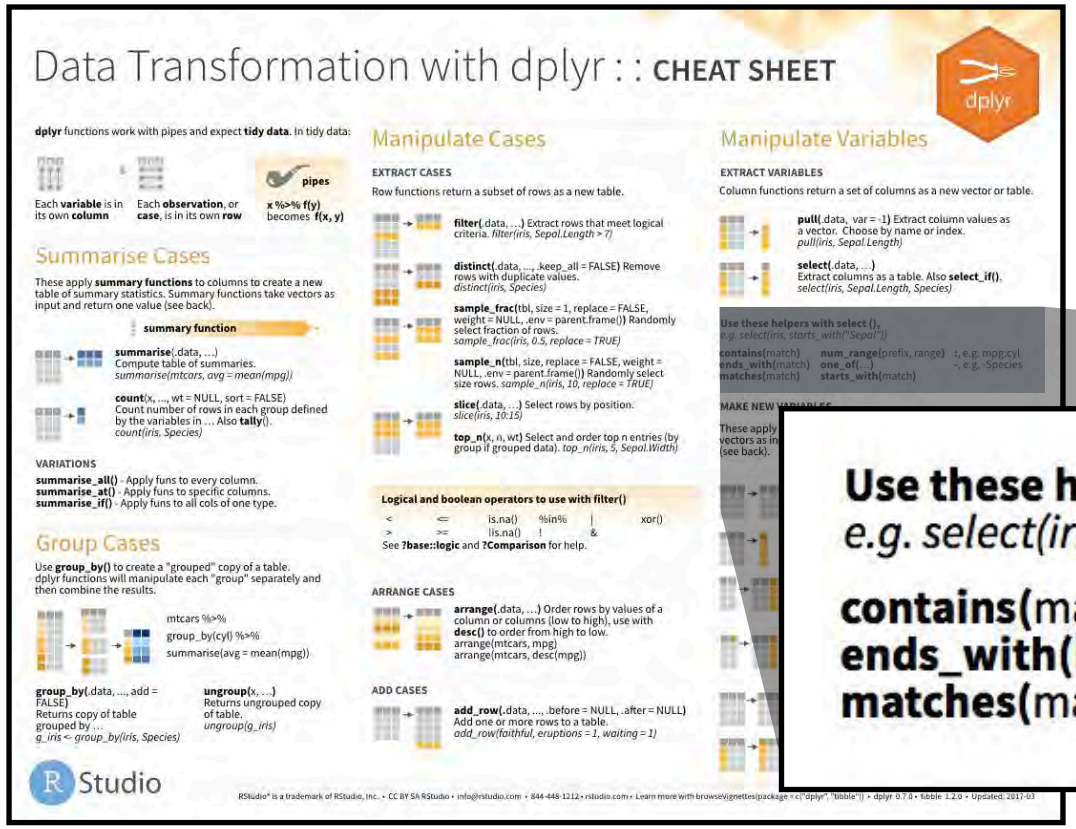
first_name	gender
sarella	female
alester	male
jhezane	female
penny	female

Your Turn 2

- Alter the code to select just the `first_name` column from `covid_testing`
- If you have time, try to remove the `first_name` column

```
covid_testing_2 <- select(covid_testing, _____)
```

select() helpers



Data Transformation with dplyr : : CHEAT SHEET

dplyr

dplyr functions work with pipes and expect tidy data. In tidy data:

Each variable is in its own column. Each observation, or case, is in its own row.

pipes

x %>% f(y) becomes f(x, y)

Summarise Cases

These apply summary functions to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

summary function

summarise(.data, ...)
Compute table of summaries.
summarise(mtcars, avg = mean(mpg))

count(x, ..., wt = NULL, sort = FALSE)
Count number of rows in each group defined by the variables in ... Also tally().
count(iris, Species)

VARIATIONS

summarise_all() - Apply funs to every column.
summarise_at() - Apply funs to specific columns.
summarise_if() - Apply funs to all cols of one type.

Group Cases

Use group_by() to create a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.

mtcars %>%
group_by(cyl) %>%
summarise(avg = mean(mpg))

group_by(.data, ..., add = FALSE)
Returns copy of table grouped by ...
g_iris <- group_by(iris, Species)

ungroup(x, ...)
Returns ungrouped copy of table.
ungroup(g_iris)

Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table.

filter(.data, ...) Extract rows that meet logical criteria. *filter(iris, Sepal.Length > 7)*

distinct(.data, ..., .keep_all = FALSE) Remove rows with duplicate values.
distinct(iris, Species)

sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select fraction of rows.
sample_frac(iris, 0.5, replace = TRUE)

sample_n(tbl, size, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select size rows. *sample_n(iris, 10, replace = TRUE)*

slice(.data, ...) Select rows by position.
slice(iris, 10:15)

top_n(x, n, wt) Select and order top n entries (by group if grouped data). *top_n(iris, 5, Sepal.Width)*

Logical and boolean operators to use with filter()

< <= is.na() %in% | xor()
> >= !is.na() ! &

See ?base::logic and ?Comparison for help.

ARRANGE CASES

arrange(.data, ...) Order rows by values of a column or columns (low to high), use with desc() to order from high to low.
arrange(mtcars, mpg)
arrange(mtcars, desc(mpg))

ADD CASES

add_row(.data, ..., .before = NULL, .after = NULL) Add one or more rows to a table.
add_row(faithful, eruptions = 1, waiting = 1)

Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.

pull(.data, var = -1) Extract column values as a vector. Choose by name or index.
pull(iris, Sepal.Length)

select(.data, ...)
Extract columns as a table. Also select_if().
select(iris, Sepal.Length, Species)

Use these helpers with select (),
e.g. select(iris, starts_with("Sepal"))

contains(match) num_range(prefix, range) :, e.g. mpg:cyl
ends_with(match) one_of(...) -, e.g, -Species
matches(match) starts_with(match)

MAKE NEW VARIABLES

These apply
vectors as in
(see back).

RStudio

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more with browseVignettes(package = c("dplyr", "tibble")) • dplyr 0.7.0 • tibble 1.2.0 • Updated: 2017-03

Use these helpers with select (),
e.g. select(iris, starts_with("Sepal"))

contains(match) **num_range**(prefix, range) **:**, e.g. mpg:cyl
ends_with(match) **one_of**(...) -, e.g, -Species
matches(match) **starts_with**(match)

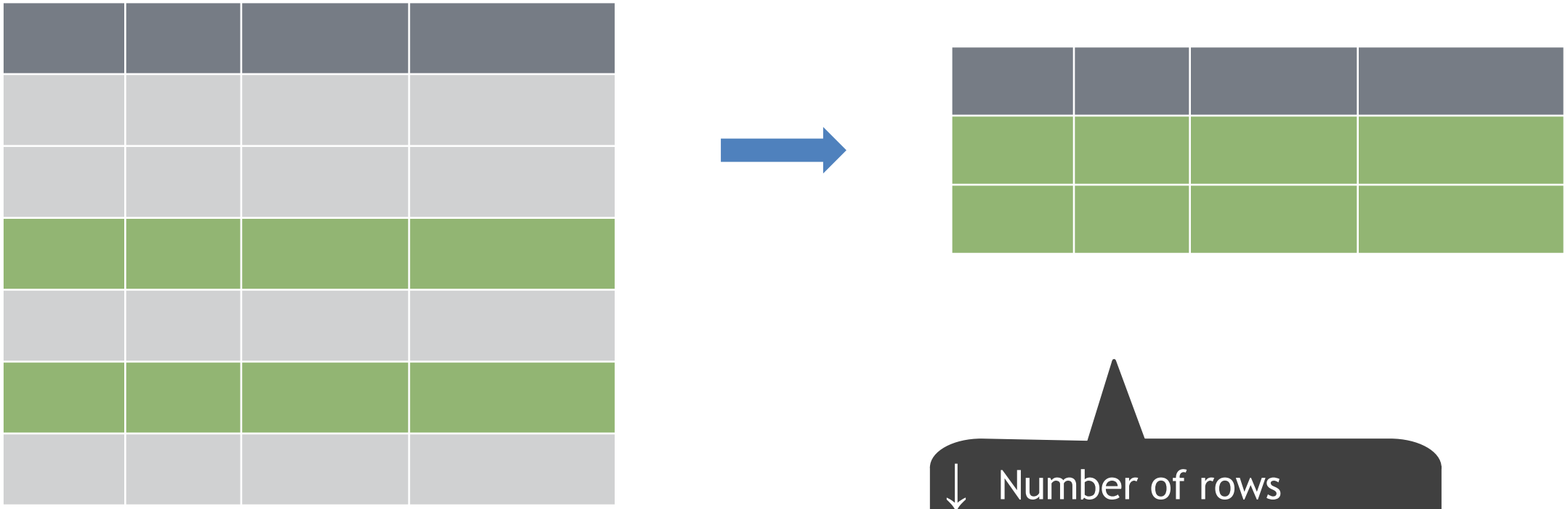
dplyr

Pulling specific rows out of your data frame

filter()

filter()

Extract rows that meet logical criteria



↓ Number of rows

= Number of Columns

Common syntax

Each function takes a data frame as its first argument and returns a data frame as its output.

```
function(data, ...)
```

dplyr function

data frame to transform

specific arguments

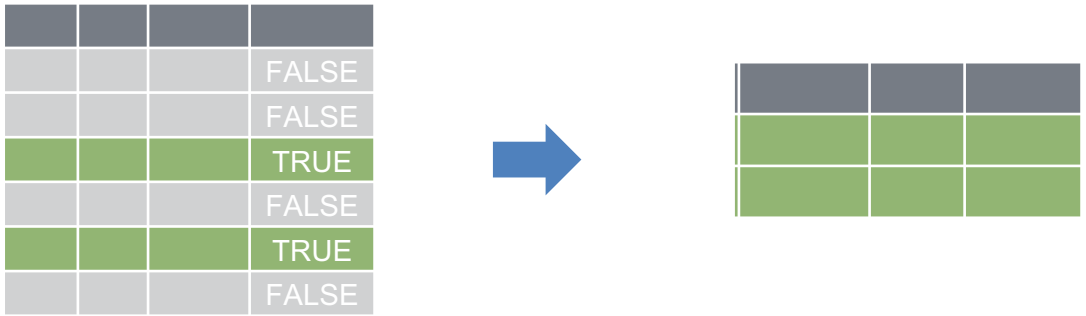
filter()

Extract rows that meet logical criteria

```
filter(data,... )
```

data frame to transform

one or more logical tests (filter returns each row for which the test is TRUE)

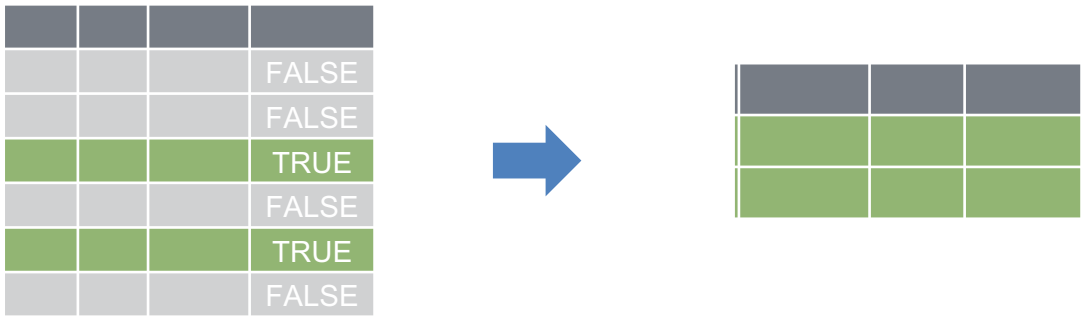


filter()

Extract rows that meet logical criteria

```
filter(data, column_name == criteria )
```

one or more logical tests (filter returns each row for which the test is TRUE)



filter()

Extract rows that meet logical criteria

```
filter(covid_testing, mrn==5000083)
```

	mrn	first_name	last_name
FALSE	5000876	sarella	stark
FALSE	5006017	alester	stark
FALSE	5001412	jhezane	westerling
TRUE	5000083	lollys	clegane

mrn	first_name	last_name
5000083	lollys	clegane

filter()

Extract rows that meet logical criteria

```
filter(covid_testing, mrn==5000083)
```

mrn	first_name	last_name
5000876	sarella	stark
5006017	alester	stark
5001412	jhezane	westerling
5000083	lollys	clegane

= sets

(returns nothing)

== tests if equal

(returns TRUE or FALSE)

filter()

Values coded as character strings must be surrounded by quotes

Extract rows that meet logical criteria.

```
filter(covid_testing, last_name=="stark")
```

mrn	first_name	last_name	
5000876	sarella	stark	TRUE
5006017	alester	stark	TRUE
5001412	jhezane	westerling	FALSE
5000083	lollys	clegane	FALSE

→

mrn	first_name	last_name
5000876	sarella	stark
5006017	alester	stark

filter()

Extract rows that meet logical criteria

```
filter(data, ...)
```

data frame to transform

one or more logical tests (filter returns each row for which the test is TRUE)

Logical tests

`x < y`	Less than
`x > y`	Greater than
`x == y`	Equal to
`x <= y`	Less than or equal to
`x >= y`	Greater than or equal to
`x != y`	Not equal to
`x %in% y`	Group membership
`is.na(x)`	Is NA
`!is.na(x)`	Is not NA

Pop Quiz

What is the result?

```
1 == 1
```

Pop Quiz

What is the result?

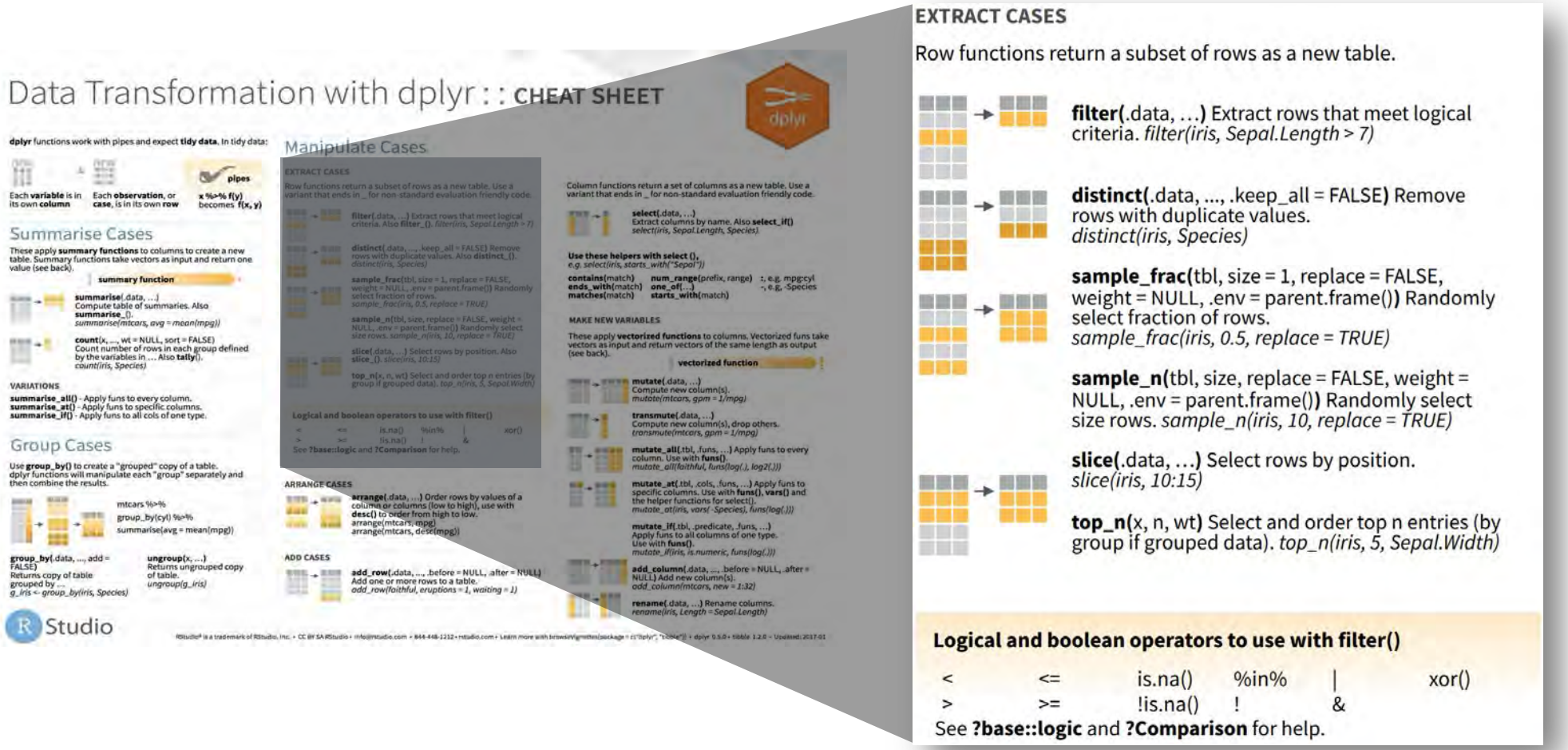
```
3 != 1
```

Your Turn 3

Use filter() with the logical operators to find:

- Every test for patients **over age 80**
- All of the covid testing where the demographic group (demo_group) is **equal to "client"**
- CHALLENGE:
  - All of the covid testing where the patient class (patient_class) **is NA** [Hint: See slide titled "Logical Tests"]

filter() variants



EXTRACT CASES

Row functions return a subset of rows as a new table.

filter(.data, …**)** Extract rows that meet logical criteria. *filter(iris, Sepal.Length > 7)*

distinct(.data, ..., .keep_all = FALSE**)** Remove rows with duplicate values. *distinct(iris, Species)*

sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame()**)** Randomly select fraction of rows. *sample_frac(iris, 0.5, replace = TRUE)*

sample_n(tbl, size, replace = FALSE, weight = NULL, .env = parent.frame()**)** Randomly select size rows. *sample_n(iris, 10, replace = TRUE)*

slice(.data, …**)** Select rows by position. *slice(iris, 10:15)*

top_n(x, n, wt**)** Select and order top n entries (by group if grouped data). *top_n(iris, 5, Sepal.Width)*

Logical and boolean operators to use with filter()

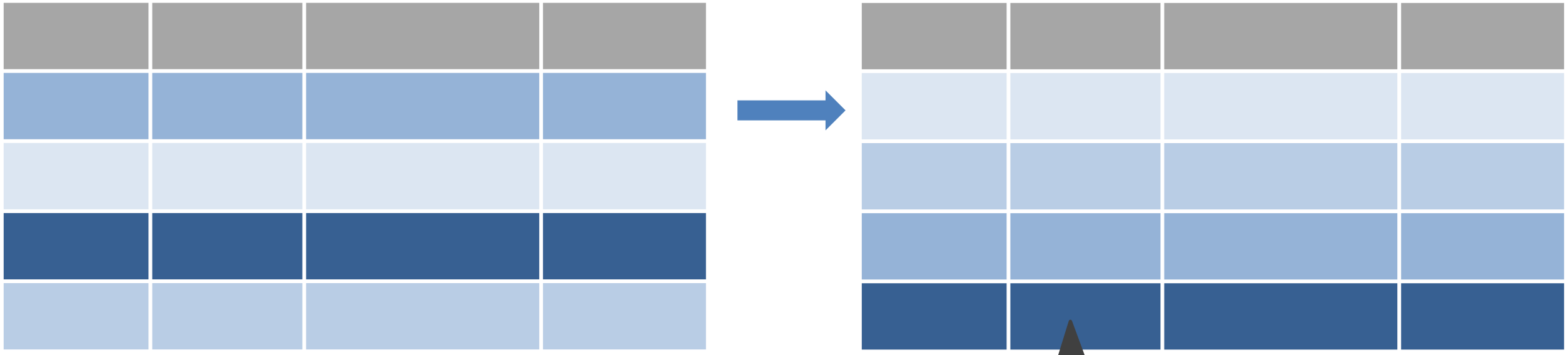
< <= is.na() %in% | xor()
> >= !is.na() ! &
See **?base::logic** and **?Comparison** for help.

What else?

arrange()

arrange()

Order rows by values in a column



= Number of rows

= Number of Columns

arrange()

Order rows by values in a column

```
arrange(data, ... )
```

data frame to transform

name(s) of columns to arrange by

arrange()

Order rows by values in a column

```
arrange(covid_testing, first_name)
```

mrn	first_name	last_name
5000876	sarella	stark
5006017	alester	stark
5001412	jhezane	targaryen
5000533	penny	targaryen

mrn	first_name	last_name
5006017	alester	stark
5001412	jhezane	targaryen
5000533	penny	targaryen
5000876	sarella	stark

arrange()

Order rows by values in a column

```
arrange(covid_testing, desc(mrn))
```

mrn	first_name	last_name
5000876	sarella	stark
5006017	alester	stark
5001412	jhezane	targaryen
5000533	penny	targaryen

mrn	first_name	last_name
5006017	alester	stark
5001412	jhezane	targaryen
5000876	sarella	stark
5000533	penny	targaryen

Your Turn 4

The column `ct_result` contains the cycle threshold (Ct) for the real-time PCR that generated the final result.

How might you use `arrange()` to determine the highest and lowest Ct result in the dataset?

Pop Quiz

The default behavior of arrange() is to order from lower to higher values.

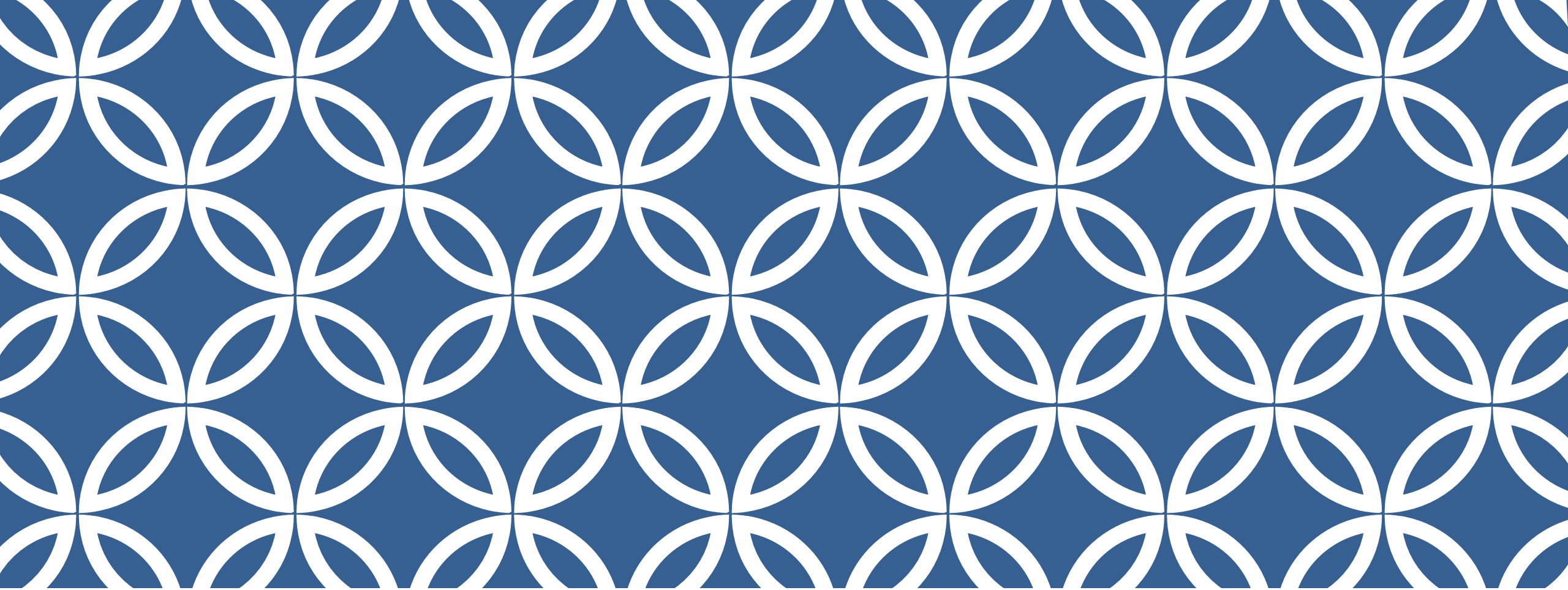
When might arrange() place "1000" before "50"?

Goal

1. Learn how to use dplyr to transform data frames

Objectives

1. List the major forms of data transformation implemented in dplyr
2. Use code templates with dplyr functions to tidy a raw data set



Data Transformation

Session 5

Dan Herman

December 13, 2020

Presentation adapted from…

Amrom Obstfeld

Assistant Professor of Clinical Pathology and Laboratory Medicine

University of Pennsylvania Perelman School of Medicine

Director of Hematology and Coagulation Laboratories

Children's Hospital of Philadelphia

Goal

1. Learn how to use dplyr to transform data frames
2. Appreciate the role of piping in facilitating data transformation

Objectives

1. Use the pipe operator to pass the output of one function as an input to the next function
2. Create new calculated columns not found in the original data frame

%>%

Data Analysis Steps

```
day_10 <- filter(covid_testing, pan_day <= 10)

day_10 <- select(day_10, clinic_name)

day_10 <- arrange(day_10, clinic_name)
```

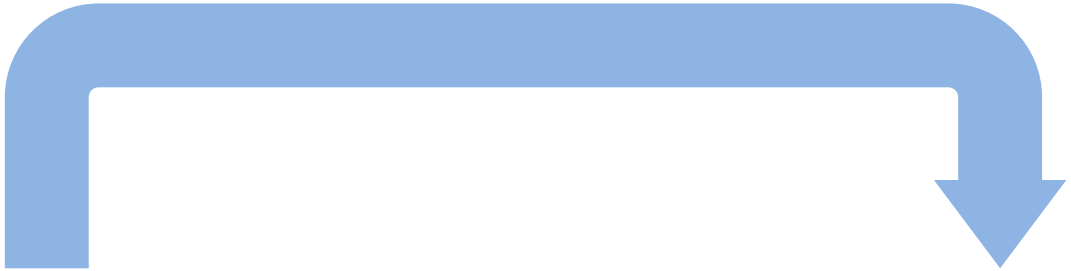
1. Filter tests to those from pandemic day 10 or earlier
2. Select the column that contains the ordering location
3. Arrange those columns by location

Data Analysis Steps

```
day_10 <- arrange(
            select(
              filter(
                covid_testing,
                pan_day <= 10
              ),
              clinic_name
            ),
            clinic_name
          )
```

The Pipe Operator %>%

Passes result on left into first argument of function on right.



```
covid_testing %>% filter(____, pan_day <= 10)
```

```
filter(covid_testing, pan_day <= 10)
covid_testing %>% filter(pan_day <= 10)
```

Data Analysis Steps

```
day_10 <- arrange(
            select(
              filter(
                covid_testing,
                pan_day <= 10
              ),
              clinic_name
            ),
            clinic_name
          )
```

Data Analysis Steps

```
covid_testing %>%
    filter(pan_day <= 10) %>%
    select(clinic_name) %>%
    arrange(clinic_name)
```

Example: TAT investigation

The PICU would like a word with you because of a recent incident involving a delay in results for a patient who required an aerosol generating procedure.

They had to wait over 10 hours before the procedure could begin!

You decide to investigate…WITH DATA

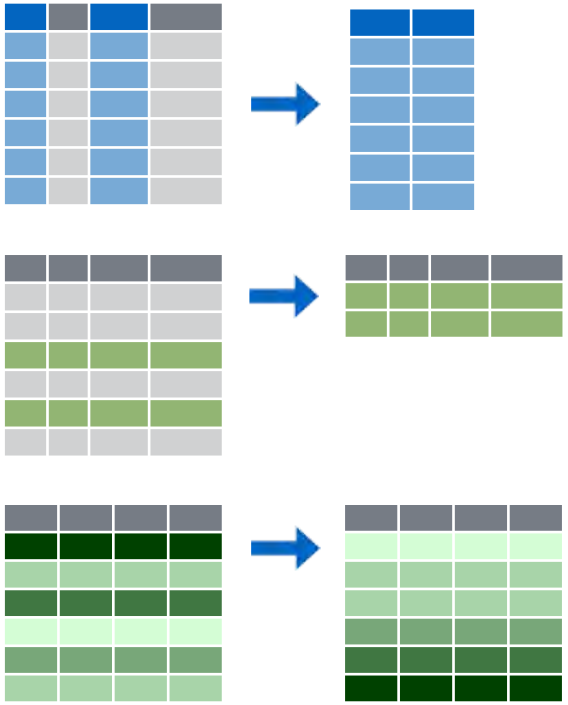
Your Turn 1

Open ‘05– Transform.Rmd’ and run the setup chunk.

Use %>% to write a sequence of three functions that:

1. Filters to orders from the clinic (**`clinic_name`**) of "picu”
2. Selects the columns with the receive to verify turnaround time (**`rec_ver_tat`**) and day from start of the pandemic (**`pan_day`**)
3. Arrange the **`pan_day`** from highest to lowest
4. Use <- to assign the result to a new variable and take a look.

Isolating data



Extract variables with **select()**

Extract rows with **filter()**

Arrange rows, with **arrange()**.

Deriving Data

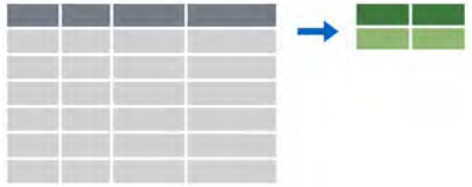
What are the average and standard deviation of the total turnaround time for each ordering clinic?

Breaking down the analytical question

1. Total TAT for each order
2. Group orders by clinic
3. Calculate average and standard deviation for each clinic

Deriving data

Make new variables with **mutate()**

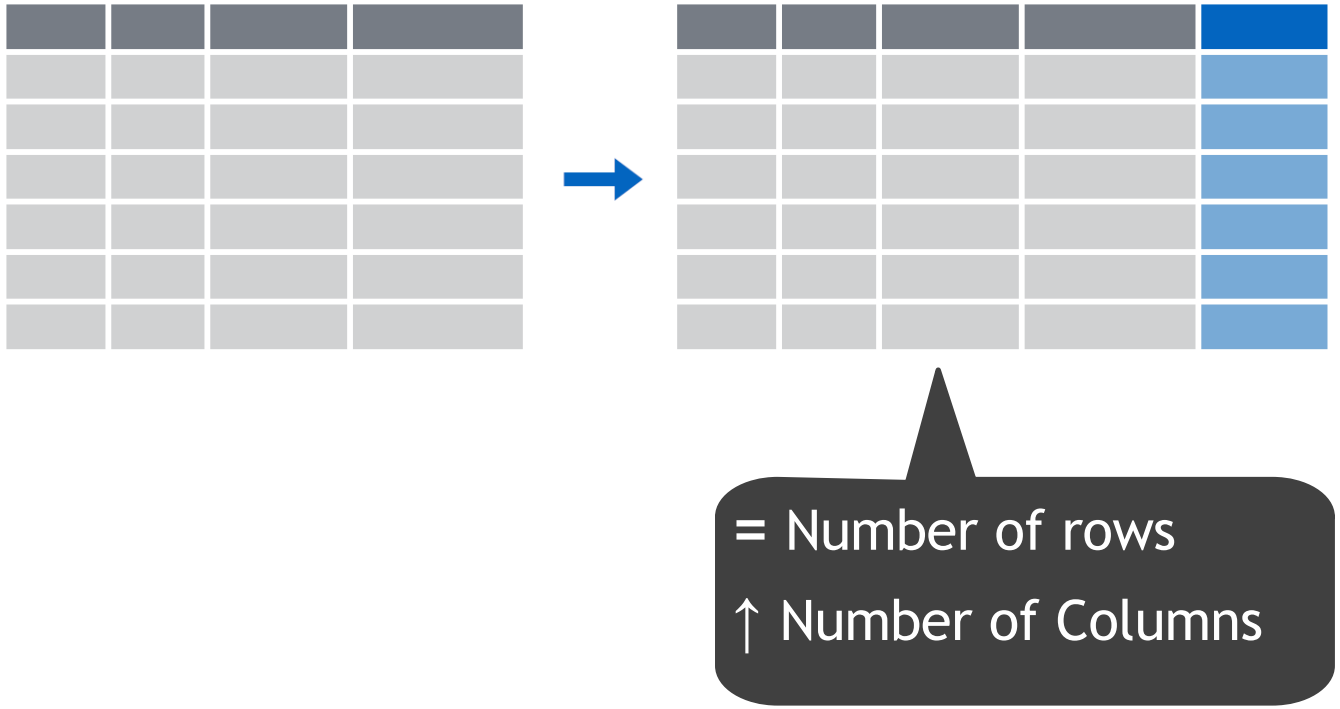


Make summaries of data with **summarize()**

mutate()

mutate()

Creating new calculated columns



mutate()

Creating new calculated columns

```
covid_testing %>%
    mutate(new_column = calculation)
```

name for new column

equals

function whose results will populate columns

mutate()

Creating new calculated columns

```
covid_testing %>%
    mutate(c_r_tat_mins = col_rec_tat * 60)
```

mrn	col_rec_tat	rec_ver_tat
5000876	29.5	11.5
5006017	3.6	5
5001412	1.4	5.2
5000533	2.3	5.8

mrn	col_rec_tat	rec_ver_tat	c_r_tat_mins
5000876	29.5	11.5	1770
5006017	3.6	5	216
5001412	1.4	5.2	84
5000533	2.3	5.8	138

Your Turn 2

Create a new column using the mutate() function that contains the total TAT (sum of **`col_rec_tat`** and **`rec_ver_tat`**)

mutate()

Replace columns

Function to "coerce" one date type into another data type

```
orders %>%
    mutate(mrn = as.character(mrn))
```

mrn <dbl>	first_name <chr>	last_name <chr>
5000876	sarella	stark
5006017	alester	stark
5001412	jhezane	westerling
5000533	penny	targaryen

mrn <chr>	first_name <chr>	last_name <chr>
5000876	sarella	stark
5006017	alester	stark
5001412	jhezane	westerling
5000533	penny	targaryen

Functions to use in mutate()

Vector Functions

TO USE WITH MUTATE ()

mutate() and **transmute()** apply vectorized functions to columns to create new columns. Vectorized functions take vectors as input and return vectors of the same length as output.

vectorized function

OFFSETS

dplyr::**lag()** - Offset elements by 1
dplyr::**lead()** - Offset elements by -1

CUMULATIVE AGGREGATES

dplyr::**cumall()** - Cumulative all()
dplyr::**cumany()** - Cumulative any()
cummax() - Cumulative max()
dplyr::**cummean()** - Cumulative mean()
cummin() - Cumulative min()
cumprod() - Cumulative prod()
cumsum() - Cumulative sum()

RANKINGS

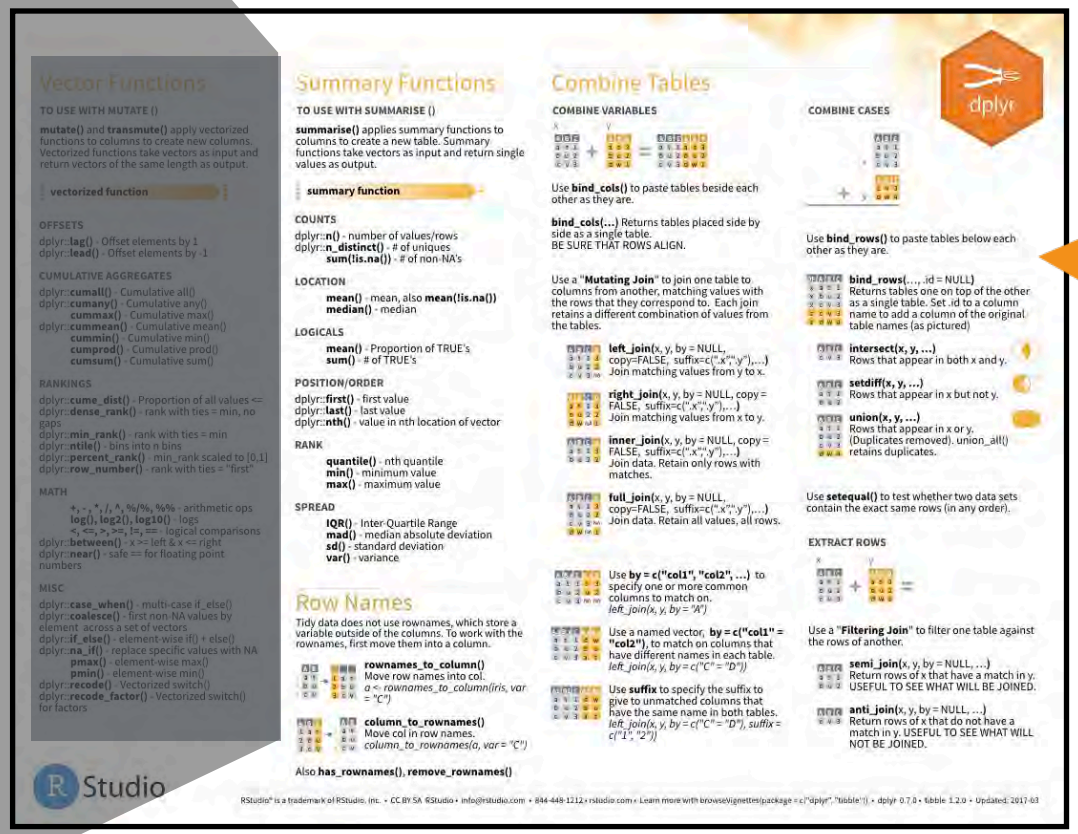
dplyr::**cume_dist()** - Proportion of all values <=
dplyr::**dense_rank()** - rank with ties = min, no gaps
dplyr::**min_rank()** - rank with ties = min
dplyr::**ntile()** - bins into n bins
dplyr::**percent_rank()** - min_rank scaled to [0,1]
dplyr::**row_number()** - rank with ties = "first"

MATH

+, - , *, /, ^, %/%, %% - arithmetic ops
log(), log2(), log10() - logs
<, <=, >, >=, !=, == - logical comparisons
dplyr::**between()** - x >= left & x <= right
dplyr::**near()** - safe == for floating point numbers

MISC

dplyr::**case_when()** - multi-case if_else()
dplyr::**coalesce()** - first non-NA values by element across a set of vectors
dplyr::**if_else()** - element-wise if() + else()
dplyr::**na_if()** - replace specific values with NA
pmax() - element-wise max()
pmin() - element-wise min()
dplyr::**recode()** - Vectorized switch()
dplyr::**recode_factor()** - Vectorized switch() for factors



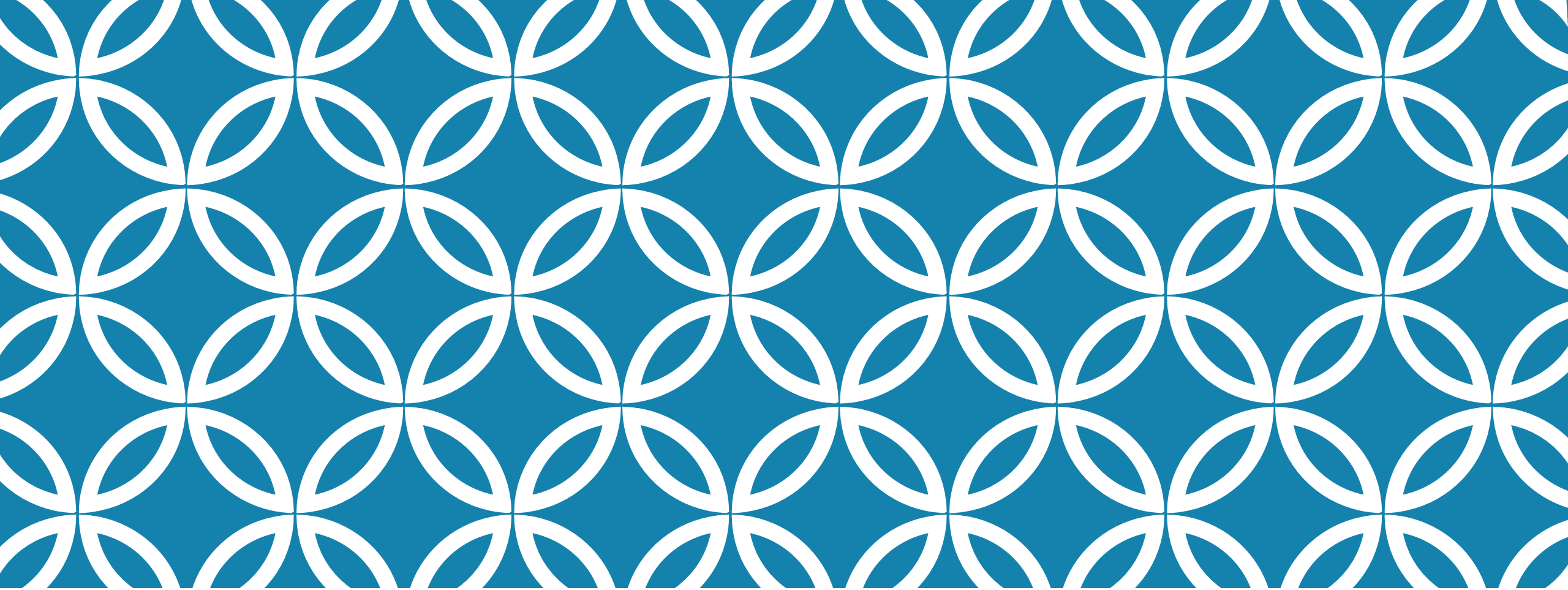
on back

Goal

1. Learn how to use dplyr to transform data frames
2. Appreciate the role of piping in facilitating data transformation

Objectives

1. Use the pipe operator to pass the output of one function as an input to the next function
2. Create new calculated columns not found in the original data frame



Statistical Analyses

Session 6

Dan Herman

December 13, 2020

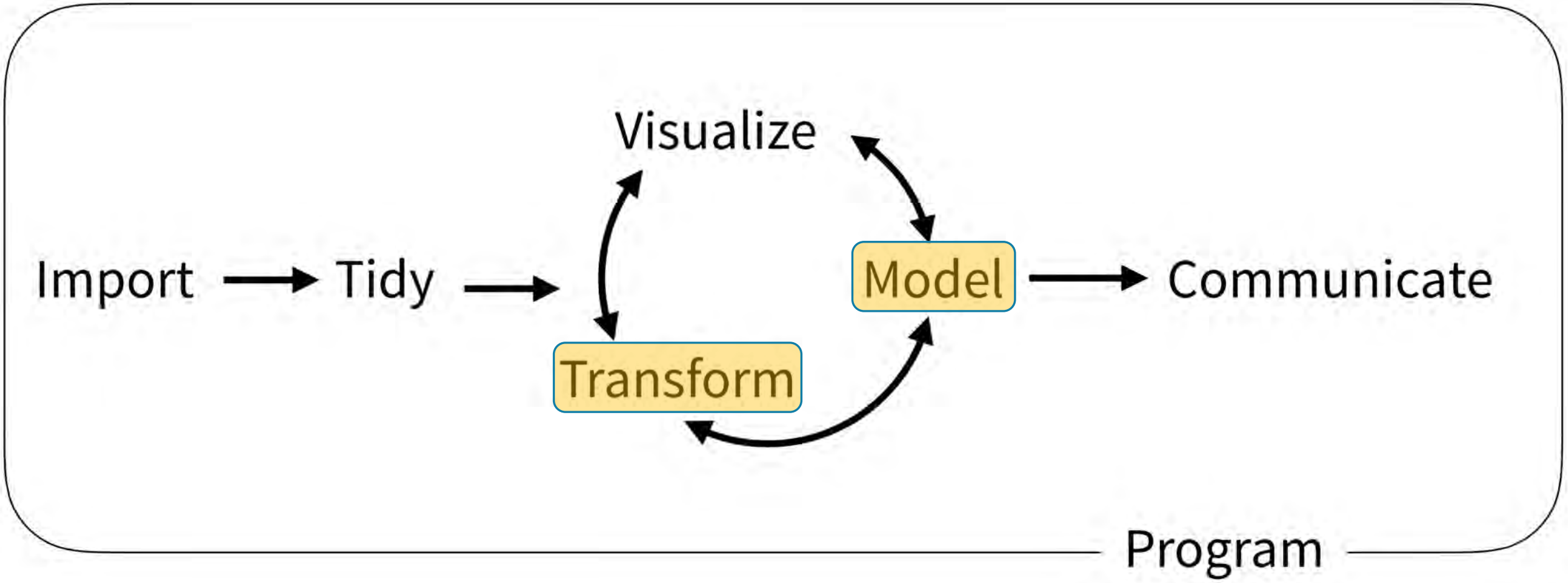
Goals

1. Learn how to summarize data and assess hypotheses

Objectives

1. Calculate a summary statistic for a variable using `summarize`
2. Calculate of summary statistic for a variable separately for a group of observations, using `group_by` and `summarize`
3. Perform a simple test for association

Typical Data Science Pipeline

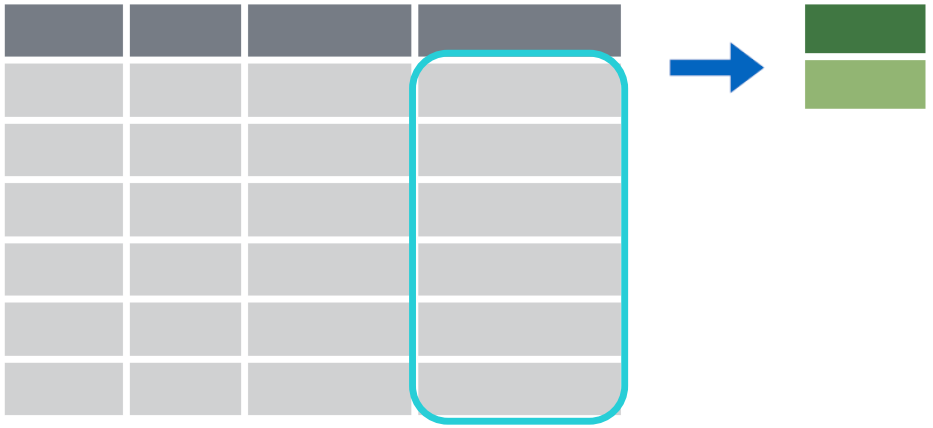


From *R for Data Science* (https://r4ds.had.co.nz/introduction.html)

Summarize()

summarize()

- Make summaries of your data



summarize()

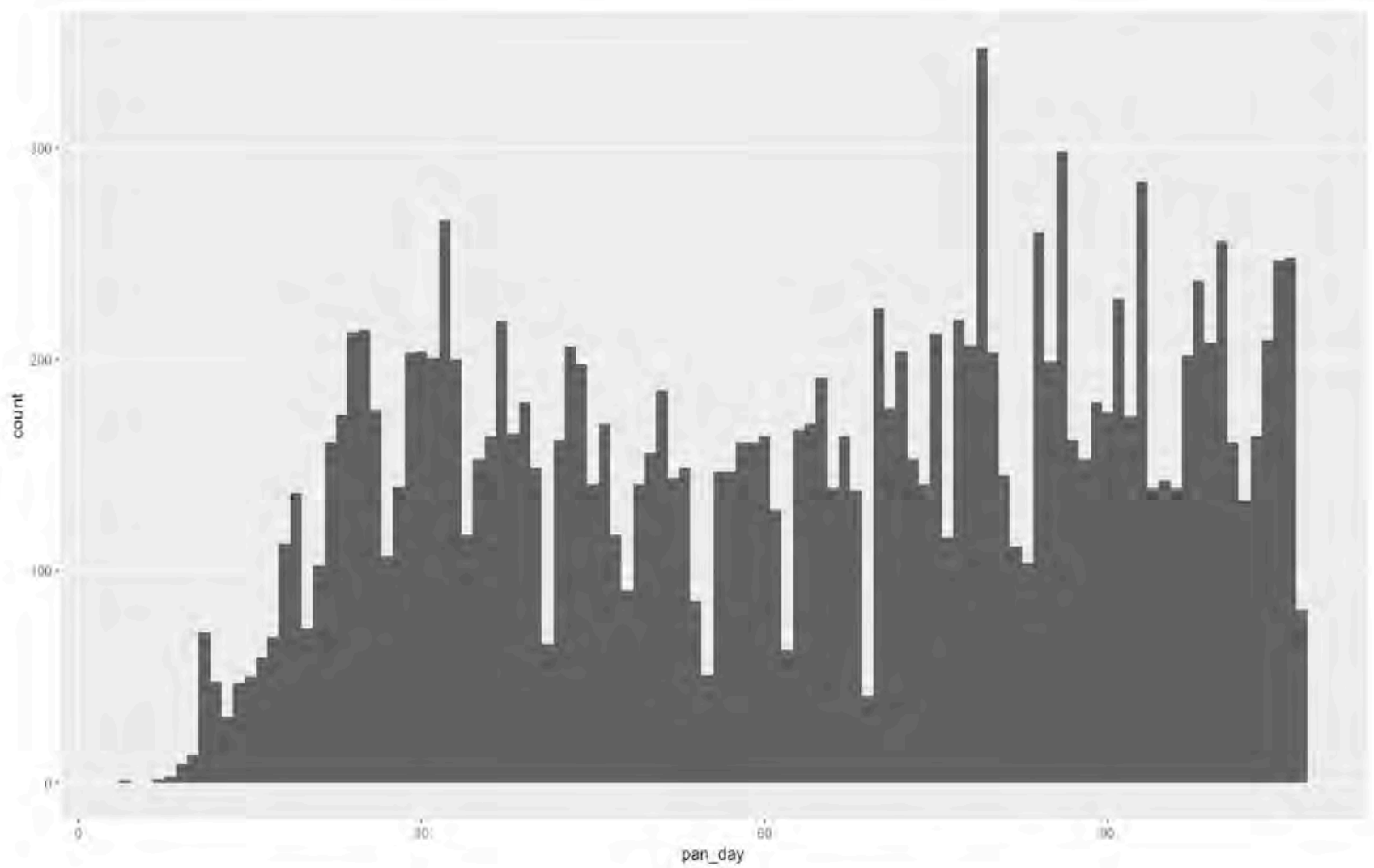
- Make summaries of your data

```
covid_testing %>%
    summarize(new_variable = calculation)
```

name for new variable

Value or function

Q: How many tests are ordered per day?



summarize()

- Make summaries of your data

```
covid_testing %>%
      select(mrn, pan_day)  %>%
      head(4) %>%
      summarize(order_count = n())
```

function that returns number of observations

mrn	pan_day
5001412	4
5000533	7
5009134	7
5008518	8

→

order_count
4

summarize()

- Make summaries of your data

```
covid_testing %>%
      select(mrn, pan_day)  %>%
      head(4) %>%
      summarize(order_count = n(),
                day_count = n_distinct(pan_day))
```

function that returns number of distinct values

mrn	pan_day
5001412	4
5000533	7
5009134	7
5008518	8

→

order_count	day_count
4	3

Your Turn 1

- Open “06 – Stats.Rmd”
- Run the setup chunk
- Fill-in gaps to calculate:

  a) Mean count of orders per `pan_day`

  b) Mean count of orders per clinic

Vector Functions

TO USE WITH MUTATE ()

COUNTS

dplyr::**n()** - number of values/rows
dplyr::**n_distinct()** - # of uniques
sum(!is.na()) - # of non-NA's

LOCATION

mean() - mean, also **mean(!is.na())**
median() - median

LOGICALS

mean() - Proportion of TRUE's
sum() - # of TRUE's

POSITION/ORDER

dplyr::**first()** - first value
dplyr::**last()** - last value
dplyr::**nth()** - value in nth location of vector

RANK

quantile() - nth quantile
min() - minimum value
max() - maximum value

SPREAD

IQR() - Inter-Quartile Range
mad() - median absolute deviation
sd() - standard deviation
var() - variance

Summary Functions

TO USE WITH SUMMARISE ()

summarise() applies summary functions to columns to create a new table. Summary functions take vectors as input and return single values as output.

summary function

COUNTS

dplyr::**n()** - number of values/rows
dplyr::**n_distinct()** - # of uniques
sum(!is.na()) - # of non-NA's

LOCATION

mean() - mean, also **mean(!is.na())**
median() - median

LOGICALS

mean() - Proportion of TRUE's
sum() - # of TRUE's

POSITION/ORDER

dplyr::**first()** - first value
dplyr::last() - last value
dplyr::nth() - value in nth location of vector

RANK

quantile() - nth quantile
min() - minimum value
max() - maximum value

SPREAD

IQR() - Inter-Quartile Range
mad() - median absolute deviation
sd() - standard deviation
var() - variance

Row Names

Tidy data does not use rownames, which store a variable outside of the columns. To work with the rownames, first move them into a column.

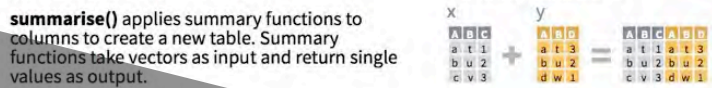
rownames_to_column()
Move row names into col.
a <- rownames_to_column(iris, var = "C")

column_to_rownames()
Move col in row names.
column_to_rownames(a, var = "C")

Also **has_rownames()**, **remove_rownames()**

Combine Tables

COMBINE VARIABLES



Use **bind_cols()** to paste tables beside each other as they are.

bind_cols(...) Returns tables placed side by side as a single table.
BE SURE THAT ROWS ALIGN.

Use a "**Mutating Join**" to join one table to columns from another, matching values with the rows that they correspond to. Each join retains a different combination of values from the tables.

left_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),...)
Join matching values from y to x.

right_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),...)
Join matching values from x to y.

inner_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),...)
Join data. Retain only rows with matches.

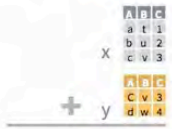
full_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),...)
Join data. Retain all values, all rows.

Use **by = c("col1", "col2")** to specify the column(s) to match on.
left_join(x, y, by = "A")

Use a named vector, **by = c("col1" = "col2")**, to match on columns with different names in each data set.
left_join(x, y, by = c("C" = "D"))

Use **suffix** to specify suffix to give to duplicate column names.
left_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))

COMBINE CASES



Use **bind_rows()** to paste tables below each other as they are.

bind_rows(..., .id = NULL)
Returns tables one on top of the other as a single table. Set .id to a column name to add a column of the original table names (as pictured)

intersect(x, y, ...)
Rows that appear in both x and y.

setdiff(x, y, ...)
Rows that appear in x but not y.

union(x, y, ...)
Rows that appear in x or y. (Duplicates removed). union_all() retains duplicates.

Use **setequal()** to test whether two data sets contain the exact same rows (in any order).

EXTRACT ROWS

Use a "**Filtering Join**" to filter one table against the rows of another.

semi_join(x, y, by = NULL, ...)
Return rows of x that have a match in y. USEFUL TO SEE WHAT WILL BE JOINED.

anti_join(x, y, by = NULL, ...)
Return rows of x that do not have a match in y. USEFUL TO SEE WHAT WILL NOT BE JOINED.

summarize() examples

- Last pandemic day (in data)
- Median turnaround time

Your Turn 2

Consider:

How would you calculate the median number of orders per day?

group_by()

group_by()

group_by()

- *Grouping observations based on a specific variable's values*

```
covid_testing %>%
    group_by(variable)
```

name of variable to group by

group_by()

- *Group observations by pan_day*

```
covid_testing %>%
    group_by(pan_day)
```

```
# A tibble: 15,524 x 17
# Groups:   pan_day [102]
      mrn first_name last_name gender pan_day
    <dbl> <chr>      <chr>     <chr>    <dbl>
1 5.00e6 jhezane    westerli… female       4
2 5.00e6 penny      targaryen female       7
3 5.01e6 grunt      rivers    male         7
4 5.01e6 melisandre swyft     female       8
5 5.01e6 rolley     karstark  male         8
```

group_by()

- *Group observations by `pan_day` and `clinic_name`*

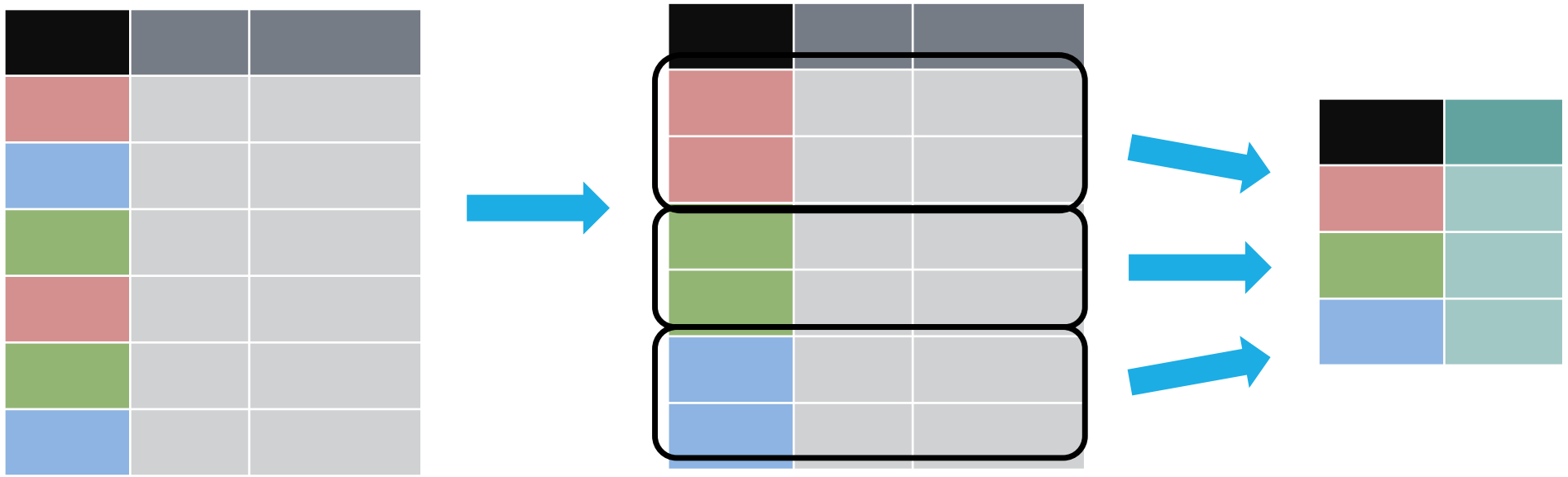
```
covid_testing %>%
    select(mrn, pan_day, clinic_name) %>%
    group_by(pan_day, clinic_name)
```

```
# A tibble: 15,524 x 3
# Groups:   pan_day, clinic_name [2,826]
      mrn pan_day clinic_name
    <dbl>   <dbl> <chr>
 1 5001412       4 inpatient ward a
 2 5000533       7 clinical lab
 3 5009134       7 clinical lab
 4 5008518       8 clinical lab
 5 5008967       8 emergency dept
```

group_by() %>% summarize()

group_by() %>% summarize()

Make summaries of your data *by group*



group_by() %>% summarize()

- Make summaries of your data

```
covid_testing %>%

    summarize(order_count = n())
```

mrn	pan_day
5001412	4
5000533	7
5009134	7
5008518	8

→

order_count
15524

group_by() %>% summarize()

- Make summaries of your data

```
covid_testing %>%
      group_by(pan_day) %>%
      summarize(order_count = n())
```

mrn	pan_day
5001412	4
5000533	7
5009134	7
5008518	8

→

pan_day	order_count
4	1
7	2
8	3
9	9

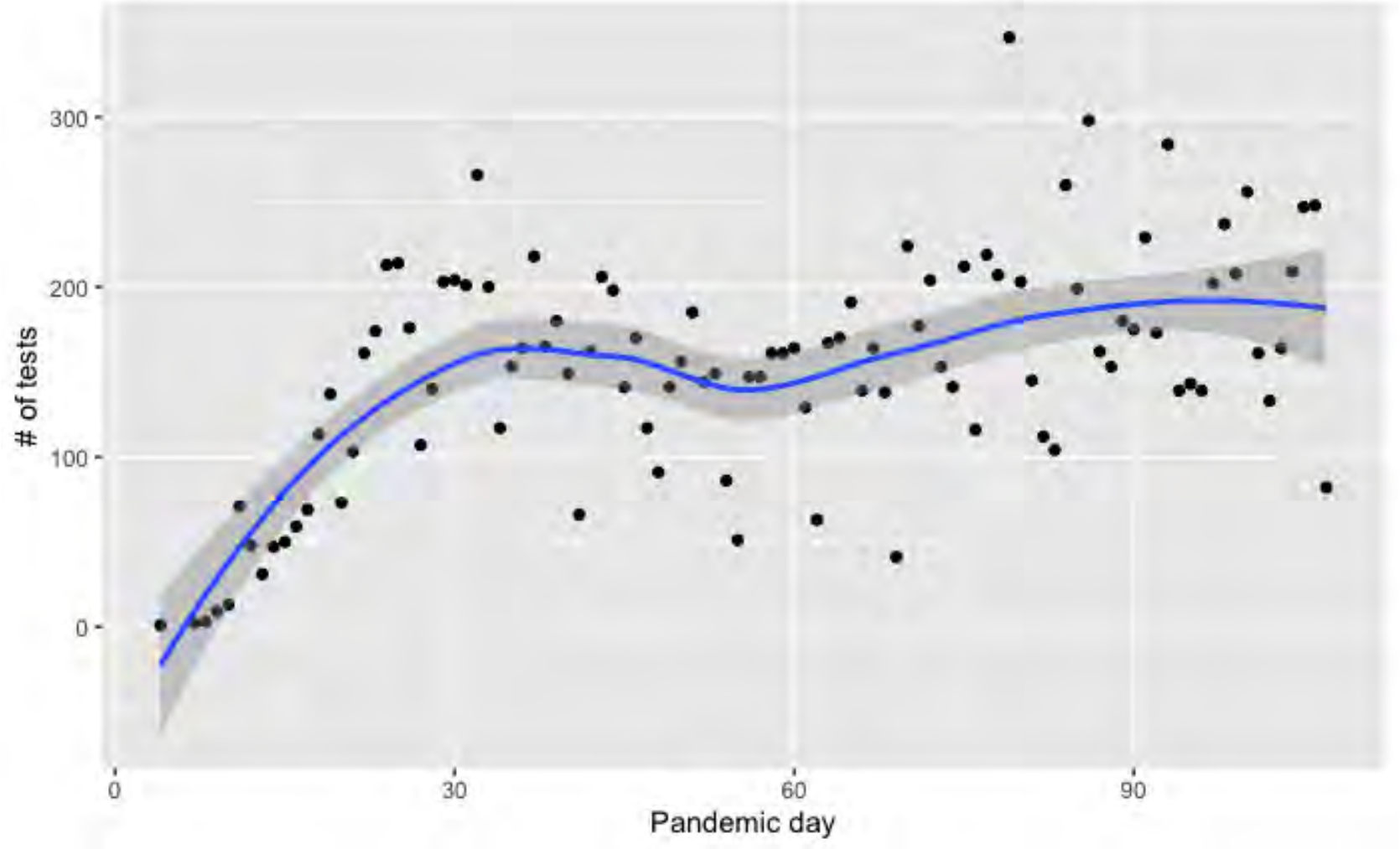
Your Turn 3

Calculate:

a) The median turnaround time for each day

b) (**Extra**) The median number of orders per day

group_by() %>% summarize(): Example



Stats: Tests for association

Q: Is there an association between insurance and SARS-CoV-2 RT-PCR positivity?

payor_group_fac <chr>	negative <int>	positive <int>
commercial	3549	86
government	3318	242
other	309	17
unassigned	7182	520

4 rows

```
data %>%
  fisher.test(simulate.p.value = T)
```

Data wrangling - 1

function that flexibly assigns values

```
covid_testing_2 <- covid_testing %>%
  mutate(payor_group_fac = case_when(
                            is.na(payor_group)                          ~ "unassigned",
                            payor_group %in% c("charity care",
                                               "medical assistance",
                                               "self pay",
                                               "other")                 ~ "other",
                                                                  TRUE ~ payor_group)
  ) %>%
  filter(result %in% c("positive", "negative"))
```

Data wrangling - 2

```
# Generate counts
tmp_table_tall <- covid_testing_2 %>%
  group_by(payor_group_fac, result) %>%
  summarize(n = n()) %>%
  ungroup()
tmp_table_tall

# Pivot from tall to wide table
tmp_table_wide <- tmp_table_tall %>%
  spread(key = "result", value = "n")
tmp_table_wide
```

Remove groupings

Maps key values to separate columns

Testing for association

payor_group_fac <chr>	negative <int>	positive <int>
commercial	3549	86
government	3318	242
other	309	17
unassigned	7182	520

4 rows

```
data %>%
  fisher.test(simulate.p.value = T)
```

```
        Fisher's Exact Test for Count Data with simulated p-value (based on 2000 replicates)

data:  .
p-value = 0.0004998
alternative hypothesis: two.sided
```

Your Turn 4

Use fisher.test() to estimate the relative odds of a positive test result for patients with government insurance compared to commercial insurance?

What Else?

Logistic regression

```
tmp_fit <- glm(result_fac ~ payor_group_fac + age,  # model formula
               data = tmp,                          # dataset
               family = "binomial")                 # type of model

summary(tmp_fit)
```

Goals

1. Learn how to summarize and assess data

Objectives

1. Calculate a summary statistic for a variable using `summarize`
2. Calculate of summary statistic for a variable separately for a group of observations, using `group_by` and `summarize`
3. Perform a simple test for association

z

Appendix

R Markdown : : CHEAT SHEET

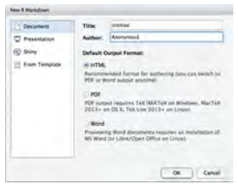
What is R Markdown?

.Rmd files • An R Markdown (.Rmd) file is a record of your research. It contains the code that a scientist needs to reproduce your work along with the narration that a reader needs to understand your work.

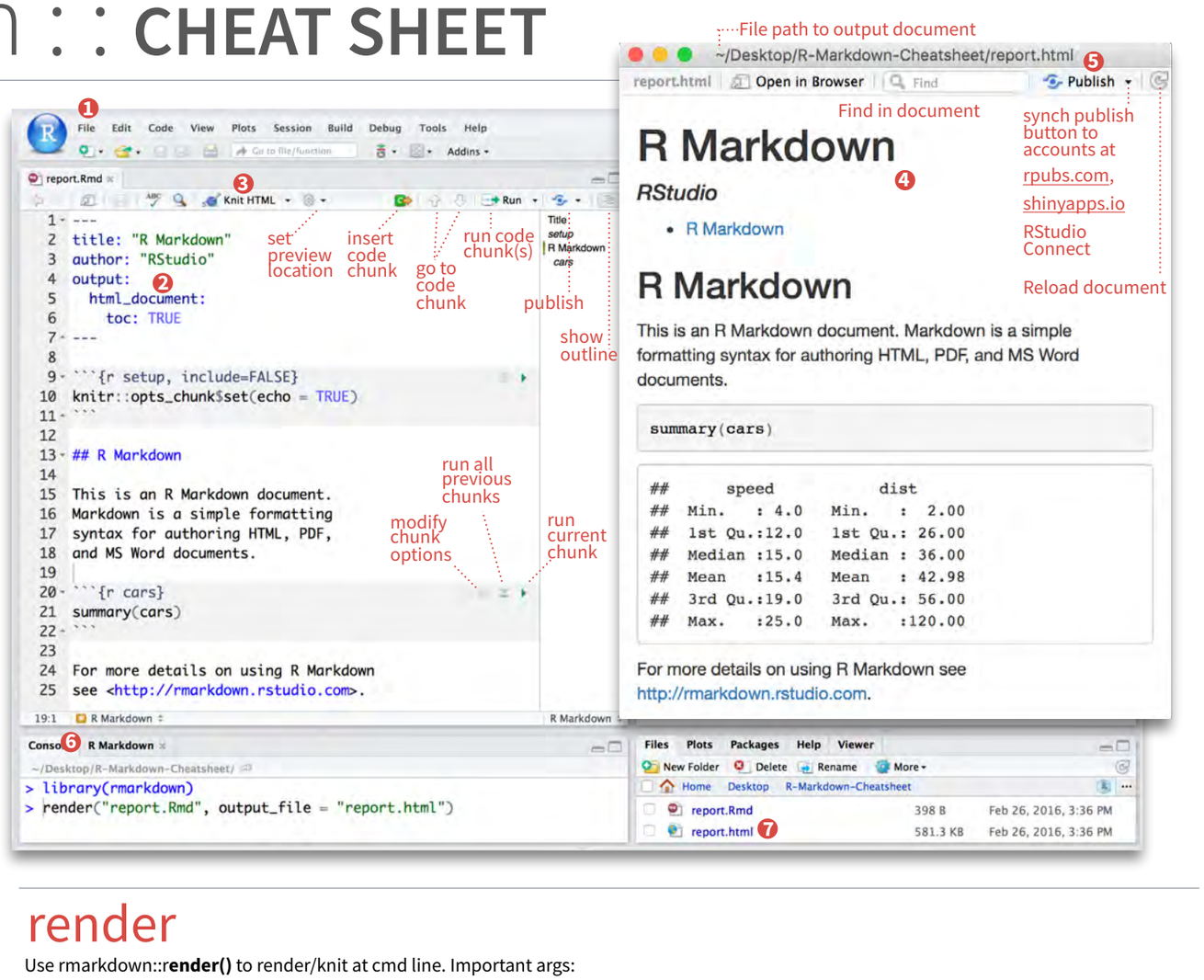
Reproducible Research • At the click of a button, or the type of a command, you can rerun the code in an R Markdown file to reproduce your work and export the results as a finished report.

Dynamic Documents • You can choose to export the finished report in a variety of formats, including html, pdf, MS Word, or RTF documents; html or pdf based slides, Notebooks, and more.

Workflow



1. **Open a new .Rmd file** at File ▶ New File ▶ R Markdown. Use the wizard that opens to pre-populate the file with a template
2. **Write document** by editing template
3. **Knit document to create report**; use knit button or **render()** to knit
4. **Preview Output** in IDE window
5. **Publish** (optional) to web server
6. **Examine build log** in R Markdown console
7. **Use output file** that is saved along side .Rmd



render

Use rmarkdown::**render()** to render/knit at cmd line. Important args:

input - file to render **output_format**	**output_options** - List of render options (as in YAML)	**output_file** **output_dir**	**params** - list of params to use	**envir** - environment to evaluate code chunks in	**encoding** - of input file

Embed code with knitr syntax

INLINE CODE

Insert with **`r <code>`**. Results appear as text without code.

Built with `` `r getRversion()` `` ➡ Built with 3.2.3

CODE CHUNKS

One or more lines surrounded with **```{r}** and **```**. Place chunk options within curly braces, after **r**. Insert with

```
```{r echo=TRUE}
getRversion()
```
```

➡

```
getRversion()
## [1] '3.2.3'
```

GLOBAL OPTIONS

Set with knitr::**opts_chunk$set()**, e.g.

```
```{r include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```
```

IMPORTANT CHUNK OPTIONS

cache - cache results for future knits (default = FALSE)

cache.path - directory to save cached results in (default = "cache/")

child - file(s) to knit and then include (default = NULL)

collapse - collapse all output into single block (default = FALSE)

comment - prefix for each line of results (default = '##')

dependson - chunk dependencies for caching (default = NULL)

echo - Display code in output document (default = TRUE)

engine - code language used in chunk (default = 'R')

error - Display error messages in doc (TRUE) or stop render when errors occur (FALSE) (default = FALSE)

eval - Run code in chunk (default = TRUE)

fig.align - 'left', 'right', or 'center' (default = 'default')

fig.cap - figure caption as character string (default = NULL)

fig.height, fig.width - Dimensions of plots in inches

highlight - highlight source code (default = TRUE)

include - Include chunk in doc after running (default = TRUE)

message - display code messages in document (default = TRUE)

results (default = 'markup')
'asis' - passthrough results
'hide' - do not display results
'hold' - put all results below all code

tidy - tidy code for display (default = FALSE)

warning - display code warnings in document (default = TRUE)

Options not listed above: R.options, aniopts, autodep, background, cache.comments, cache.lazy, cache.rebuild, cache.vars, dev, dev.args, dpi, engine.opts, engine.path, fig.asp, fig.env, fig.ext, fig.keep, fig.lp, fig.path, fig.pos, fig.process, fig.retina, fig.scap, fig.show, fig.showtext, fig.subcap, interval, out.extra, out.height, out.width, prompt, purl, ref.label, render, size, split, tidy.opts

.rmd Structure

YAML Header
Optional section of render (e.g. pandoc) options written as key:value pairs (YAML).

At start of file

Between lines of `---`

Text
Narration formatted with markdown, mixed with:

Code Chunks
Chunks of embedded code. Each chunk:

Begins with **```{r}**

ends with **```**

R Markdown will run the code and append the results to the doc.

It will use the location of the .Rmd file as the **working directory**

Parameters

Parameterize your documents to reuse with new inputs (e.g., data, values, etc.)

1. **Add parameters** • Create and set parameters in the header as sub-values of params

2. **Call parameters** • Call parameter values in code as params$<name>

3. **Set parameters** • Set values wth Knit with parameters or the params argument of render():

render("doc.Rmd", params = list(n = 1, d = as.Date("2015-01-01"))

```
---
params:
  n: 100
  d: !r Sys.Date()
---
```

```
Today's date
is `r params$d`
```

Interactive Documents

Turn your report into an interactive Shiny document in 4 steps

1. Add runtime: shiny to the YAML header.
2. Call Shiny input functions to embed input objects.
3. Call Shiny render functions to embed reactive output.
4. Render w rmarkdown::run or click Run Document in RStudio IDE

```
---
output: html_document
runtime: shiny
---

```{r, echo = FALSE}
numericInput("n",
  "How many cars?", 5)

renderTable({
  head(cars, input$n)
})
```
```

Embed a complete app into your document with shiny::**shinyAppDir()**

Publish on RStudio Connect, to share R Markdown documents securely, schedule automatic updates, and interact with parameters in real time. www.rstudio.com/products/connect/

RStudio

Pandoc's Markdown

Write with syntax on the left to create effect on right (after render)

```
Plain text
End a line with two spaces
to start a new paragraph.
*italics* and **bold**
`verbatim code`
sub/superscript^2^~2~
~~strikethrough~~
escaped: \* \_ \\
endash: --, emdash: ---
equation: $A = \pi*r^{2}$
equation block:

$$E = mc^{2}$$

> block quote

# Header1  {#anchor}

## Header 2  {#css_id}

### Header 3  {.css_class}

#### Header 4

##### Header 5

###### Header 6

<!--Text comment-->

\textbf{Tex ignored in HTML}
<em>HTML ignored in pdfs</em>

<http://www.rstudio.com>
[link](www.rstudio.com)
Jump to [Header 1](#anchor)
image:

![Caption](smallorb.png)

* unordered list
  + sub-item 1
  + sub-item 2
    - sub-sub-item 1

* item 2

  Continued (indent 4 spaces)

1. ordered list
2. item 2
   i) sub-item 1
      A.  sub-sub-item 1

(@)  A list whose numbering

continues after

(@)  an interruption

Term 1

:  Definition 1

| Right | Left | Default | Center |
|------:|:-----|---------|:------:|
|  12 | 12 |  12 |  12 |
| 123 | 123 | 123 | 123 |
|   1 |   1 |   1 |   1 |

- slide bullet 1
- slide bullet 2

(>- to have bullets appear on click)

horizontal rule/slide break:

***

A footnote [^1]

[^1]: Here is the footnote.
```

Plain text
End a line with two spaces
to start a new paragraph.
italics and **bold**
`verbatim code`
sub/superscript^2~2~
~~strikethrough~~
escaped: * _ \
endash: –, emdash: —
equation: $A = \pi * r^2$
equation block:

$$E = mc^2$$

> block quote

Header1

Header 2

Header 3

Header 4

Header 5

Header 6

HTML ignored in pdfs

http://www.rstudio.com
link
Jump to Header 1
image:

Caption

- unordered list
  - sub-item 1
  - sub-item 2
    - sub-sub-item 1
- item 2

  Continued (indent 4 spaces)

1. ordered list
2. item 2
   i. sub-item 1
      A. sub-sub-item 1

1. A list whose numbering

continues after

2. an interruption

Term 1

Definition 1

Right	Left	Default	Center
12	12	12	12
123	123	123	123
1	1	1	1

- slide bullet 1
- slide bullet 2

(>- to have bullets appear on click)

horizontal rule/slide break:

A footnote [1]

1. Here is the footnote.↩

Set render options with YAML

When you render, R Markdown

1. runs the R code, embeds results and text into .md file with knitr
2. then converts the .md file into the finished format with pandoc

Set a document's default output format in the YAML header:

```
---
output: html_document
---
# Body
```

output value	creates
html_document	html
pdf_document	pdf (requires Tex)
word_document	Microsoft Word (.docx)
odt_document	OpenDocument Text
rtf_document	Rich Text Format
md_document	Markdown
github_document	Github compatible markdown
ioslides_presentation	ioslides HTML slides
slidy_presentation	slidy HTML slides
beamer_presentation	Beamer pdf slides (requires Tex)

Customize output with sub-options (listed to the right):

Indent 2 spaces — Indent 4 spaces

```
---
output: html_document:
  code_folding: hide
  toc_float: TRUE
---
# Body
```

html tabsets

Use tablet css class to place sub-headers into tabs

```
# Tabset {.tabset .tabset-fade .tabset-pills}
## Tab 1
text 1
## Tab 2
text 2
### End tabset
```

Tabset

Tab 1 | Tab 2

text 1
End tabset

sub-option	description	html	pdf	word	odt	rtf	md	github	ioslides	slidy	beamer
citation_package	The LaTeX package to process citations, natbib, biblatex or none		X				X				X
code_folding	Let readers to toggle the display of R code, "none", "hide", or "show"	X									
colortheme	Beamer color theme to use										X
css	CSS file to use to style document	X							X	X	
dev	Graphics device to use for figure output (e.g. "png")	X	X				X	X	X	X	X
duration	Add a countdown timer (in minutes) to footer of slides									X	
fig_caption	Should figures be rendered with captions?	X	X	X	X				X	X	X
fig_height, fig_width	Default figure height and width (in inches) for document	X	X	X	X	X	X	X	X	X	X
highlight	Syntax highlighting: "tango", "pygments", "kate","zenburn", "textmate"	X	X	X						X	X
includes	File of content to place in document (in_header, before_body, after_body)	X	X		X		X	X	X	X	X
incremental	Should bullets appear one at a time (on presenter mouse clicks)?								X	X	X
keep_md	Save a copy of .md file that contains knitr output	X		X	X	X			X	X	
keep_tex	Save a copy of .tex file that contains knitr output		X								X
latex_engine	Engine to render latex, "pdflatex", "xelatex", or "lualatex"		X								X
lib_dir	Directory of dependency files to use (Bootstrap, MathJax, etc.)	X							X	X	
mathjax	Set to local or a URL to use a local/URL version of MathJax to render equations	X							X	X	
md_extensions	Markdown extensions to add to default definition or R Markdown	X	X	X	X	X	X	X	X	X	X
number_sections	Add section numbering to headers	X	X								
pandoc_args	Additional arguments to pass to Pandoc	X	X	X	X	X	X	X	X	X	X
preserve_yaml	Preserve YAML front matter in final document?						X				
reference_docx	docx file whose styles should be copied when producing docx output			X							
self_contained	Embed dependencies into the doc	X							X	X	
slide_level	The lowest heading level that defines individual slides										X
smaller	Use the smaller font size in the presentation?								X		
smart	Convert straight quotes to curly, dashes to em-dashes, … to ellipses, etc.	X							X	X	
template	Pandoc template to use when rendering file quarterly_report.html).	X	X		X					X	X
theme	Bootswatch or Beamer theme to use for page	X									X
toc	Add a table of contents at start of document	X	X	X		X	X	X			X
toc_depth	The lowest level of headings to add to table of contents	X	X	X		X	X	X			
toc_float	Float the table of contents to the left of the main content	X									

Create a Reusable Template

1. **Create a new package** with a inst/rmarkdown/templates directory
2. In the directory, **Place a folder** that contains:
   template.yaml (see below)
   skeleton.Rmd (contents of the template)
   any supporting files
3. **Install the package**
4. **Access template** in wizard at File ▶ New File ▶ R Markdown

template.yaml

```
---
name: My Template
—
```

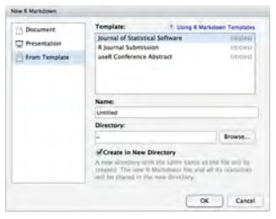


Table Suggestions

Several functions format R data into tables

Table with kable

eruptions	waiting
3.600	79
1.800	54
3.333	74
2.283	62

	eruptions	waiting
1	3.60	79.00
2	1.80	54.00
3	3.33	74.00
4	2.28	62.00

Table with xtable

Table with stargazer

	eruptions	waiting
1	3.600	79
2	1.800	54
3	3.333	74
4	2.283	62

```
data <- faithful[1:4, ]
```

```{r results = 'asis'}
knitr::kable(data, caption = "Table with kable")
```

```{r results = "asis"}
print(xtable::xtable(data, caption = "Table with xtable"),
      type = "html", html.table.attributes = "border=0"))
```

```{r results = "asis"}
stargazer::stargazer(data, type = "html", title = "Table
                     with stargazer")
```

Learn more in the **stargazer, xtable**, and **knitr** packages.

Citations and Bibliographies

Create citations with .bib, .bibtex, .copac, .enl, .json, .medline, .mods, .ris, .wos, and .xml files

1. **Set bibliography file** and CSL 1.0 Style file (optional) in the YAML header
2. **Use citation keys in text**

```
---
bibliography: refs.bib
csl: style.csl
---
```

```
Smith cited [@smith04].
Smith cited without author [-@smith04].
@smith04 cited in line.
```

3. **Render.** Bibliography will be added to end of document

```
Smith cited (Joe Smith 2004).
Smith cited without author (2004).
Joe Smith (2004) cited in line.
```

Data Import : : CHEAT SHEET

R's **tidyverse** is built around **tidy data** stored in **tibbles**, which are enhanced data frames.

The front side of this sheet shows how to read text files into R with **readr**.

The reverse side shows how to create tibbles with **tibble** and to layout tidy data with **tidyr**.

OTHER TYPES OF DATA

Try one of the following packages to import other types of files

- **haven** - SPSS, Stata, and SAS files
- **readxl** - excel files (.xls and .xlsx)
- **DBI** - databases
- **jsonlite** - json
- **xml2** - XML
- **httr** - Web APIs
- **rvest** - HTML (Web Scraping)

Save Data

Save **x**, an R object, to **path**, a file path, as:

Comma delimited file
write_csv(x, path, na = "NA", append = FALSE, col_names = !append**)**

File with arbitrary delimiter
write_delim(x, path, delim = " ", na = "NA", append = FALSE, col_names = !append**)**

CSV for excel
write_excel_csv(x, path, na = "NA", append = FALSE, col_names = !append**)**

String to file
write_file(x, path, append = FALSE**)**

String vector to file, one element per line
write_lines(x,path, na = "NA", append = FALSE**)**

Object to RDS file
write_rds(x, path, compress = c("none", "gz", "bz2", "xz"), ...**)**

Tab delimited files
write_tsv(x, path, na = "NA", append = FALSE, col_names = !append**)**

Read Tabular Data - These functions share the common arguments:

read_*(file, col_names = TRUE, col_types = NULL, locale = default_locale(), na = c("", "NA"), quoted_na = TRUE, comment = "", trim_ws = TRUE, skip = 0, n_max = Inf, guess_max = min(1000, n_max), progress = interactive()**)**

a,b,c
1,2,3
4,5,NA

A	B	C
1	2	3
4	5	NA

Comma Delimited Files
read_csv("file.csv"**)**
To make file.csv run:
write_file(x = "a,b,c\n1,2,3\n4,5,NA", path = "file.csv")

a;b;c
1;2;3
4;5;NA

A	B	C
1	2	3
4	5	NA

Semi-colon Delimited Files
read_csv2("file2.csv"**)**
write_file(x = "a;b;c\n1;2;3\n4;5;NA", path = "file2.csv")

a|b|c
1|2|3
4|5|NA

A	B	C
1	2	3
4	5	NA

Files with Any Delimiter
read_delim("file.txt", delim = "|"**)**
write_file(x = "a|b|c\n1|2|3\n4|5|NA", path = "file.txt")

a b c
1 2 3
4 5 NA

A	B	C
1	2	3
4	5	NA

Fixed Width Files
read_fwf("file.fwf", col_positions = c(1, 3, 5)**)**
write_file(x = "a b c\n1 2 3\n4 5 NA", path = "file.fwf")

Tab Delimited Files
read_tsv("file.tsv"**)** Also **read_table().**
write_file(x = "a\tb\tc\n1\t2\t3\n4\t5\tNA", path = "file.tsv")

USEFUL ARGUMENTS

a,b,c
1,2,3
4,5,NA

Example file
write_file("a,b,c\n1,2,3\n4,5,NA","file.csv")
f <- "file.csv"

A	B	C
1	2	3
4	5	NA

No header
read_csv(f, **col_names = FALSE**)

x	y	z
A	B	C
1	2	3
4	5	NA

Provide header
read_csv(f, **col_names = c("x", "y", "z")**)

1	2	3
4	5	NA

Skip lines
read_csv(f, **skip = 1**)

A	B	C
1	2	3

Read in a subset
read_csv(f, **n_max = 1**)

A	B	C
NA	2	3
4	5	NA

Missing Values
read_csv(f, **na = c("1", ".")**)

Read Non-Tabular Data

Read a file into a single string
read_file(file, locale = default_locale()**)**

Read each line into its own string
read_lines(file, skip = 0, n_max = -1L, na = character(), locale = default_locale(), progress = interactive()**)**

Read Apache style log files
read_log(file, col_names = FALSE, col_types = NULL, skip = 0, n_max = -1, progress = interactive()**)**

Read a file into a raw vector
read_file_raw(file**)**

Read each line into a raw vector
read_lines_raw(file, skip = 0, n_max = -1L, progress = interactive()**)**

Data types

readr functions guess the types of each column and convert types when appropriate (but will NOT convert strings to factors automatically).

A message shows the type of each column in the result.

```
## Parsed with column specification:
## cols(
##   age = col_integer(),
##   sex = col_character(),
##   earn = col_double()
## )
```

age is an integer
sex is a character
earn is a double (numeric)

1. Use **problems()** to diagnose problems.
x <- read_csv("file.csv"); problems(x)

2. Use a col_ function to guide parsing.
- **col_guess()** - the default
- **col_character()**
- **col_double()**, **col_euro_double()**
- **col_datetime(**format = ""**)** Also **col_date(**format = ""**)**, **col_time(**format = ""**)**
- **col_factor(**levels, ordered = FALSE**)**
- **col_integer()**
- **col_logical()**
- **col_number()**, **col_numeric()**
- **col_skip()**

```
x <- read_csv("file.csv", col_types = cols(
  A = col_double(),
  B = col_logical(),
  C = col_factor()))
```

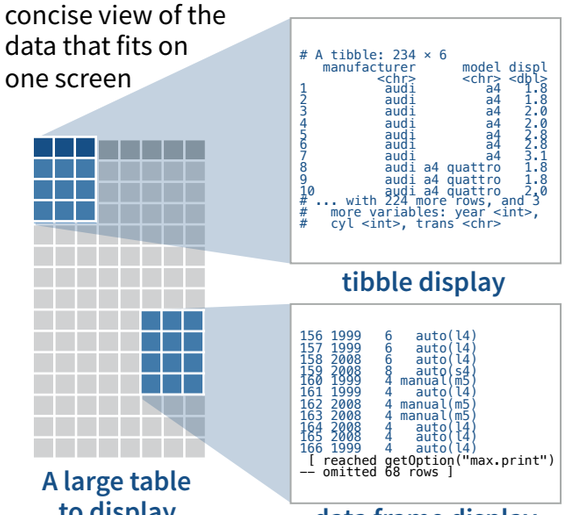
3. Else, read in as character vectors then parse with a parse_ function.
- **parse_guess()**
- **parse_character()**
- **parse_datetime()** Also **parse_date()** and **parse_time()**
- **parse_double()**
- **parse_factor()**
- **parse_integer()**
- **parse_logical()**
- **parse_number()**

```
x$A <- parse_number(x$A)
```

Tibbles - an enhanced data frame

The **tibble** package provides a new S3 class for storing tabular data, the tibble. Tibbles inherit the data frame class, but improve three behaviors:

- **Subsetting** - [always returns a new tibble, [[and $ always return a vector.
- **No partial matching** - You must use full column names when subsetting
- **Display** - When you print a tibble, R provides a concise view of the data that fits on one screen



- Control the default appearance with options:

  options(tibble.print_max = n, tibble.print_min = m, tibble.width = Inf**)**

- View full data set with **View()** or **glimpse()**
- Revert to data frame with **as.data.frame()**

CONSTRUCT A TIBBLE IN TWO WAYS

tibble(…)
Construct by columns.
tibble(x = 1:3, y = c("a", "b", "c"))

tribble(…)
Construct by rows.
tribble(~x, ~y, 1, "a", 2, "b", 3, "c")

Both make this tibble

```
A tibble: 3 × 2
      x     y
  <int> <chr>
1     1     a
2     2     b
3     3     c
```

as_tibble(x, …) Convert data frame to tibble.

enframe(x, name = "name", value = "value")
Convert named vector to a tibble

is_tibble(x) Test whether x is a tibble.

Tidy Data with tidyr

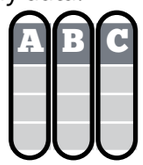
Tidy data is a way to organize tabular data. It provides a consistent data structure across packages.

A table is tidy if:

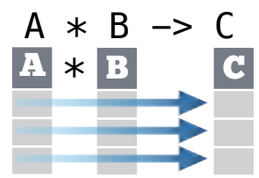
Each **variable** is in its own **column**

Each **observation**, or **case**, is in its own **row**

Tidy data:



Makes variables easy to access as vectors



Preserves cases during vectorized operations

Reshape Data - change the layout of values in a table

Use **gather()** and **spread()** to reorganize the values of a table into a new layout.

gather(data, key, value, ..., na.rm = FALSE, convert = FALSE, factor_key = FALSE**)**

gather() moves column names into a **key** column, gathering the column values into a single **value** column.

table4a

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

gather(table4a, `1999`, `2000`, key = "year", value = "cases")

spread(data, key, value, fill = NA, convert = FALSE, drop = TRUE, sep = NULL**)**

spread() moves the unique values of a **key** column into the column names, spreading the values of a **value** column across the new columns.

table2

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

key value

→

country	year	cases	pop
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

spread(table2, type, count)

Handle Missing Values

drop_na(data, ...**)**
Drop rows containing NA's in … columns.

x

x1	x2
A	1
B	NA
C	NA
D	3
E	NA

→

x1	x2
A	1
D	3

drop_na(x, x2)

fill(data, ..., .direction = c("down", "up")**)**
Fill in NA's in … columns with most recent non-NA values.

x

x1	x2
A	1
B	NA
C	NA
D	3
E	NA

→

x1	x2
A	1
B	1
C	1
D	3
E	3

fill(x, x2)

replace_na(data, replace = list(), ...**)**
Replace NA's by column.

x

x1	x2
A	1
B	NA
C	NA
D	3
E	NA

→

x1	x2
A	1
B	2
C	2
D	3
E	2

replace_na(x, list(x2 = 2))

Expand Tables - quickly create tables with combinations of values

complete(data, ..., fill = list()**)**
Adds to the data missing combinations of the values of the variables listed in …
complete(mtcars, cyl, gear, carb)

expand(data, ...**)**
Create new tibble with all possible combinations of the values of the variables listed in …
expand(mtcars, cyl, gear, carb)

Split Cells

Use these functions to split or combine cells into individual, isolated values.

separate(data, col, into, sep = "[^[:alnum:]]+", remove = TRUE, convert = FALSE, extra = "warn", fill = "warn", ...**)**

Separate each cell in a column to make several columns.

table3

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(table3, rate, sep = "/", into = c("cases", "pop"))

separate_rows(data, ..., sep = "[^[:alnum:].]+", convert = FALSE**)**

Separate each cell in a column to make several rows.

table3

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

separate_rows(table3, rate, sep = "/")

unite(data, col, ..., sep = "_", remove = TRUE**)**

Collapse cells across several columns to make a single column.

table5

country	century	year
Afghan	19	99
Afghan	20	00
Brazil	19	99
Brazil	20	00
China	19	99
China	20	00

→

country	year
Afghan	1999
Afghan	2000
Brazil	1999
Brazil	2000
China	1999
China	2000

unite(table5, century, year, col = "year", sep = "")

Data Transformation with dplyr : : CHEAT SHEET

dplyr functions work with pipes and expect **tidy data**. In tidy data:

Each **variable** is in its own **column**

&

Each **observation**, or **case**, is in its own **row**

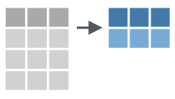
pipes

x %>% f(y) becomes **f(x, y)**

Summarise Cases

These apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

summary function



summarise(.data, …)
Compute table of summaries.
summarise(mtcars, avg = mean(mpg))

count(x, ..., wt = NULL, sort = FALSE)
Count number of rows in each group defined by the variables in … Also **tally**().
count(iris, Species)

VARIATIONS

summarise_all() - Apply funs to every column.
summarise_at() - Apply funs to specific columns.
summarise_if() - Apply funs to all cols of one type.

Group Cases

Use **group_by()** to create a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.

```
mtcars %>%
group_by(cyl) %>%
summarise(avg = mean(mpg))
```

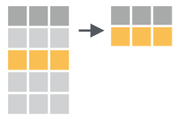
group_by(.data, ..., add = FALSE)
Returns copy of table grouped by …
g_iris <- group_by(iris, Species)

ungroup(x, …)
Returns ungrouped copy of table.
ungroup(g_iris)

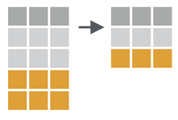
Manipulate Cases

EXTRACT CASES

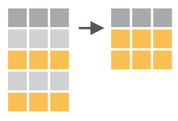
Row functions return a subset of rows as a new table.



filter(.data, …**)** Extract rows that meet logical criteria. *filter(iris, Sepal.Length > 7)*

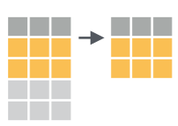


distinct(.data, ..., .keep_all = FALSE**)** Remove rows with duplicate values.
distinct(iris, Species)



sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame()**)** Randomly select fraction of rows.
sample_frac(iris, 0.5, replace = TRUE)

sample_n(tbl, size, replace = FALSE, weight = NULL, .env = parent.frame()**)** Randomly select size rows. *sample_n(iris, 10, replace = TRUE)*



slice(.data, …**)** Select rows by position.
slice(iris, 10:15)

top_n(x, n, wt**)** Select and order top n entries (by group if grouped data). *top_n(iris, 5, Sepal.Width)*

Logical and boolean operators to use with filter()

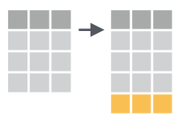
<	<=	is.na()	%in%	|	xor()
>	>=	!is.na()	!	&	

See **?base::Logic** and **?Comparison** for help.

ARRANGE CASES

arrange(.data, …**)** Order rows by values of a column or columns (low to high), use with **desc()** to order from high to low.
arrange(mtcars, mpg)
arrange(mtcars, desc(mpg))

ADD CASES



add_row(.data, ..., .before = NULL, .after = NULL**)**
Add one or more rows to a table.
add_row(faithful, eruptions = 1, waiting = 1)

Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.

pull(.data, var = -1**)** Extract column values as a vector. Choose by name or index.
pull(iris, Sepal.Length)

select(.data, …**)**
Extract columns as a table. Also **select_if()**.
select(iris, Sepal.Length, Species)

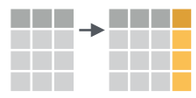
Use these helpers with select (),
e.g. select(iris, starts_with("Sepal"))

contains(match**)**	**num_range(**prefix, range**)**	**:**, e.g. mpg:cyl
ends_with(match**)**	**one_of(**…**)**	**-**, e.g, -Species
matches(match**)**	**starts_with(**match**)**	

MAKE NEW VARIABLES

These apply **vectorized functions** to columns. Vectorized funs take vectors as input and return vectors of the same length as output (see back).

vectorized function

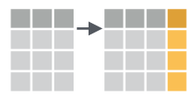


mutate(.data, …**)**
Compute new column(s).
mutate(mtcars, gpm = 1/mpg)

transmute(.data, …**)**
Compute new column(s), drop others.
transmute(mtcars, gpm = 1/mpg)

mutate_all(.tbl, .funs, …**)** Apply funs to every column. Use with **funs()**. Also **mutate_if()**.
mutate_all(faithful, funs(log(.), log2(.)))
mutate_if(iris, is.numeric, funs(log(.)))

mutate_at(.tbl, .cols, .funs, …**)** Apply funs to specific columns. Use with **funs()**, **vars()** and the helper functions for select().
mutate_at(iris, vars(-Species), funs(log(.)))



add_column(.data, ..., .before = NULL, .after = NULL**)** Add new column(s). Also **add_count()**, **add_tally()**. *add_column(mtcars, new = 1:32)*

rename(.data, …**)** Rename columns.
rename(iris, Length = Sepal.Length)

Vector Functions

TO USE WITH MUTATE ()

mutate() and **transmute()** apply vectorized functions to columns to create new columns. Vectorized functions take vectors as input and return vectors of the same length as output.

vectorized function →

OFFSETS

dplyr::**lag()** - Offset elements by 1
dplyr::**lead()** - Offset elements by -1

CUMULATIVE AGGREGATES

dplyr::**cumall()** - Cumulative all()
dplyr::**cumany()** - Cumulative any()
cummax() - Cumulative max()
dplyr::**cummean()** - Cumulative mean()
cummin() - Cumulative min()
cumprod() - Cumulative prod()
cumsum() - Cumulative sum()

RANKINGS

dplyr::**cume_dist()** - Proportion of all values <=
dplyr::**dense_rank()** - rank w ties = min, no gaps
dplyr::**min_rank()** - rank with ties = min
dplyr::**ntile()** - bins into n bins
dplyr::**percent_rank()** - min_rank scaled to [0,1]
dplyr::**row_number()** - rank with ties = "first"

MATH

+, - , *, /, ^, %/%, %% - arithmetic ops
log(), log2(), log10() - logs
<, <=, >, >=, !=, == - logical comparisons
dplyr::**between()** - x >= left & x <= right
dplyr::**near()** - safe == for floating point numbers

MISC

dplyr::**case_when()** - multi-case if_else()

```
iris %>% mutate(Species = case_when(
                Species == "versicolor" ~ "versi",
                Species == "virginica"  ~ "virgi",
                           TRUE ~ Species))
```

dplyr::**coalesce()** - first non-NA values by element across a set of vectors
dplyr::**if_else()** - element-wise if() + else()
dplyr::**na_if()** - replace specific values with NA
pmax() - element-wise max()
pmin() - element-wise min()
dplyr::**recode()** - Vectorized switch()
dplyr::**recode_factor()** - Vectorized switch() for factors

Summary Functions

TO USE WITH SUMMARISE ()

summarise() applies summary functions to columns to create a new table. Summary functions take vectors as input and return single values as output.

summary function →

COUNTS

dplyr::**n()** - number of values/rows
dplyr::**n_distinct()** - # of uniques
sum(!is.na()) - # of non-NA's

LOCATION

mean() - mean, also **mean(!is.na())**
median() - median

LOGICALS

mean() - Proportion of TRUE's
sum() - # of TRUE's

POSITION/ORDER

dplyr::**first()** - first value
dplyr::**last()** - last value
dplyr::**nth()** - value in nth location of vector

RANK

quantile() - nth quantile
min() - minimum value
max() - maximum value

SPREAD

IQR() - Inter-Quartile Range
mad() - median absolute deviation
sd() - standard deviation
var() - variance

Row Names

Tidy data does not use rownames, which store a variable outside of the columns. To work with the rownames, first move them into a column.

rownames_to_column()
Move row names into col.
a <- rownames_to_column(iris, var = "C")

column_to_rownames()
Move col in row names.
column_to_rownames(a, var = "C")

Also **has_rownames()**, **remove_rownames()**

Combine Tables

COMBINE VARIABLES

x + y =

Use **bind_cols()** to paste tables beside each other as they are.

bind_cols(…) Returns tables placed side by side as a single table.
BE SURE THAT ROWS ALIGN.

Use a "**Mutating Join**" to join one table to columns from another, matching values with the rows that they correspond to. Each join retains a different combination of values from the tables.

left_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),…**)**
Join matching values from y to x.

right_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),…**)**
Join matching values from x to y.

inner_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),…**)**
Join data. Retain only rows with matches.

full_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),…**)**
Join data. Retain all values, all rows.

Use **by = c("col1", "col2", …)** to specify one or more common columns to match on.
left_join(x, y, by = "A")

Use a named vector, **by = c("col1" = "col2")**, to match on columns that have different names in each table.
left_join(x, y, by = c("C" = "D"))

Use **suffix** to specify the suffix to give to unmatched columns that have the same name in both tables.
left_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))

COMBINE CASES

x + y =

Use **bind_rows()** to paste tables below each other as they are.

bind_rows(…, .id = NULL**)**
Returns tables one on top of the other as a single table. Set .id to a column name to add a column of the original table names (as pictured)

intersect(x, y, …)
Rows that appear in both x and y.

setdiff(x, y, …)
Rows that appear in x but not y.

union(x, y, …)
Rows that appear in x or y. (Duplicates removed). union_all() retains duplicates.

Use **setequal()** to test whether two data sets contain the exact same rows (in any order).

EXTRACT ROWS

x + y =

Use a "**Filtering Join**" to filter one table against the rows of another.

semi_join(x, y, by = NULL, …**)**
Return rows of x that have a match in y. USEFUL TO SEE WHAT WILL BE JOINED.

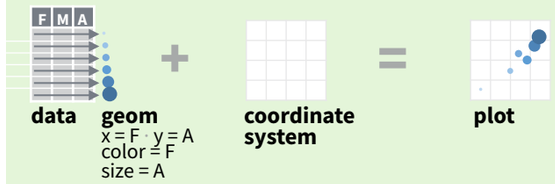
anti_join(x, y, by = NULL, …**)**
Return rows of x that do not have a match in y. USEFUL TO SEE WHAT WILL NOT BE JOINED.

Data Visualization with ggplot2 : : CHEAT SHEET

Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same components: a **data** set, a **coordinate system**, and geoms—visual marks that represent data points.

To display values, map variables in the data to visual properties of the geom (**aesthetics**) like **size**, **color**, and **x** and **y** locations.



Complete the template below to build a graph.

```
ggplot (data = <DATA>) +                                  required
<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>),
 stat = <STAT>, position = <POSITION> ) +                 Not required, sensible defaults supplied
<COORDINATE_FUNCTION> +
<FACET_FUNCTION> +
<SCALE_FUNCTION> +
<THEME_FUNCTION>
```

ggplot(data = mpg, **aes(**x = cty, y = hwy**))** Begins a plot that you finish by adding layers to. Add one geom function per layer.

aesthetic mappings · data · geom

qplot(x = cty, y = hwy, data = mpg, geom = "point"**)** Creates a complete plot with given data, geom, and mappings. Supplies many useful defaults.

last_plot() Returns the last plot

ggsave("plot.png", width = 5, height = 5) Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

Geoms

Use a geom function to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

GRAPHICAL PRIMITIVES

a <- ggplot(economics, aes(date, unemploy))
b <- ggplot(seals, aes(x = long, y = lat))

a + geom_blank()
(Useful for expanding limits)

b + geom_curve(aes(yend = lat + 1, xend=long+1),curvature=1**)** - x, xend, y, yend, alpha, angle, color, curvature, linetype, size

a + geom_path(lineend="butt", linejoin="round", linemitre=1**)**
x, y, alpha, color, group, linetype, size

a + geom_polygon(aes(group = group)**)**
x, y, alpha, color, fill, group, linetype, size

b + geom_rect(aes(xmin = long, ymin=lat, xmax= long + 1, ymax = lat + 1)**)** - xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

a + geom_ribbon(aes(ymin=unemploy - 900, ymax=unemploy + 900)**)** - x, ymax, ymin, alpha, color, fill, group, linetype, size

LINE SEGMENTS

common aesthetics: x, y, alpha, color, linetype, size

b + geom_abline(aes(intercept=0, slope=1)**)**
b + geom_hline(aes(yintercept = lat)**)**
b + geom_vline(aes(xintercept = long)**)**

b + geom_segment(aes(yend=lat+1, xend=long+1)**)**
b + geom_spoke(aes(angle = 1:1155, radius = 1)**)**

ONE VARIABLE continuous

c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)

c + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size

c + geom_density(kernel = "gaussian"**)**
x, y, alpha, color, fill, group, linetype, size, weight

c + geom_dotplot()
x, y, alpha, color, fill

c + geom_freqpoly() x, y, alpha, color, group, linetype, size

c + geom_histogram(binwidth = 5**)** x, y, alpha, color, fill, linetype, size, weight

c2 + geom_qq(aes(sample = hwy)**)** x, y, alpha, color, fill, linetype, size, weight

discrete

d <- ggplot(mpg, aes(fl))

d + geom_bar()
x, alpha, color, fill, linetype, size, weight

TWO VARIABLES

continuous x , continuous y

e <- ggplot(mpg, aes(cty, hwy))

e + geom_label(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE**)** x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

e + geom_jitter(height = 2, width = 2**)**
x, y, alpha, color, fill, shape, size

e + geom_point(), x, y, alpha, color, fill, shape, size, stroke

e + geom_quantile(), x, y, alpha, color, group, linetype, size, weight

e + geom_rug(sides = "bl"**)**, x, y, alpha, color, linetype, size

e + geom_smooth(method = lm**)**, x, y, alpha, color, fill, group, linetype, size, weight

e + geom_text(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE**)**, x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

discrete x , continuous y

f <- ggplot(mpg, aes(class, hwy))

f + geom_col(), x, y, alpha, color, fill, group, linetype, size

f + geom_boxplot(), x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight

f + geom_dotplot(binaxis = "y", stackdir = "center"**)**, x, y, alpha, color, fill, group

f + geom_violin(scale = "area"**)**, x, y, alpha, color, fill, group, linetype, size, weight

discrete x , discrete y

g <- ggplot(diamonds, aes(cut, color))

g + geom_count(), x, y, alpha, color, fill, shape, size, stroke

continuous bivariate distribution

h <- ggplot(diamonds, aes(carat, price))

h + geom_bin2d(binwidth = c(0.25, 500)**)**
x, y, alpha, color, fill, linetype, size, weight

h + geom_density2d()
x, y, alpha, colour, group, linetype, size

h + geom_hex()
x, y, alpha, colour, fill, size

continuous function

i <- ggplot(economics, aes(date, unemploy))

i + geom_area()
x, y, alpha, color, fill, linetype, size

i + geom_line()
x, y, alpha, color, group, linetype, size

i + geom_step(direction = "hv")
x, y, alpha, color, group, linetype, size

visualizing error

df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
j <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))

j + geom_crossbar(fatten = 2**)**
x, y, ymax, ymin, alpha, color, fill, group, linetype, size

j + geom_errorbar(), x, ymax, ymin, alpha, color, group, linetype, size, width (also **geom_errorbarh()**)

j + geom_linerange()
x, ymin, ymax, alpha, color, group, linetype, size

j + geom_pointrange()
x, y, ymin, ymax, alpha, color, fill, group, linetype, shape, size

maps

data <- data.frame(murder = USArrests$Murder, state = tolower(rownames(USArrests)))
map <- map_data("state")
k <- ggplot(data, aes(fill = murder))

k + geom_map(aes(map_id = state), map = map**)**
+ expand_limits(x = map$long, y = map$lat**)**,
map_id, alpha, color, fill, linetype, size

THREE VARIABLES

seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2)); l <- ggplot(seals, aes(long, lat))

l + geom_contour(aes(z = z))
x, y, z, alpha, colour, group, linetype, size, weight

l + geom_raster(aes(fill = z), hjust=0.5, vjust=0.5, interpolate=FALSE**)**
x, y, alpha, fill

l + geom_tile(aes(fill = z)), x, y, alpha, color, fill, linetype, size, width

Stats An alternative way to build a layer

A stat builds new variables to plot (e.g., count, prop).

Visualize a stat by changing the default stat of a geom function, **geom_bar(stat="count")** or by using a stat function, **stat_count(geom="bar")**, which calls a default geom to make a layer (equivalent to a geom function). Use **..name..** syntax to map stat variables to aesthetics.

geom to use | stat function | geommappings

i + stat_density2d(aes(fill = ..level..), geom = "polygon")

variable created by stat

c + stat_bin(binwidth = 1, origin = 10)
x, y | ..count.., ..ncount.., ..density.., ..ndensity..

c + stat_count(width = 1) **x, y,** | ..count.., ..prop..

c + stat_density(adjust = 1, kernel = "gaussian")
x, y, | ..count.., ..density.., ..scaled..

e + stat_bin_2d(bins = 30, drop = T)
x, y, fill | ..count.., ..density..

e + stat_bin_hex(bins=30) **x, y, fill** | ..count.., ..density..

e + stat_density_2d(contour = TRUE, n = 100)
x, y, color, size | ..level..

e + stat_ellipse(level = 0.95, segments = 51, type = "t")

l + stat_contour(aes(z = z)) **x, y, z, order** | ..level..

l + stat_summary_hex(aes(z = z), bins = 30, fun = max)
x, y, z, fill | ..value..

l + stat_summary_2d(aes(z = z), bins = 30, fun = mean)
x, y, z, fill | ..value..

f + stat_boxplot(coef = 1.5) **x, y** | ..lower.., ..middle.., ..upper.., ..width.. , ..ymin.., ..ymax..

f + stat_ydensity(kernel = "gaussian", scale = "area") **x, y** | ..density.., ..scaled.., ..count.., ..n.., ..violinwidth.., ..width..

e + stat_ecdf(n = 40) **x, y** | ..x.., ..y..

e + stat_quantile(quantiles = c(0.1, 0.9), formula = y ~ log(x), method = "rq") **x, y** | ..quantile..

e + stat_smooth(method = "lm", formula = y ~ x, se=T, level=0.95) **x, y** | ..se.., ..x.., ..y.., ..ymin.., ..ymax..

ggplot() + stat_function(aes(x = -3:3), n = 99, fun = dnorm, args = list(sd=0.5)) **x** | ..x.., ..y..

e + stat_identity(na.rm = TRUE)

ggplot() + stat_qq(aes(sample=1:100), dist = qt, dparam=list(df=5)) **sample, x, y** | ..sample.., ..theoretical..

e + stat_sum() x, y, size | ..n.., ..prop..

e + stat_summary(fun.data = "mean_cl_boot")

h + stat_summary_bin(fun.y = "mean", geom = "bar")

e + stat_unique()

Scales

Scales map data values to the visual values of an aesthetic. To change a mapping, add a new scale.

(n <- d + geom_bar(aes(fill = fl)))

scale_ | aesthetic to adjust | prepackaged scale to use | scale-specific arguments

n + scale_fill_manual(
values = c("skyblue", "royalblue", "blue", "navy"),
limits = c("d", "e", "p", "r"), breaks =c("d", "e", "p", "r"),
name = "fuel", labels = c("D", "E", "P", "R"))

range of values to include in mapping | title to use in legend/axis | labels to use in legend/axis | breaks to use in legend/axis

GENERAL PURPOSE SCALES

Use with most aesthetics

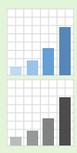
scale_*_continuous() - map cont' values to visual ones
scale_*_discrete() - map discrete values to visual ones
scale_*_identity() - use data values **as** visual ones
scale_*_manual(values = c()) - map discrete values to manually chosen visual ones
scale_*_date(date_labels = "%m/%d"), date_breaks = "2 weeks") - treat data values as dates.
scale_*_datetime() - treat data x values as date times. Use same arguments as scale_x_date(). See ?strptime for label formats.

X & Y LOCATION SCALES

Use with x or y aesthetics (x shown here)

scale_x_log10() - Plot x on log10 scale
scale_x_reverse() - Reverse direction of x axis
scale_x_sqrt() - Plot x on square root scale

COLOR AND FILL SCALES (DISCRETE)

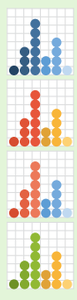


n <- d + geom_bar(aes(fill = fl))

n + scale_fill_brewer(palette = "Blues")
For palette choices:
RColorBrewer::display.brewer.all()

n + scale_fill_grey(start = 0.2, end = 0.8, na.value = "red")

COLOR AND FILL SCALES (CONTINUOUS)



o <- c + geom_dotplot(aes(fill = ..x..))

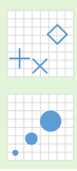
o + scale_fill_distiller(palette = "Blues")

o + scale_fill_gradient(low="red", high="yellow")

o + scale_fill_gradient2(low="red", high="blue", mid = "white", midpoint = 25)

o + scale_fill_gradientn(colours=topo.colors(6))
Also: rainbow(), heat.colors(), terrain.colors(), cm.colors(), RColorBrewer::brewer.pal()

SHAPE AND SIZE SCALES



p <- e + geom_point(aes(shape = fl, size = cyl))
p + scale_shape() + scale_size()
p + scale_shape_manual(values = c(3:7))

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

p + scale_radius(range = c(1,6))
p + scale_size_area(max_size = 6)

Coordinate Systems

r <- d + geom_bar()

r + coord_cartesian(xlim = c(0, 5))
xlim, ylim
The default cartesian coordinate system

r + coord_fixed(ratio = 1/2)
ratio, xlim, ylim
Cartesian coordinates with fixed aspect ratio between x and y units

r + coord_flip()
xlim, ylim
Flipped Cartesian coordinates

r + coord_polar(theta = "x", direction=1)
theta, start, direction
Polar coordinates

r + coord_trans(ytrans = "sqrt")
xtrans, ytrans, limx, limy
Transformed cartesian coordinates. Set xtrans and ytrans to the name of a window function.

π + coord_quickmap()
π + coord_map(projection = "ortho", orientation=c(41, -74, 0))projection, xlim, ylim
Map projections from the mapproj package (mercator (default), azequalarea, lagrange, etc.)

Position Adjustments

Position adjustments determine how to arrange geoms that would otherwise occupy the same space.

s <- ggplot(mpg, aes(fl, fill = drv))

s + geom_bar(position = "dodge")
Arrange elements side by side

s + geom_bar(position = "fill")
Stack elements on top of one another, normalize height

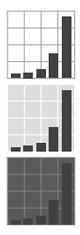
e + geom_point(position = "jitter")
Add random noise to X and Y position of each element to avoid overplotting

e + geom_label(position = "nudge")
Nudge labels away from points

s + geom_bar(position = "stack")
Stack elements on top of one another

Each position adjustment can be recast as a function with manual **width** and **height** arguments
s + geom_bar(position = position_dodge(width = 1))

Themes



r + theme_bw()
White background with grid lines

r + theme_gray()
Grey background (default theme)

r + theme_dark()
dark for contrast

r + theme_classic()

r + theme_light()

r + theme_linedraw()

r + theme_minimal()
Minimal themes

r + theme_void()
Empty theme

Faceting

Facets divide a plot into subplots based on the values of one or more discrete variables.

t <- ggplot(mpg, aes(cty, hwy)) + geom_point()

t + facet_grid(cols = vars(fl))
facet into columns based on fl

t + facet_grid(rows = vars(year))
facet into rows based on year

t + facet_grid(rows = vars(year), cols = vars(fl))
facet into both rows and columns

t + facet_wrap(vars(fl))
wrap facets into a rectangular layout

Set **scales** to let axis limits vary across facets

t + facet_grid(rows = vars(drv), cols = vars(fl), scales = "free")
x and y axis limits adjust to individual facets
"free_x" - x axis limits adjust
"free_y" - y axis limits adjust

Set **labeller** to adjust facet labels

t + facet_grid(cols = vars(fl), labeller = label_both)

fl: c	fl: d	fl: e	fl: p	fl: r

t + facet_grid(rows = vars(fl), labeller = label_bquote(alpha ^ .(fl)))

α^c	α^d	α^e	α^p	α^r

Labels

t + labs(**x** = "New x axis label", **y** = "New y axis label",
title ="Add a title above the plot",
subtitle = "Add a subtitle below title",
caption = "Add a caption below plot",
<AES> = "New <AES> legend title")

Use scale functions to update legend labels

t + annotate(geom = "text", x = 8, y = 9, label = "A")

geom to place | manual values for geom's aesthetics

Legends

n + theme(legend.position = "bottom")
Place legend at "bottom", "top", "left", or "right"

n + guides(fill = "none")
Set legend type for each aesthetic: colorbar, legend, or none (no legend)

n + scale_fill_discrete(name = "Title", labels = c("A", "B", "C", "D", "E"))
Set legend title and labels with a scale function.

Zooming

Without clipping (preferred)

t + coord_cartesian(
xlim = c(0, 100), ylim = c(10, 20))

With clipping (removes unseen data points)

t + xlim(0, 100) **+ ylim**(10, 20)

t + scale_x_continuous(limits = c(0, 100)) **+ scale_y_continuous(**limits = c(0, 100))

Resources for Future Learning

After this workshop you will have a foundation for building future knowledge and skills in R and data analysis. However, we have barely scratched the surface in terms of what R is, what R can do, and more generally how to code or do data analysis. Below are our favorite resources for learning R.

The most comprehensive inventory of R related resources and educational material can be found on RStudio's website. Check out the huge inventory of guidelines, workshop material, videos, and other useful content.

RStudio Cheatsheets

Very well-designed (if somewhat dense) two-page overview cards for specific R packages or topics. We recommend:

Data Import Cheatsheet: https://github.com/rstudio/cheatsheets/raw/master/data-import.pdf

Data Visualization Cheatsheet: https://github.com/rstudio/cheatsheets/raw/master/data-visualization-2.1.pdf

Data Transformation Cheatsheet: https://github.com/rstudio/cheatsheets/raw/master/data-transformation.pdf

R Markdown Cheatsheet: https://github.com/rstudio/cheatsheets/raw/master/rmarkdown-2.0.pdf

---

Massive Open Online Courses (MOOCs)/online courses

The Johns Hopkins University Data Science Specialization

- An excellent series of lessons teaching the A to Z of data science from some of the earliest and best team of R educators.

Harvard Data Science Specialization

- A fantastic data science course with a leading biostatistician and data scientist.

stat545

- Data wrangling, exploration, and analysis with R, one of best courses teaching data munging and all things R, initially taught by statistician Jenny Bryan at UBC.

Online/Free Textbooks

R for Data Science by Hadley Wickham and Garrett Grolemund

- The definitive text on data science via the tidyverse by the leading R developer Hadley Wickham.

Introduction to Data Science by Rafael Irizarry

- Raf Irizarry, Harvard Professor and fantastic teacher has published a wonderful introductory Data Science Book.

An Introduction to Statistical and Data Sciences via R byChester Ismay and Albert Y. Kim

- A fantastic introduction to R via statistical inference.

Fundamentals of Data Visualization by Clause Wilke

- Claus Wilke, a professor from UT Austin has written a highly useful ggplot add-on package and now has a new book on data visualization.

Youtube videos

Anything with Hadley Wickham, including:

- Hadley Wickham's "dplyr" tutorial at useR 2014
- Stanford Seminar - Expressing yourself in R

As well as Garrett Grolemund's Data Wrangling Series

Websites/Blogs

R Bloggers is a blog aggregator which captures a lot of the most prominent R bloggers on the internet

Rweekly.org is a weekly manual review of the latest and greatest in R internet content.

Rviews is the RStudio blog devoted to the R Community and the R Language.