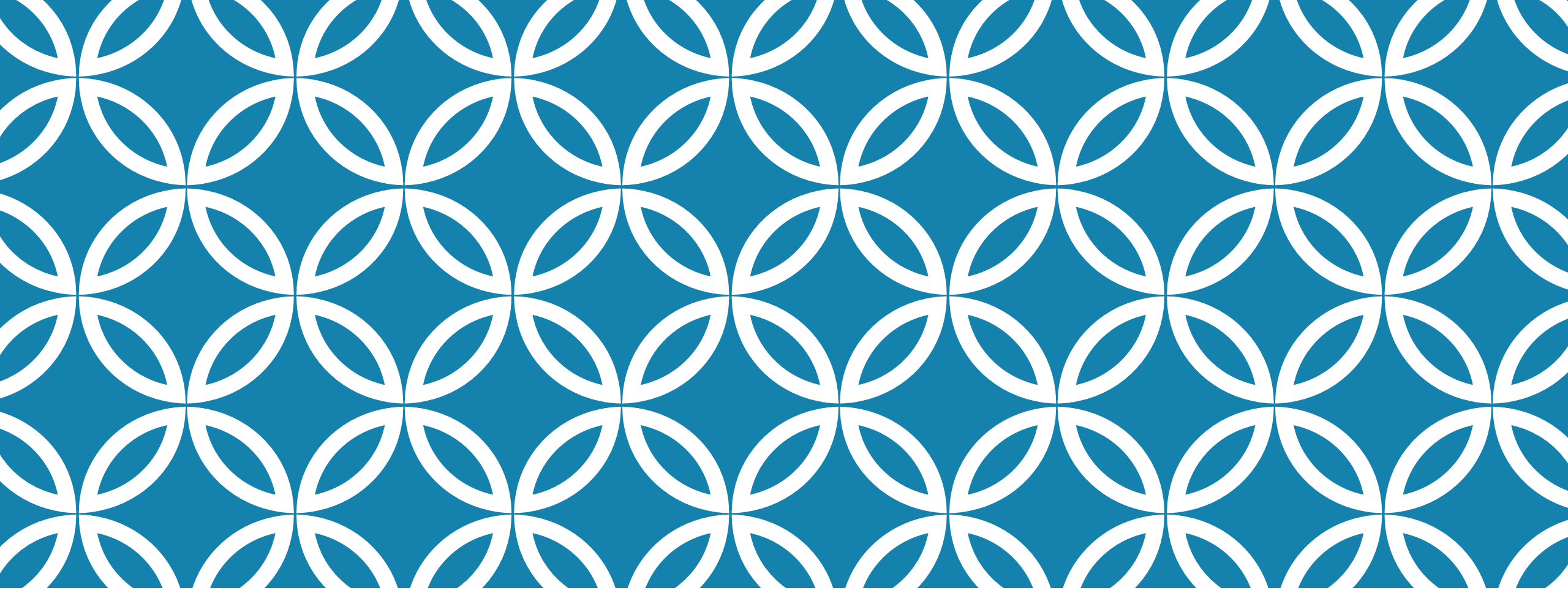
z

API Introduction to R Workshop

May 9, 2022



Introduction to R Workshop

Amrom Obstfeld
MD PhD

Course Introduction

Goals and Objectives

- Advocate for the use of R as a means of improving reproducibility in clinical data analysis
- Demonstrate how R is used to perform analyses of laboratory operational data
- Establish a basis of understanding in the 'tidy' approach to data analysis within the framework of R

Session	Instructor
Instructor Introductions, Introduction to technology	Amrom Obstfeld
Introduction to R and RStudio	Joe Rudolf
Reproducible Reporting	Joe Rudolf
Data Visualization	Stephan Kadauke
Data Transformation	Amrom Obstfeld
Group and Summarize	Patrick Mathias
Advanced Reporting	Patrick Mathias

Who are we?

Joseph Rudolf

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

Medical Director, Automated Core Laboratory, ARUP Laboratories

Patrick Mathias

Assistant Professor, Department of Laboratory Medicine and Pathology

University of Washington School of Medicine

Associate Medical Director, Laboratory Medicine and Pathology Informatics

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

Amrom Obstfeld

Assistant Professor of Clinical Pathology and Laboratory Medicine

University of Pennsylvania Perelman School of Medicine

Director of Pathology Informatics

Children's Hospital of Philadelphia

Workshop Workflow

Sessions

Loading Data to Create a Dataframe

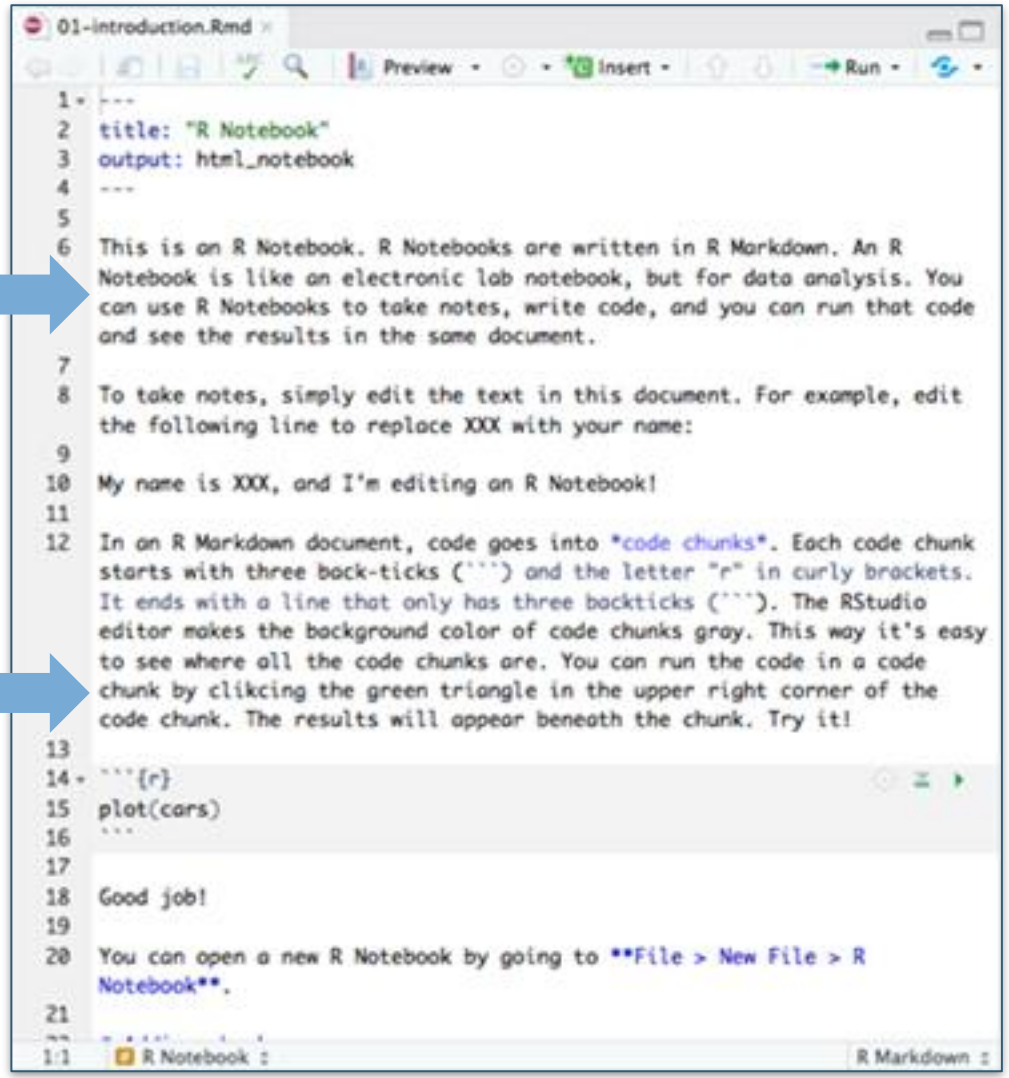
```
data_frame <- read_csv("file_name")
```

Your Turn

Introduce yourself to your neighbors

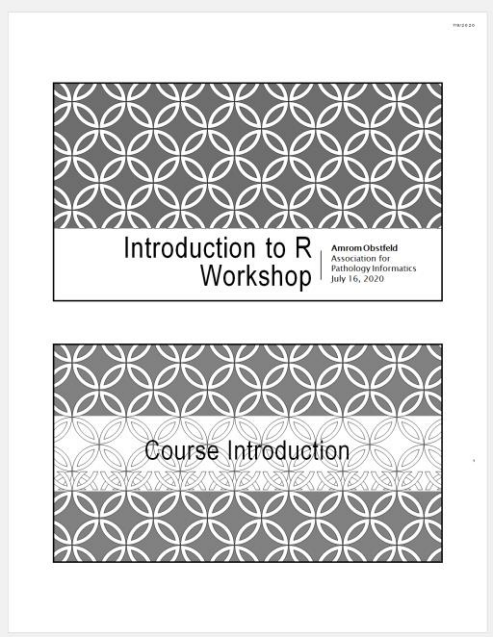
- Who are you?
- Where are you from?
- What do you do with data?
- Have you ever used R?

3:00



Workshop Coursebook

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



Who are you?

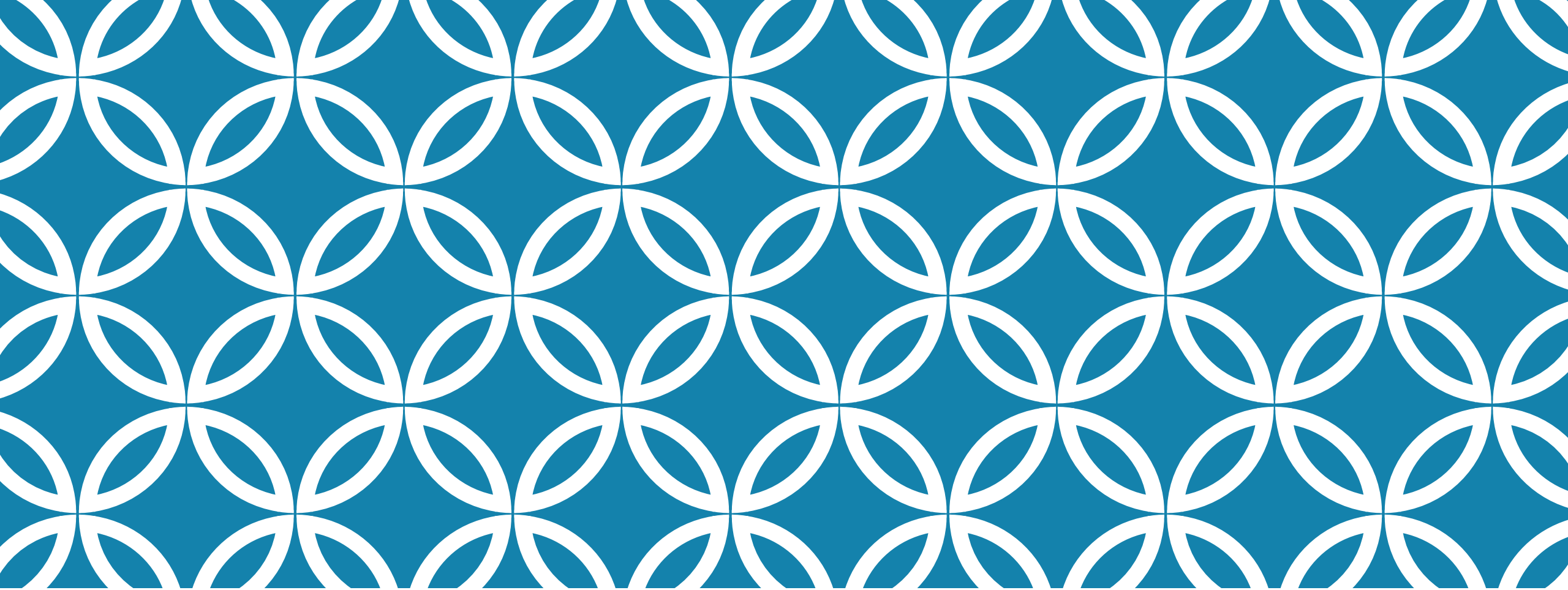
Your Turn

Introduce yourself to your breakout roommates

Who are you?
Where are you from?
Why are you here?
Have you ever used R?

Final Tips

- The best way to learn to code is by doing
- Practice is key!



Introduction to R, RStudio, and R Markdown

Joseph Rudolf

API-R 2022

Part I

R

Programming language for data analysis

RStudio

Interactive development environment (IDE)

R Markdown

Computational document format

Getting Started with RStudio

RStudio: On the Web and In Your Home

RStudio Server
Hosted on a server
(in the cloud)

RStudio Desktop
Installed locally on
your computer

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

Your Turn #1

Go to https://api-r.cloud in your browser and log in using the username and password provided in the course email.

Click “thumbs up” in zoom once you see the RStudio panes.

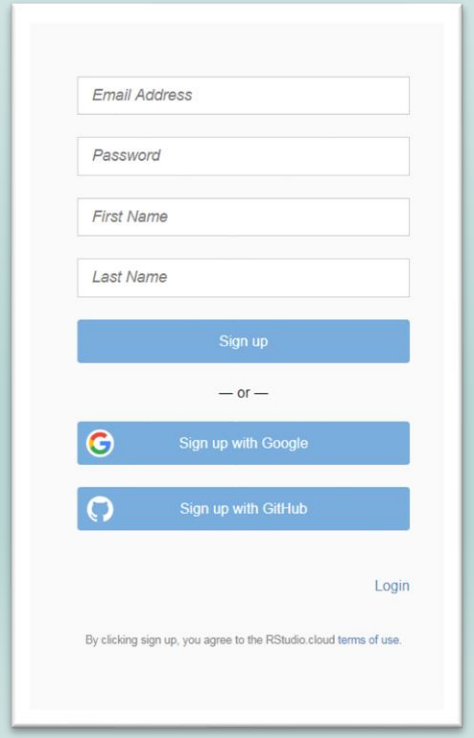
If you can’t access the site click “thumbs down” and we will set you up in a backup configuration shortly.

If You Can't Access api-r.cloud site

1.

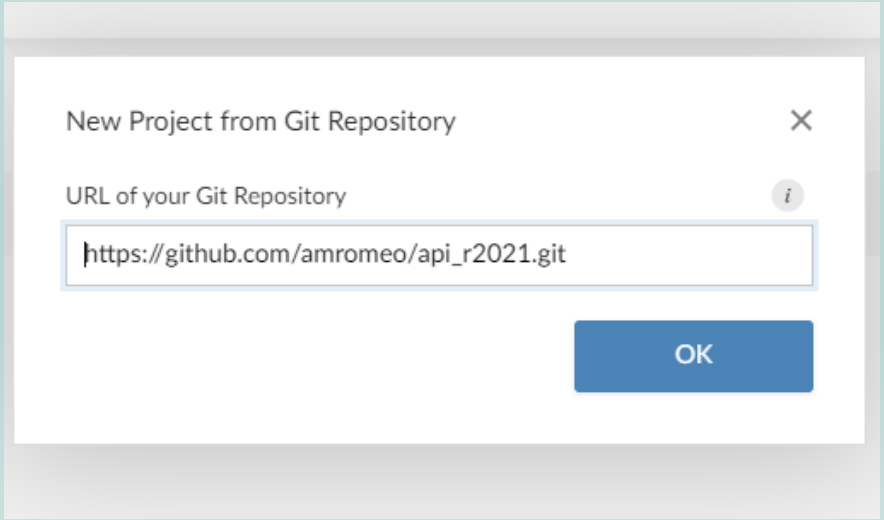
rstudio.cloud

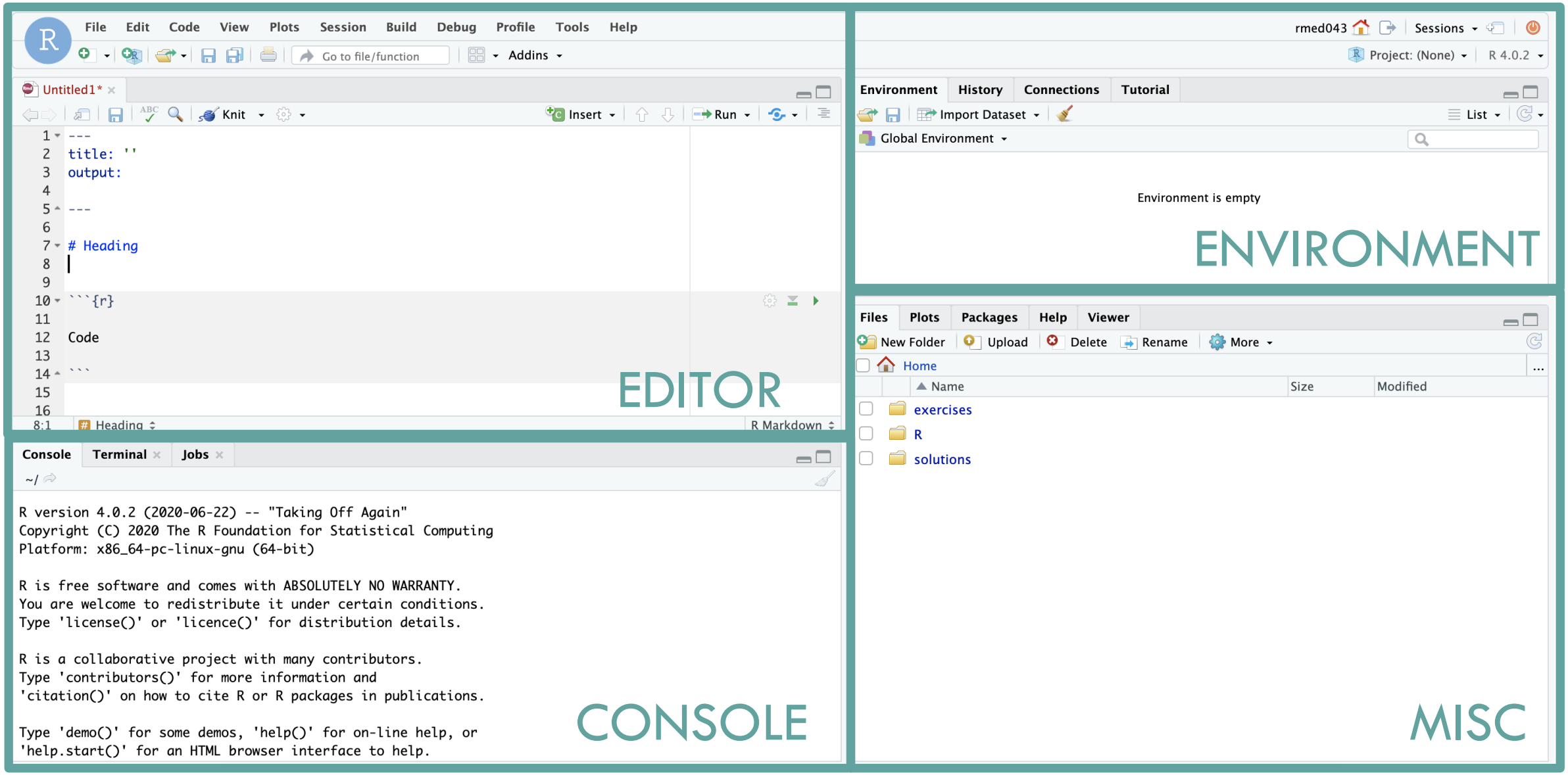
2.



3.

4.




File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Untitled1*
Knit
Insert
Run
1 ---
2 title: ''
3 output:
4
5 ---
6
7 # Heading
8
9
10 ```{r}
11
12 Code
13
14 ```
15
16
8:1
Heading
R Markdown
EDITOR
Console
Terminal
Jobs
~/
R version 4.0.2 (2020-06-22) -- "Taking Off Again"
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.
CONSOLE
rmed043
Sessions
Project: (None)
R 4.0.2
Environment
History
Connections
Tutorial
Import Dataset
List
Global Environment
Environment is empty
ENVIRONMENT
Files
Plots
Packages
Help
Viewer
New Folder
Upload
Delete
Rename
More
Home
Name
Size
Modified
exercises
R
solutions
MISC

Reproducible Data Analysis and R Markdown

The Duke Cancer Scandal

- Chemo sensitivity from microarrays
- Errors first, then cover-up
- Clinical trials based on flawed models
- Papers retracted, lawsuits settled

Duke	MD Anderson
"18881_at"	"18882_g_at"
"31321_at"	"31322_at"
"31725_s_at"	"31726_at"
"32307_r_at"	"32308_r_at"
. . .	

Off-by-one indexing error

“Common problems are simple…

Off-by-one **indexing error**

Sensitive / resistant **label reversal**

Confounding in experimental design

Inclusion of data from **non-reported sources**

Wrong figure shown

… and simple problems are common.”

Point-and-click is not reproducible

Computer code can precisely document each step of the analysis

Why YOU should analyze your data reproducibly

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

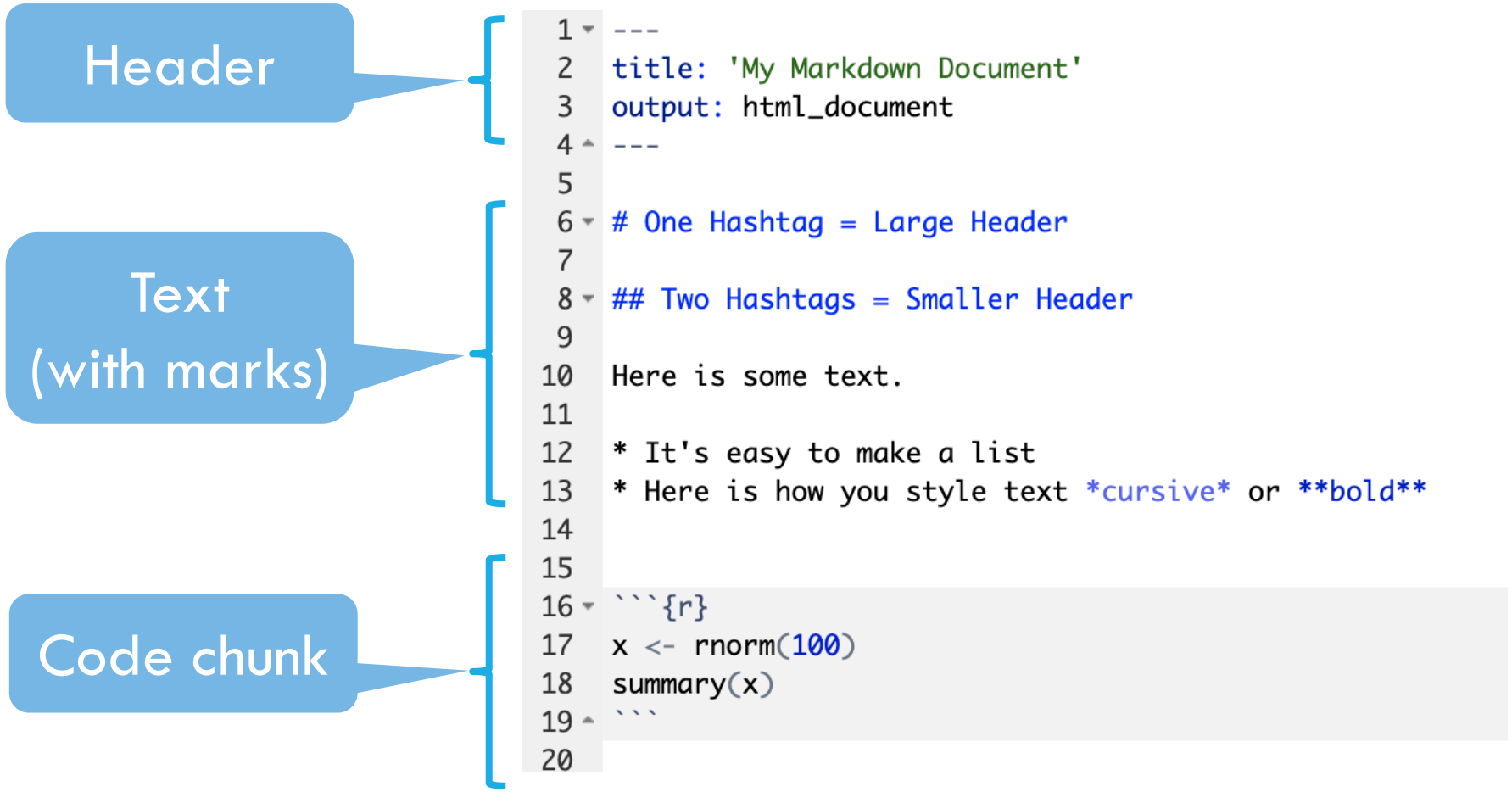
"Why do the data in Table 1 not seem to agree with Figure 2?"

"Why did I decide to omit these six samples from my analysis?"

YOUR CLOSEST COLLABORATOR IS **YOU** FROM 6 MONTHS AGO


rmarkdown
www.rstudio.com

Anatomy of an R Markdown Document



```
---
title: 'My Markdown Document'
output: html_document
---


# One Hashtag = Large Header

## Two Hashtags = Smaller Header

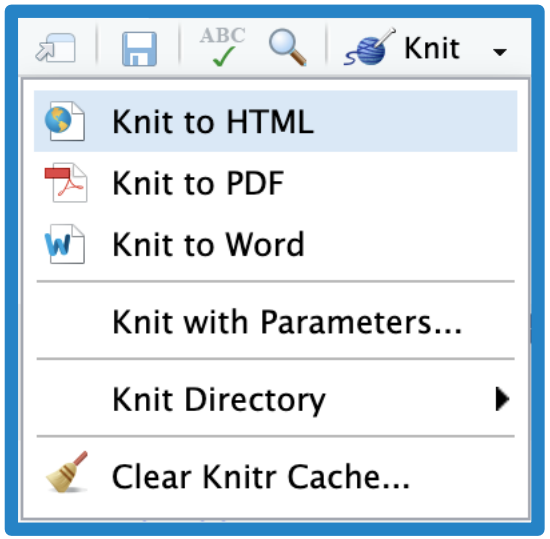
Here is some text.

* It's easy to make a list
* Here is how you style text *cursive* or **bold**


```{r}
x <- rnorm(100)
summary(x)
```


Including Plots

```{r, echo=FALSE}
hist(x)
```
```



My Markdown Document

One Hashtag = Large Header

Two Hashtags = Smaller Header

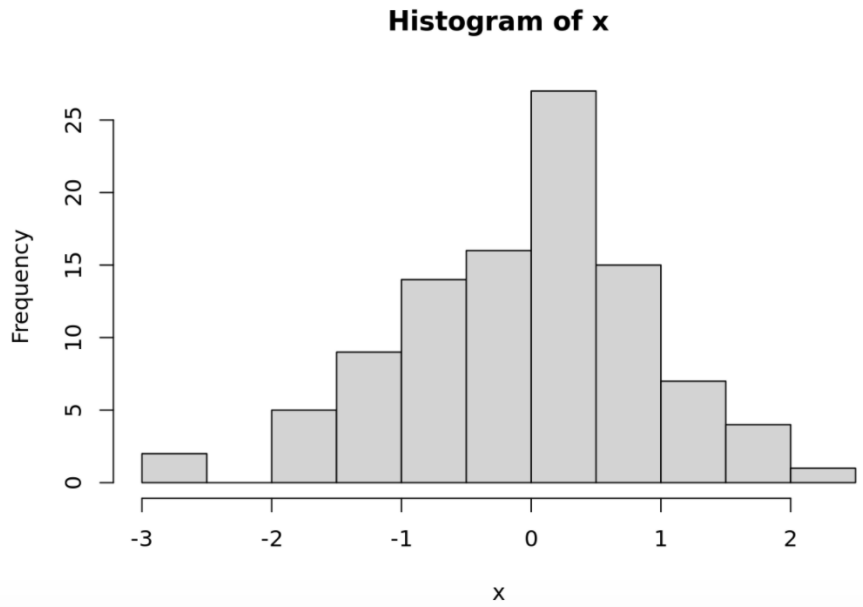
Here is some text.

- It's easy to make a list
- Here is how you style text *cursive* or **bold**

```
x <- rnorm(100)
summary(x)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -2.99204 -0.64726  0.14853 -0.02832  0.58218  2.07410
```

Including Plots



```
---
title: 'My Markdown Document'
output: html_document
---


# One Hashtag = Large Header

## Two Hashtags = Smaller Header

Here is some text.

* It's easy to make a list
* Here is how you style text *cursive* or **bold**


```{r}
x <- rnorm(100)
summary(x)
```


Including Plots

```{r, echo=FALSE}
hist(x)
```
```

My Markdown Document

One Hashtag = Large Header

Two Hashtags = Smaller Header

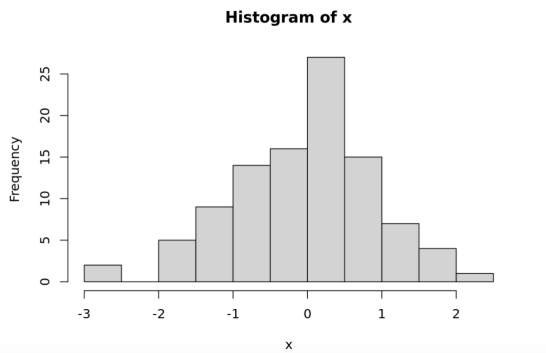
Here is some text.

- It's easy to make a list
- Here is how you style text *cursive* or **bold**

```
x <- rnorm(100)
summary(x)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -2.99204 -0.64726  0.14853 -0.02832  0.58218  2.07410
```

Including Plots



Your Turn #2

Open a sample R Markdown document (File -> New File -> R Markdown).

Review the format of the document: header, text, code chunks

Execute the individual code chunks by selecting the Run Current Chunk arrow.

Knit the document to HTML (Preview or Knit Button -> Knit to HTML). You may be prompted to save your R Markdown first. In this case select a name for your document and click save. Review the knitted document.

Part II

The Basics of Coding

The Basics of Coding: Calculation

- R is a calculator!

press play button to execute code

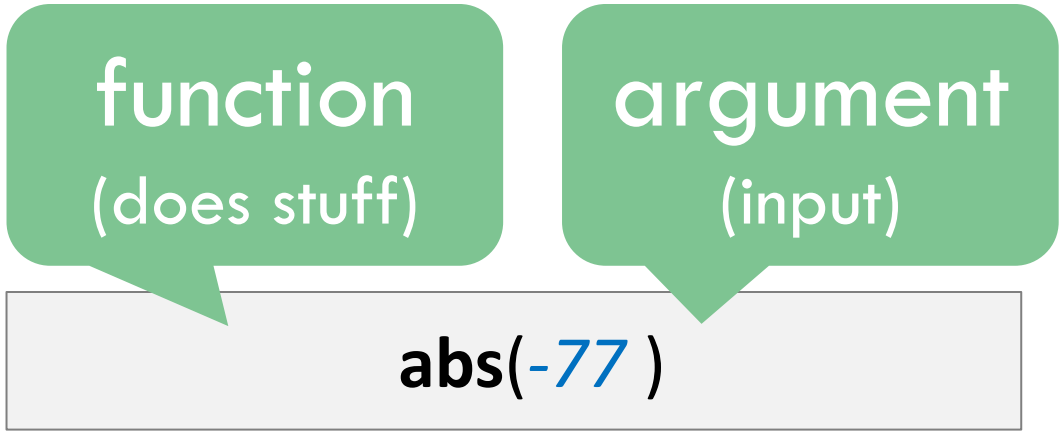
answer returned here

The Basics of Coding: Functions

- Code that extends our reach beyond the basic operators

```
1
2 ```{r}
3
4 abs(-77)
5
6 ```
```

```
[1] 77
```



Putting Functions to Work

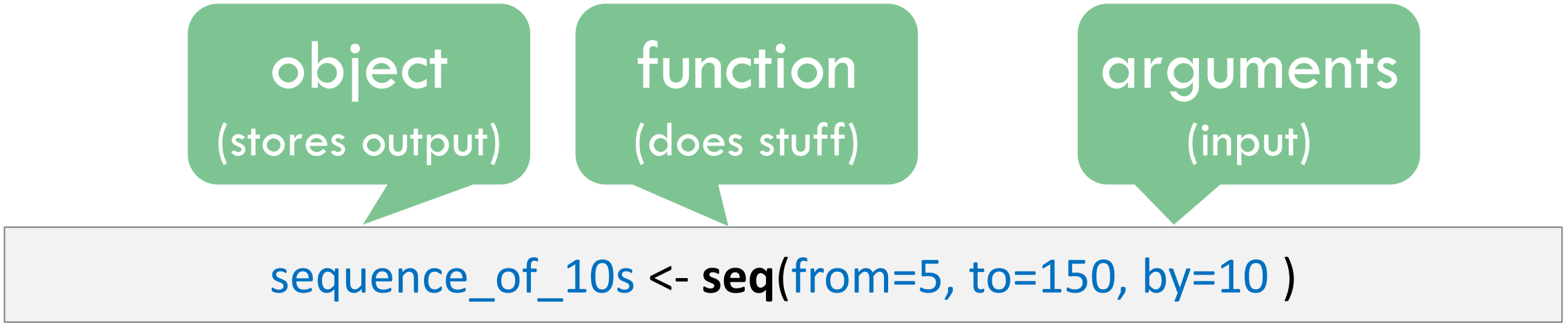
- We can use functions to do more than simple math, we can make things!
- We can create a series of integers (a vector) using the seq() function

````
1
2 ```{r}
3
4 seq(from=5, to=150, by=10)
5
6 ```
````

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

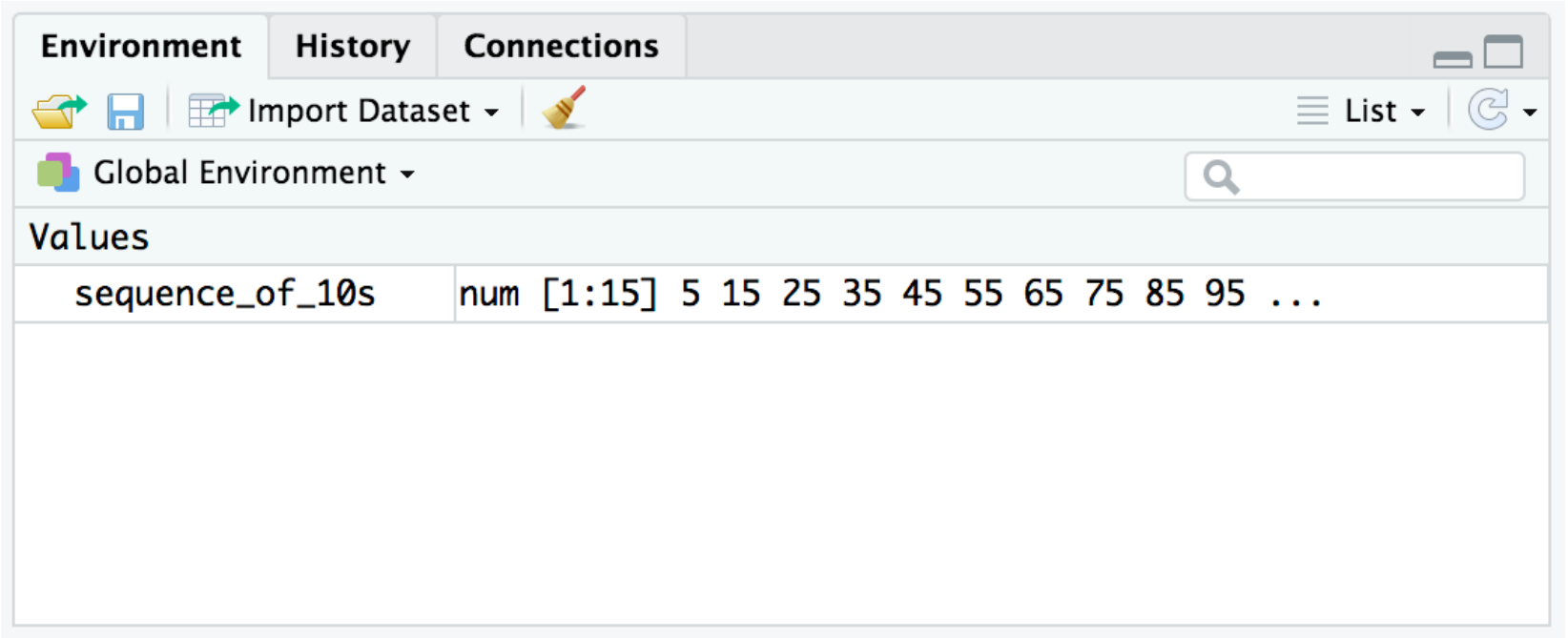
The Basics of Coding: Objects

- Objects are the container for your output



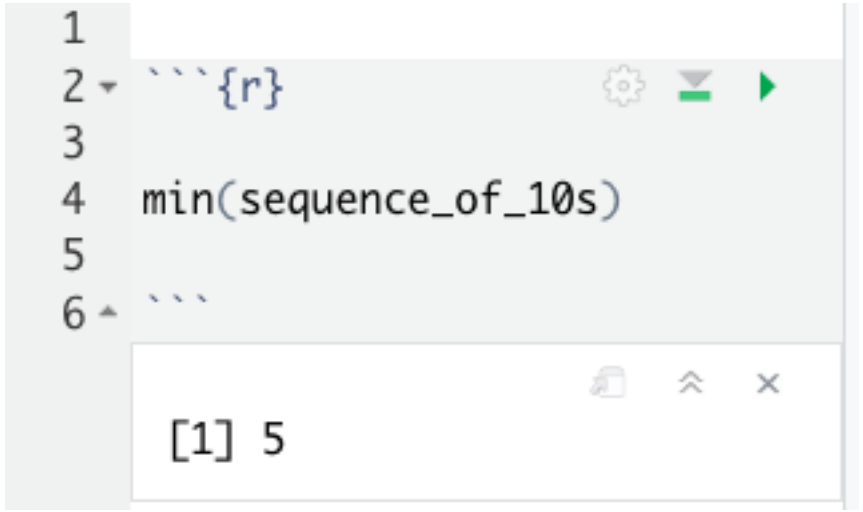
Checking the contents of an object

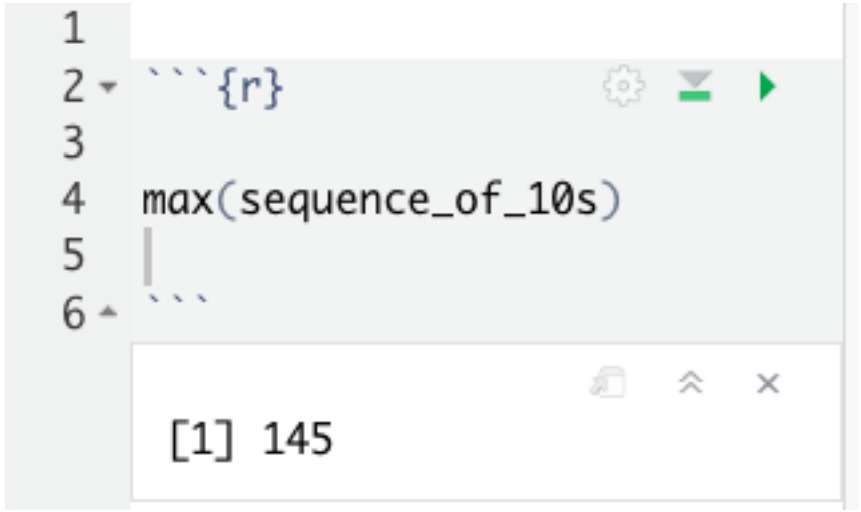
- The environment tab shows us the objects we have created.



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!





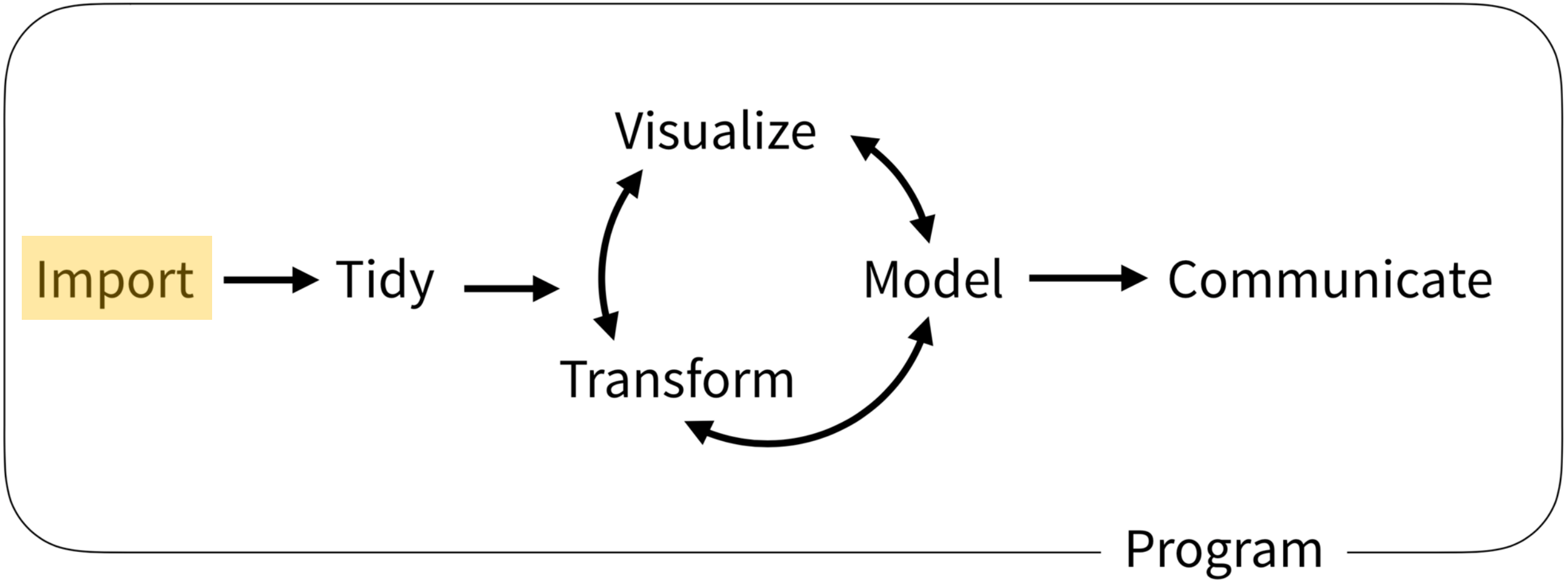
Your Turn #3

I've written some code to create a sequence from 0 to 500 in increments of 25 called sequence_of_25s. Ultimately I want to calculate the median value of this sequence. Unfortunately I've made some mistakes in my code and I am hoping you can help me find them.

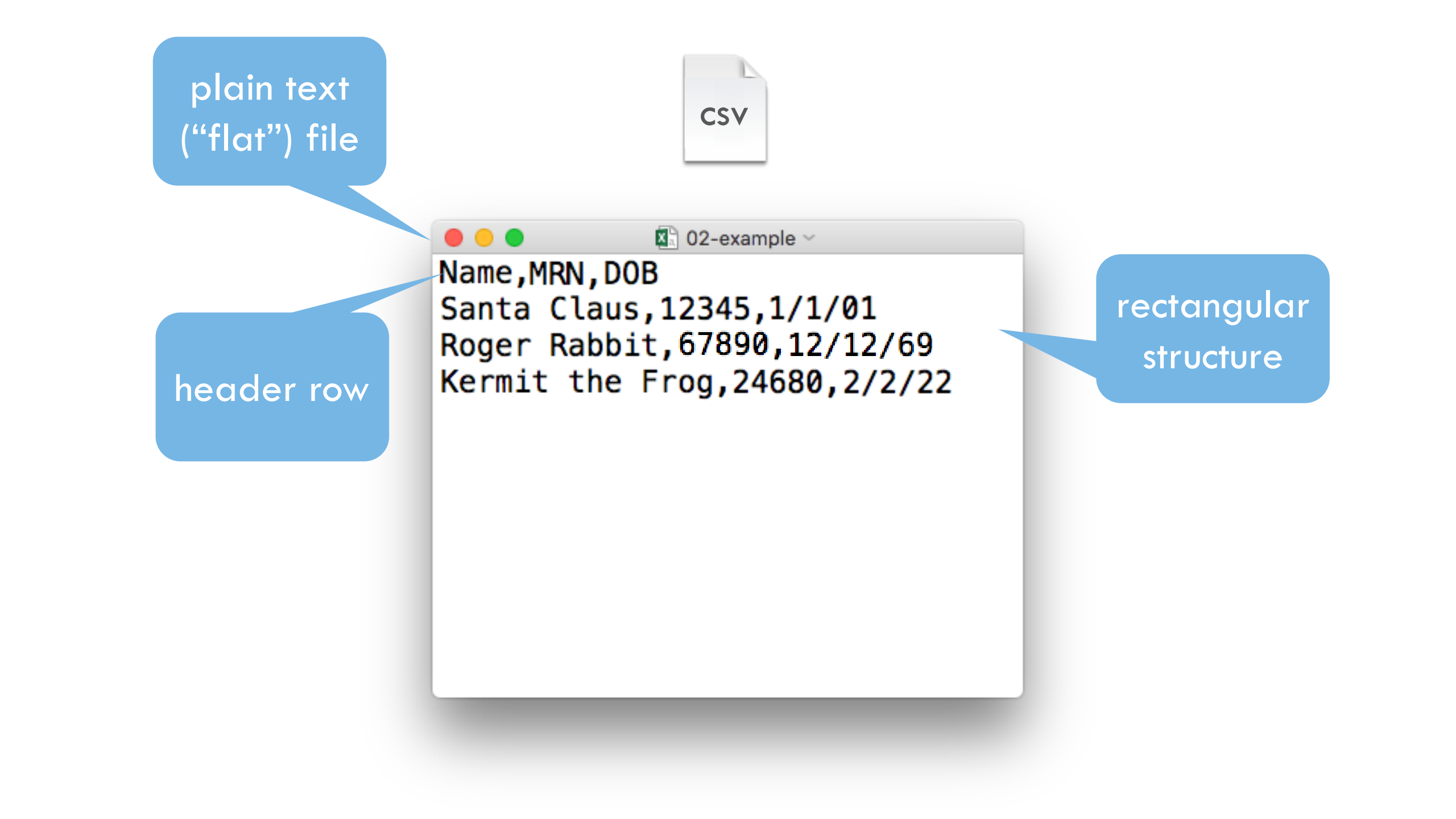
````
1
2  ```{r}
3
4  sequence_of_25s -< seq(from=0 to=50, by=25)
5
6  ```
7
8  ```{r}
9
10 median(sequence of_25s]
11
12 ```
13
````

Importing Data

The Data Analysis Pipeline



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

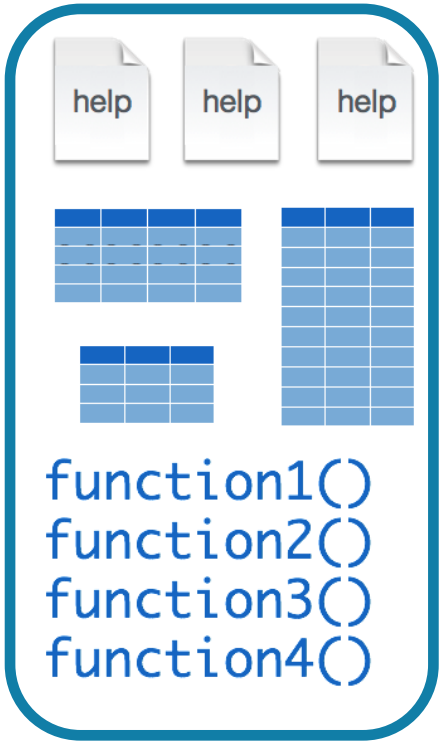

plain text
("flat") file
CSV
02-example
Name,MRN,DOB
Santa Claus,12345,1/1/01
Roger Rabbit,67890,12/12/69
Kermit the Frog,24680,2/2/22
header row
rectangular
structure

Tidyverse: R Packages for Data Science

- A consistent way to organize data
- Human readable, concise, consistent code
- Build pipelines from atomic data analysis steps

Installing and loading R packages

tidyverse




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

Downloads files to computer

1 x per computer

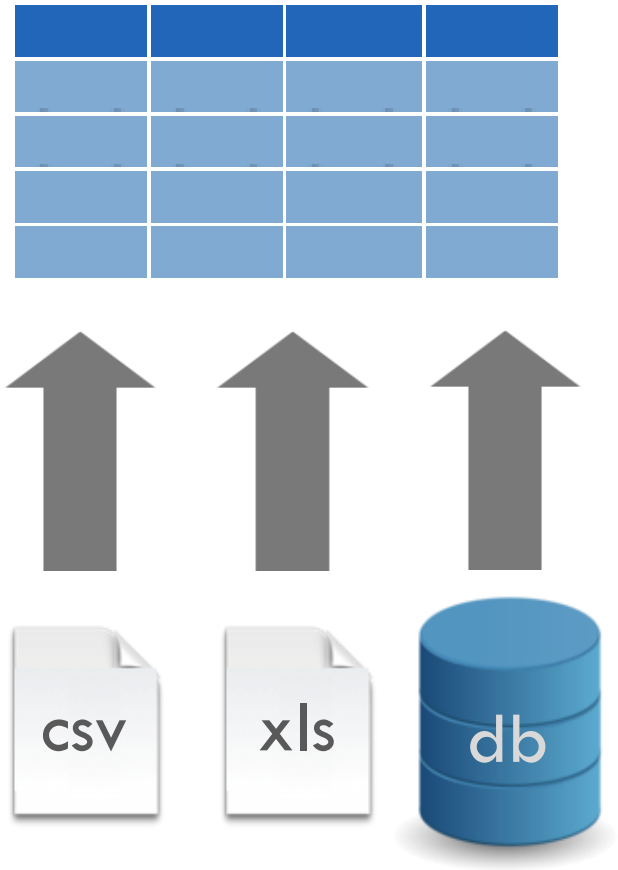
```
library("tidyverse")
```

Loads package

1 x per R Session

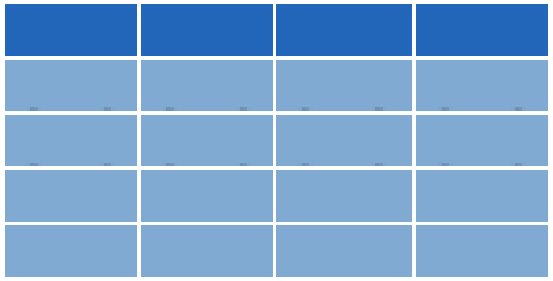
Dataframes: Beyond the Vector

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



read_csv()

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

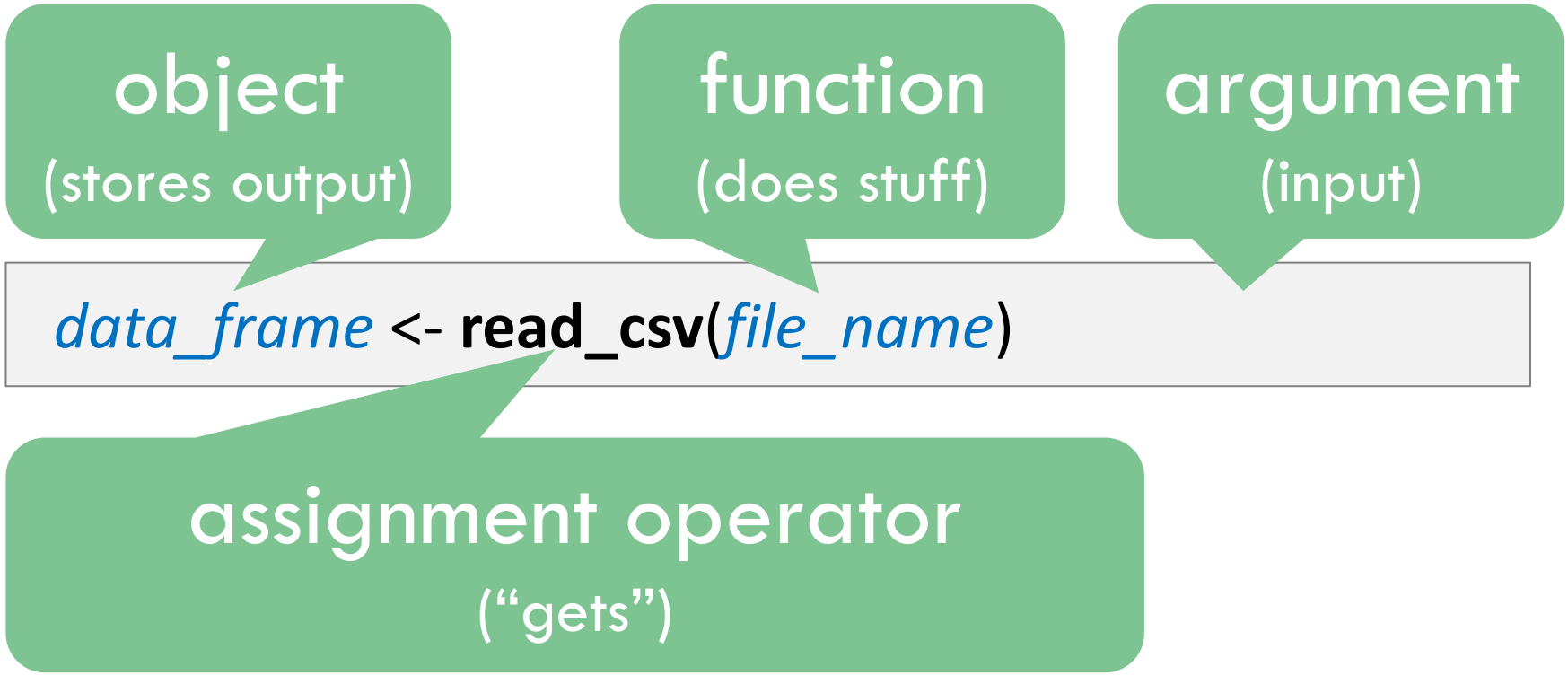


function
(does stuff)

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


function
(does stuff)
argument
(input)
data_frame <- read_csv(file_name)


object
(stores output)
function
(does stuff)
argument
(input)
data_frame <- read_csv(file_name)

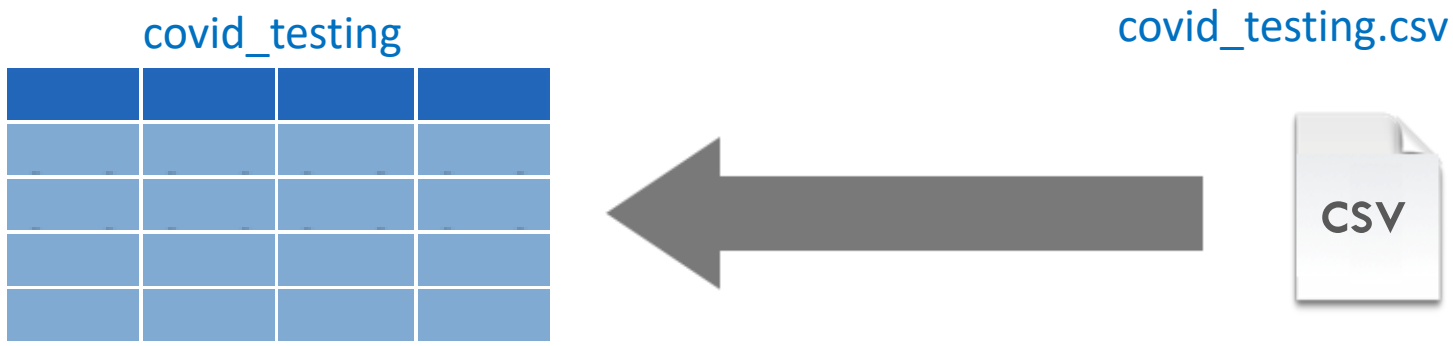

object
(stores output)
function
(does stuff)
argument
(input)
data_frame <- read_csv(file_name)
assignment operator
("gets")

read_csv()

data frame to read data into

name of CSV file

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



Your Turn #4

In the MISC pane, select the folder:

"exercises"

Select the R Markdown file:

"01 - Importing and Exploring Data.Rmd"

In the Editor pane, follow the instructions to complete the exercise.

Recap

Programming Language

IDE

Document Format

Packages extend the functionality of R. They need to be installed once per computer and loaded each session.

Functions do stuff. They accept **Arguments** to define parameters. We can store the output of functions in **Objects** using the assignment operator (<-).

Importing Data is the first step data analysis pipeline. read_csv() is a function from the tidyverse that we can use for importing data.

What else?

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

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

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

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"

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

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

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

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

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)

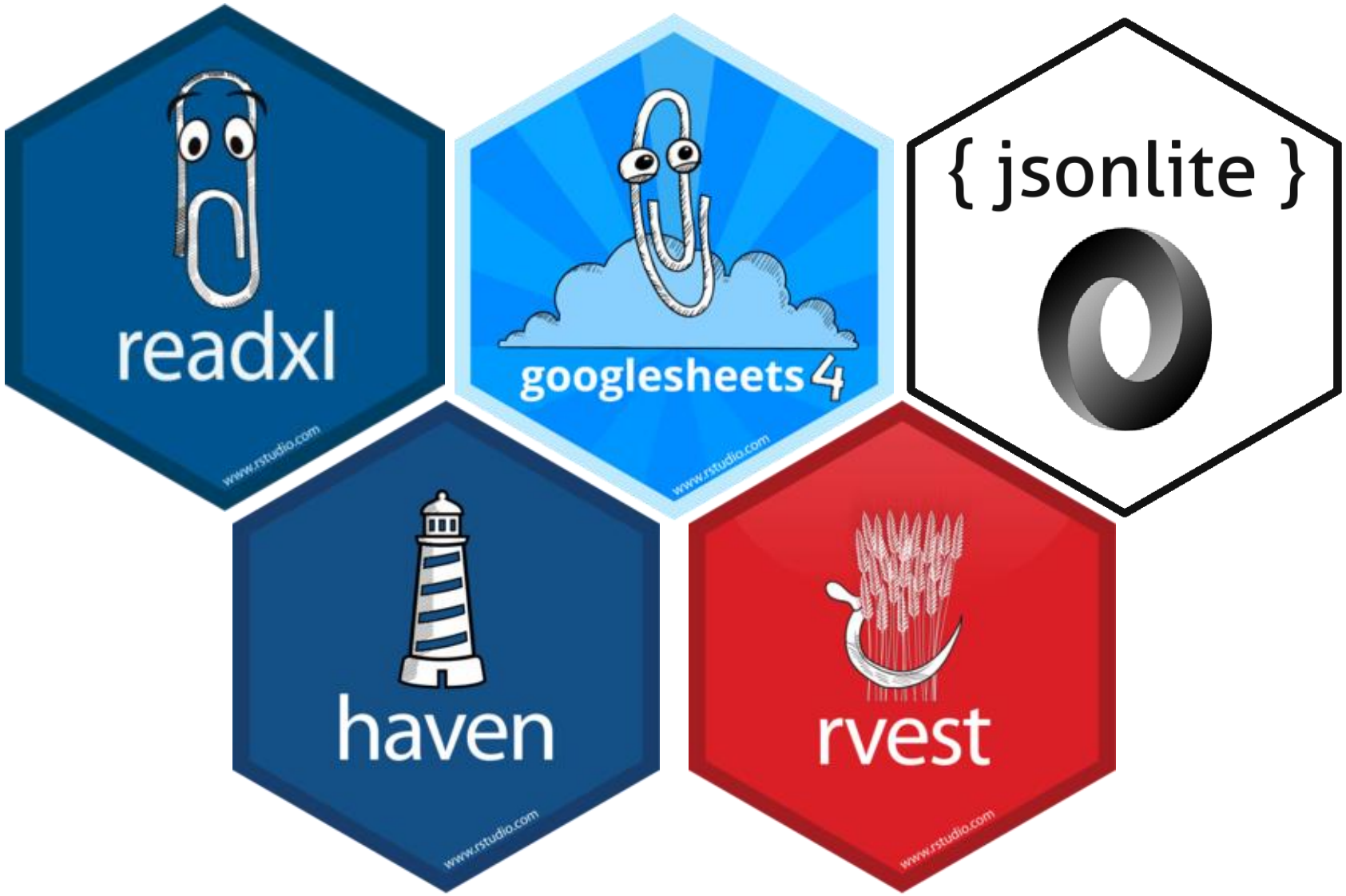
2. Use a col_ function to guide parsing
- **col_guess()** - the default
- **col_character()**
- **col_double()**, **col_euro_double()**
- **col_datetime**(format = "") Also **col_date**(format = ""), **col_time**(format = "")
- **col_factor**(levels, ordered = FALSE)
- **col_integer()**
- **col_logical()**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

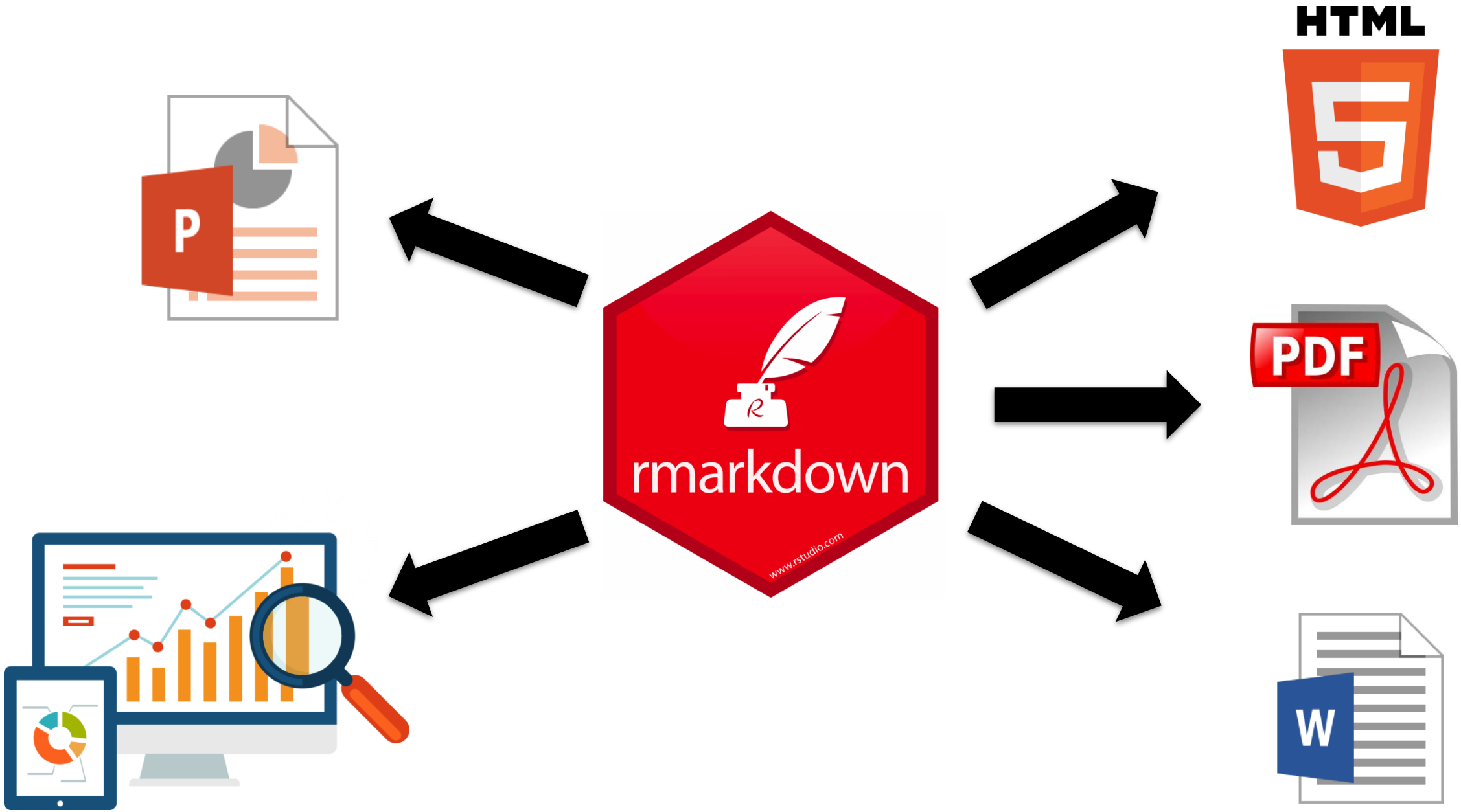
RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more with tidyverse.org • readr 1.1.0 • tibble 1.2.12 • tidyr 0.6.0 • Updated: 2017-01


readxl
www.rstudio.com
googlesheets4
www.rstudio.com
{ jsonlite }
haven
www.rstudio.com
rvest
www.rstudio.com

Databases

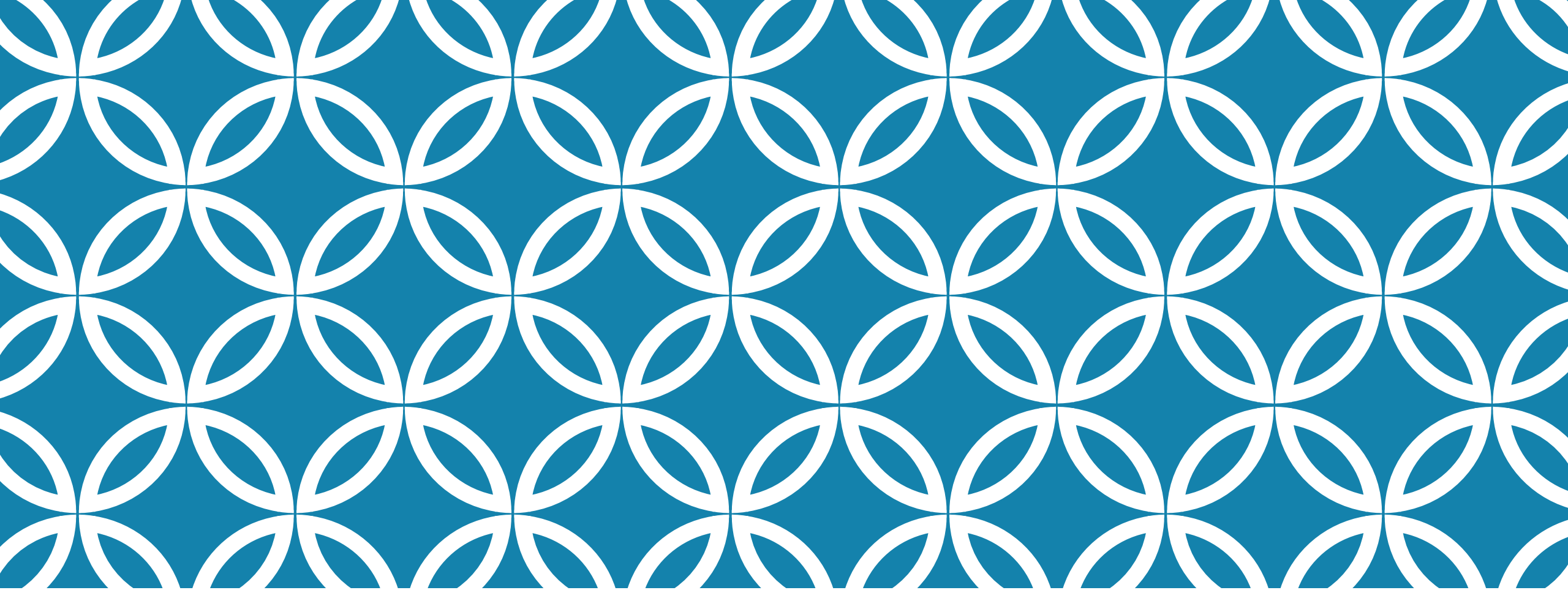
- Microsoft SQL Serve
- MonetDB
- MongoDB
- MySQL
- Netezza
- Oracle
- Amazon Redshift
- Apache Hive
- Apache Impala
- Athena
- Cassandra
- Google BigQuery
- Other Databases
- PostgreSQL
- SQLite
- Salesforce
- Teradata

https://db.rstudio.com/databases/


HTML
rmarkdown
www.rstudio.com
PDF

R Interface to Python

```{python}
import pandas
covid_testing.info()
```



Data Visualization

Session 3

Stephan Kadauke

Session	Instructor
Instructor Introductions, Introduction to technology	Amrom Obstfeld
Introduction to R and RStudio	Joe Rudolf
Reproducible Reporting	Joe Rudolf
Data Visualization	Stephan Kadauke
Data Transformation	Amrom Obstfeld
Statistical Analysis	Dan Herman
Advanced Reporting	Patrick Mathias

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

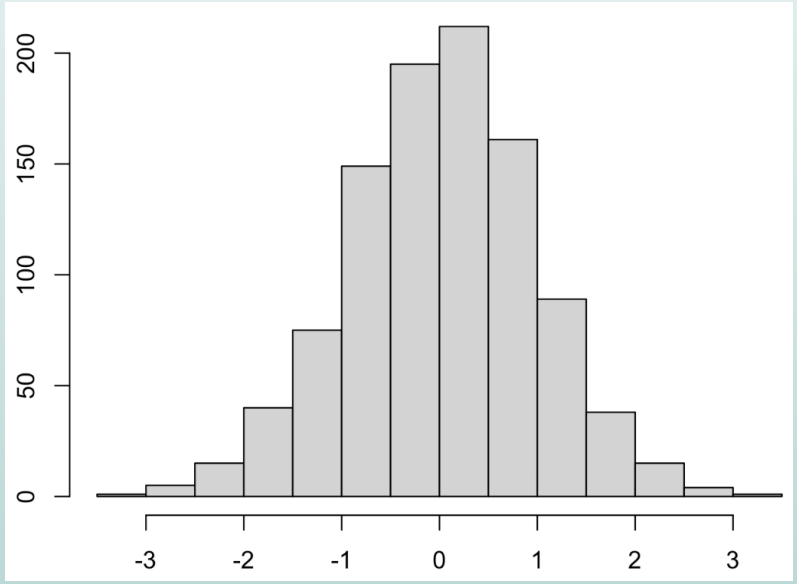
Your Turn 1

Consider the `covid_testing` data frame.

What do you think 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!

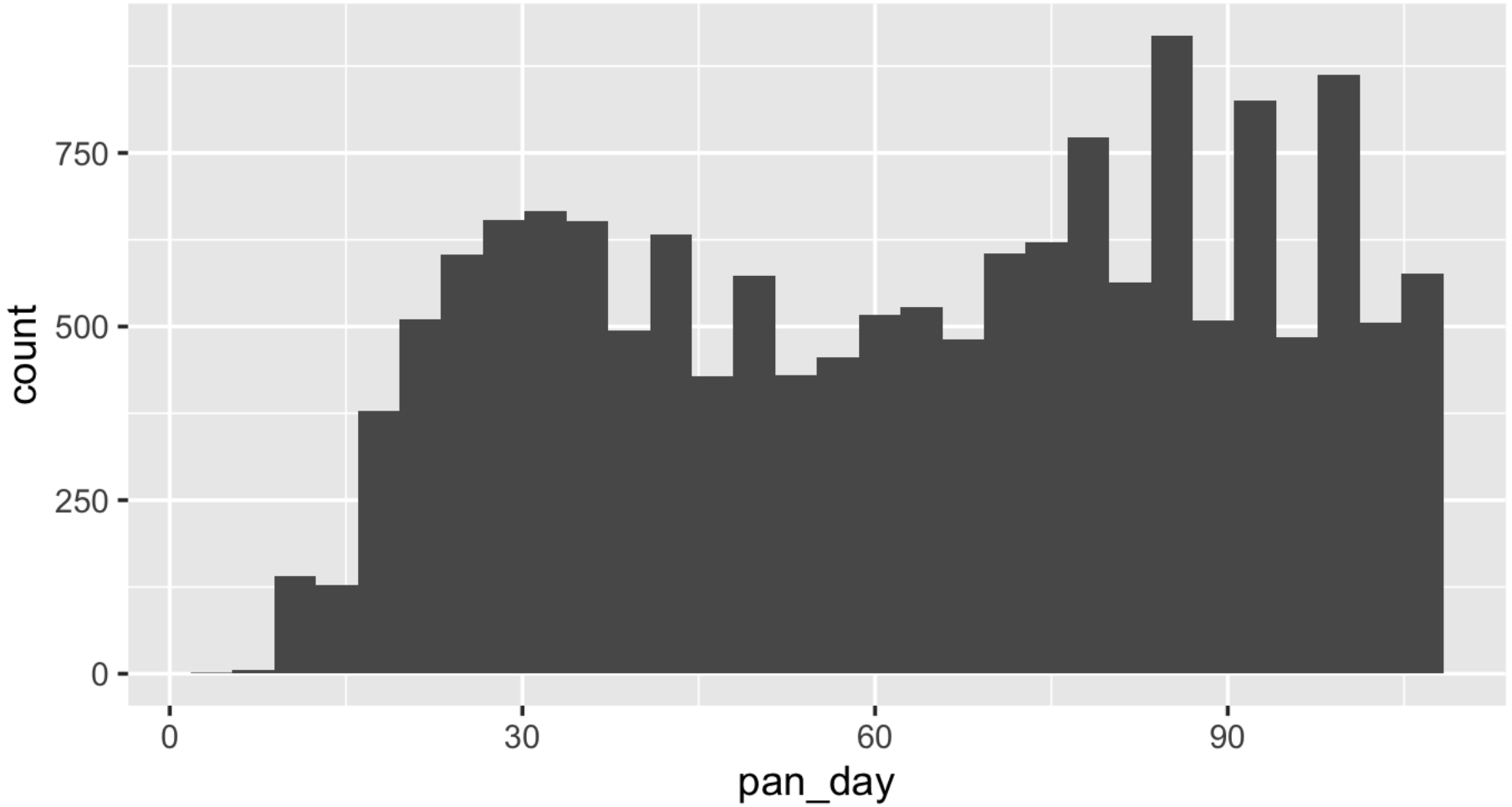
```
ggplot(data = covid_testing) +
  geom_histogram(mapping = aes(x = pan_day))
```

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

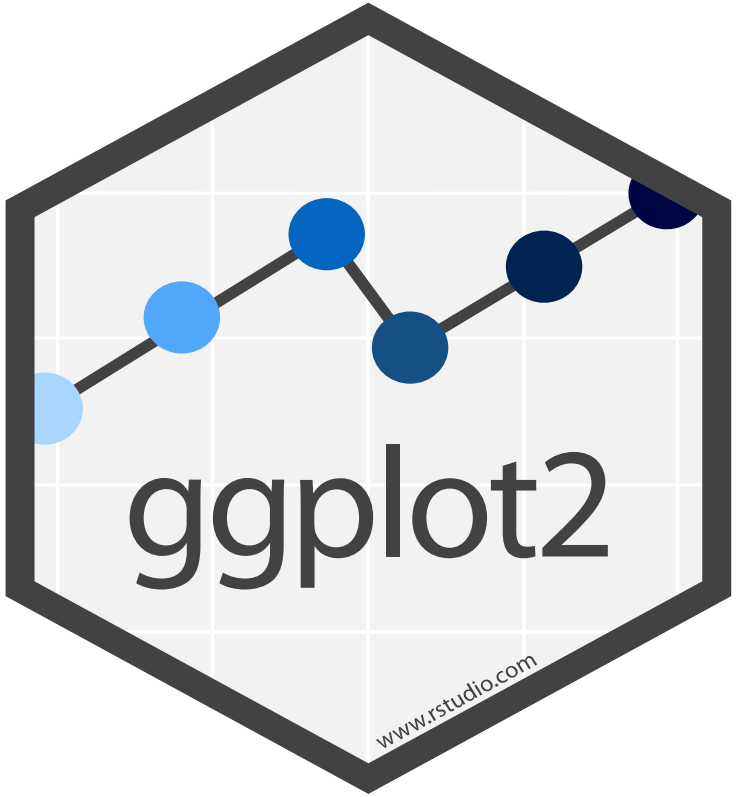
Often times, things that look like an error in R are actually just a message.

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.

```
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. Pick a "tidy" **data** frame

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

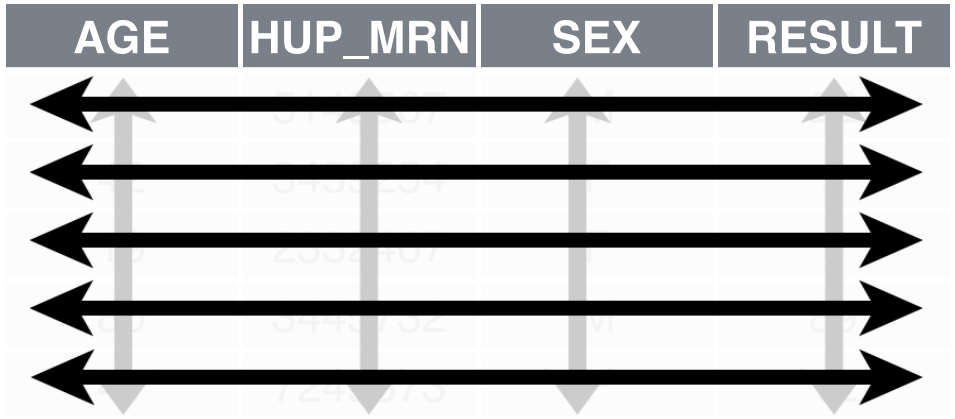
2. Pick a "**geom**" function

3. Write aesthetic **mappings**

1. Pick 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**

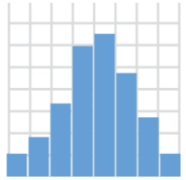
Wickham H. **Tidy Data.** *J Stat Soft* 2014.

1. Pick a “tidy” **data** frame

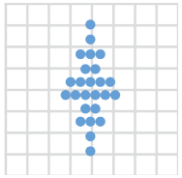
```
ggplot(data = data_frame) +
  geom_function(mapping = aes(mappings))
```

2. Pick a “**geom**” function

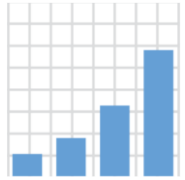
2. Pick a “Geom” Function



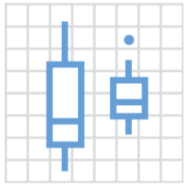
```
geom_histogram()
```



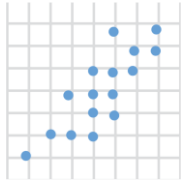
```
geom_dotplot()
```



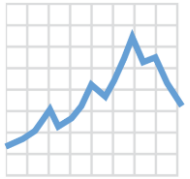
```
geom_bar()
```



```
geom_boxplot()
```



```
geom_point()
```



```
geom_line()
```

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

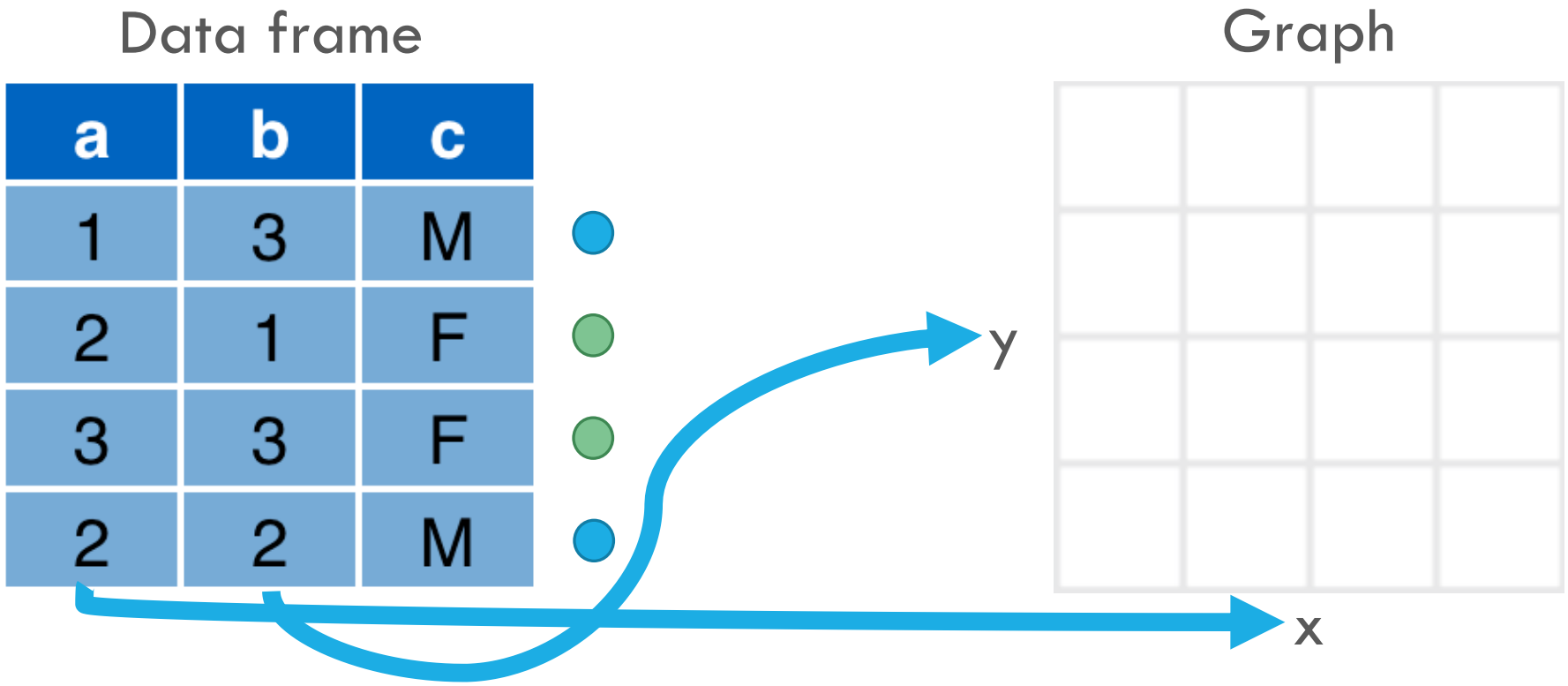
```
ggplot(data = data_frame) +
  geom_function(mapping = aes(mappings))
```

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

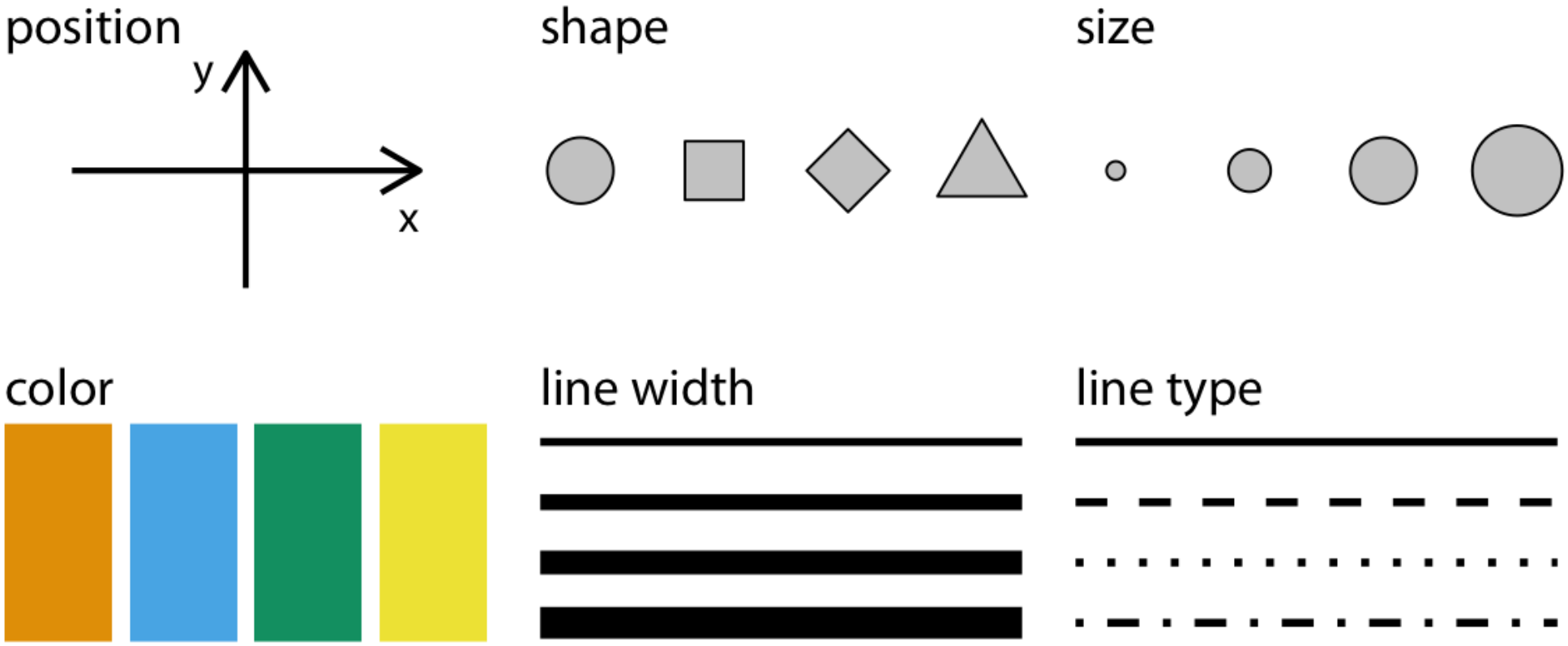
3. Write aesthetic **mappings**

3. Write Aesthetic Mappings

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



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 4

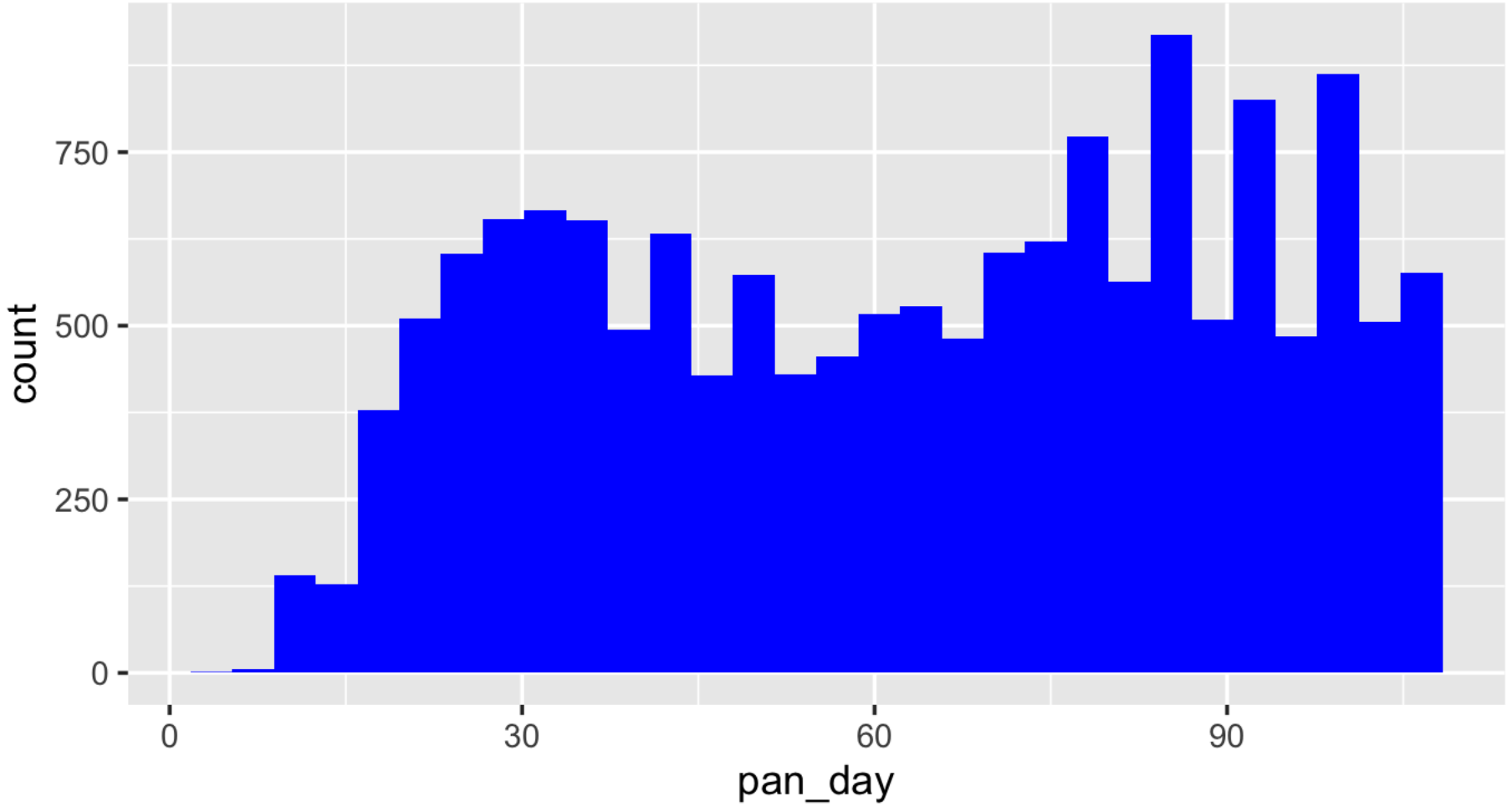
Open `03 - Visualize.Rmd`. Work through the exercises of the section titled "Your Turn 4".

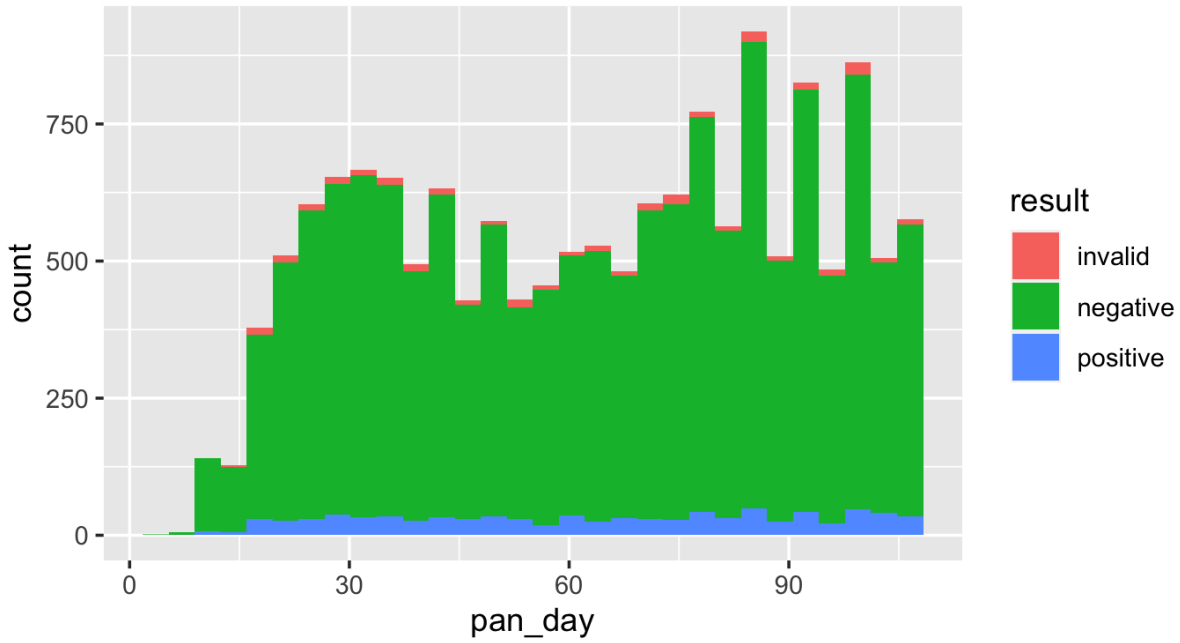
Stop when it says "Stop Here".

Click "yes" when you're done!

Setting vs Mapping Aesthetics

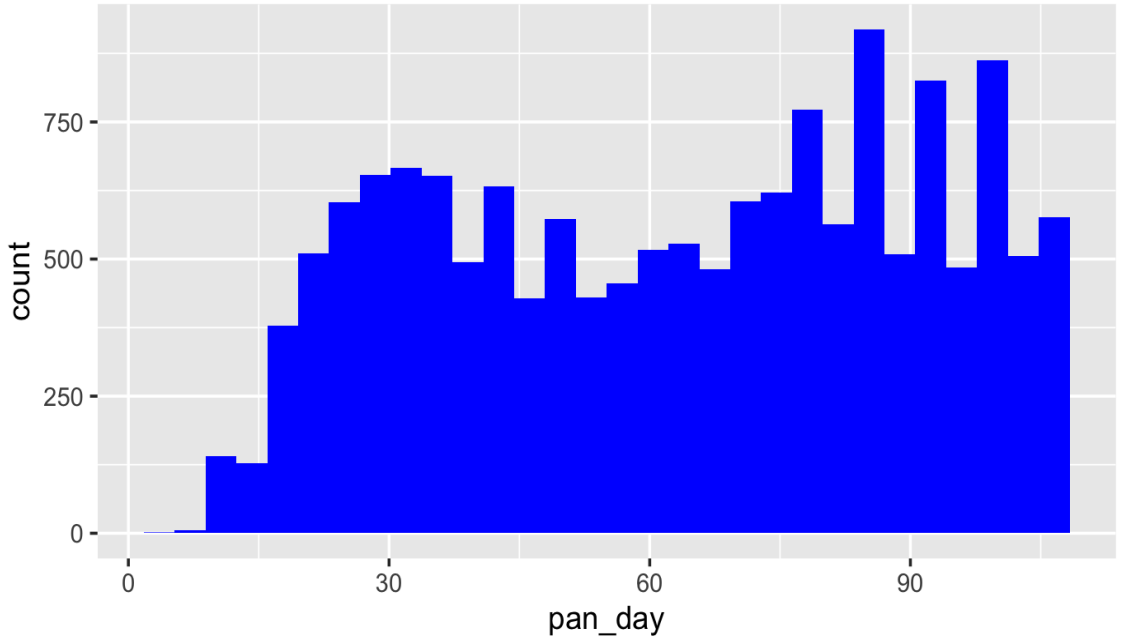
How would you make this plot?





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

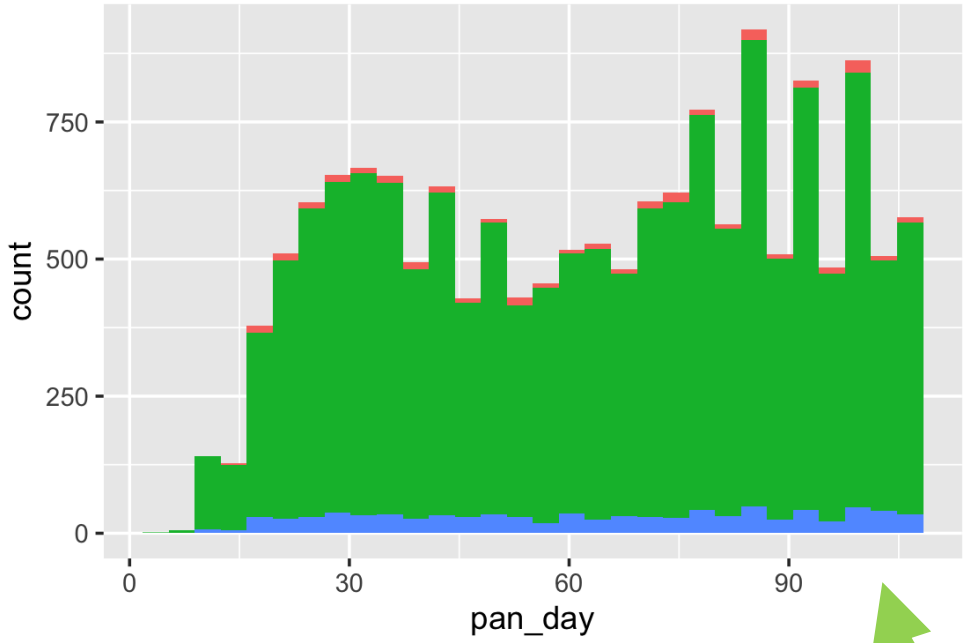
```
ggplot(data = covid_testing) +
  geom_histogram(mapping = aes(x = pan_day, fill = result))
```

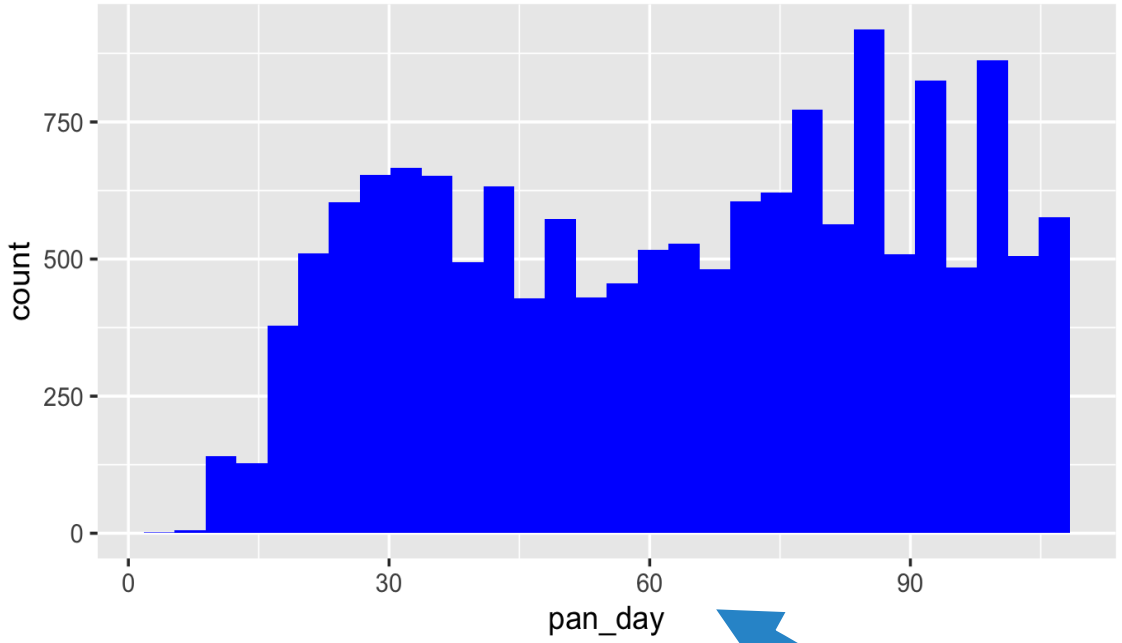


Outside of `aes()`:
set an aesthetic to a **value**

color name in "quotes"

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



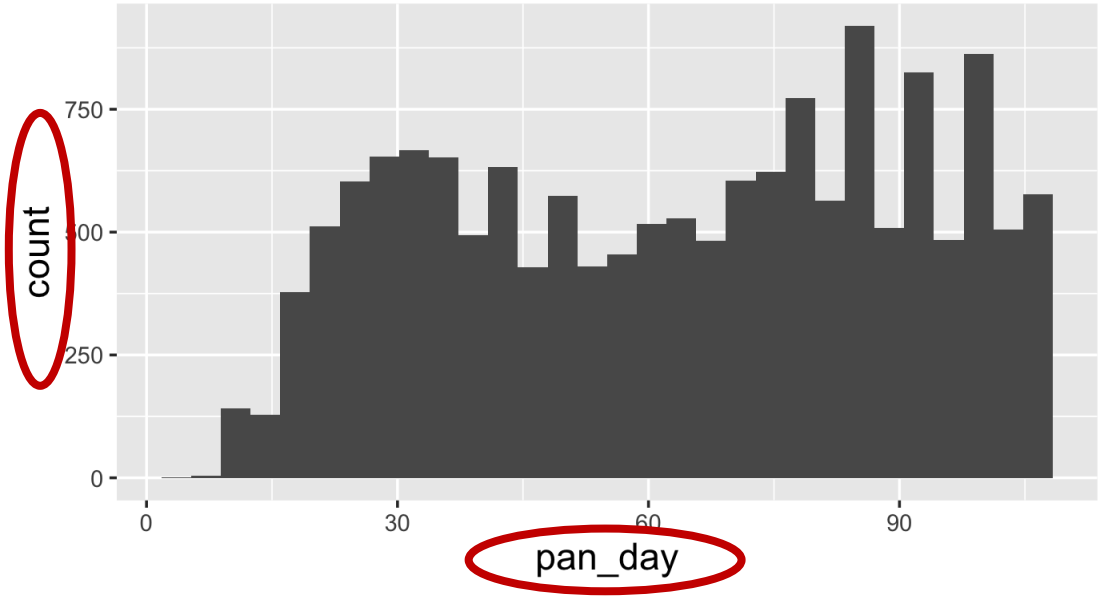


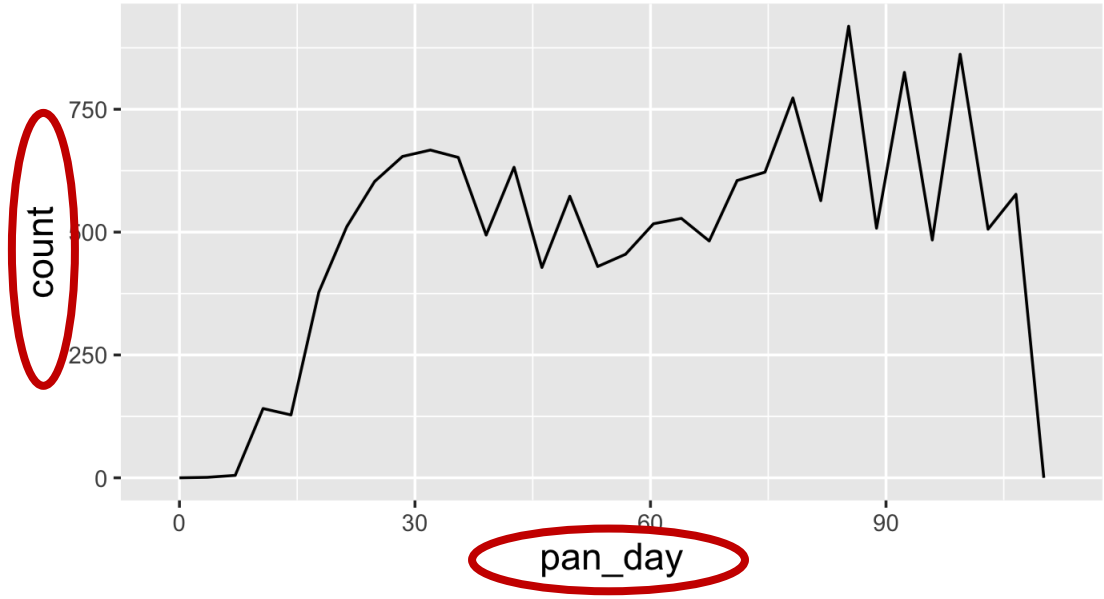

```
ggplot(data = covid_testing) +
  geom_histogram(mapping = aes(x = pan_day, fill = result))
```

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

Geom Functions

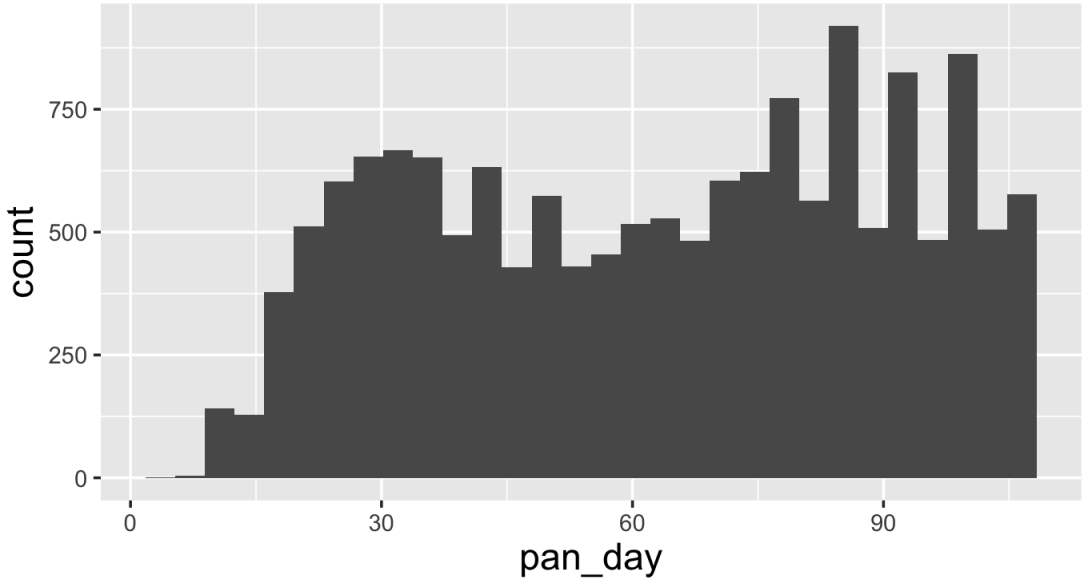
How are these plots similar?

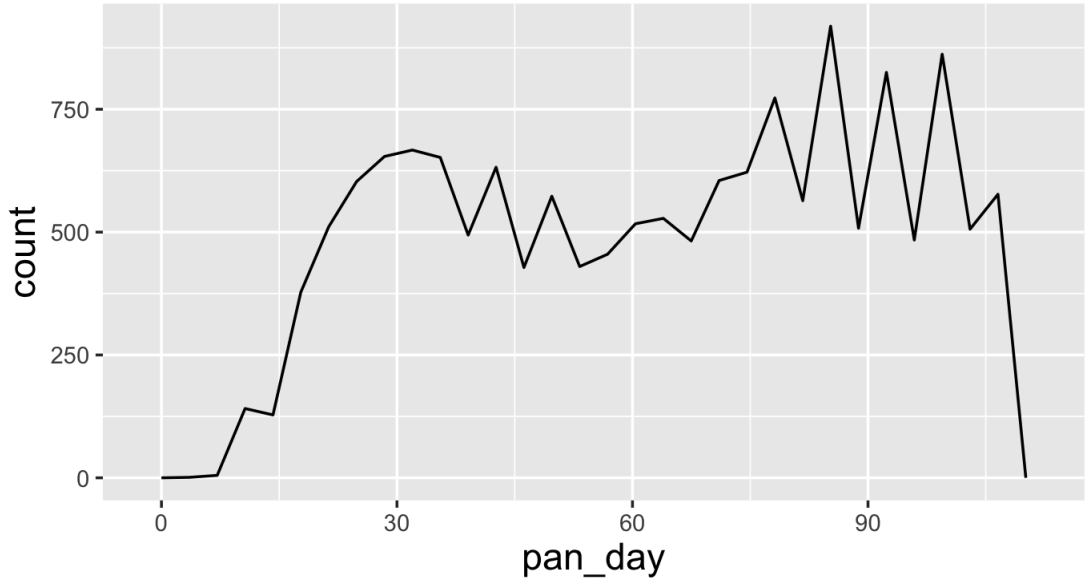




Same: x axis, y axis, data

How are these plots different?





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

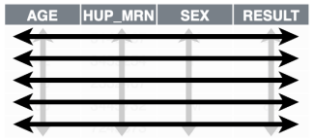
Your Turn 5

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

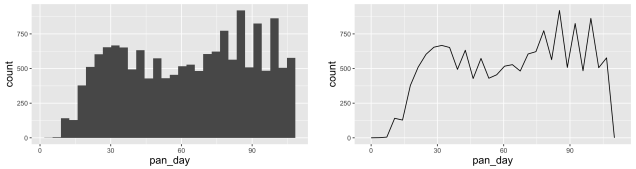
Click "yes" when you're done!

Recap

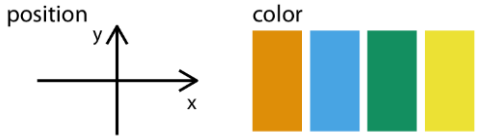
ggplot2 is a package that provides a **grammar of graphics.** You can create **any type of plot** using a simple template to which you provide:



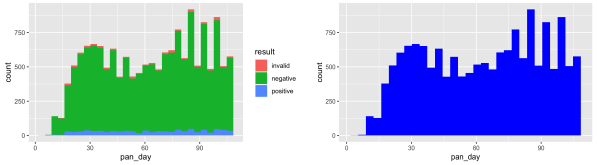
1. A **tidy data frame,** in which each **variable** is in its own **column**, each **observation** is in its own **row**, each **value** is in its own **cell**;



2. A **geom function,** which tells R what kind of plot to make; and



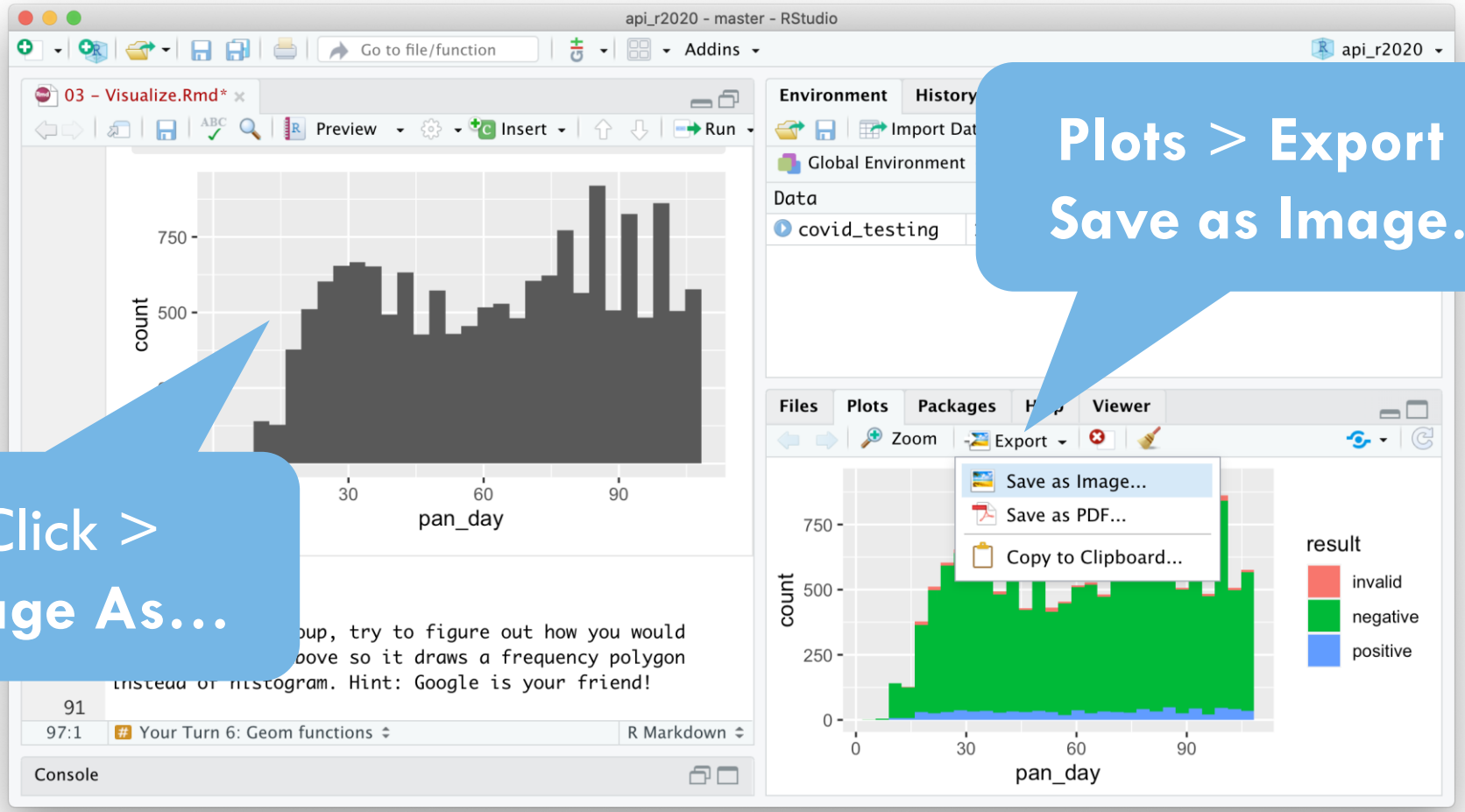
3. **Aesthetic mappings,** which tell R how to represent data as graphical markings on the plot.



Aesthetics can be **mapped** to a variable or **set** to a constant value.

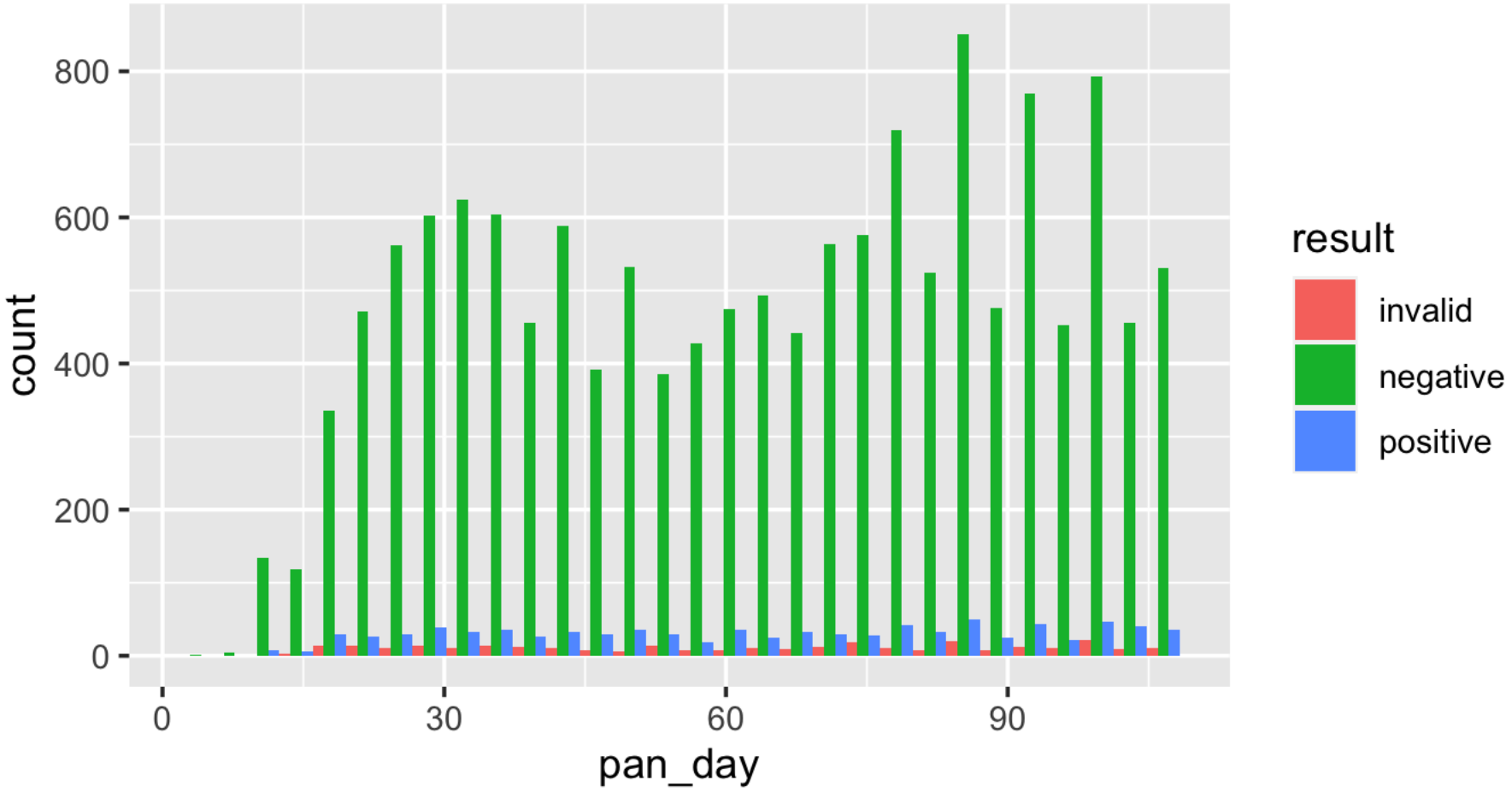
What Else?

Manually saving plots



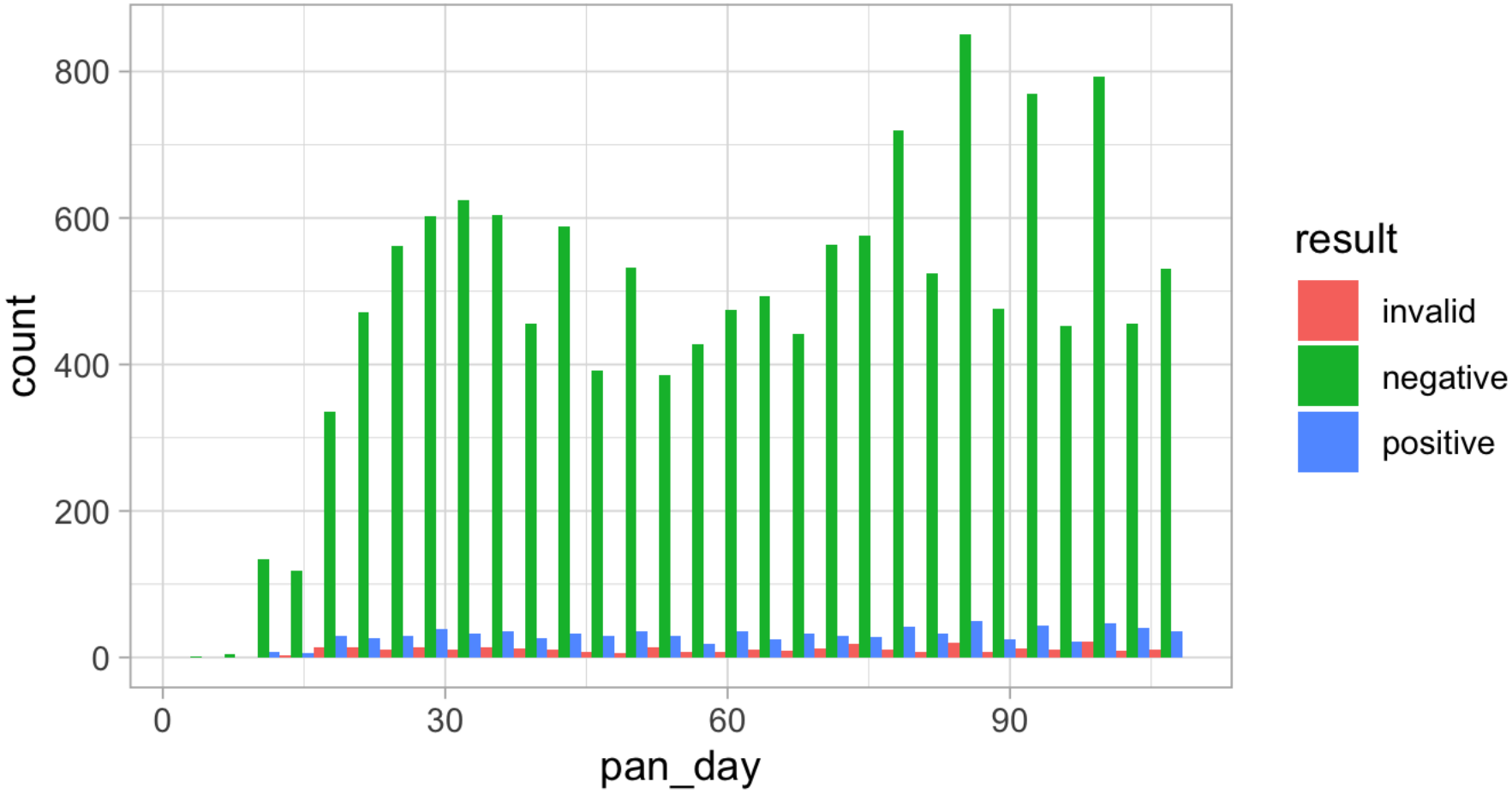
Position adjustments

How overlapping objects are arranged



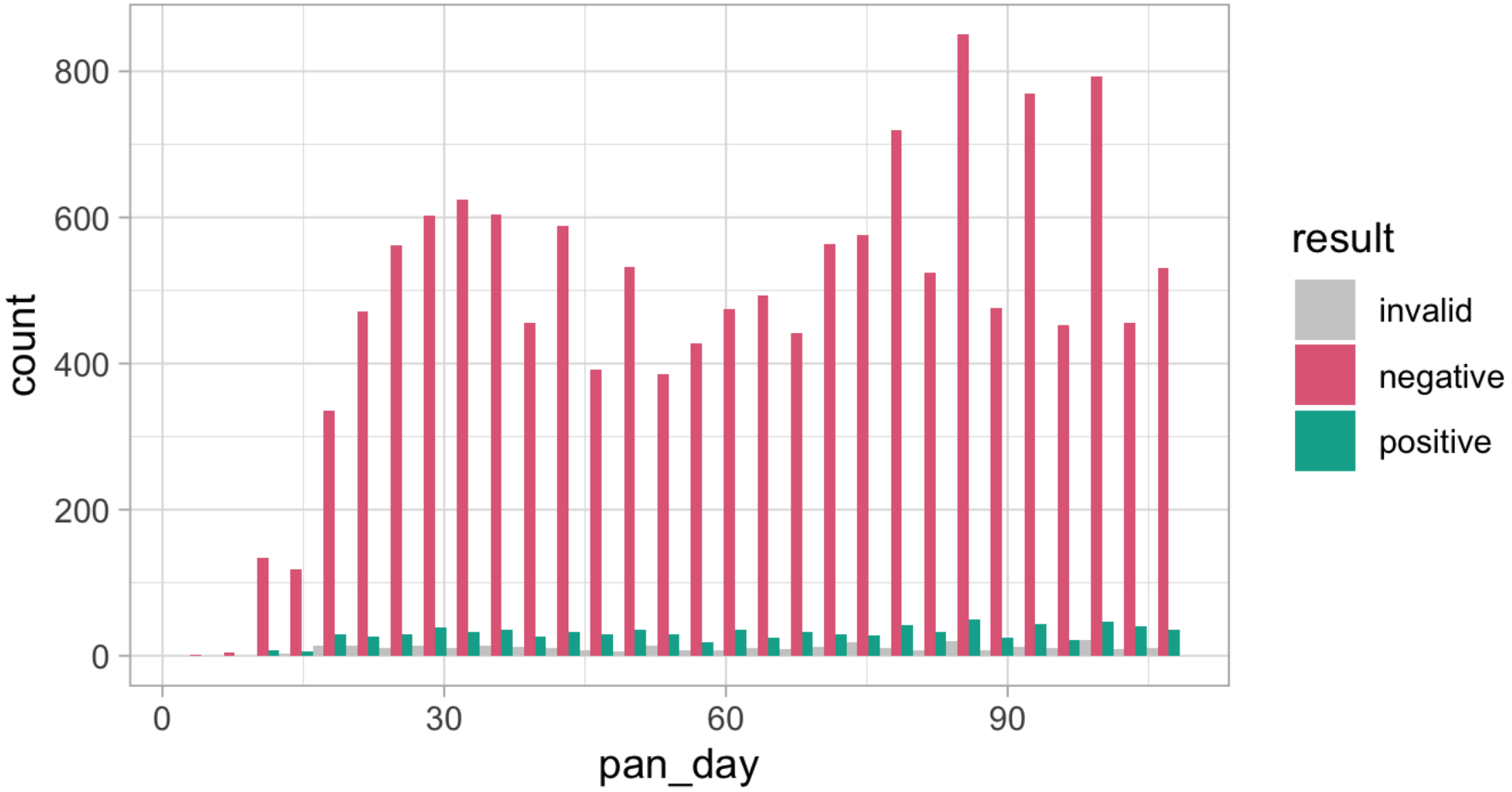
Themes

Visual appearance of non-data elements



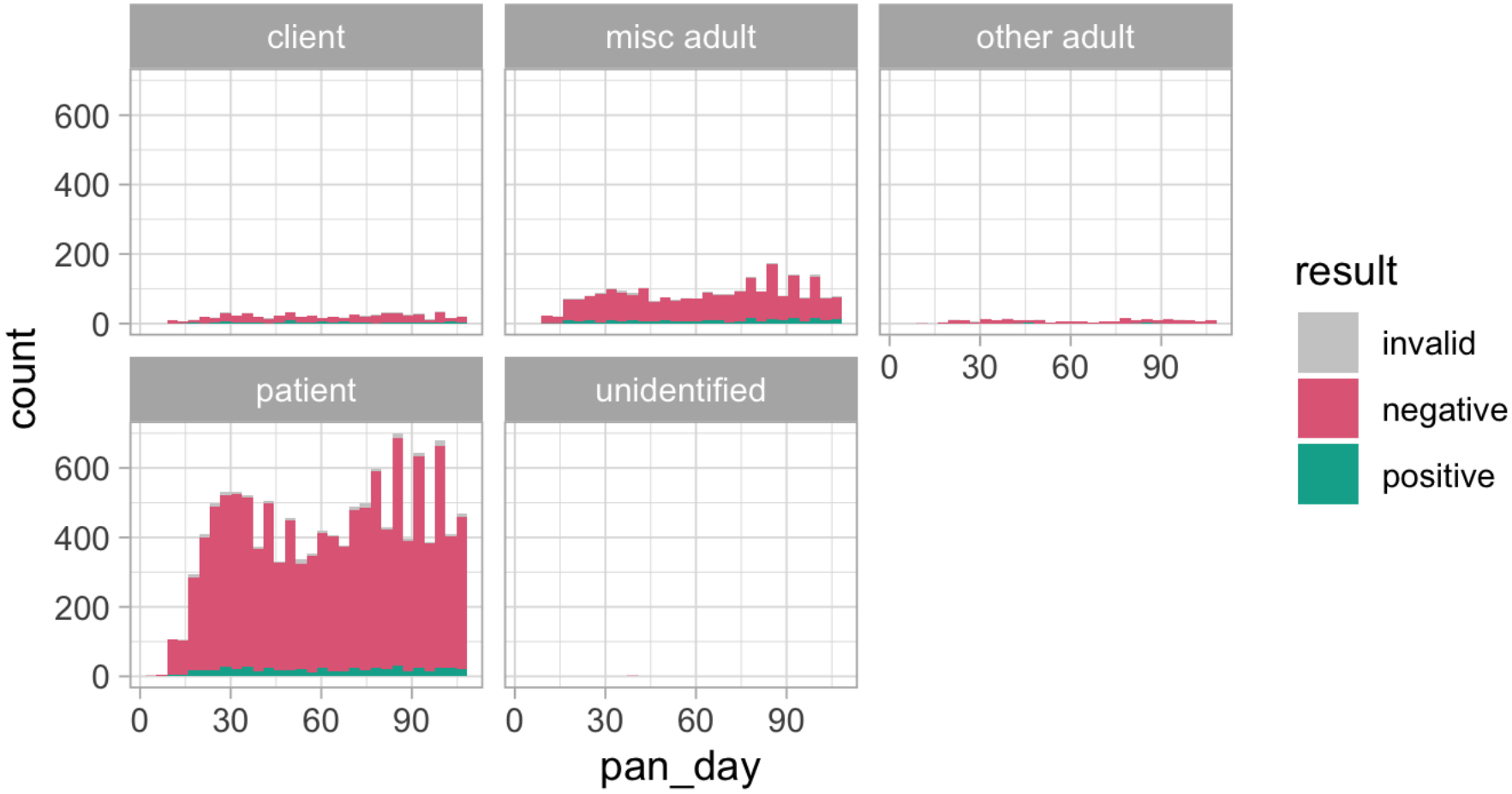
Scales

Customize color scales and other mappings

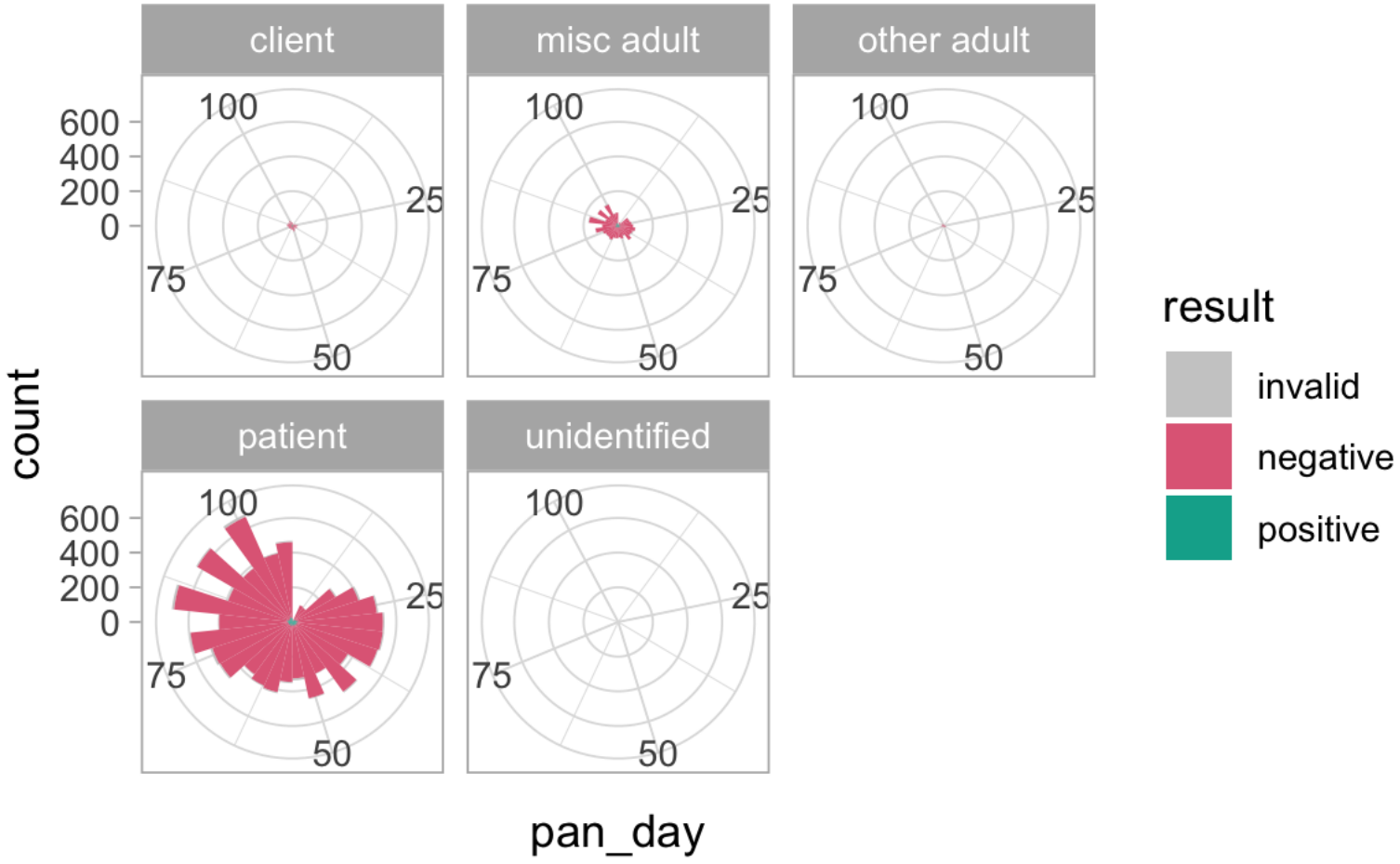


Facets

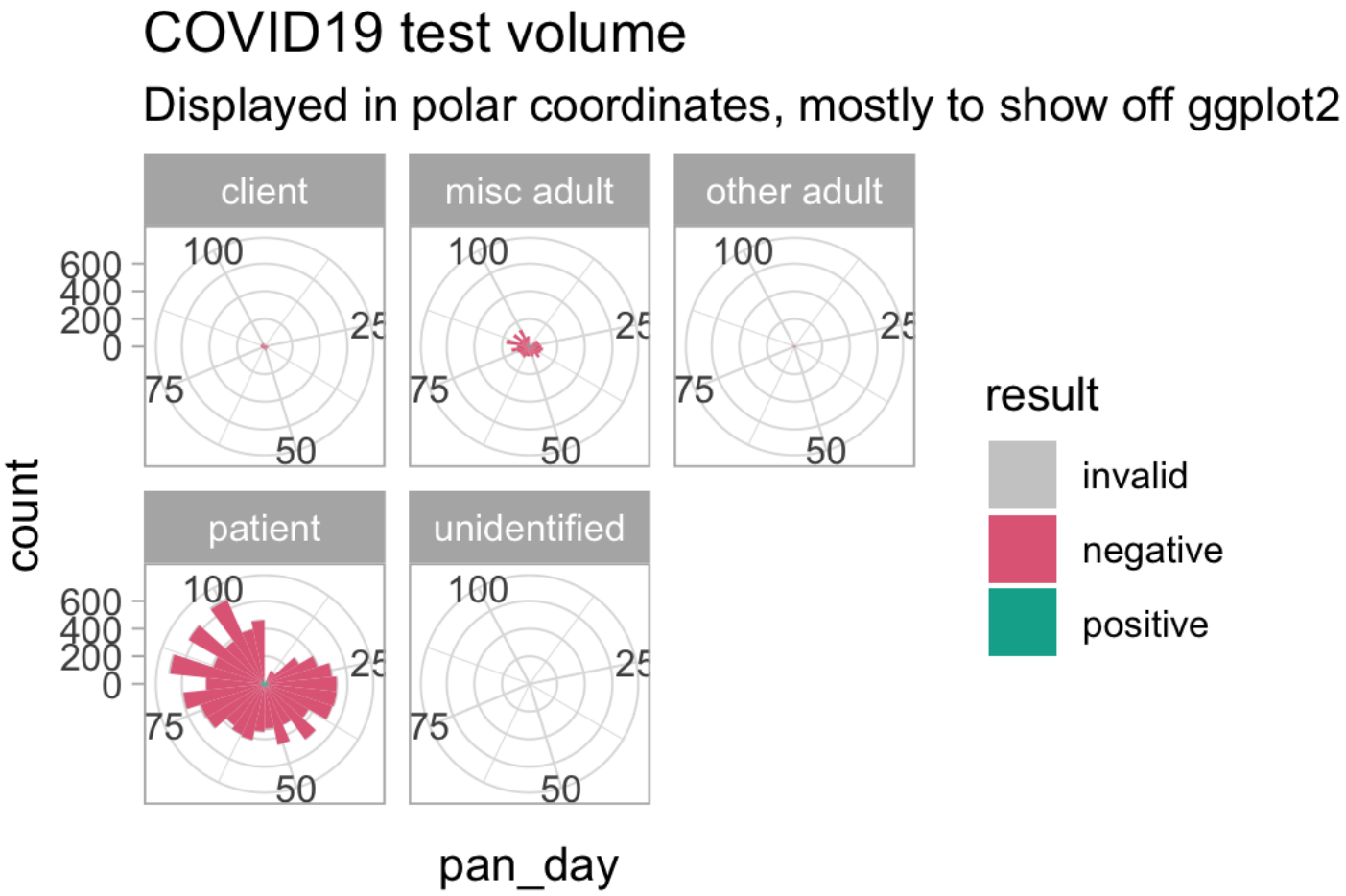
Subplots that display subsets of the data



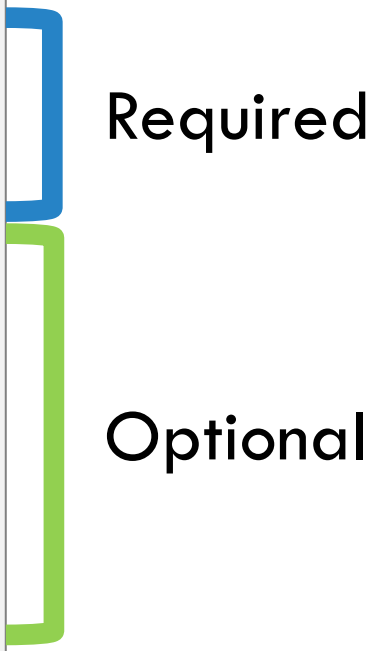
Coordinate systems



Titles and captions



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



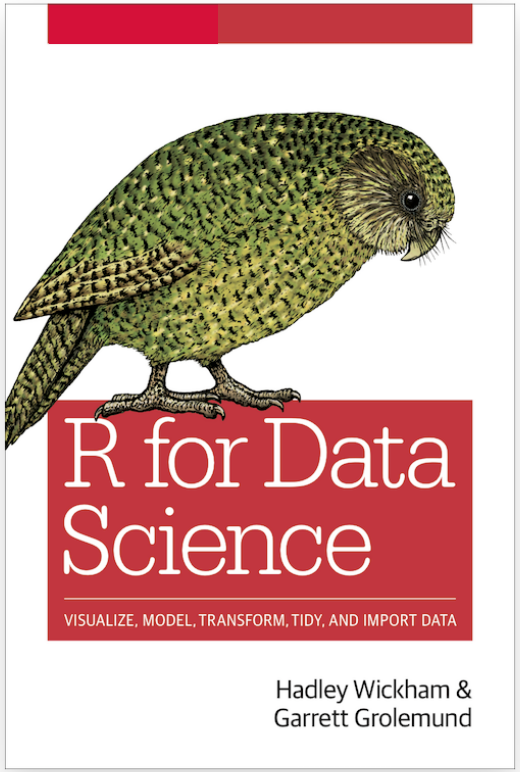
https://r4ds.had.co.nz/

R for Data Science

Search

Table of contents

Welcome



This is the website for **"R for Data Science"**. This book will teach you how to do data science with R: You'll learn how to get your data into R, get it into the most useful structure, transform it, visualise it and model it. In this book, you will find a practicum of skills for data science. Just as a chemist learns how to clean test tubes and stock a lab, you'll learn how to clean data and draw plots—and many other things besides. These are the skills that allow data science to happen, and here you will find the best practices for doing each of these things with R. You'll learn how to use the grammar of graphics, literate programming, and reproducible research to save time. You'll also learn how to manage cognitive resources to facilitate discoveries when wrangling, visualising, and exploring data.

https://www.rstudio.com/resources/cheatsheets/

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

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

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,

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

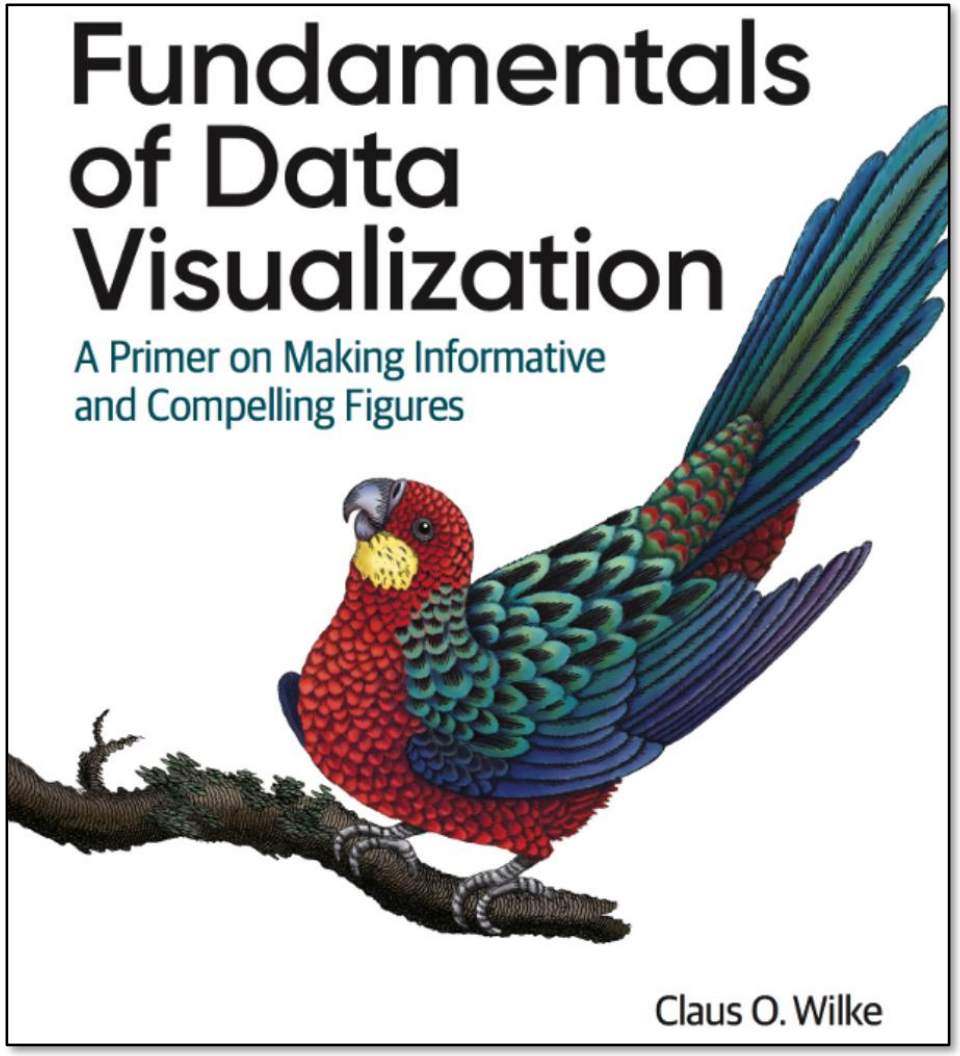
df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
j <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))

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,



https://clauswilke.com/dataviz/

Creating Survival Plots

Informative and Elegant with *survminer*

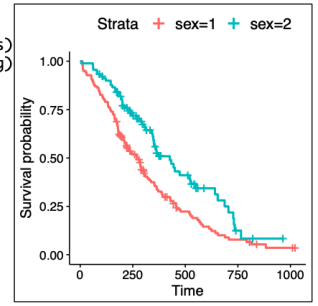
Survival Curves

The **ggsurvplot()** function creates **ggplot2** plots from **survfit** objects.

```
library("survival")
fit <- survfit(Surv(time,status)
               ~ sex, data = lung)

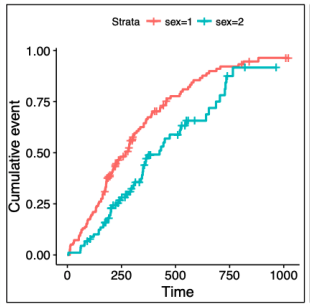
class(fit)
## [1] "survfit"

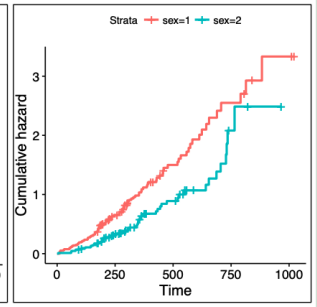
library("survminer")
ggsurvplot(fit, data = lung)
```



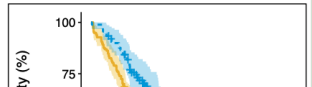
Use the **fun** argument to set the transformation of the survival curve. E.g. **"event"** for cumulative events, **"cumhaz"** for the cumulative hazard function or **"pct"** for survival probability in percentage.

```
ggsurvplot(fit, data = lung, fun = "event")
ggsurvplot(fit, data = lung, fun = "cumhaz")
```





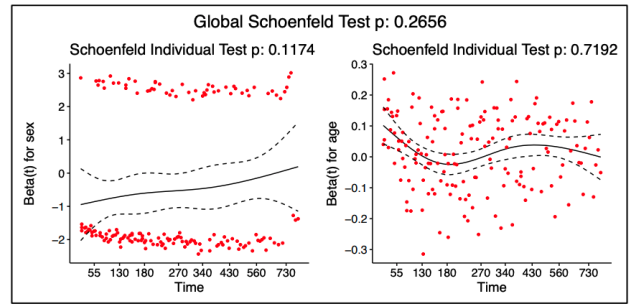
With lots of graphical parameters you have full control over look and feel of the survival



Diagnostics of Cox Model

The function **cox.zph()** from **survival** package may be used to test the proportional hazards assumption for a Cox regression model fit. The graphical verification of this assumption may be performed with the function **ggcoxzph()** from the **survminer** package. For each covariate it produces plots with scaled Schoenfeld residuals against the time.

```
library("survival")
fit <- coxph(Surv(time, status) ~ sex + age, data = lung)
ftest <- cox.zph(fit)
ftest
##            rho chisq     p
## sex     0.1236 2.452 0.117
## age    -0.0275 0.129 0.719
## GLOBAL      NA 2.651 0.266
library("survminer")
ggcoxzph(ftest)
```

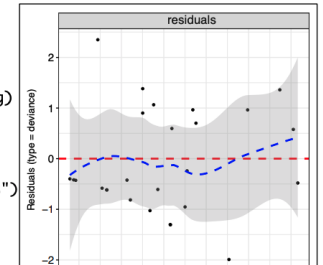


The function **ggcoxdiagnostics()** plots different types of residuals as a function of time, linear predictor or observation id. The type of residual is selected with **type** argument. Possible values are "martingale", "deviance", "score", "schoenfeld", "dfbeta"', "dfbetas", and "scaledsch".
The **ox.scale** argument defines what shall be plotted on the OX axis. Possible values are "linear.predictions", "observation.id", "time".
Logical arguments **hline** and **sline** may be used to add horizontal line or smooth line to the plot.

```
library("survival")
library("survminer")
fit <- coxph(Surv(time, status) ~
             sex + age, data = lung)


ggcoxdiagnostics(fit,
     type = "deviance",
     ox.scale = "linear.predictions")


ggcoxdiagnostics(fit,
     type = "schoenfeld",
```



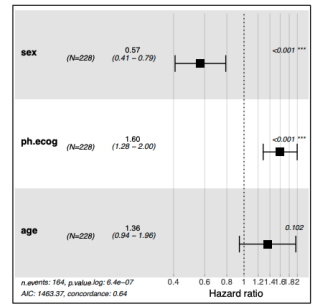
Summary of Cox Model

The function **ggforest()** from the **survminer** package creates a forest plot for a Cox regression model fit. Hazard ratio estimates along with confidence intervals and p-values are plotter for each variable.

```
library("survival")
library("survminer")
lung$age <- ifelse(lung$age > 70, ">70","<= 70")
fit <- coxph( Surv(time, status) ~ sex + ph.ecog + age, data = lung)
fit

## Call:
## coxph(formula = Surv(time, status) ~ sex+ph.ecog+age, data=lung)
##
##            coef exp(coef) se(coef)     z       p
## sex      -0.567     0.567    0.168 -3.37 0.00075
## ph.ecog   0.470     1.600    0.113  4.16 3.1e-05
## age>70    0.307     1.359    0.187  1.64 0.10175
##
## Likelihood ratio test=31.6  on
## n= 227, number of events= 164

ggforest(fit)
```



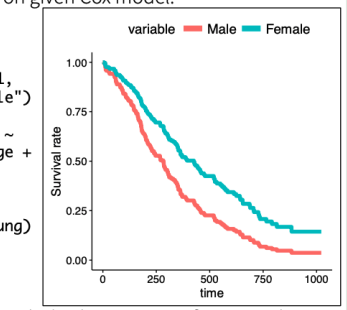
The function **ggadjustedcurves()** from the **survminer** package plots Adjusted Survival Curves for Cox Proportional Hazards Model. Adjusted Survival Curves show how a selected factor influences survival estimated from a Cox model.
Note that these curves differ from Kaplan Meier estimates since they present expected ssurvival based on given Cox model.

```
library("survival")
library("survminer")

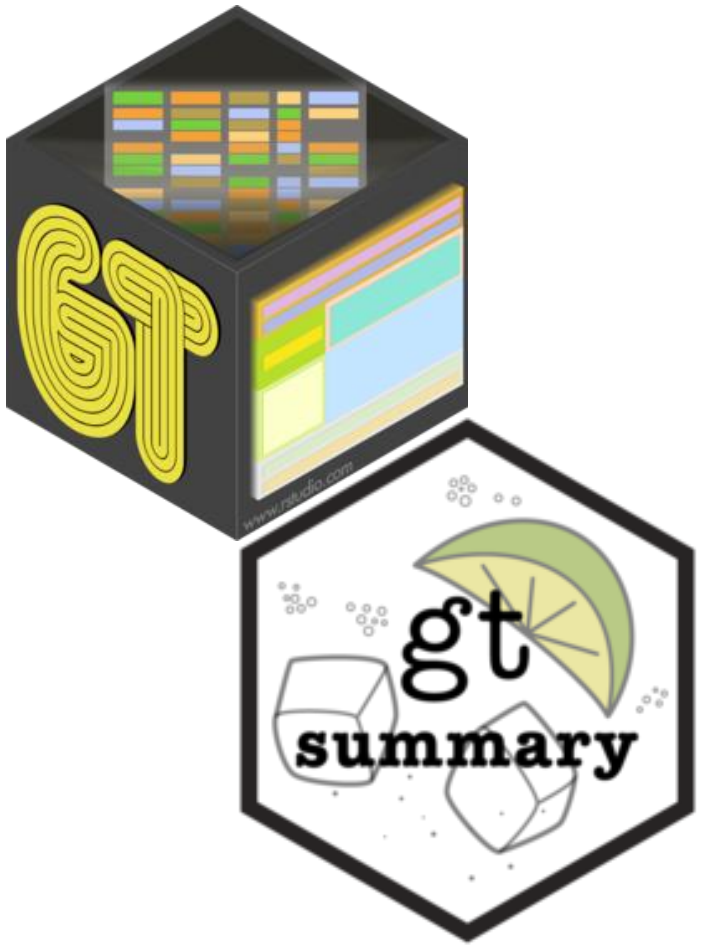
lung$sex <- ifelse(lung$sex == 1,
                   "Male", "Female")

fit <- coxph(Surv(time, status) ~
             sex + ph.ecog + age +
             strata(sex),
             data = lung)

ggcoxadjustedcurves(fit, data=lung)
```



A grammar for **tables**



Variable	**N**	**Drug A**, N = 98[1]	**Drug B**, N = 102[1]	**p-value**[2]
Age	189	46 (37, 59)	48 (39, 56)	0.7
Grade	200			0.9
I		35 (36%)	33 (32%)	
II		32 (33%)	36 (35%)	
III		31 (32%)	33 (32%)	
Tumor Response	193	28 (29%)	33 (34%)	0.6

[1] Statistics presented: median (IQR); n (%)

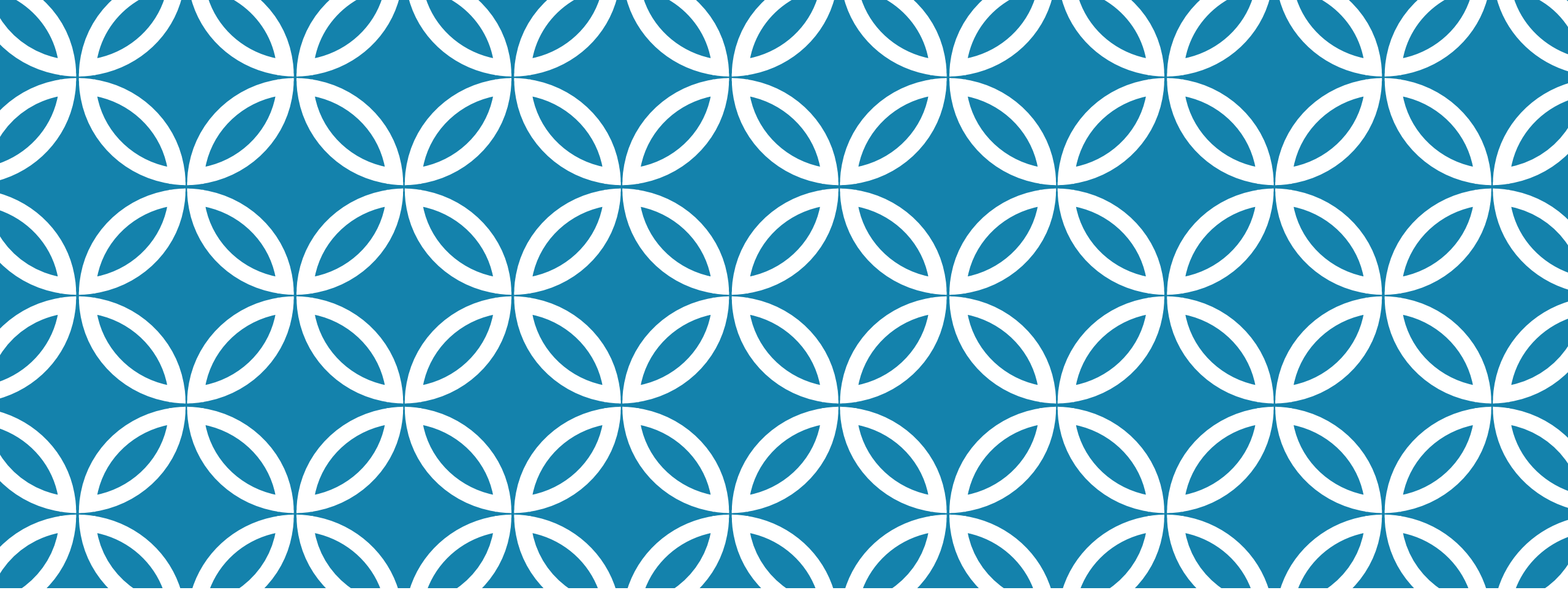
[2] Statistical tests performed: Wilcoxon rank-sum test; chi-square test of independence

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

Session 4

Amrom Obstfeld

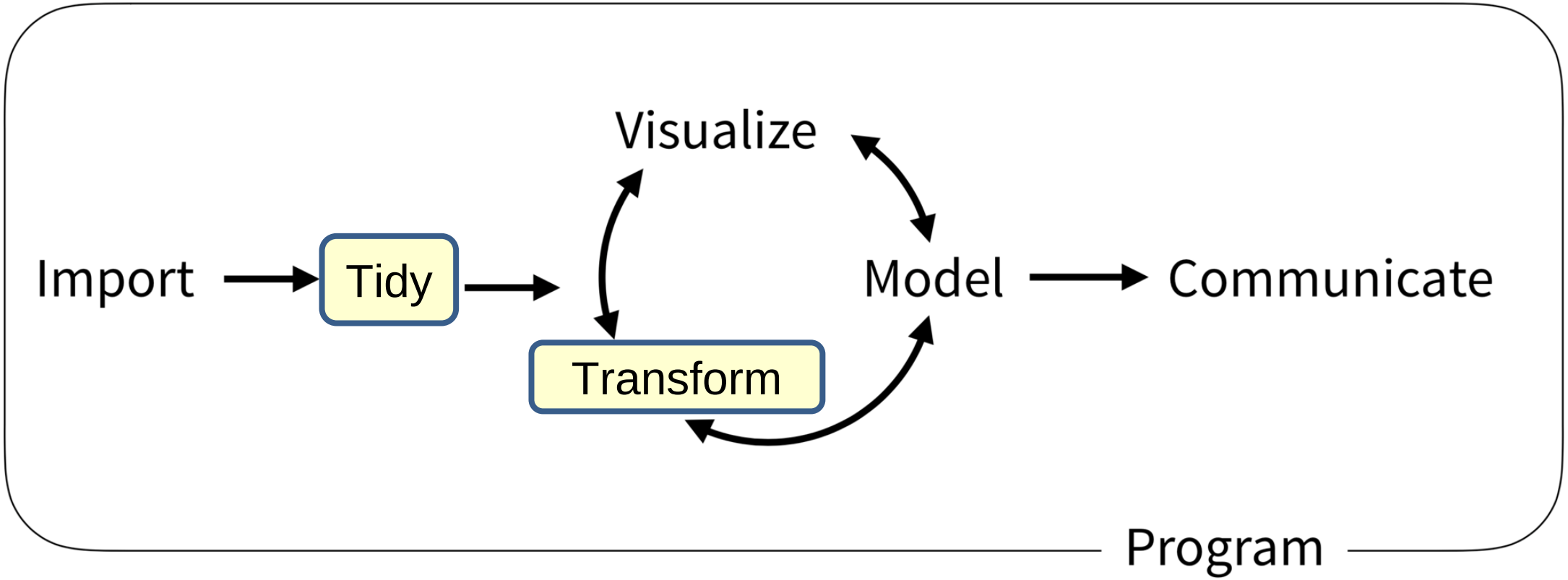
Goal

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

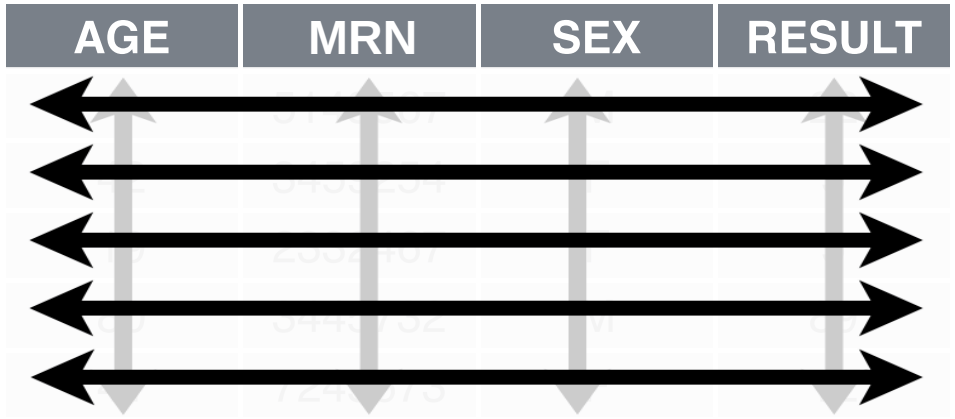
Objectives

1. List the major forms of data transformation implemented in dplyr
2. Use code templates with dplyr functions to tidy a raw data set
3. Use the pipe operator to pass the output of one function as an input to the next function
4. Create new calculated columns not found in the original data frame

Typical Data Science Pipeline



What is a “Tidy” Data Frame



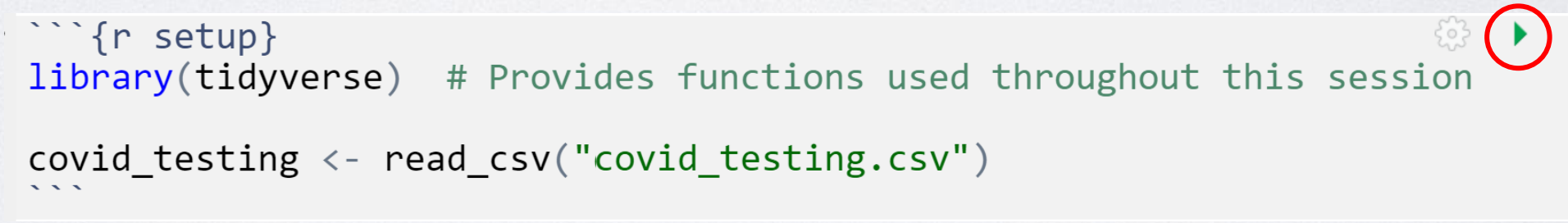
A data set is **tidy** if:

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



Pop Quiz

How can you confirm that you have successfully loaded the data file into RStudio?

1. The code that imported the data did not yield an error
2. Code that references the `covid_testing` object runs without errors
3. The `covid_testing` object is present in the environment pane
4. All of the above

Transform Data with

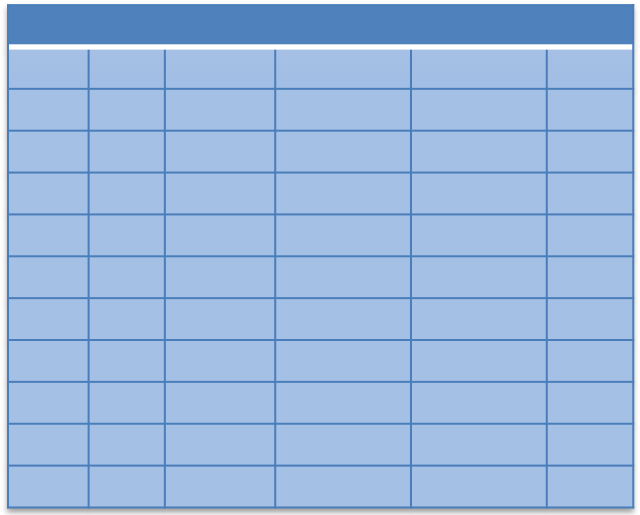
dplyr

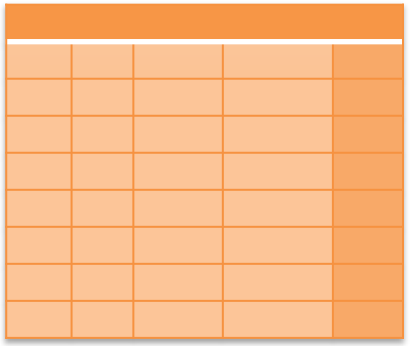
dplyr implements a *grammar* for transforming tabular data.

dplyr: a grammar for transforming data

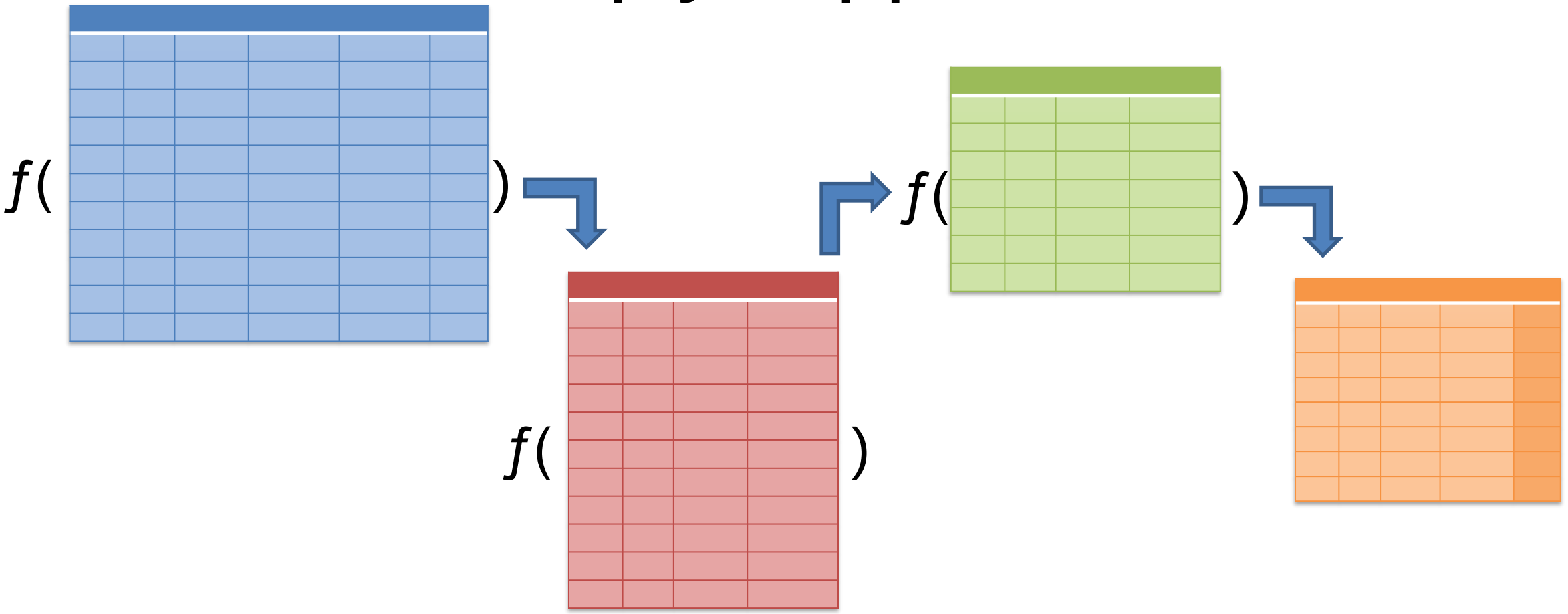
1. **Choose** columns. — **select()**
2. **Extract** rows. — **filter()**
3. **Derive** new columns. — **mutate()**
4. **Change** the unit of analysis. — **summarize()**

Dplyr Approach

$f($  $)$



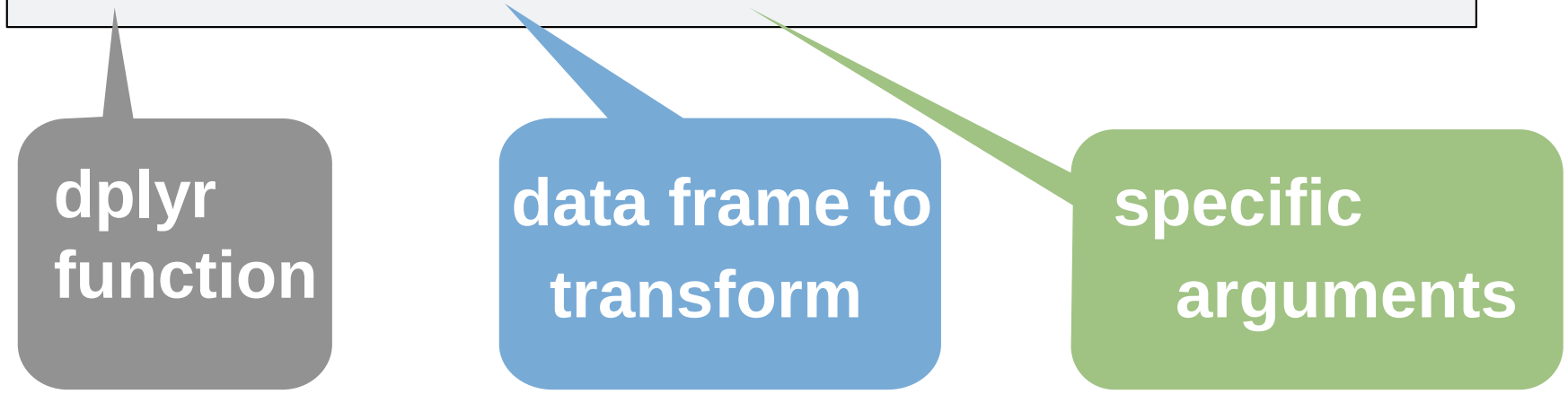
Dplyr Approach



Common syntax

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

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

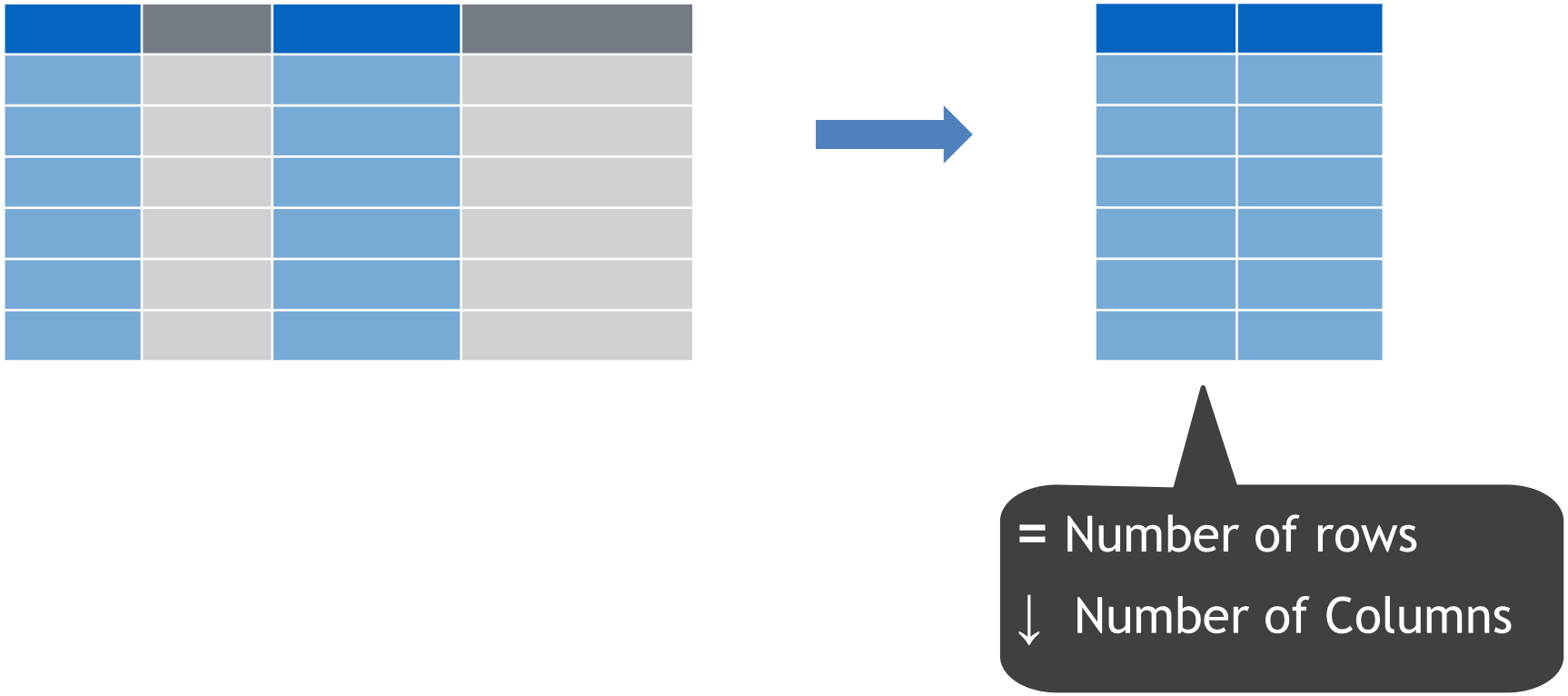


Isolating data

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**

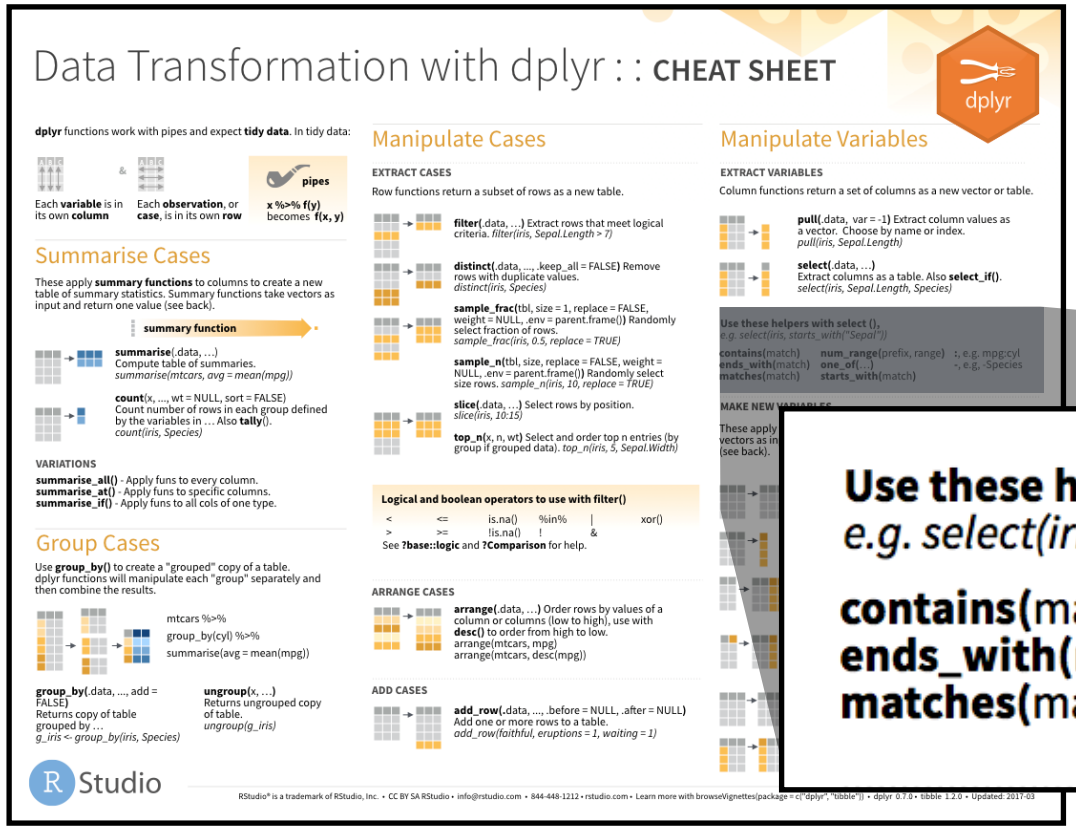
```
select(covid_testing, -mrn, -last_name)
```

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

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

Your Turn 2

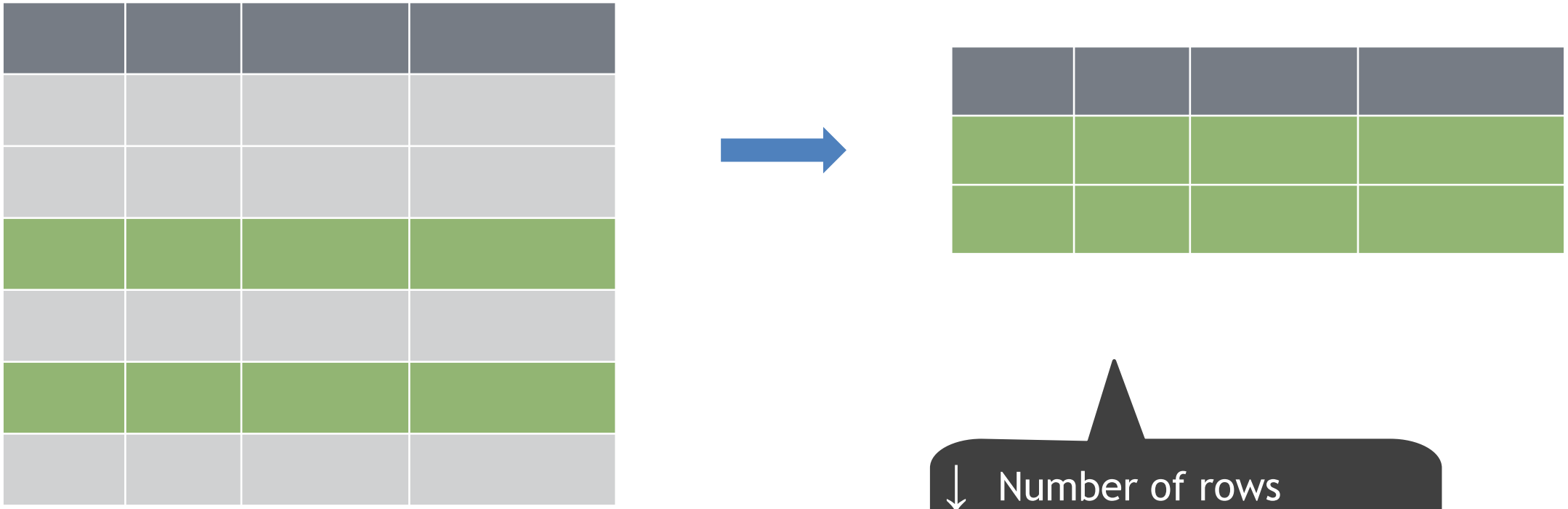
- Alter the code to select just the `first_name` column from `covid_testing`
- Use the second code chunk to see if you can remove the `first_name` column

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

filter()

filter()

Extract rows that meet logical criteria



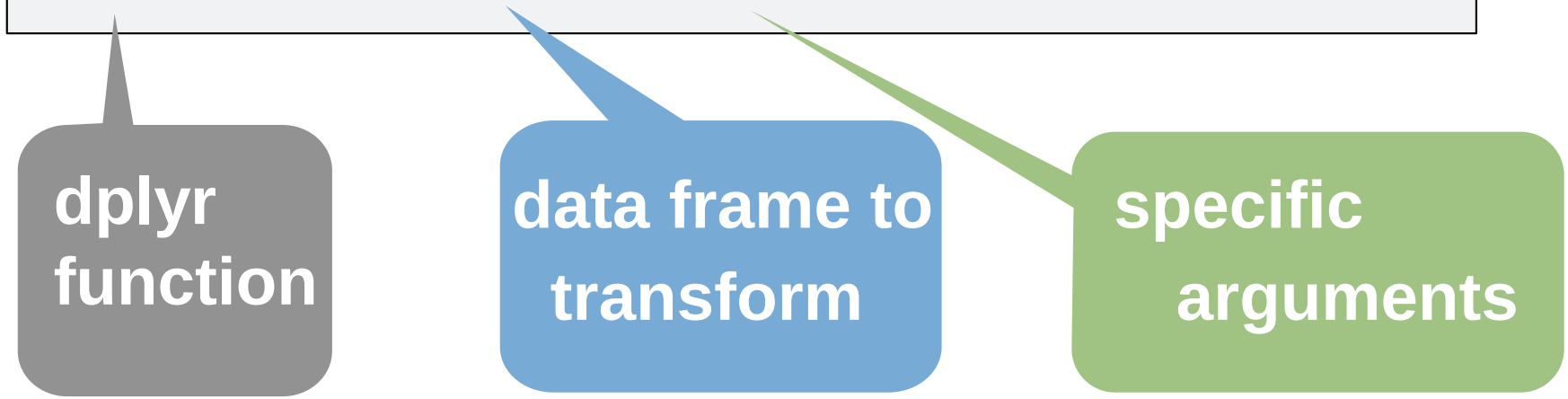
↓ Number of rows

= 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, ...)
```



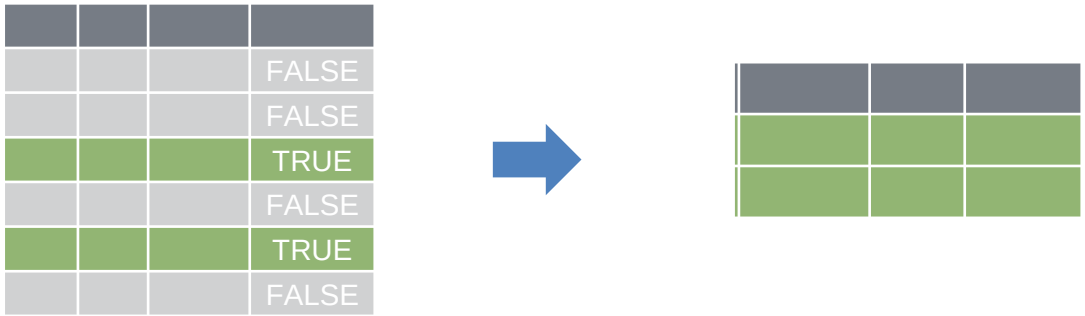
filter()

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)

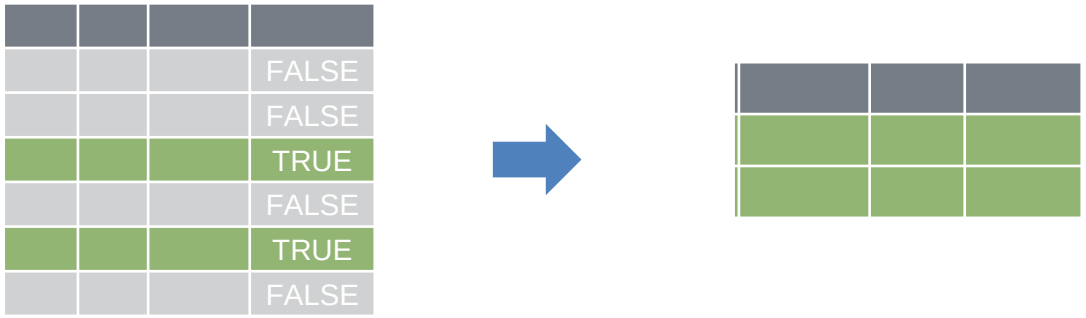


filter()

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?

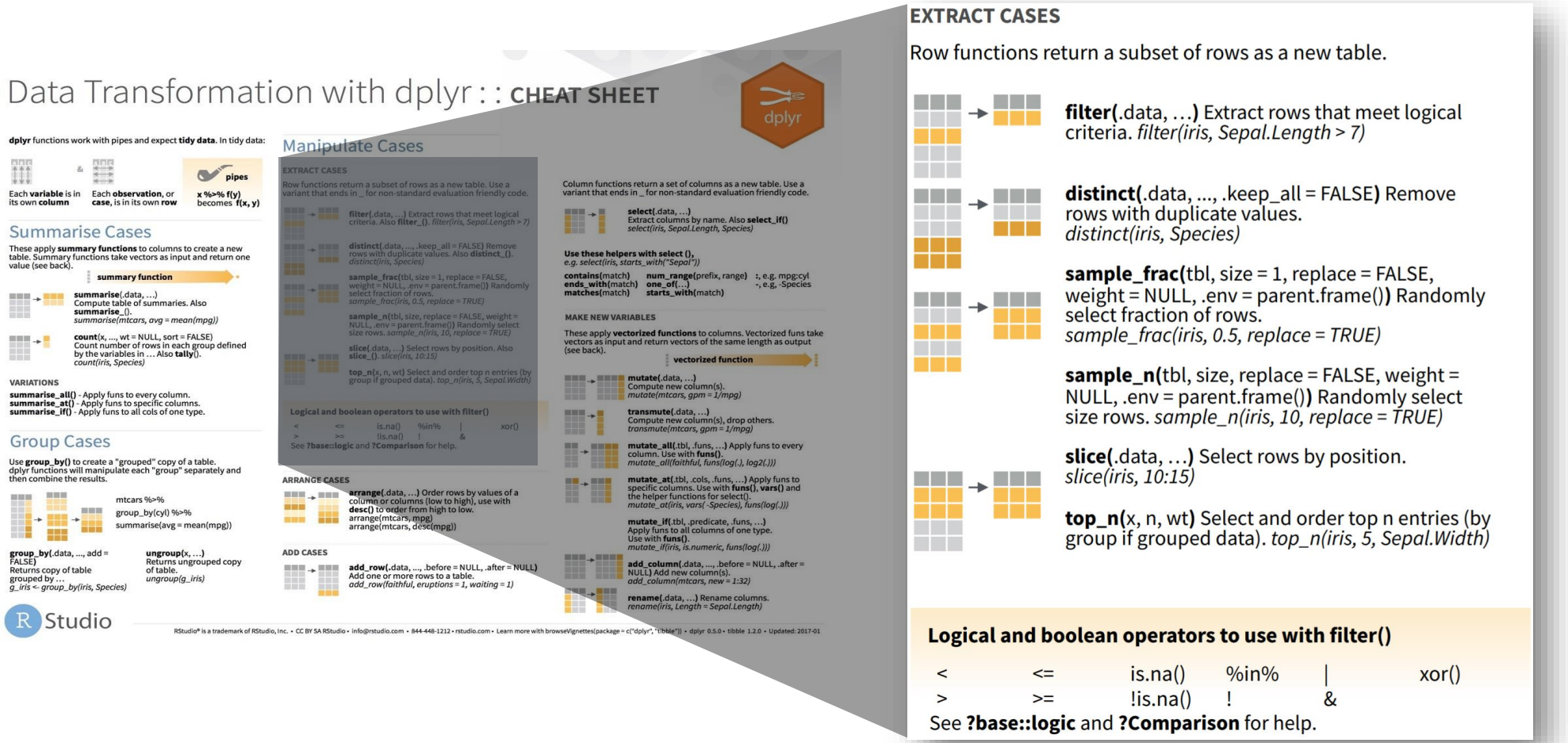
```
1 == 1
```

Pop Quiz

What is the result?

```
3 != 1
```

filter() variants



Your Turn 3

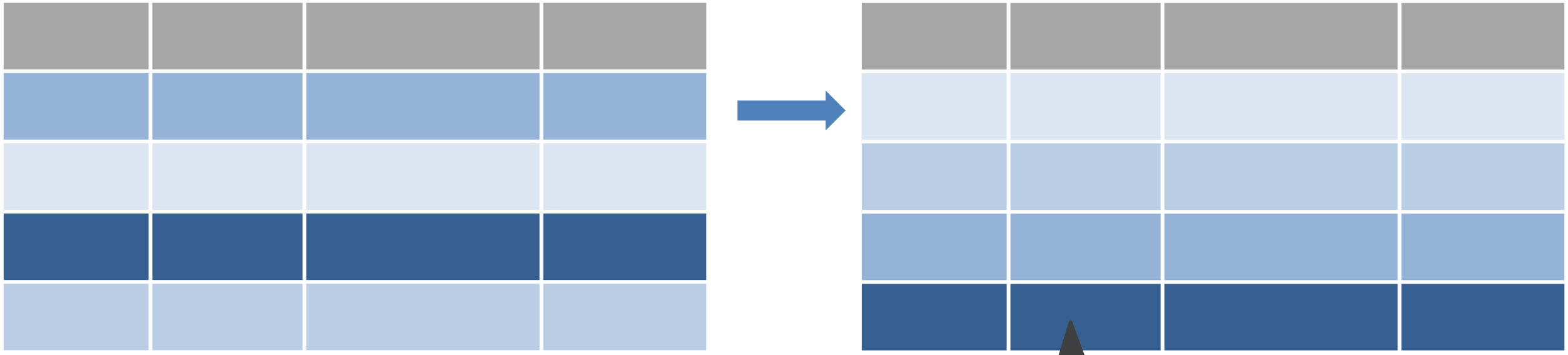
Use filter() with the logical operators to find:

- Every test for patients **over age 80**
- All of the covid testing where the demographic group (demo_group) is **equal to "client"**

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_value` 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"?

%>%

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)
```

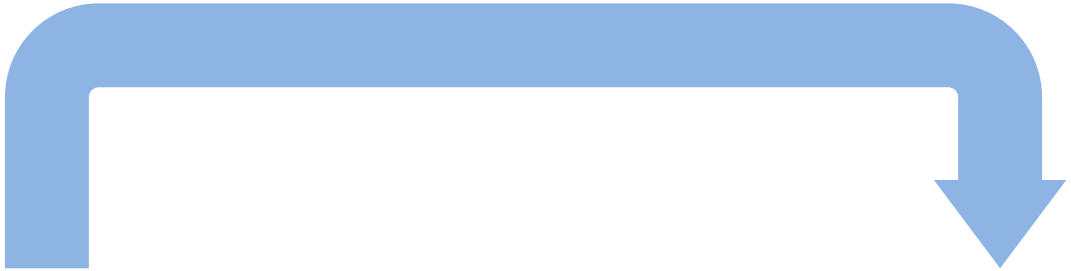
1. Filter tests to those on pandemic day less than 10
2. Select the column that contains ordering location
3. Arrange those columns by location

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_tesing, pan_day <= 10)
covid_tesing %>% 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)
```

Shortcut to type %>%

(Mac)

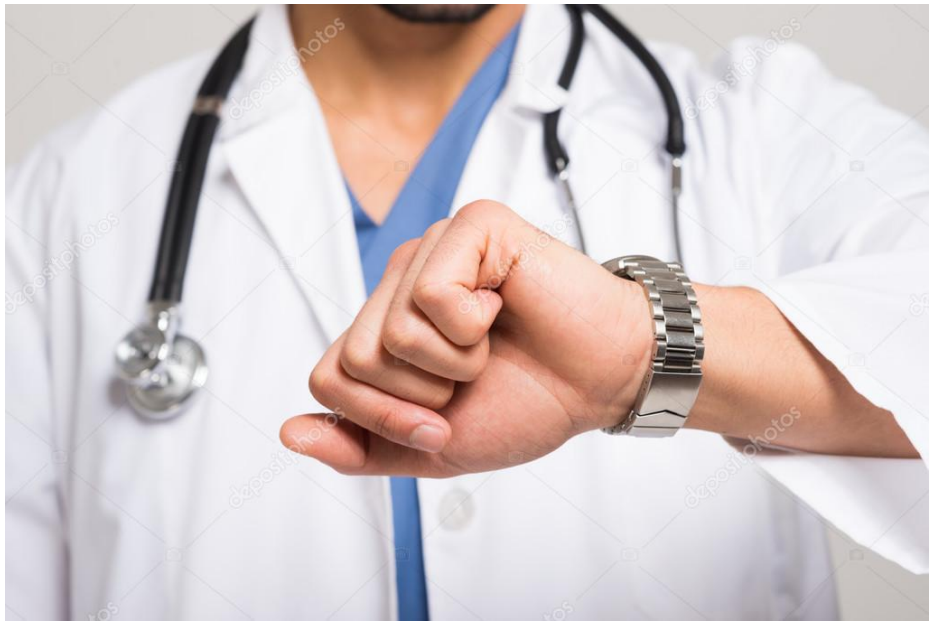
(Windows)

Scene

The PICU would like a word with you because of a recent incident involving a delay in results for a patient who required a AGP

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

You decide to investigate… WITH DATA



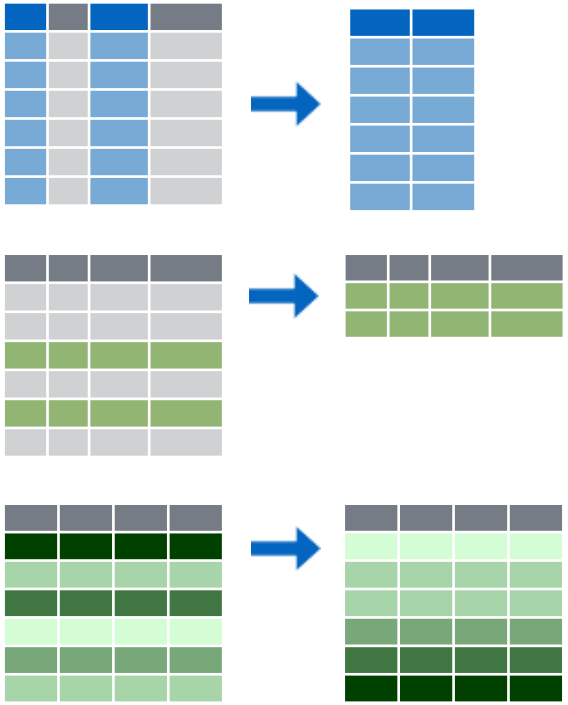
Your Turn 5

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

1. Filters to tests from the clinic (**`clinic_name`**) of "picu"
2. Selects the column with the receive to verify turnaround time (**`rec_ver_tat`**) as well as the day from start of the pandemic (**`pan_day`**)
3. Arrange the `` `pan_day` `` from highest to lowest

Using <-, assign the result to a new variable, call it whatever you want.

Isolating data



Extract variables with **select()**

Extract rows with **filter()**

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

Deriving Data

What is the mean and median collect to verify turnaround time by clinic?

Breaking down the analytical question

1. Total TAT for each test
2. Group tests by clinic
3. Calculate mean and median for each clinic

Deriving data

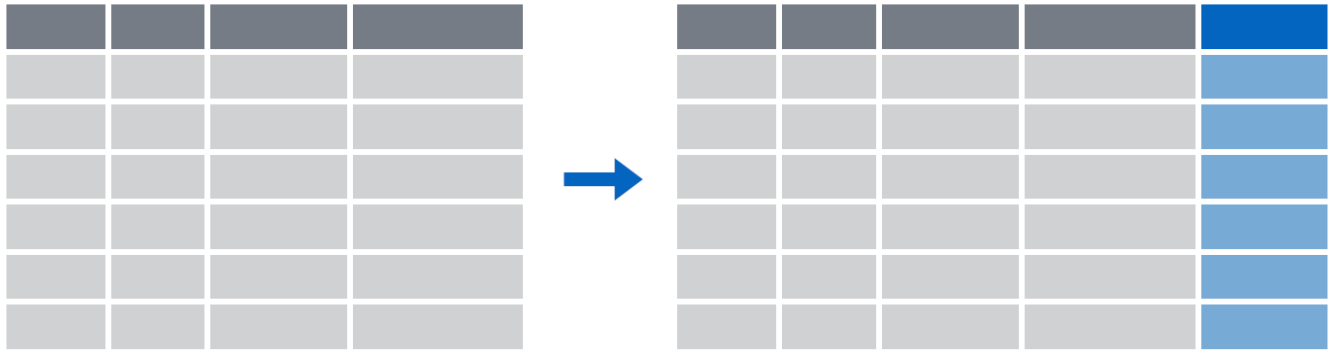
Make new variables with **mutate()**

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

mutate()

mutate()

Creating new calculated columns



= Number of rows

↑ Number of 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 6

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

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

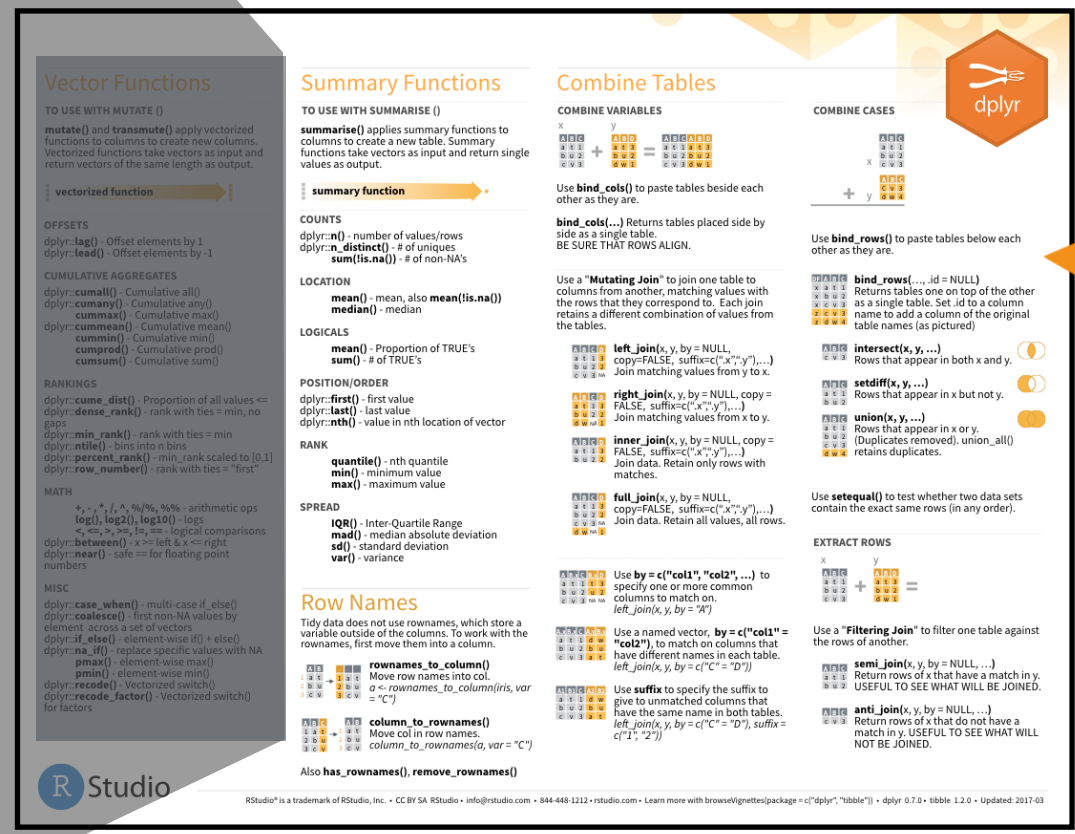
dplyr::**cume_dist()** - Proportion of all values <=
dplyr::**dense_rank()** - rank with ties = min, no gaps
dplyr::**min_rank()** - rank with ties = min
dplyr::**ntile()** - bins into n bins
dplyr::**percent_rank()** - min_rank scaled to [0,1]
dplyr::**row_number()** - rank with ties = "first"

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. List the major forms of data transformation implemented in dplyr
2. Use code templates containing dplyr functions to tidy a raw data set
3. Use the pipe operator to pass the output of one function as an input to the next function
4. Create new calculated columns not found in the original data frame

Dplyr tips and tricks

select()

Renaming columns

```
covid_testing %>%
    select(MRN = mrn, first_name, last_name)
```

mrn <dbl>	first_name <chr>	last_name <chr>
5001412	jhezane	westerling
5000533	penny	targaryen
5009134	grunt	rivers
5008518	melisandre	swyft

MRN <dbl>	first_name <chr>	last_name <chr>
5001412	jhezane	westerling
5000533	penny	targaryen
5009134	grunt	rivers
5008518	melisandre	swyft

filter()

Filter to multiple matches

```
covid_testing %>%
    filter(first_name %in% c("jon","daenerys"))
```

mrn <dbl>	first_name <chr>
5001412	jhezane
5000533	penny
5009134	grunt
5008518	melisandre
5008967	rolley

mrn <dbl>	first_name <chr>
5002427	daenerys
5011120	jon
5001092	jon
5004082	jon
5005197	daenerys

mutate()

Replacing columns

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

```
covid_testing %>%
    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

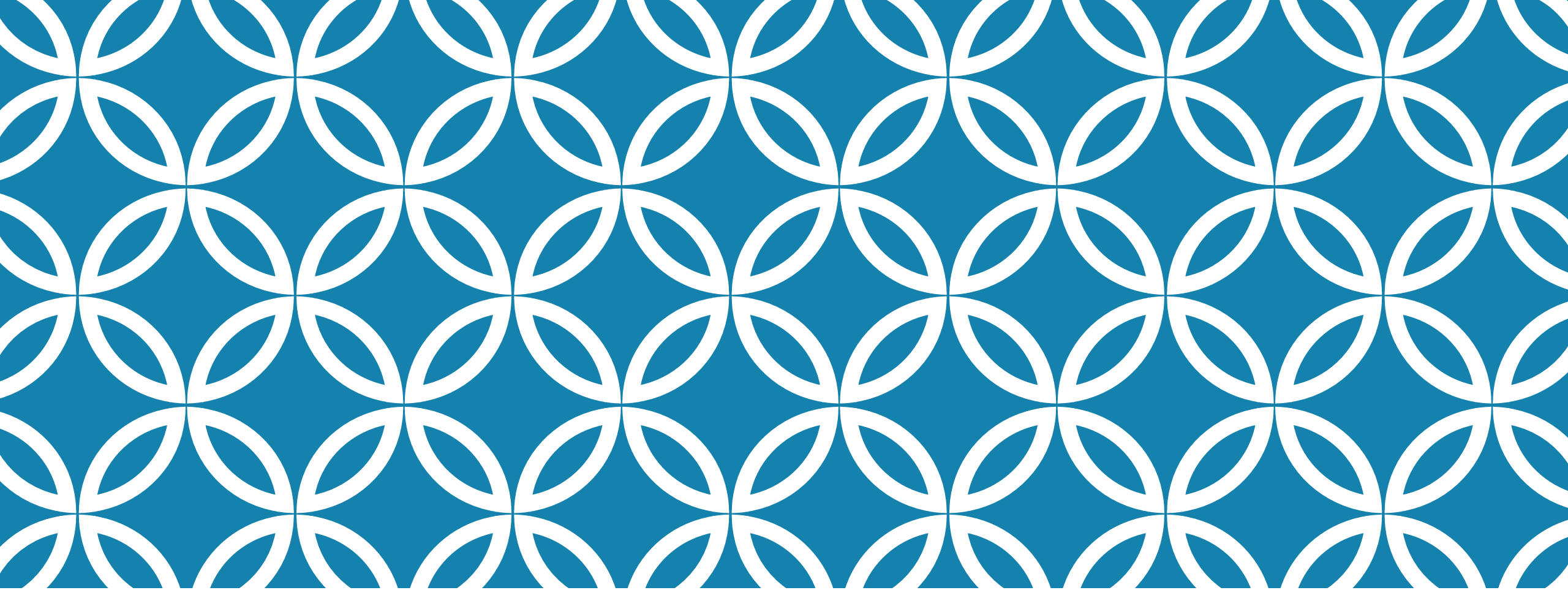
mutate()

Conditionally replacing values

```
covid_testing %>%
     mutate(last_name = if_else(last_name=="targaryen",
                                "TARGARYEN",last_name))
```

mrn <dbl>	first_name <chr>	last_name <chr>
5001412	jhezane	westerling
5000533	penny	targaryen
5009134	grunt	rivers
5008518	melisandre	swyft

mrn <dbl>	first_name <chr>	last_name <chr>
5001412	jhezane	westerling
5000533	penny	TARGARYEN
5009134	grunt	rivers
5008518	melisandre	swyft



Data Understanding: Grouping and Summarizing Data

Patrick Mathias

May 9, 2022

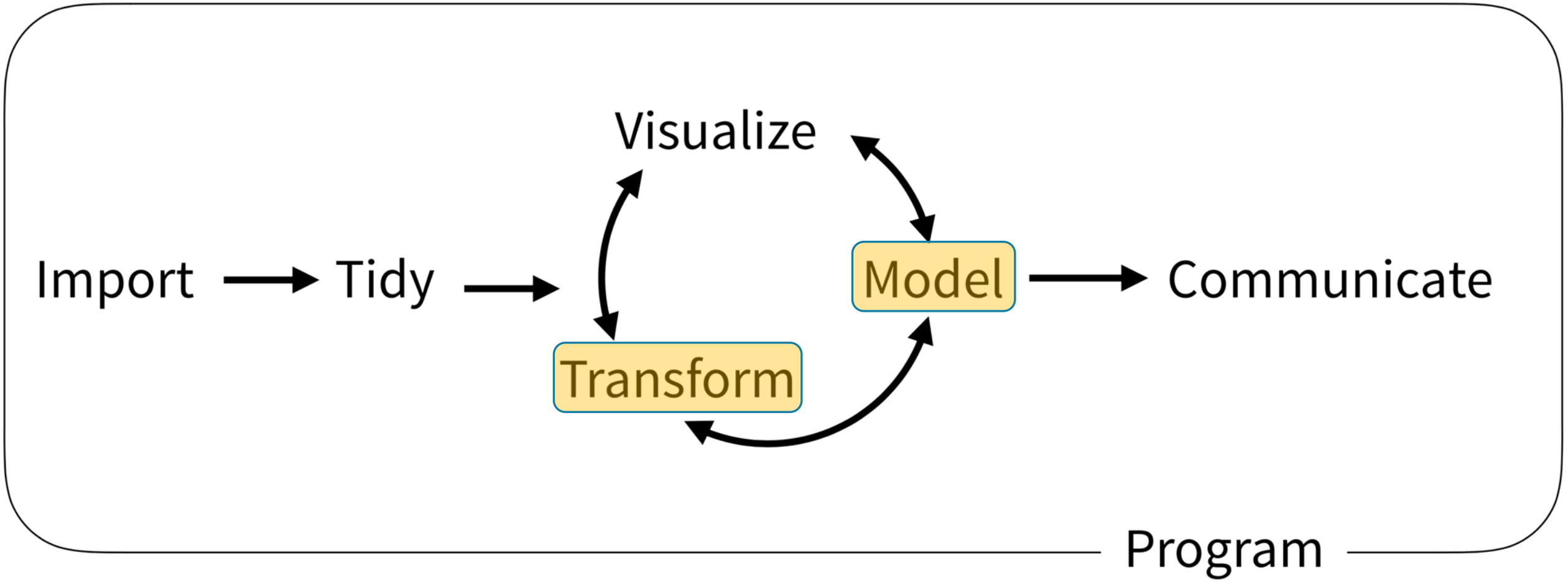
Goals

1. Learn dplyr tools for grouping and summarizing data in R

Objectives

1. Calculate a summary statistic for a variable using the summarize() function
2. Creates groupings of data using the group_by() function
3. Combine group_by() and summarize() functions to calculate summary statistics for groups of data

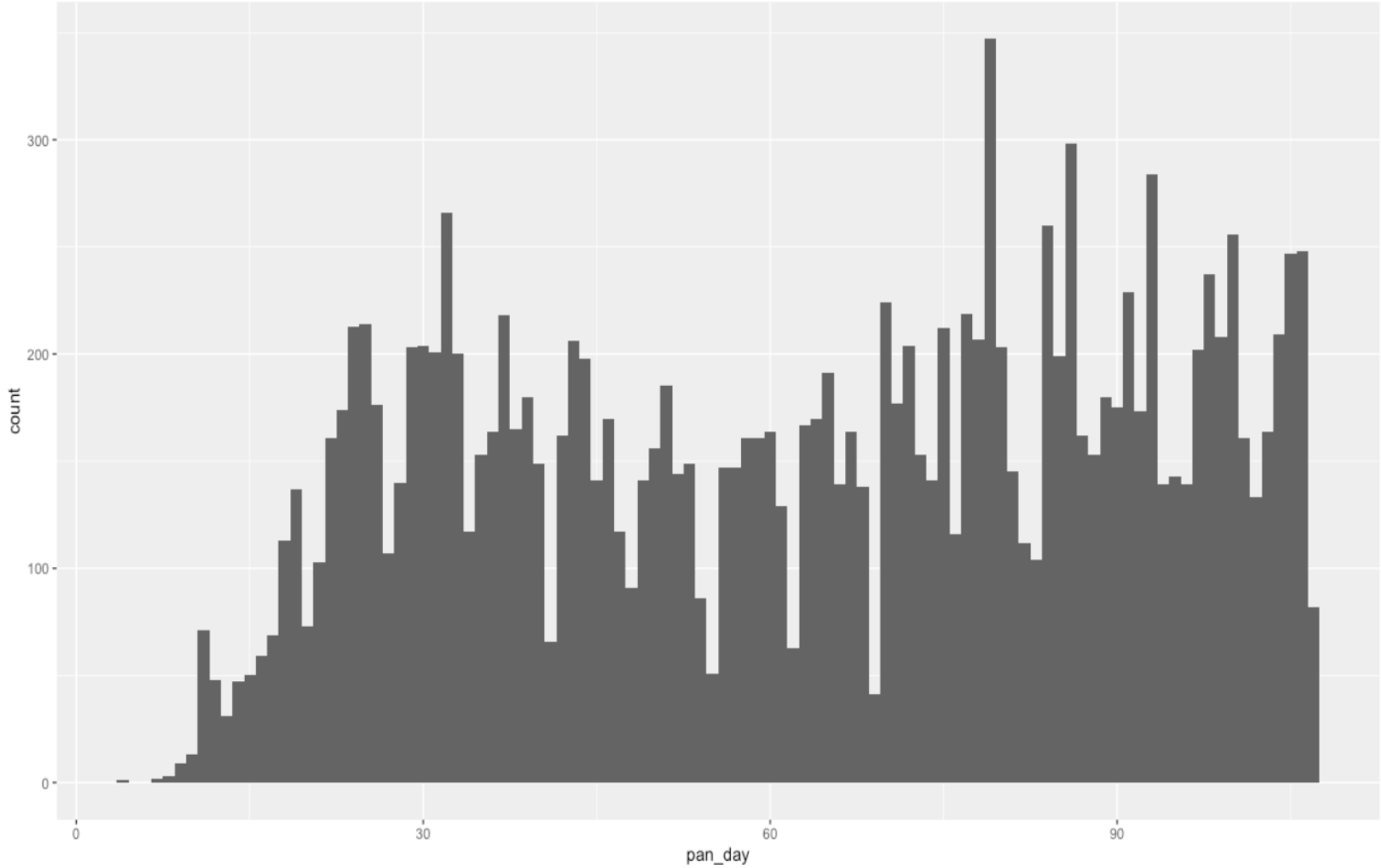
Typical Data Science Pipeline



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

Summarize the data set

Q: How many tests are ordered per day?



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

Performs calculation across all rows of data frame

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()

- Additional summaries = new columns

```
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

summarize()

- Summarize supports calculations on summary stats

```
covid_testing %>%
      summarize(order_count = n(),
                day_count = n_distinct(pan_day),
                orders_per_day = order_count/day_count)
```

mrn	pan_day
5001412	4
5000533	7
5009134	7
5008518	8

→

order_count	day_count	orders_per_day
15524	102	152

Your Turn #1

- Open “05 – Group and Summarize.Rmd”
- Run the setup chunk
- Fill-in the gaps to calculate the mean count of orders per clinic

Vector Functions

TO USE WITH MUTATE ()

torized
olumns.
nput and
utput.

n()

l values <=
= min, no

nin

aled to [0,1]
= "first"

etic ops

nparisons

nt
se()
s by

lse()
s with NA

witch()

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.

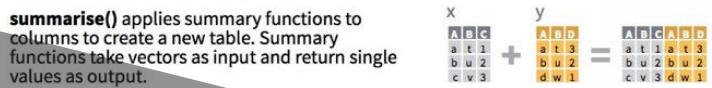
rownames_to_column()
Move row names into col.
a <- rownames_to_column(iris, var = "C")

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.

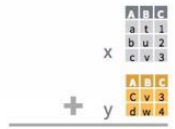
full_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),…)
Join data. Retain all values, all rows.

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

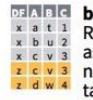
Use a named vector, **by = c("col1" = "col2")**, to match on columns with different names in each data set.
left_join(x, y, by = c("C" = "D"))

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)

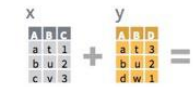
intersect(x, y, …)
Rows that appear in both x and y.

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.

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 0.5.0 • tibble 1.2.0 • Updated: 2017-01

Output the last day

```
covid_testing %>%
    summarize(last_day = last(pan_day))
```

mrn	pan_day
5001412	4
5000533	7
5009134	7
5008518	8

last_day
8

Calculate the mean turnaround time

```
covid_testing %>%
  mutate(col_ver_tat = col_rec_tat + rec_ver_tat) %>%
  summarize(col_ver_tat_mean = mean(col_ver_tat))
```

mrn	pan_day	col_ver_tat
5001412	4	6
5000533	7	8
5009134	7	10
5008518	8	11

→

col_ver_tat_mean
8.75

Calculate the 75th percentile turnaround time

```
covid_testing %>%
 mutate(col_ver_tat = col_rec_tat + rec_ver_tat) %>%
 summarize(col_ver_tat_mean = mean(col_ver_tat),
           col_ver_75_pctile = quantile(col_ver_tat, 0.75)
```

mrn	pan_day	col_ver_tat
5001412	4	6
5000533	7	8
5009134	7	10
5008518	8	11

col_ver_tat_mean	col_ver_tat_mean
8.75	9.6

Your Turn #2

For the covid_testing data frame, calculate both the median and the 95th percentile collect-to-verify turnaround time.

Pop Quiz

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

Grouping your data

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`*

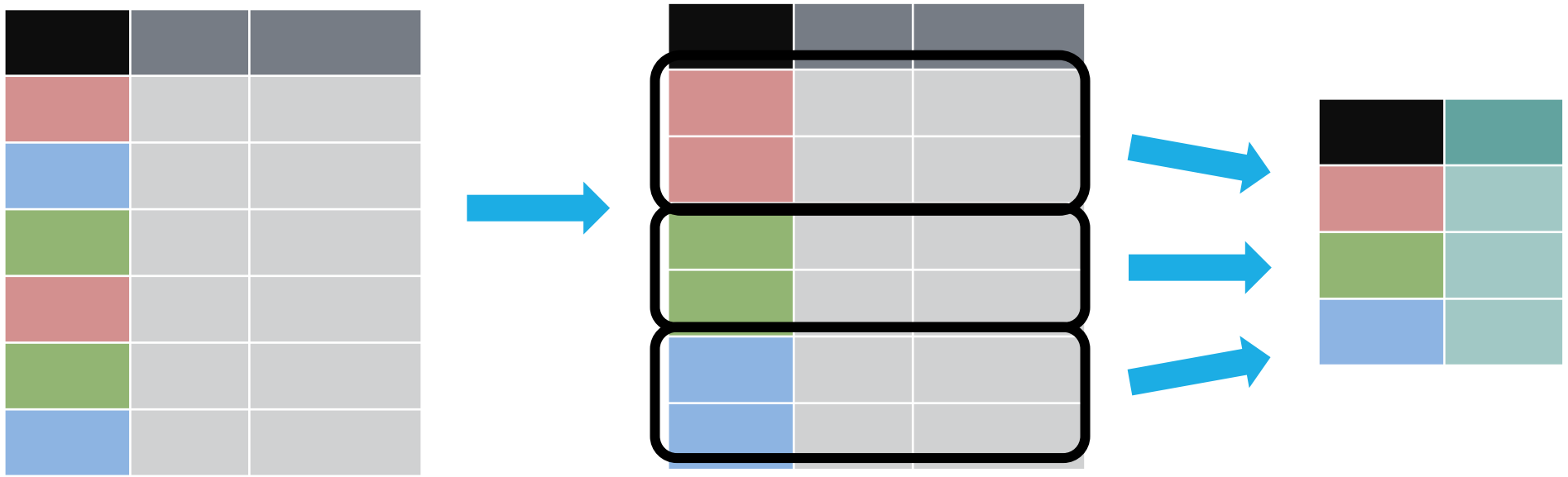
```
covid_testing %>%
    select(mrn, pan_day, clinic_name) %>%
    group_by(pan_day, clinic_name)
```

```
# A tibble: 15,524 x 3
# Groups:   pan_day, clinic_name [2,526]
       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

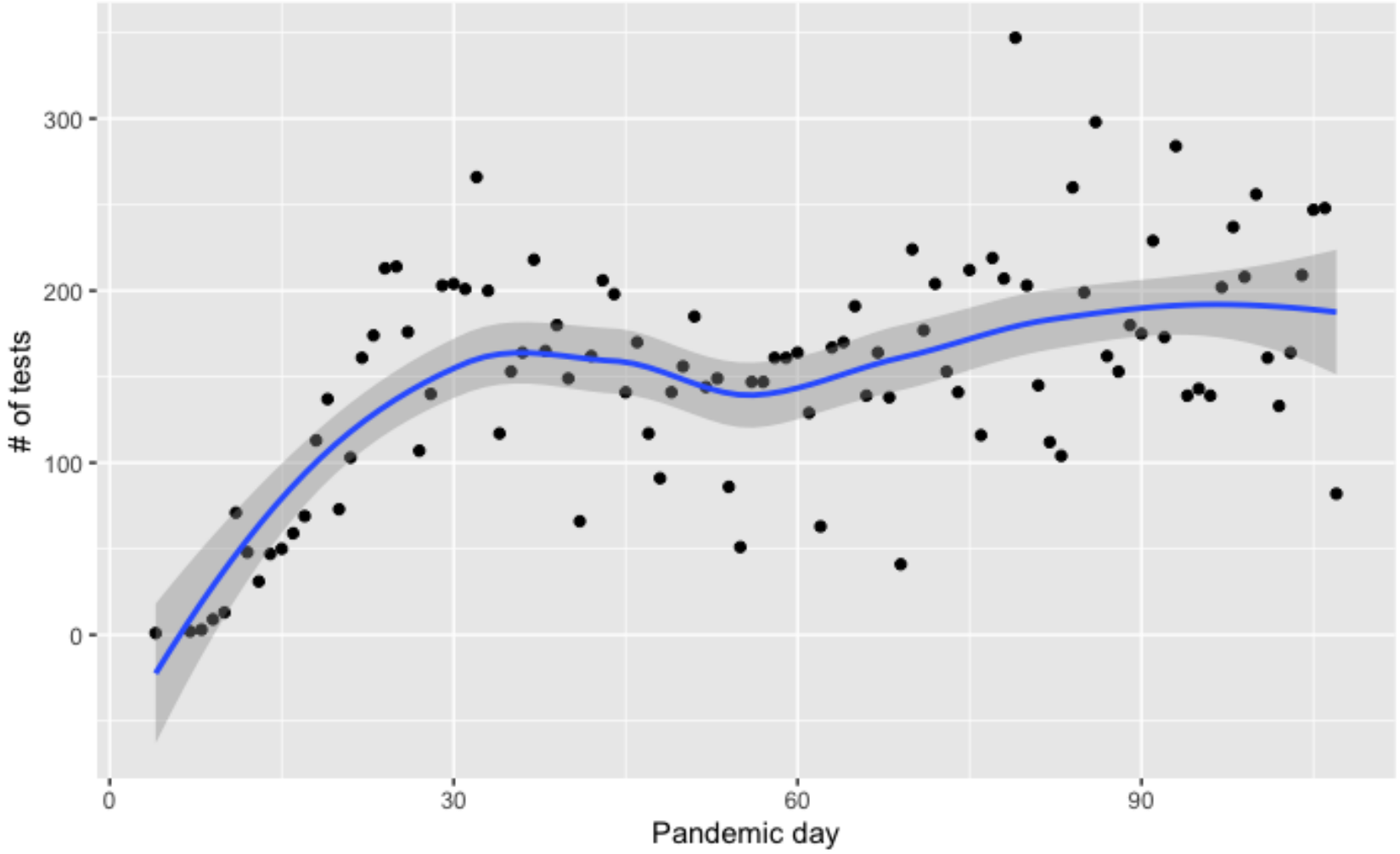
Calculate:

a) The median collect-to-verify turnaround time for each day

b) The median collect-to-verify turnaround time for each clinic/unit

c) The median number of orders per day

group_by() %>% summarize(): Example



Recap

Summarize() is a function that enables us to calculate summaries of variables (columns).

Common summary activities include counting observations using **n()**, counting unique observations using **n_distinct()**, and calculating means using **mean()**.

Group_by() is a function that enables us to create subsets of data by a variable. Data can also be grouped by multiple variables.

Combining the **group_by()** and **summarize()** functions is a powerful way to look at summarizations across groups.

What else?

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

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(.data, …, wt = NULL, sort = FALSE, name = NULL**)** Count number of rows in each group defined by the variables in … Also **tally()**.
count(mtcars, cyl)

Group Cases

Use **group_by(**.data, …, .add = FALSE, .drop = TRUE**)** to create a "grouped" copy of a table grouped by columns in ... dplyr functions will manipulate each "group" separately and combine the results.

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

Use **rowwise(**.data, …**)** to group data into individual rows. dplyr functions will compute results for each row. Also apply functions to list-columns. See tidyr cheat sheet for list-column workflow.

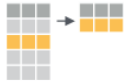
```
starwars %>%
  rowwise() %>%
  mutate(film_count = length(films))
```

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

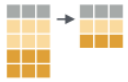
Manipulate Cases

EXTRACT CASES

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

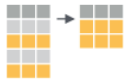


filter(.data, …, .preserve = FALSE**)** Extract rows that meet logical criteria.
filter(mtcars, mpg > 20)



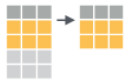
distinct(.data, …, .keep_all = FALSE**)** Remove rows with duplicate values.
distinct(mtcars, gear)

slice(.data, …, .preserve = FALSE**)** Select rows by position.
slice(mtcars, 10:15)



slice_sample(.data, …, n, prop, weight_by = NULL, replace = FALSE**)** Randomly select rows. Use n to select a number of rows and prop to select a fraction of rows.
slice_sample(mtcars, n = 5, replace = TRUE)

slice_min(.data, order_by, …, n, prop, with_ties = TRUE**)** and **slice_max()** Select rows with the lowest and highest values.
slice_min(mtcars, mpg, prop = 0.25)



slice_head(.data, …, n, prop**)** and **slice_tail()** Select the first or last rows.
slice_head(mtcars, n = 5)

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, …, .by_group = FALSE**)** 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(cars, speed = 1, dist = 1)

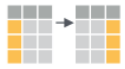
Manipulate Variables

EXTRACT VARIABLES

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

pull(.data, var = -1, name = NULL, …**)** Extract column values as a vector, by name or index.
pull(mtcars, wt)

select(.data, …**)** Extract columns as a table.
select(mtcars, mpg, wt)



relocate(.data, …, .before = NULL, .after = NULL**)** Move columns to new position.
relocate(mtcars, mpg, cyl, .after = last_col())

Use these helpers with select() and across()
e.g. select(mtcars, mpg:cyl)

contains(match**)**	**num_range(**prefix, range**)**	**:**, e.g. mpg:cyl
ends_with(match**)**	**all_of(**x**)/any_of(**x, …, vars**)**	**-**, e.g, -gear
starts_with(match**)**	**matches(**match**)**	**everything()**

MANIPULATE MULTIPLE VARIABLES AT ONCE

across(.cols, .funs, …, .names = NULL**)** Summarise or mutate multiple columns in the same way.
summarise(mtcars, across(everything(), mean))

c_across(.cols**)** Compute across columns in row-wise data.
transmute(rowwise(UKgas), total = sum(c_across(1:2)))

MAKE NEW VARIABLES

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

vectorized function

mutate(.data, …, .keep = "all", .before = NULL, .after = NULL**)** Compute new column(s). Also **add_column()**, **add_count()**, and **add_tally()**.
mutate(mtcars, gpm = 1 / mpg)

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

rename(.data, …**)** Rename columns. Use **rename_with()** to rename with a function.
rename(cars, distance = dist)

Tests for Association

Q: Is there an association between insurance product 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
```

Regression Modeling

Q: Is the association between test positivity and a government insurance product explained by the age of the patient?

```
tmp <- covid_testing_2 %>%
  filter(payor_group_fac %in% c("commercial", "government")) %>%
  mutate(result_fac = factor(result,
                             levels=c("negative", "positive"),
                             ordered=T),
         payor_group_fac = (payor_group == "government"))
tmp_fit <- glm(result_fac ~ payor_group_fac + age,     # model formula
               data = tmp,                             # dataset
               family = "binomial"                     # type of model
  )
summary(tmp_fit)
exp(coefficients(tmp_fit))                             # odds
```

Output for logistic regression

```
Call:
glm(formula = result_fac ~ payor_group_fac + age, family = "binomial",
    data = tmp)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.6365  -0.3468  -0.2532  -0.1985   2.8393

Coefficients:
                    Estimate Std. Error z value Pr(>|z|)
(Intercept)        -4.016195   0.119611 -33.577  < 2e-16 ***
payor_group_facTRUE  1.136566   0.128761   8.827  < 2e-16 ***
age                  0.032897   0.004436   7.416 1.21e-13 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 2666.6  on 7194  degrees of freedom
Residual deviance: 2535.2  on 7192  degrees of freedom
AIC: 2541.2

Number of Fisher Scoring iterations: 6

        (Intercept) payor_group_facTRUE                 age
         0.01802141          3.11604971          1.03344368
```

Odds for Payor Group and Age

Data Understanding: Advanced Reporting

Patrick Mathias

May 9, 2022

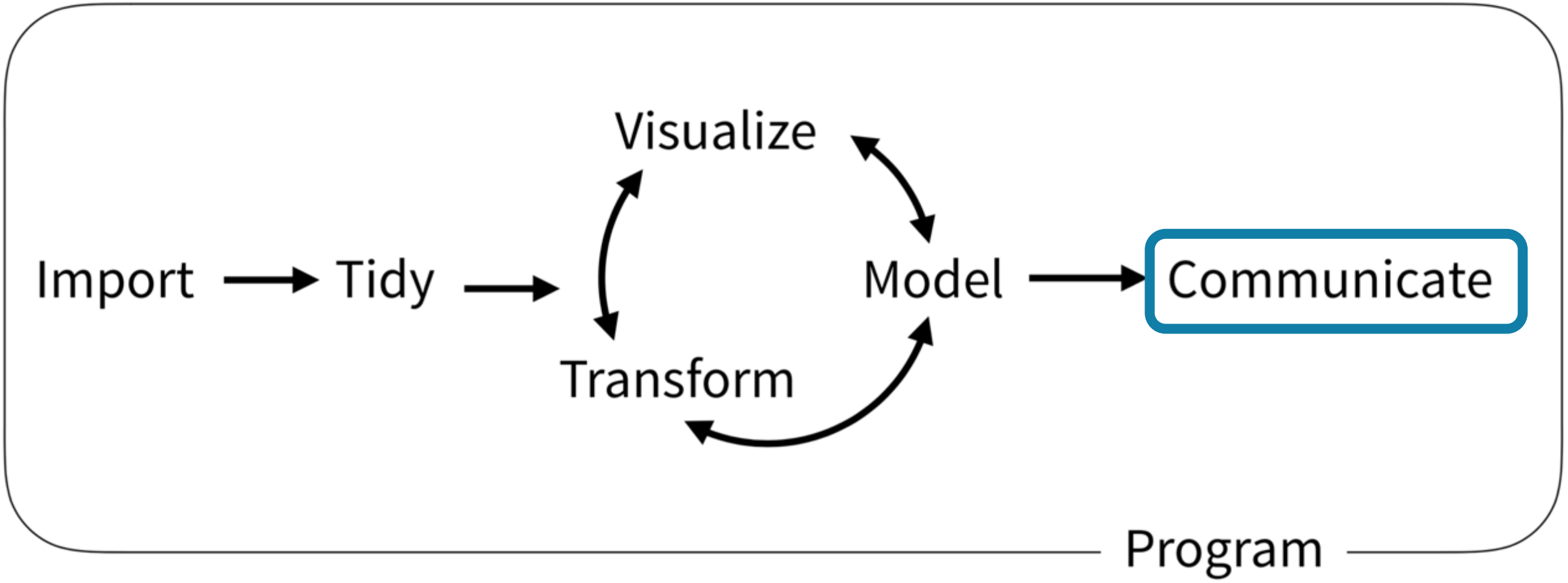
Goals

1. Build R Markdown reports using formatting outputs beyond standard document formats

Objectives

1. Format a flexdashboard to improve display of multiple plots
2. Convert a static plot into an interactive plot

Typical Data Science Pipeline



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

Refresher Quiz

You need to install 3 new packages you've never used before. What function do you run if you need to install the *flexdashboard*, *plotly*, and *DT* packages?

Refresher: Installing packages

- Installing a package

```
install.packages(c("flexdashboard", "plotly", "DT"))
```

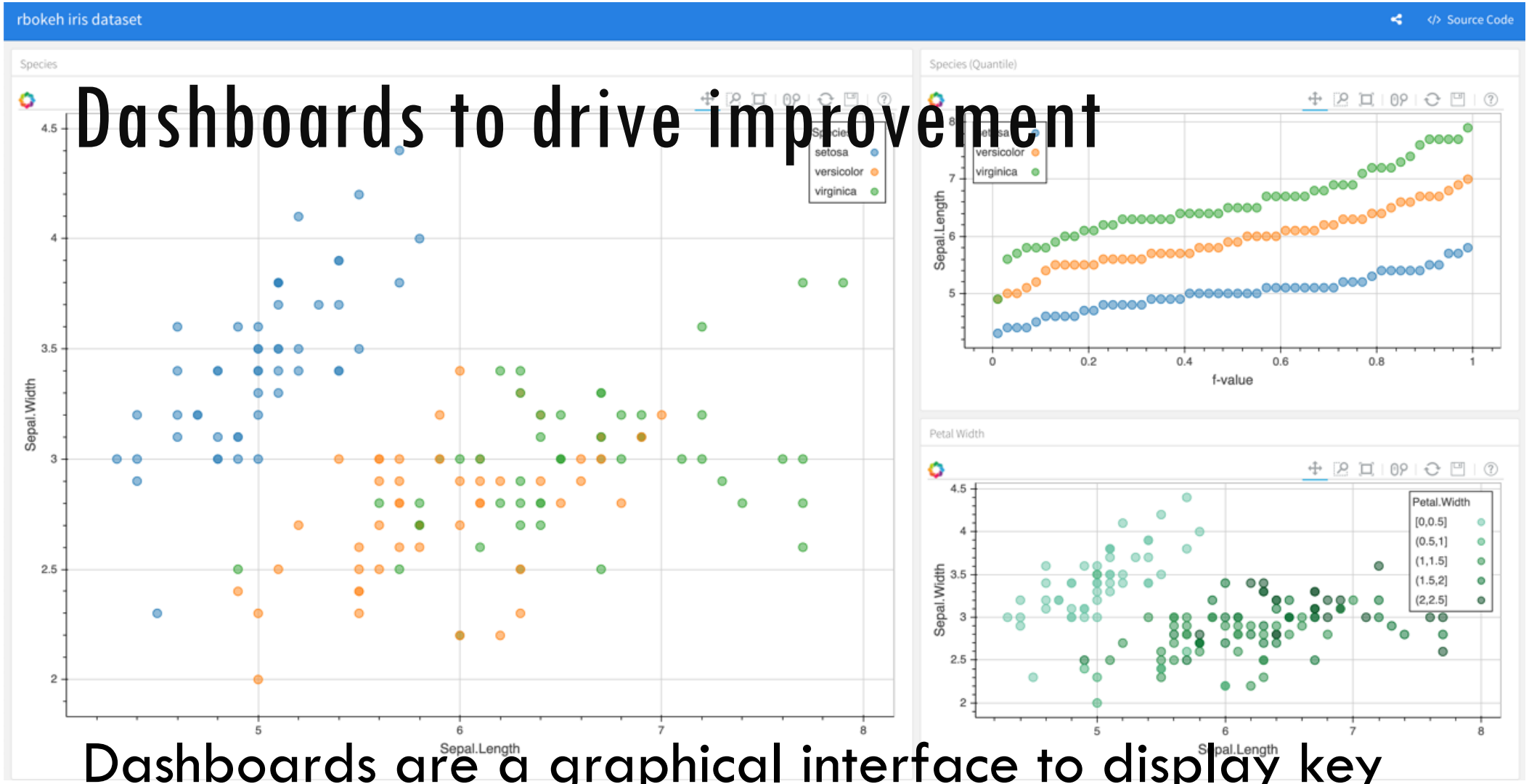
- Loading into your environment

```
library(flexdashboard)
library(plotly)
library(DT)
```

The flexdashboard and plotly packages were already installed in your Rstudio Cloud instance. To install them locally use install.packages.

From R Markdown to Quick and Painless Dashboards

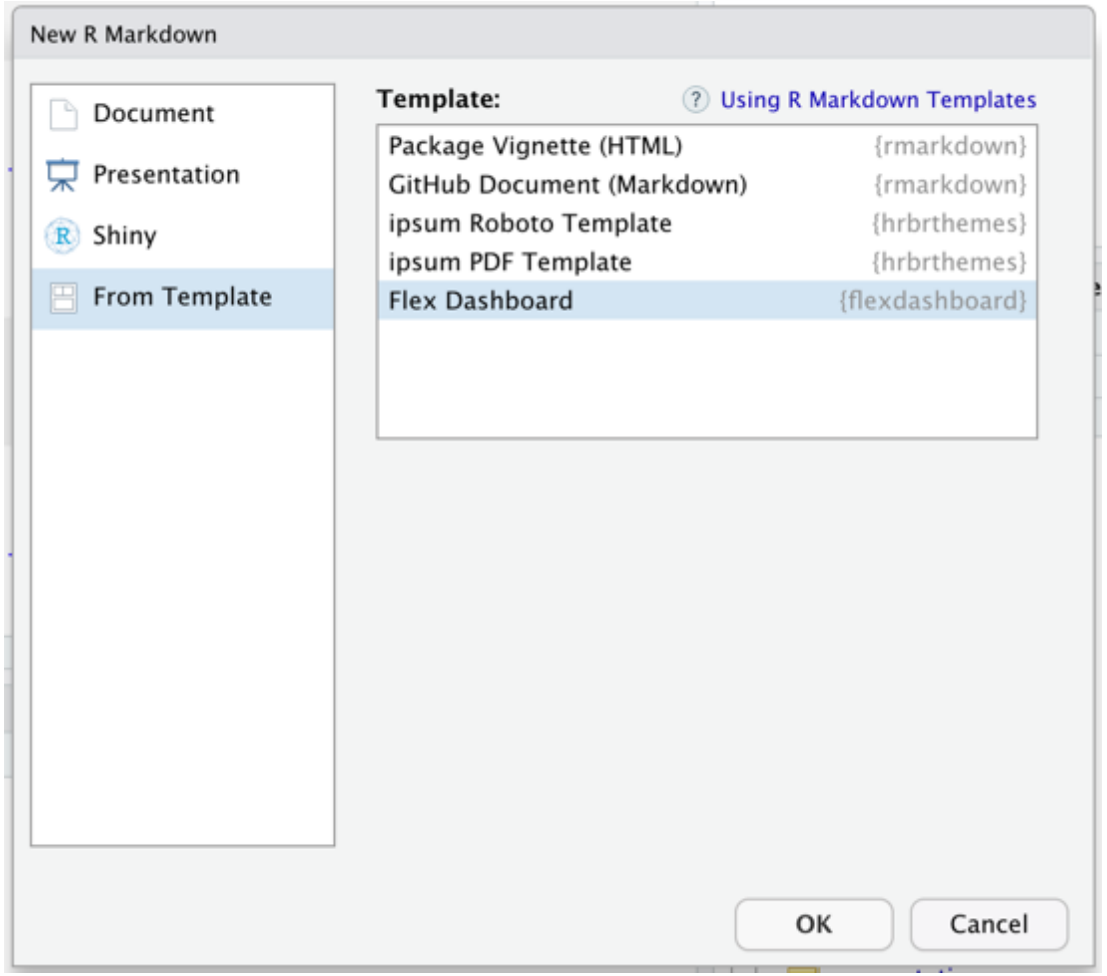
Dashboards to drive improvement



Dashboards are a graphical interface to display key performance indicators or other metrics

Intended to represent multiple pieces of information at a glance

flexdashboard provides easy dashboard templates for reporting



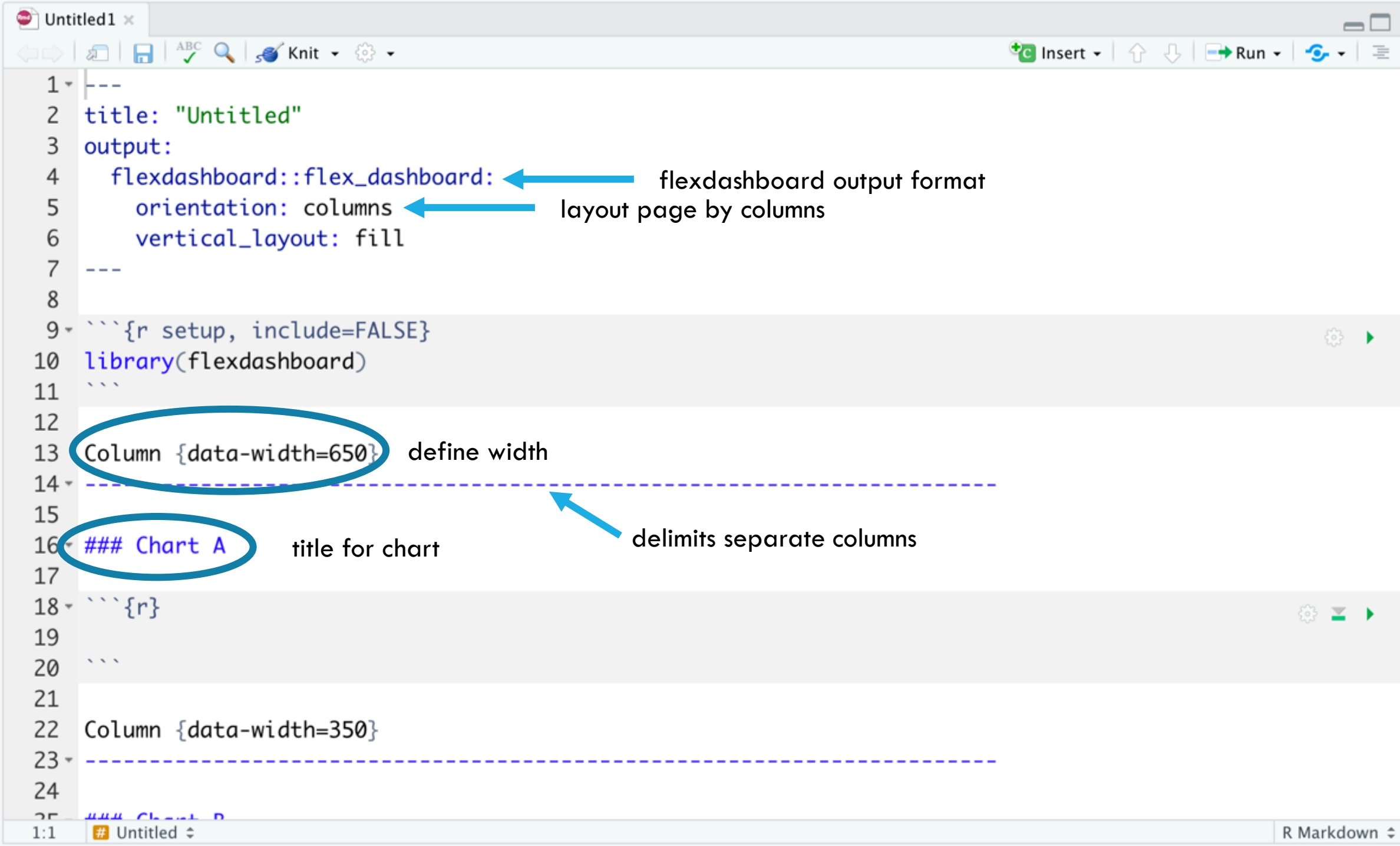
Produces HTML file that can be opened on web browsers

Or deployed on existing web server

Provides row or column based layouts

Get started on your desktop by running: install.packages("flexdashboard")

https://rmarkdown.rstudio.com/flexdashboard/


Untitled1
Knit
Insert
Run
---
title: "Untitled"
output:
flexdashboard::flex_dashboard:
orientation: columns
vertical_layout: fill

```{r setup, include=FALSE}
library(flexdashboard)
```
Column {data-width=650}

Chart A
```{r}
```
Column {data-width=350}
---
flexdashboard output format
layout page by columns
define width
delimits separate columns
title for chart
1:1
Untitled
R Markdown

```
---
title: "Column Orientation"
output: flexdashboard::flex_dashboard
---

Column
-------------------------------------

### Chart 1

```{r}
```

Column

Chart 2

```{r}
```

Chart 3

```{r}
```
```

```
---
title: "Row Orientation"
output:
  flexdashboard::flex_dashboard:
    orientation: rows
---

Row
-------------------------------------

### Chart 1

```{r}
```

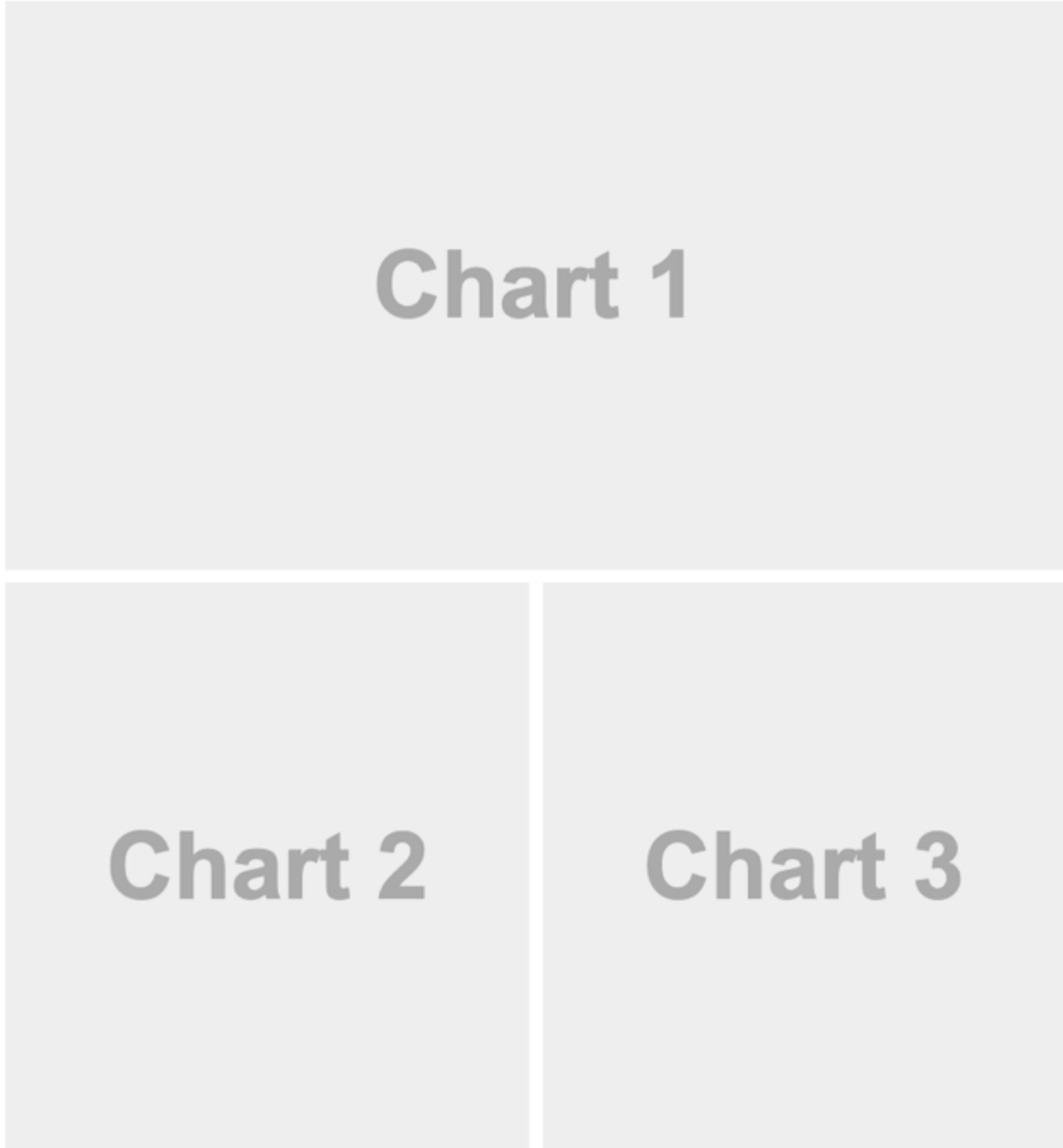
Row

Chart 2

```{r}
```

Chart 3

```{r}
```
```



```
---
title: "Chart Stack (Scrolling)"
output:
  flexdashboard::flex_dashboard:
    vertical_layout: scroll
---

### Chart 1

```{r}
```

Chart 2

```{r}
```

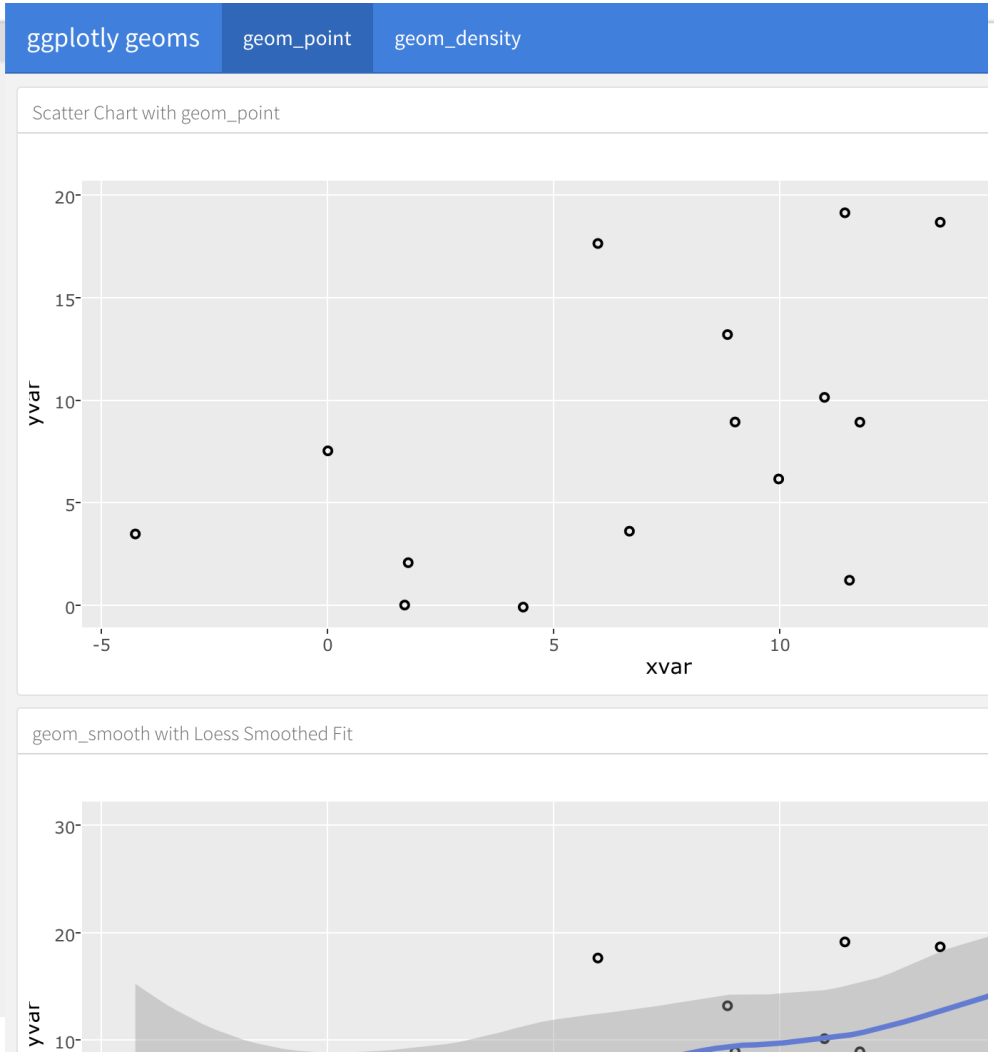
Chart 3

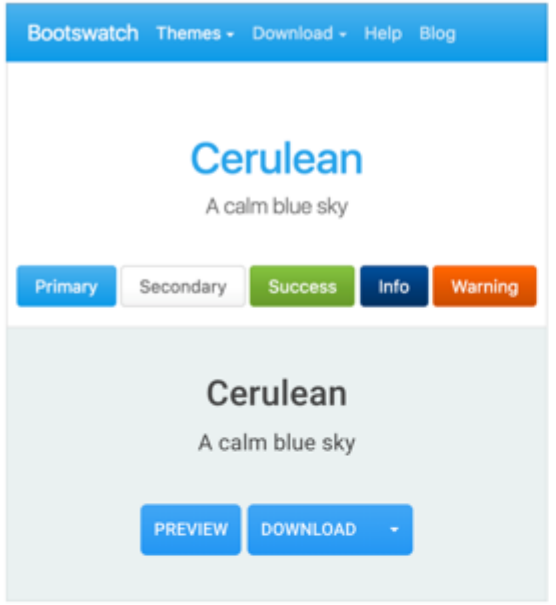
```{r}
```
```

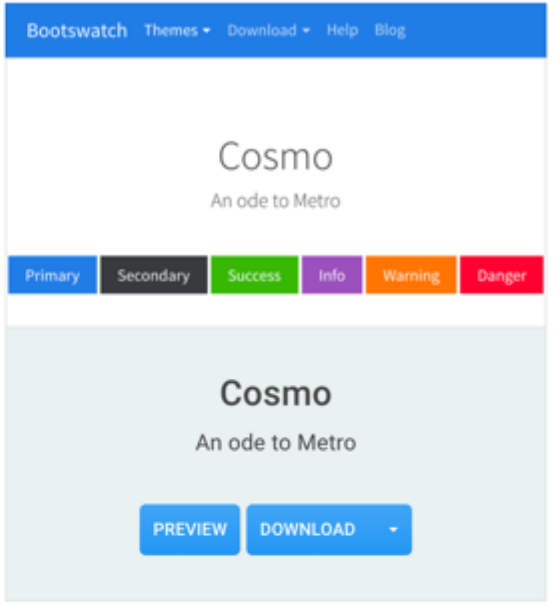
Your Turn #1

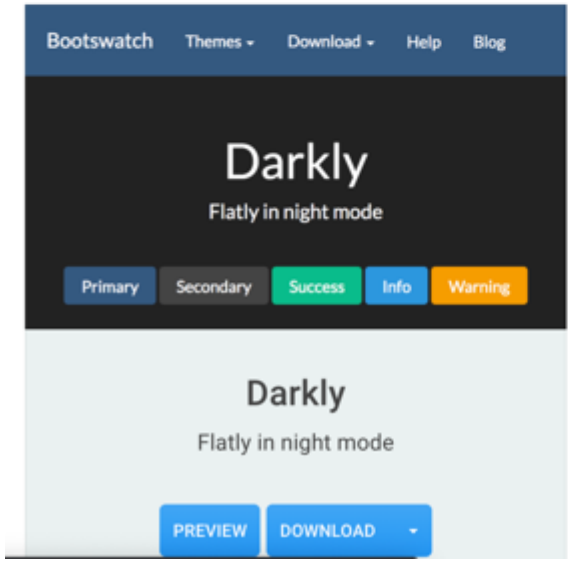
1. Open “06 - Advanced Reporting.Rmd” to work with a draft COVID-19 flexdashboard and run the setup chunk. Knit the document to see the dashboard output.
2. The ”Test Volumes Over Time” plot could show additional information regarding positive tests. Add fill to your barplot to show the result field in addition to overall test volume by day. Run that code chunk to see the output.
3. Too much information is crunched on the right side. Change the layout from columns to a row orientation. The 2nd and 3rd plots (Turnaround Times and Cycle Thresholds) should appear on the 2nd row.

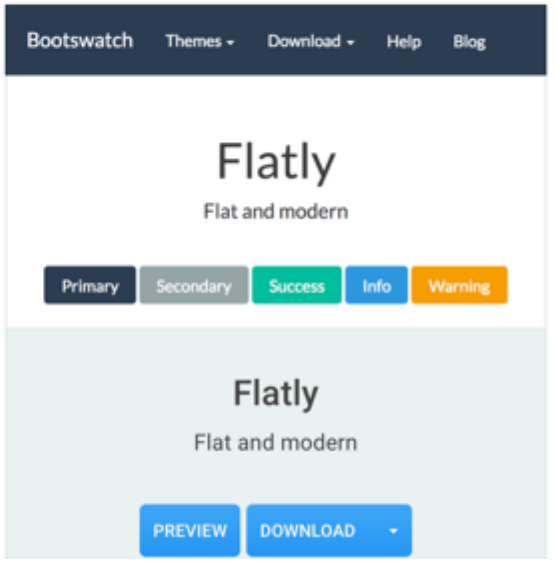
Customization











Making plots interactive

htmlwidgets for R support interactive visuals

Packages using htmlwidgets use R code to call Javascript visualization libraries (http://www.htmlwidgets.org/)

Use one line of code to convert a static plot into an interactive one

Plotly package converts ggplot with a simple command

To use Plotly on your desktop install the plotly package using the following command:

install.packages("plotly")

Examples of visualizations at Plotly website:
https://plotly.com/r/

Store plot as object and add one line to make interactive

```
plot_name <- ggplot(data = data_frame) +
  geom_function(mapping = aes(mappings))
ggplotly(plot_name)
```

Your Turn #2

1. Load the plotly package in your setup chunk
2. Convert each of the plots into an interactive plot by storing the ggplot in an object and using the ggplotly() function.
3. Knit the dashboard and hover over the interactive plots.

Other options for interactive plots

Other interactive plot packages:

- rbokeh
- Highcharter

Time series graphs with dygraphs package

Maps with leaflet package

Interactive tables with one line

DataTables library quickly converts tables into interactive element

DT package in R – to install use:

```
install.packages("DT")
```

Use `datatable()` function on a data frame to allow:

- Filter number of entries
- Search entries
- Sort by column

datatable example

```
datatable(head(iris), class = 'cell-border stripe')
```

Show 10 entries

Search:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

Showing 1 to 6 of 6 entries

Previous 1 Next

Your Turn #3

We are going to replace one of our panels of content with an interactive table.

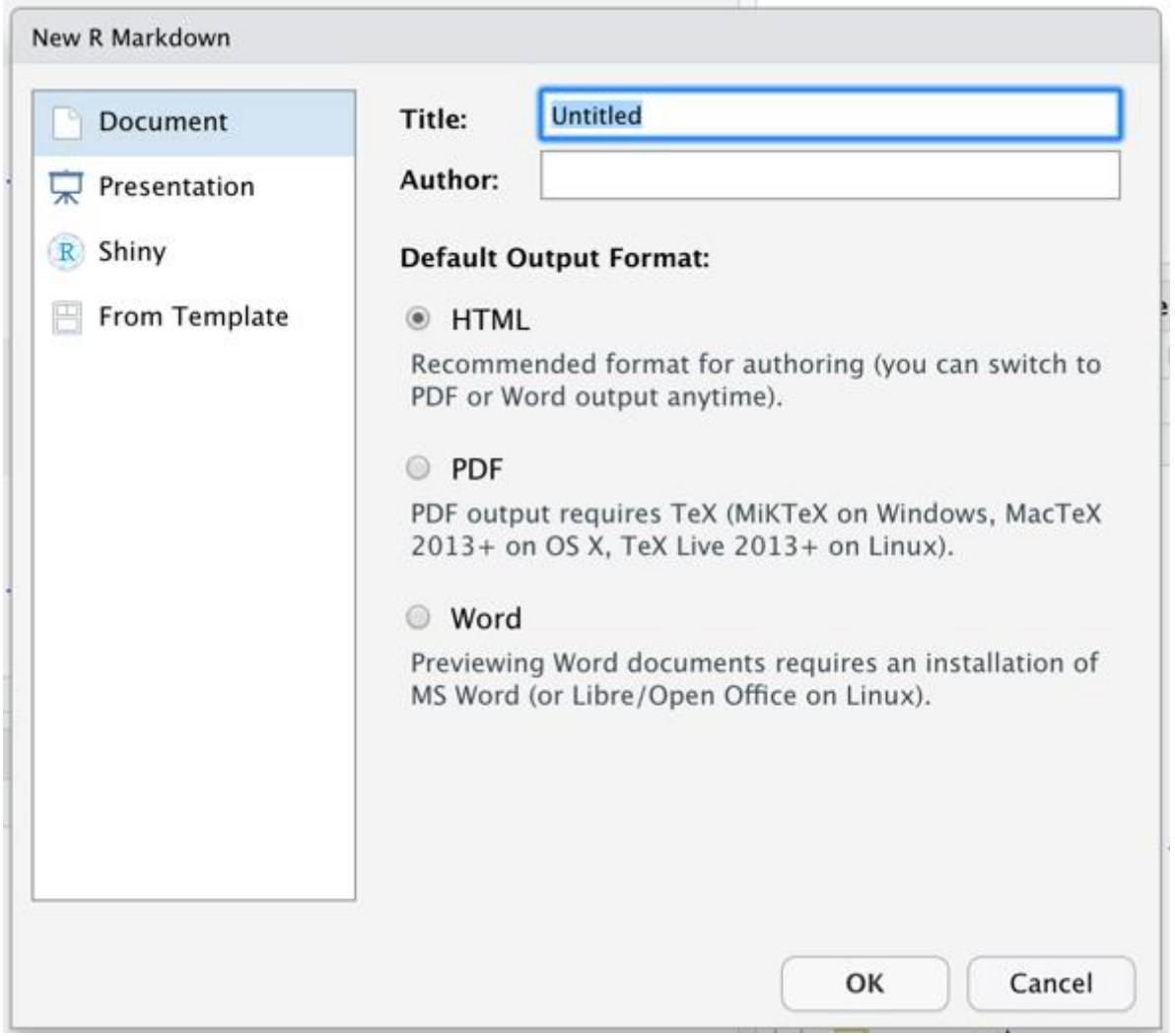
1. Load the DataTables package in your setup chunk
2. Rename the cycle threshold distribution panel to “Positive Result Details”. Use filter() to create a dataframe that only includes positive results.
3. Display an interactive table that includes the content from the positive result dataframe you created.
4. Knit the dashboard. Search “lannister” in the search box to confirm the interactive table is working.

Your Turn #4

Customize your dashboard. Use any of the data available in your covid testing dataset to generate new plots or tables that provide insight into the underlying data.

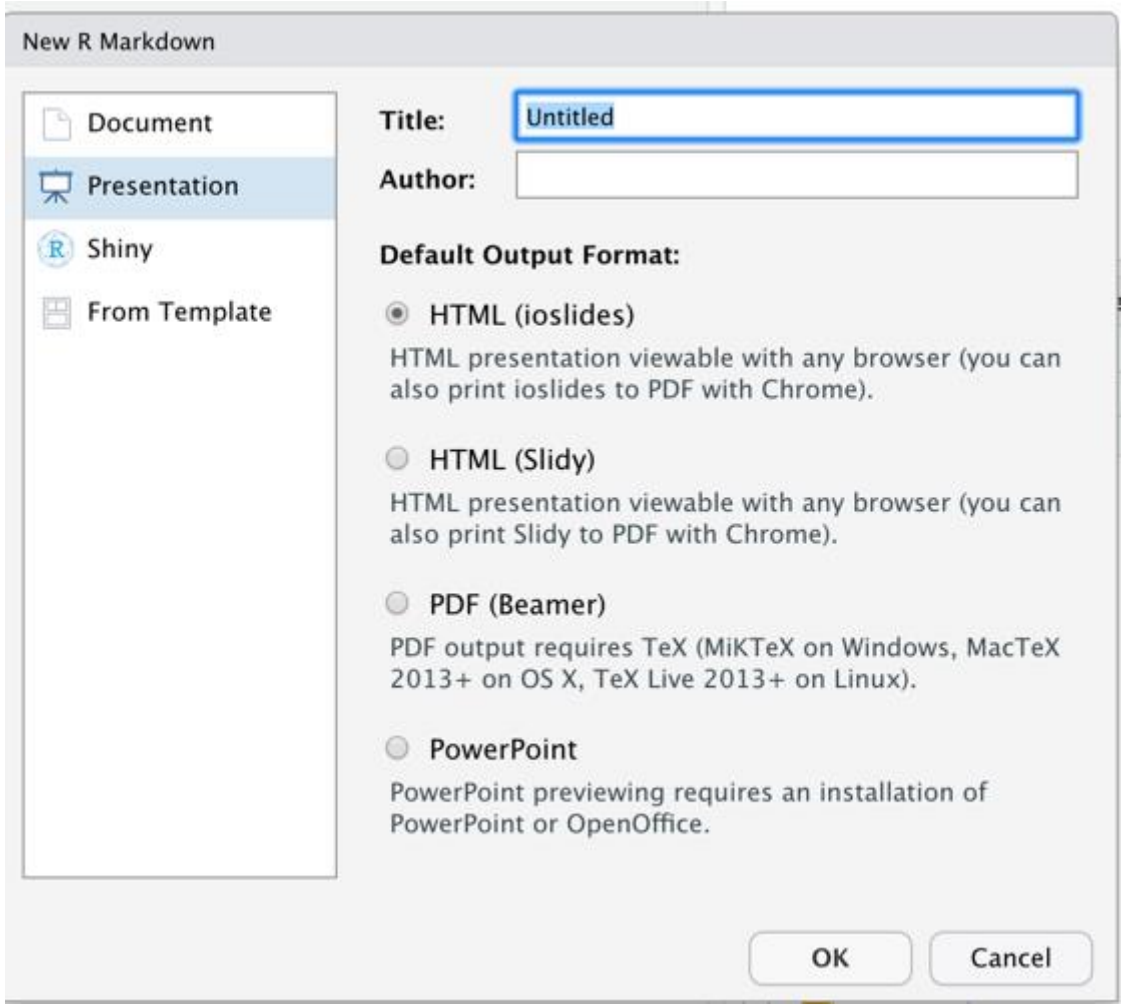
What Else?

Standard Markdown Reporting Formats



- HTML file - open with any web browser
- PDF – requires LaTeX dependencies
  - install.packages('tinytex')
  - tinytex::install_tinytex()
- Word – default format for collaborating with those who aren't familiar with R

Formats to go straight from code to slides



Multiple HTML formats create webpage that's advanceable like slides

PDF presentation uses LaTeX in the background

Powerpoint produces simple slides

README.md | Untitled1

Knit | Insert | Run

```
---
title: "Untitled"
output: powerpoint_presentation
---

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

R Markdown

This is an R Markdown presentation. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document.

Slide with Bullets

- Bullet 1
- Bullet 2
- Bullet 3

Slide with R Output

```{r cars, echo = TRUE}
summary(cars)
```

Slide with Plot

```{r pressure}
plot(pressure)
```
```

Output format

Each slide has its own header

R Markdown

This is an R Markdown presentation. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document.

Example output slide

Slide with Bullets

- Bullet 1
- Bullet 2
- Bullet 3

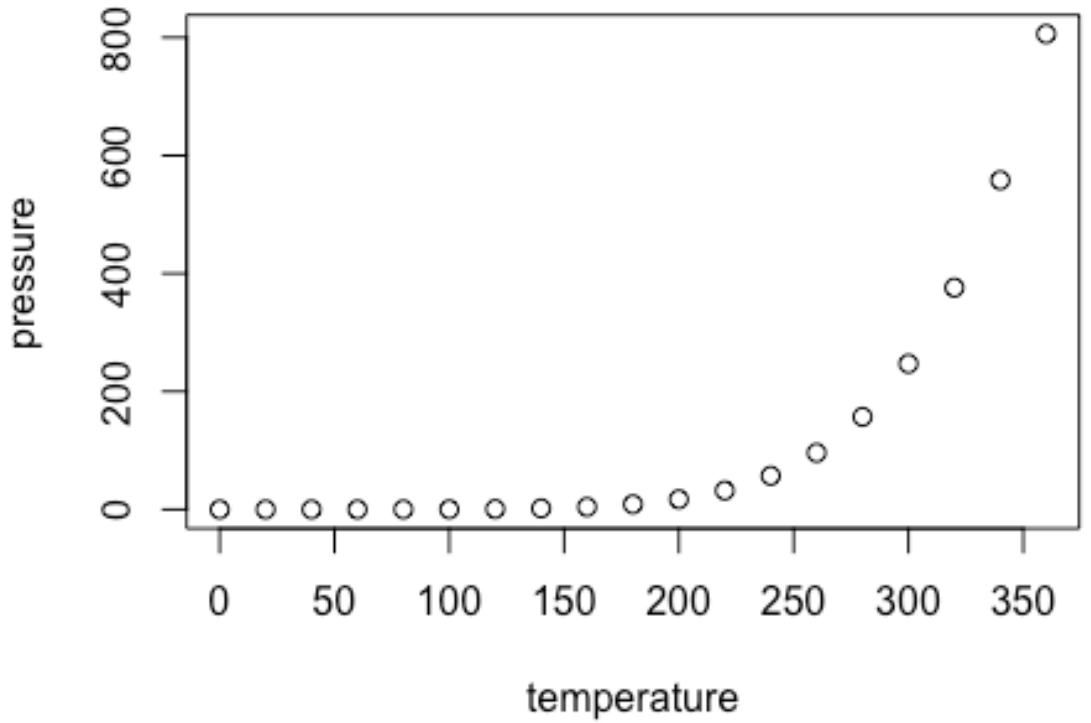
Example output slide

Slide with R Output

```
summary(cars)
##      speed           dist       
##  Min.   : 4.0   Min.   :  2.00  
##  1st Qu.:12.0   1st Qu.: 26.00  
##  Median :15.0   Median : 36.00  
##  Mean   :15.4   Mean   : 42.98  
##  3rd Qu.:19.0   3rd Qu.: 56.00  
##  Max.   :25.0   Max.   :120.00
```

Example output slide

Slide with Plot



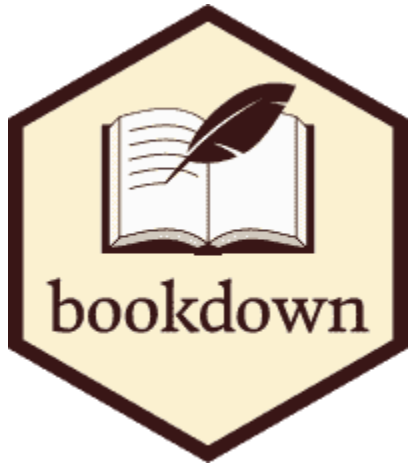
Example output slide

Books and longer documents also generated from R Markdown

Can generate printer ready books and ebooks

Supports LaTeX features such as equations

Generates blog formatted websites



https://github.com/rstudio/bookdown

https://bookdown.org/yihui/bookdown/

https://bookdown.org/yihui/blogdown/

Goals

1. Build R Markdown reports using formatting outputs beyond standard document formats

Objectives

1. Format a flexdashboard to improve display of multiple plots
2. Convert a static plot into an interactive plot

z

Appendix

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()**

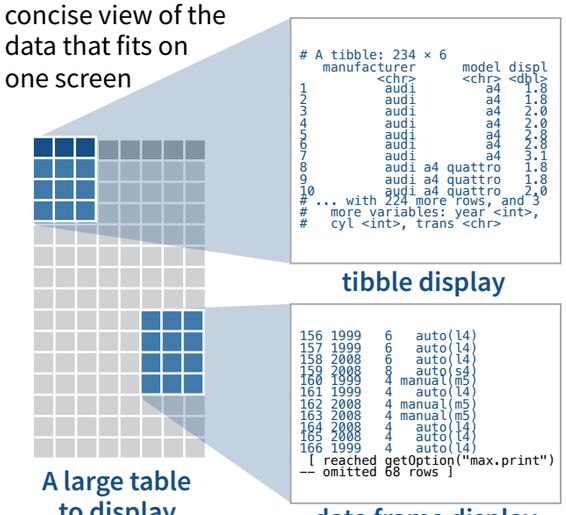
```
x$A <- parse_number(x$A)
```

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at tidyverse.org • readr 1.1.0 • tibble 1.2.12 • tidyr 0.6.0 • Updated: 2019–08

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

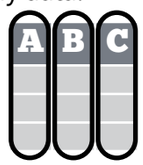
Tidy data is a way to organize tabular data. It provides a consistent data structure across packages.

A table is tidy if:

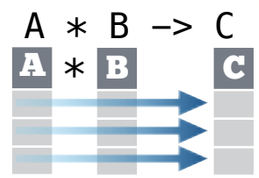
Each **variable** is in its own **column**

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 = "")

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at tidyverse.org • readr 1.1.0 • tibble 1.2.12 • tidyr 0.6.0 • Updated: 2019–08

R Markdown : : CHEAT SHEET

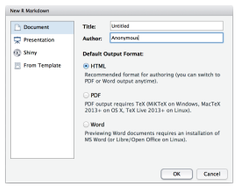
What is R Markdown?

.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.

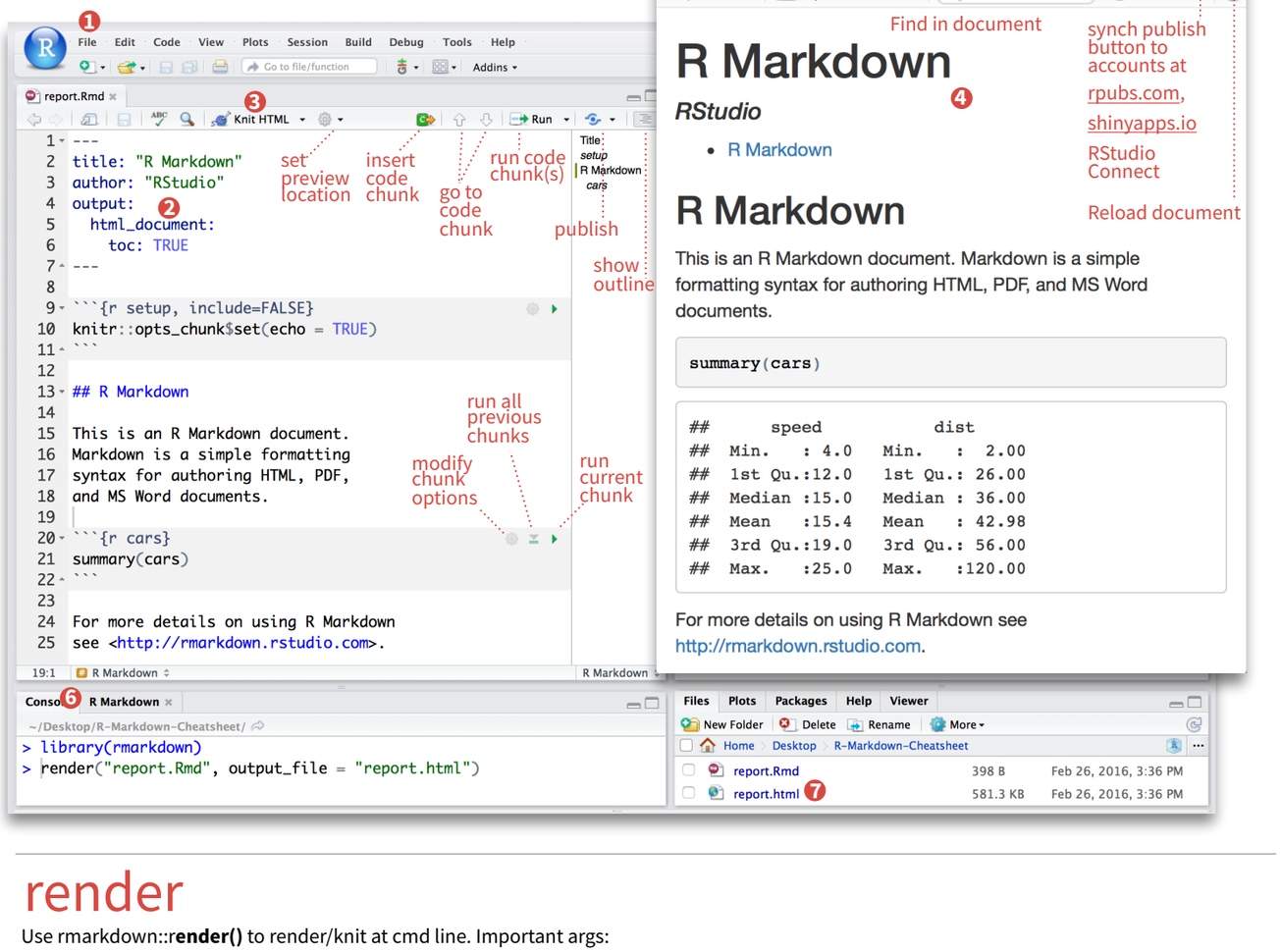
Reproducible Research • At the click of a button, or the type of a command, you can rerun the code in an R Markdown file to reproduce your work and export the results as a finished report.

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

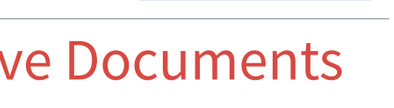
2. **Call parameters** • Call parameter values in code as params$<name>

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

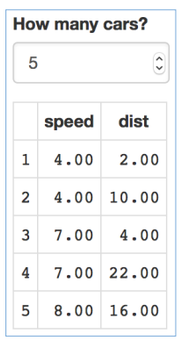
Turn your report into an interactive Shiny document in 4 steps

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

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at **rmarkdown.rstudio.com** • rmarkdown 1.6 • Updated: 2016-02

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
—
```

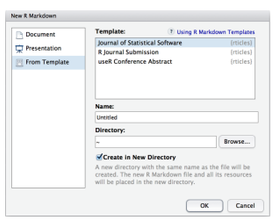


Table Suggestions

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

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

2. **Use citation keys in text**

```
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.
```

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at **rmarkdown.rstudio.com** • rmarkdown 1.6 • Updated: 2016-02

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**

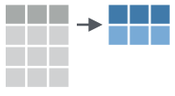
pipes

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))

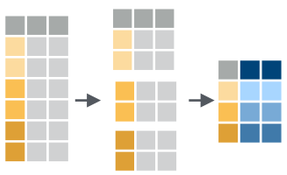
count(x, ..., wt = NULL, sort = FALSE)
Count number of rows in each group defined by the variables in … Also **tally**().
count(iris, Species)

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))
```

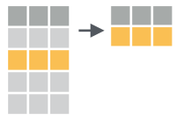
group_by(.data, ..., add = FALSE)
Returns copy of table grouped by …
g_iris <- group_by(iris, Species)

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

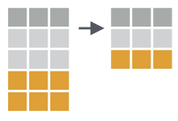
Manipulate Cases

EXTRACT CASES

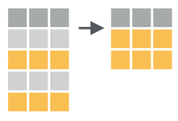
Row functions return a subset of rows as a new table.



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

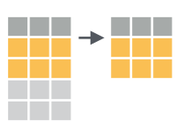


distinct(.data, ..., .keep_all = FALSE**)** Remove rows with duplicate values.
distinct(iris, Species)



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()

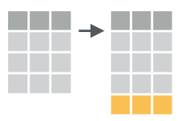
<	<=	is.na()	%in%	|	xor()
>	>=	!is.na()	!	&	

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)

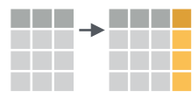
Use these helpers with select (),
e.g. select(iris, starts_with("Sepal"))

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

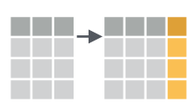


mutate(.data, …**)**
Compute new column(s).
mutate(mtcars, gpm = 1/mpg)

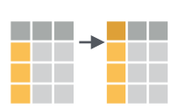
transmute(.data, …**)**
Compute new column(s), drop others.
transmute(mtcars, gpm = 1/mpg)

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.

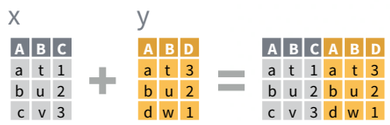
rownames_to_column()
Move row names into col.
a <- rownames_to_column(iris, var = "C")

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

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

Combine Tables

COMBINE VARIABLES



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.

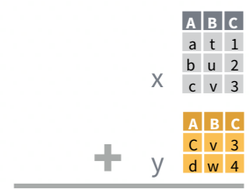
full_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),…**)**
Join data. Retain all values, all rows.

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



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)

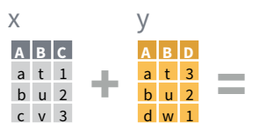
intersect(x, y, ...)
Rows that appear in both x and y.

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.

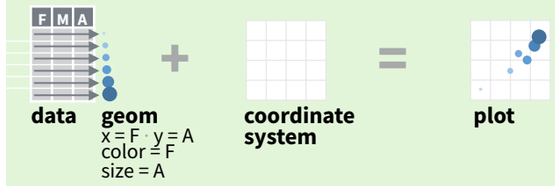
RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more with browseVignettes(package = c("dplyr", "tibble")) • dplyr 0.7.0 • tibble 1.2.0 • Updated: 2019-08

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

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at **http://ggplot2.tidyverse.org** • ggplot2 3.1.0 • Updated: 2018-12

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.

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)))

GENERAL PURPOSE SCALES

Use with most aesthetics

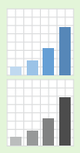
scale_*_continuous() - map cont' values to visual ones
scale_*_discrete() - map discrete values to visual ones
scale_*_identity() - use data values **as** visual ones
scale_*_manual(values = c()**)** - map discrete values to manually chosen visual ones
scale_*_date(date_labels = "%m/%d"), date_breaks = "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)

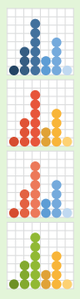


n <- d + geom_bar(aes(fill = fl)**)**

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..))

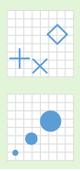
o + scale_fill_distiller(palette = "Blues"**)**

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)**)**

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

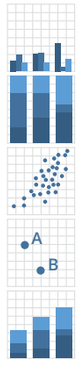
r + coord_polar(theta = "x", direction=1 **)**
theta, start, direction
Polar coordinates

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

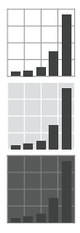
e + geom_point(position = "jitter")
Add random noise to X and Y position of each element to avoid overplotting

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)**)**

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at **http://ggplot2.tidyverse.org** • ggplot2 3.1.0 • Updated: 2018-12

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.