

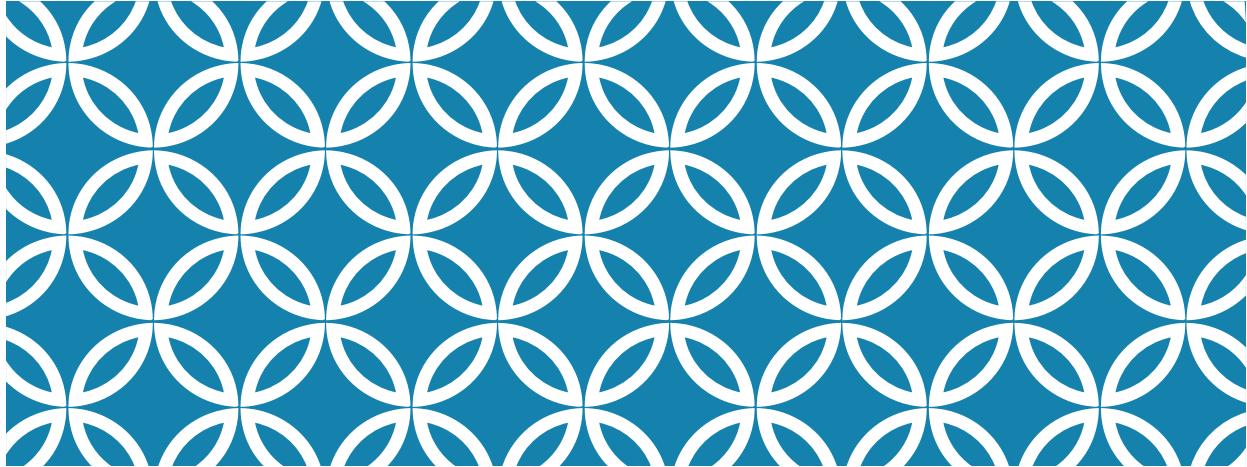
# CHOP R 101

INTRO TO R FOR CLINICAL DATA



Arcus Education

Children's Hospital  
of Philadelphia®



## CHOP R 101: Intro to R for Clinical Data

### Introduction



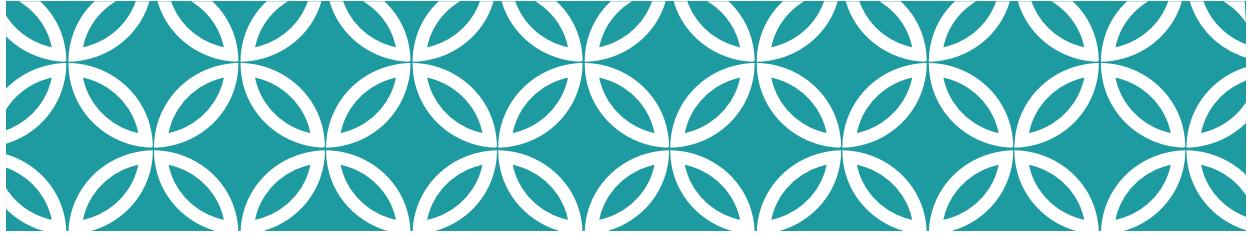
**R**  
Programming  
language for  
data analysis



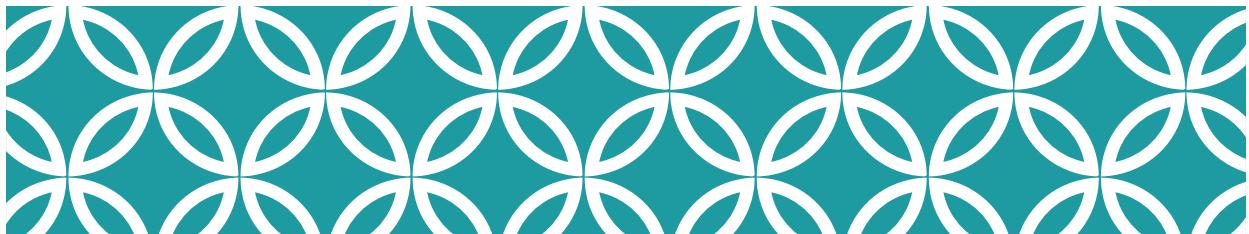
**RStudio**  
Integrated  
development  
environment (IDE)



**R Markdown**  
Computational  
document format



## Getting Started with RStudio



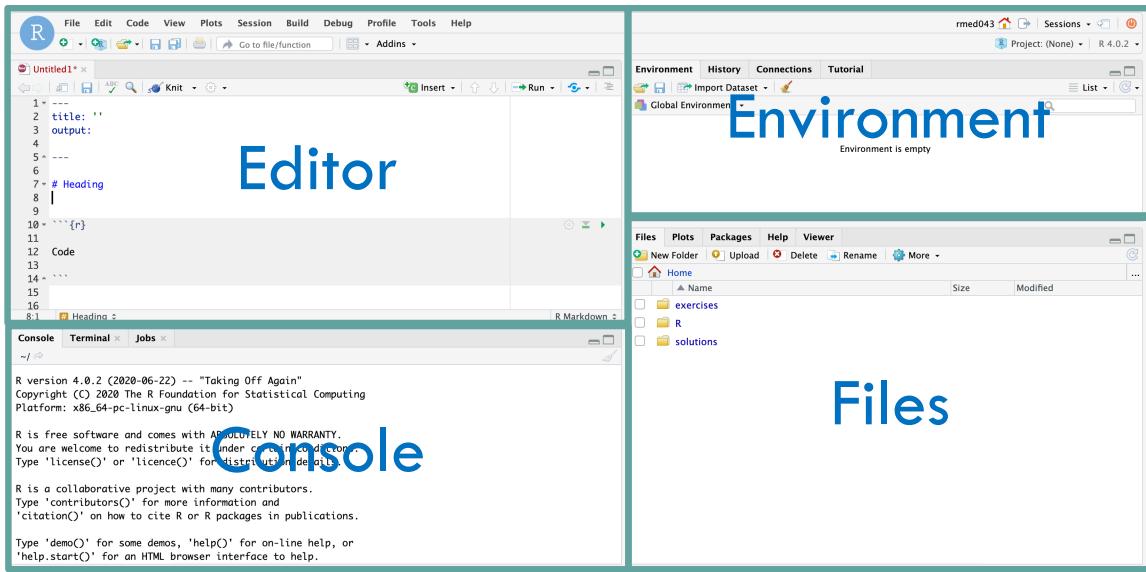
RStudio: On the Web and In Your Home



**RStudio Server**  
Hosted on a server  
(in the cloud)



**RStudio Desktop**  
Installed on your  
computer

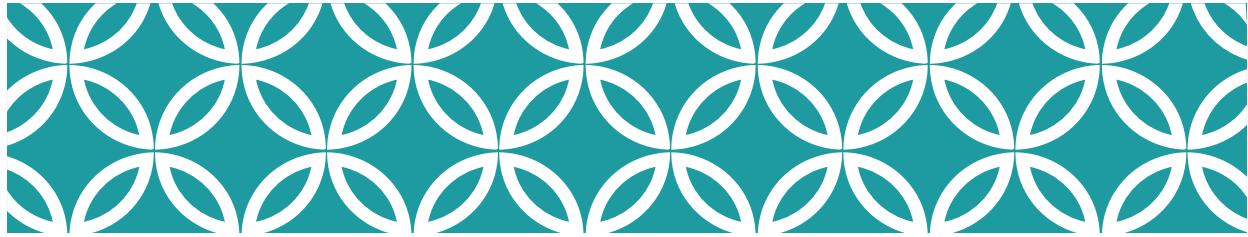


## Your Turn #1

Click the link in the chat to access the RStudio training environment.

Log in using your username and password.

Click “yes” once you see the RStudio panes.



## Reproducible Data Analysis and R Markdown



### The Duke Cancer Scandal

- ❖ Chemo sensitivity from microarrays
- ❖ Serious errors in data analysis
- ❖ Clinical trials based on flawed models
- ❖ Papers retracted, lawsuits settled



Duke

MD Anderson

"1881_at"	"1882_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



## 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!**



## Anatomy of an R Markdown document

```
1 --
2 title: 'My Markdown Document'
3 output: html_document
4 ---
5
6 # One Hashtag = Large Header
7
8 ## Two Hashtags = Smaller Header
9
10 Here is some text.
11
12 * It's easy to make a list
13 * Here is how you style text *cursive* or **bold**
14
15
16 ```{r}
17 x <- rnorm(100)
18 summary(x)
19 ```
20
```

Header

Text  
(with marks)

Code chunk

## Running a single code chunk

A screenshot of the RStudio interface. On the left, there is a code editor window containing the following R code:

```
16 ````{r}
17 x <- rnorm(100)
18 summary(x)
19 ````
```

To the right of the code editor is a preview pane displaying the output of the `summary` function:

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
x	-2.1028	-1.0127	-0.2316	-0.2374	0.4334	1.8401

A blue callout bubble with the text "run chunk" points to the green "Run" button in the toolbar above the preview pane.

## Rendering (“Knitting”) an R Markdown document

A screenshot of the RStudio interface demonstrating the knitting process. On the left, the code editor shows an R Markdown file with the following content:

```
1 ---  
2 title: 'My Markdown Document'  
3 output: html_document  
4 ---  
5  
6 # One Hashtag = Large Header  
7  
8 ## Two Hashtags = Smaller Header  
9  
10 Here is some text.  
11  
12 * It's easy to make a list  
13 * Here is how you style text cursive or bold  
14  
15  
16 ````{r}
17 x <- rnorm(100)
18 summary(x)
19 ````
```

The "Knit" button in the toolbar is highlighted with a green box. To the right, the preview pane shows the rendered document:

### 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**

At the bottom of the preview pane, the R code and its output are shown:

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

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

```

1 ---  

2 title: 'My Markdown Document'  

3 output: html_document  

4 ---  

5  

6 # One Hashtag = Large Header  

7  

8 ## Two Hashtags = Smaller Header  

9  

10 Here is some text.  

11  

12 * It's easy to make a list  

13 * Here is how you style text *cursive* or bold  

14

```

```

15  

16 ````{r}  

17 x <- rnorm(100)  

18 summary(x)  

19 ````  


```

```

20  

21 ## Including Plots|  

22  

23 ````{r, echo=FALSE}  

24 hist(x)  

25 ````  


```

## 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**

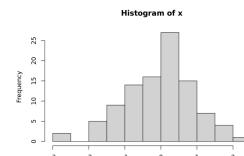
```

x <- rnorm(100)
summary(x)

##   Min. 1st Qu. Median Mean 3rd Qu. Max.
## -2.993204 -0.64726  0.14853 -0.02832  0.58218  2.07410

```

Including Plots



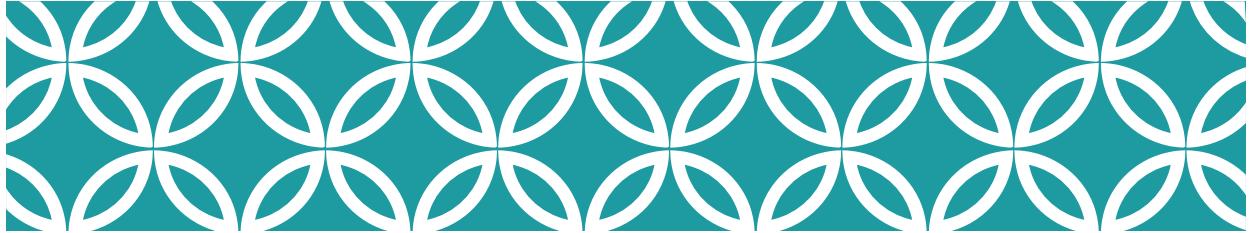
## Your Turn #2

Go to File > New File > R Markdown. Click OK.

Run each chunk by clicking . Note what happens.

Knit the document (). Type **test** and click **Save** to save the HTML file. Inspect the HTML document.

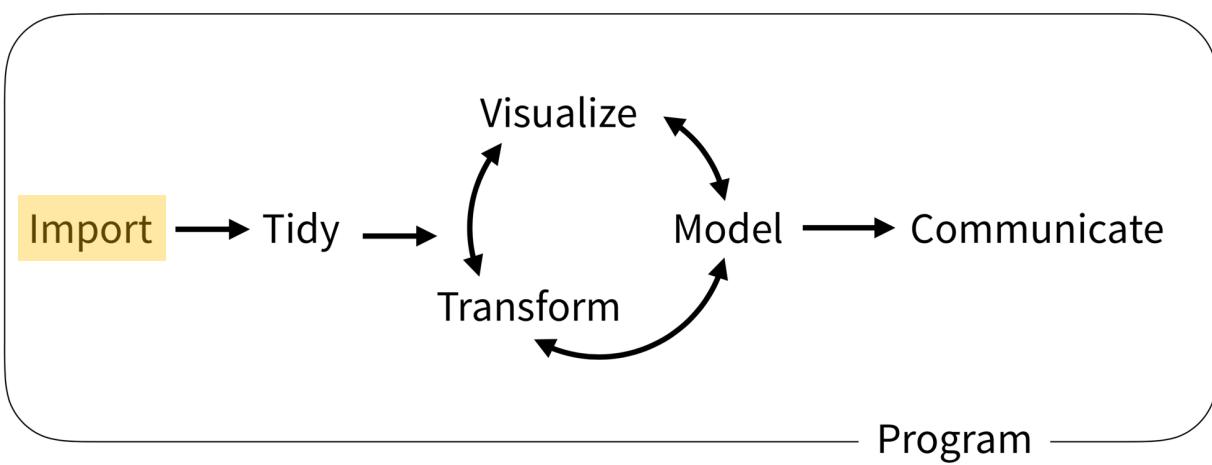




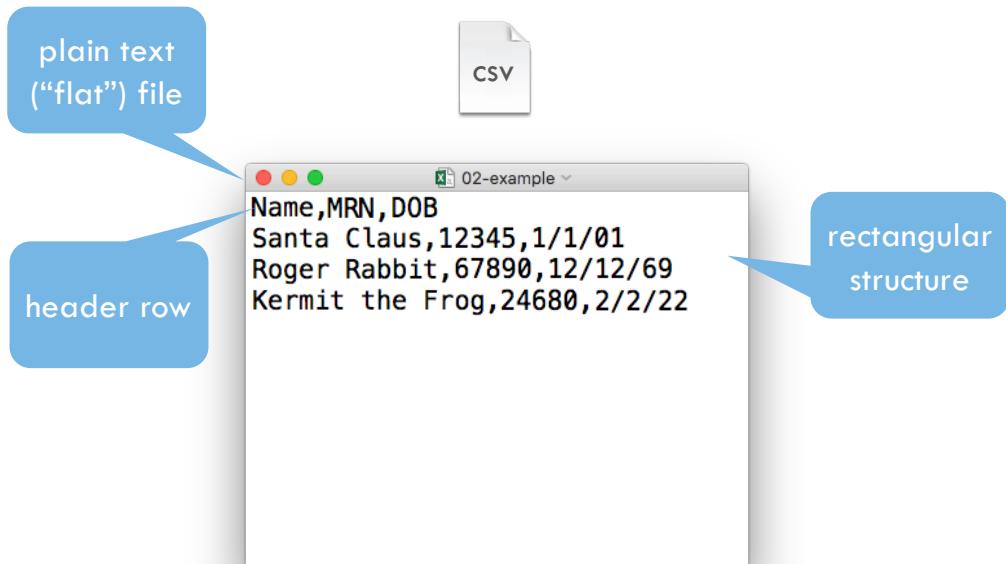
## Importing Data



## The Data Analysis Pipeline



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



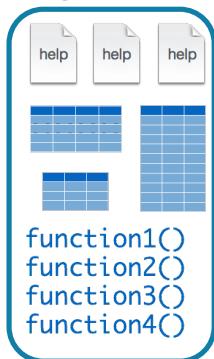
## Tidyverse: R Packages for Data Science

- 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**

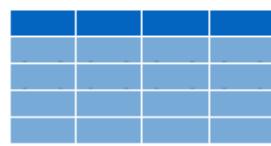
```
library("tidyverse")
```

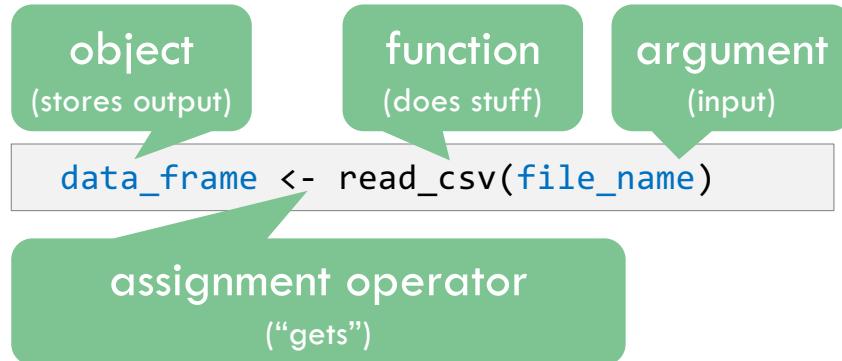
Loads package

**1 x per R Session**

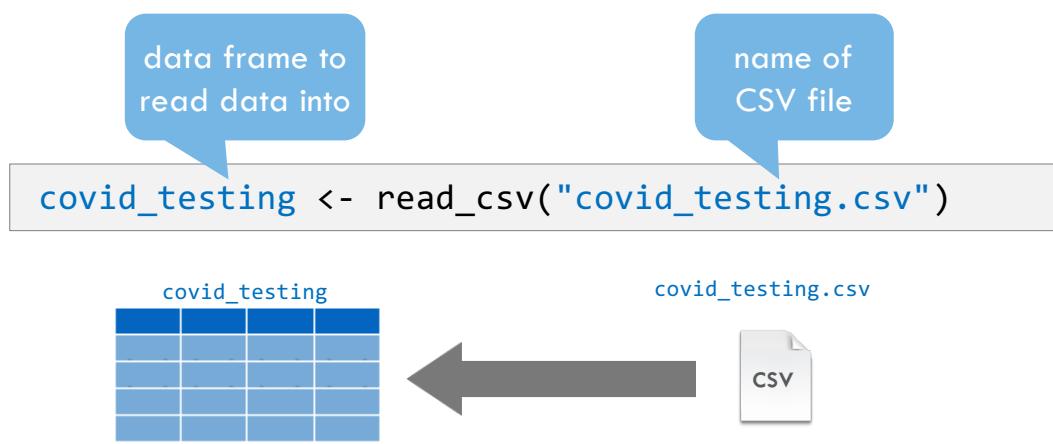
read\_csv()

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





## read\_csv()



# Your Turn #3

In the **Files** pane, click on the folder **exercises**.  
Open the R Markdown file titled

## 01 - Introduction.Rmd

Instructions for this exercise are in the text of the R Markdown document.

Click  when you are done.

05:00

## Recap



Programming Language

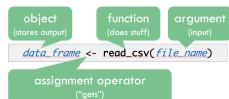


IDE (Editor)

Document Format



**Packages** extend the functionality of R. Install with `install.packages()` and load with `library()`.



**Functions** do stuff. They accept **Arguments** as input and return an **Output**. Capture an output in an **Object** using the **assignment operator** (`<-`).



**Importing Data** is the first step of data analysis. Use `read_csv()` from the `tidyverse` package to import data stored in a **CSV file**.

# What else?

<https://rstudio.com/resources/cheatsheets/>

## 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
- **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 = lappend)`

**File with arbitrary delimiter**  
`write_delim(x, path, delim = " ", na = "NA",`

### 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

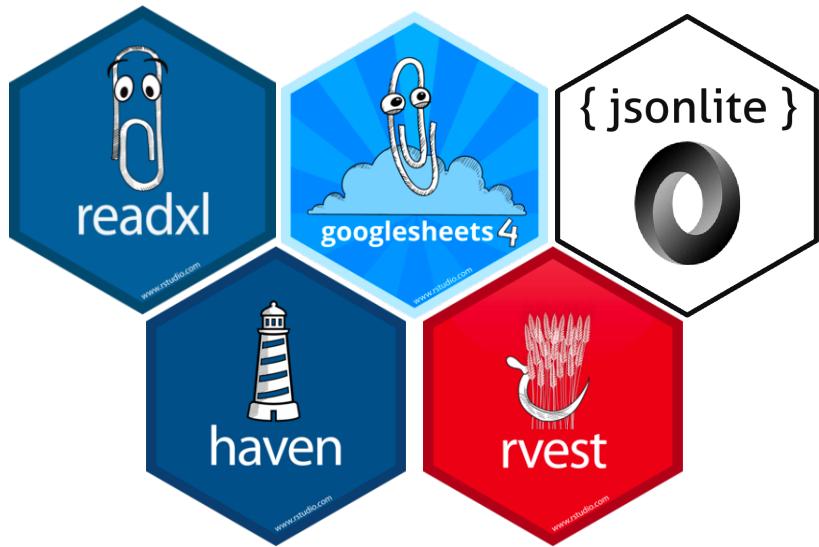
a,b,c  
1,2,3  
4,5,NA

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

a b c  
1 2 3  
4 5 NA

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

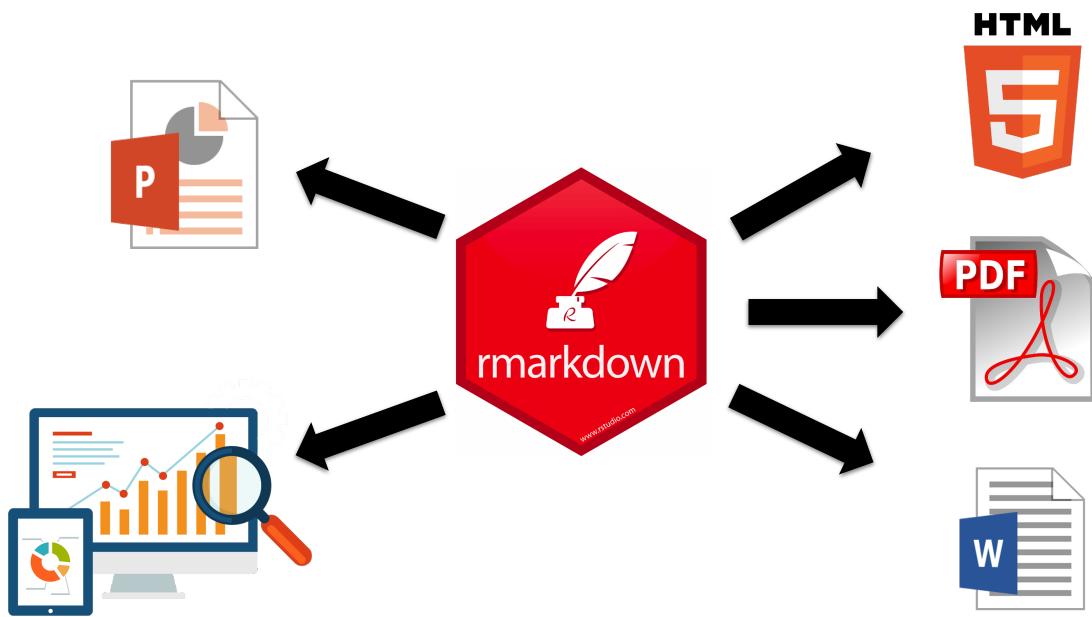
a,b,c  
1,2



## Databases



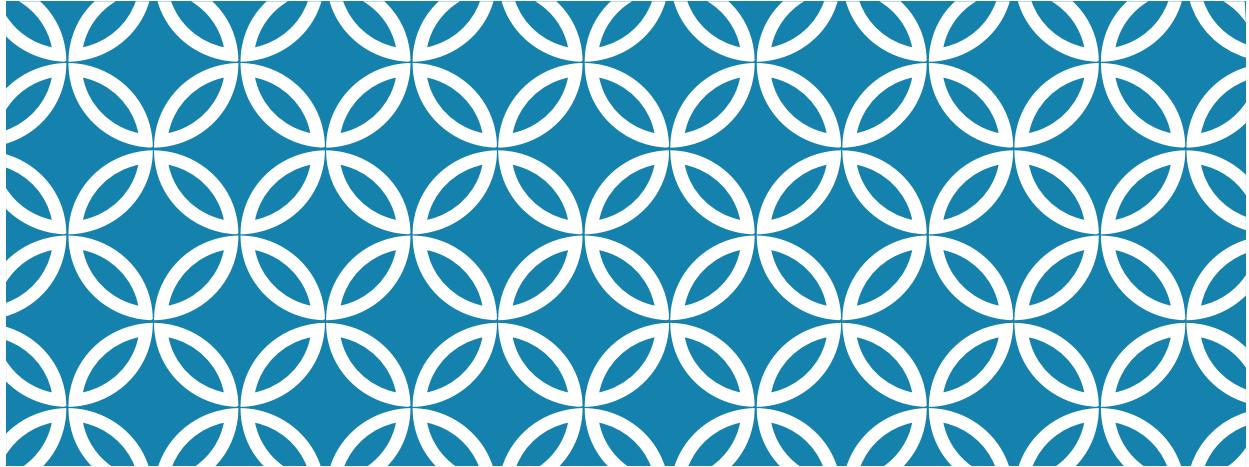
<https://db.rstudio.com/databases/>



## R Interface to Python



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



# CHOP R 101

## Intro to R for Clinical Data

# Visualize

covid\_testing

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

## Your Turn #1

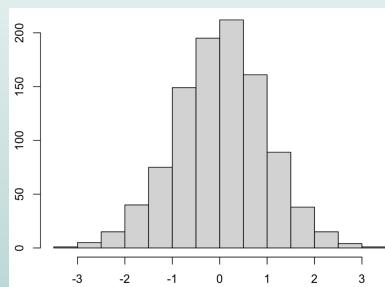
Consider the `covid_testing` data frame.

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?

01:00

## Your Turn #2



What is the name of this kind of plot?

Type the answer into the chat!

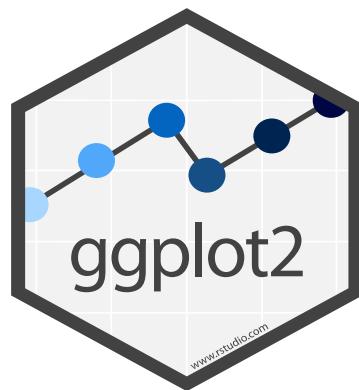
## Your Turn #3

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))
```

01:00



# 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**

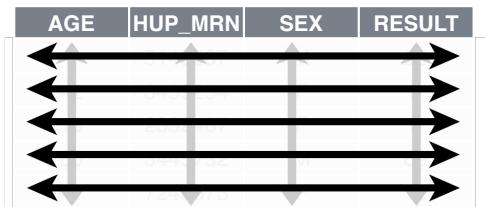
```
ggplot(data = data_frame) +  
  geom_function(mapping = aes(mappings))
```

2. Pick a “**geom**”  
function

3. Write aesthetic  
**mappings**

## 1. Pick a “Tidy” Data Frame

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.

## 2. Pick a “Geom” Function



`geom_histogram()`



`geom_dotplot()`



`geom_bar()`



`geom_boxplot()`



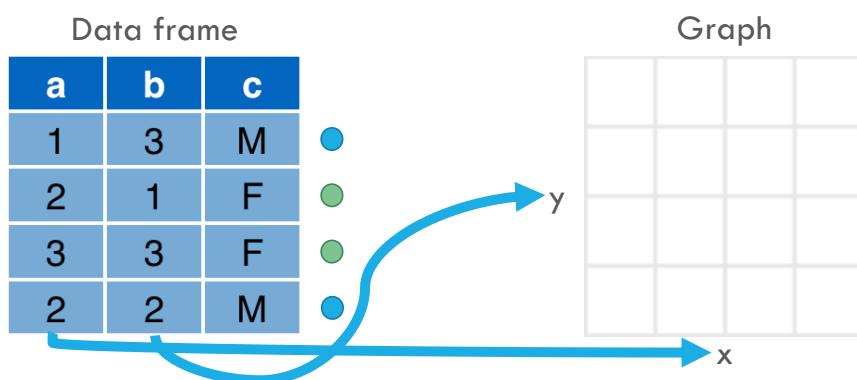
`geom_point()`



`geom_line()`

### 3. Write Aesthetic Mappings

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



### Your Turn #4



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

Type your answers in the chat!

## Your Turn #5

Open 02 - Visualize.Rmd. Work through the exercises of the section titled “Your Turn 5”.

**Stop** when it says “**Stop Here**”.

Click “yes” when you’re done!

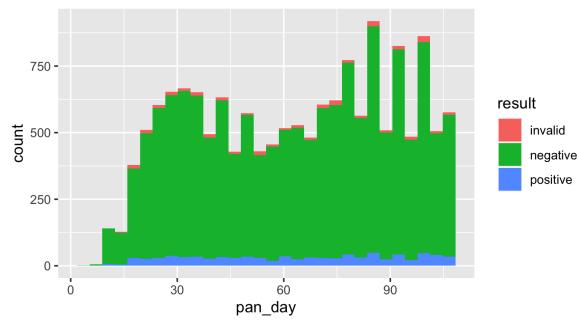
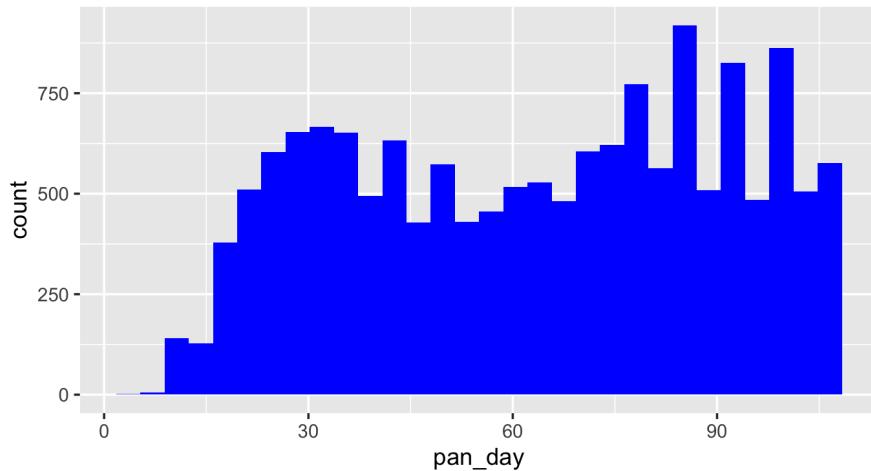
05:00



## Setting vs Mapping Aesthetics

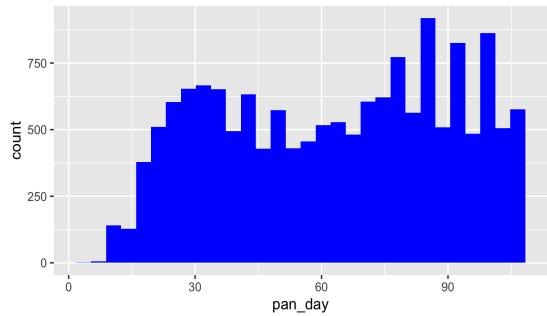


How would you make this plot?



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

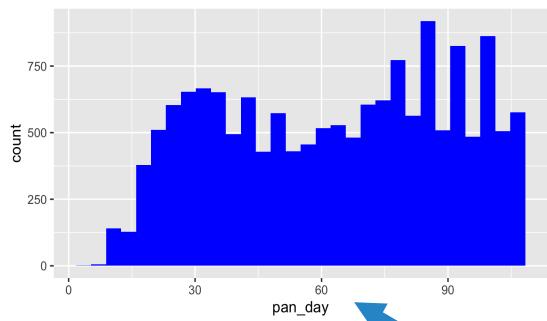
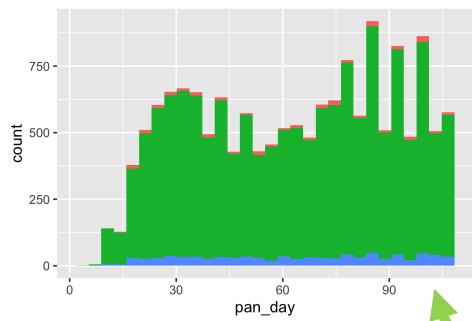
```
ggplot(data = covid_testing) +  
  geom_histogram(mapping = aes(x = pan_day, fill = result))
```



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

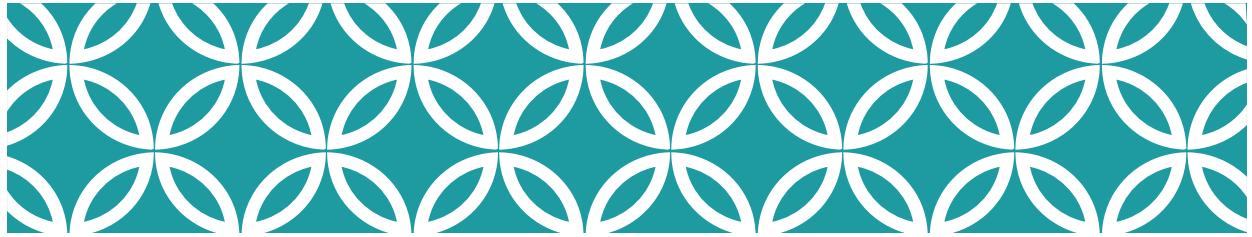
color name in  
“quotes”

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

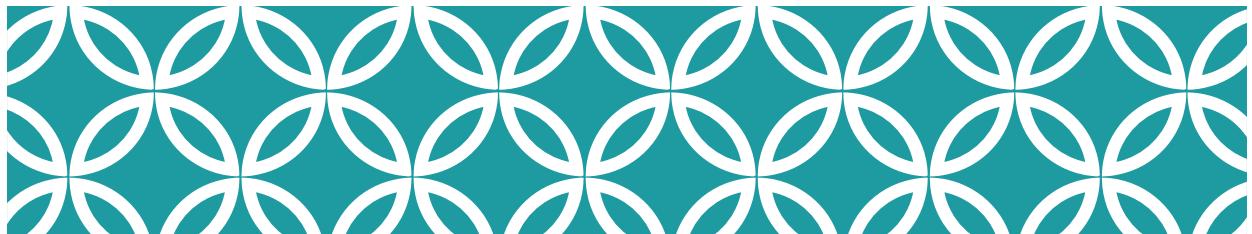


```
ggplot(data = covid_testing) +  
  geom_histogram(mapping = aes(x = pan_day, fill = result))
```

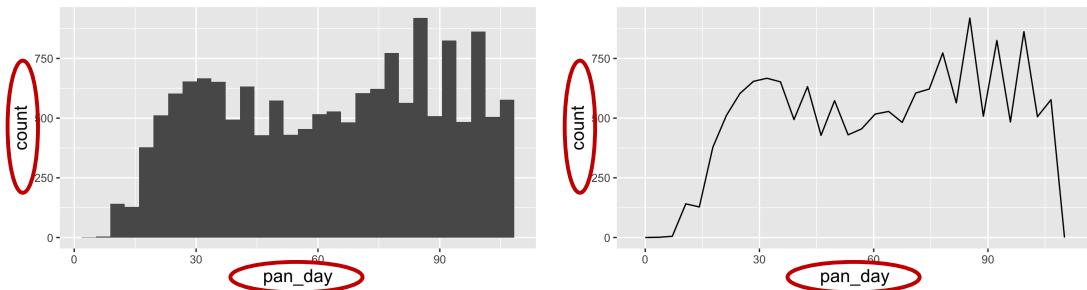
```
ggplot(data = covid_testing) +  
  geom_histogram(mapping = aes(x = pan_day), fill = "blue")
```



## Geom Functions

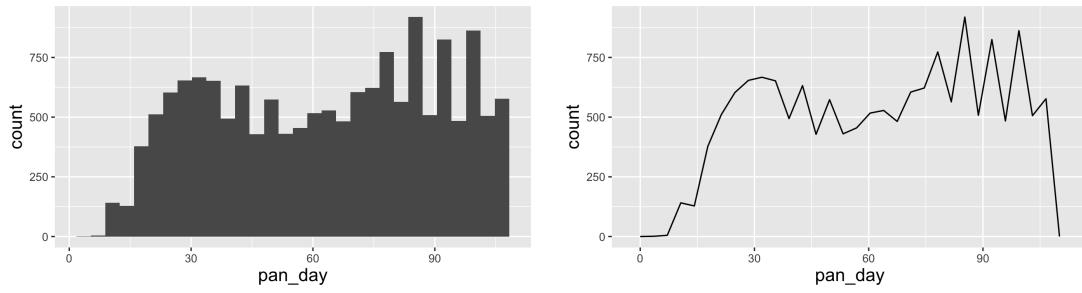


How are these plots similar?



Same: x axis, y axis, data

How are these plots different?



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

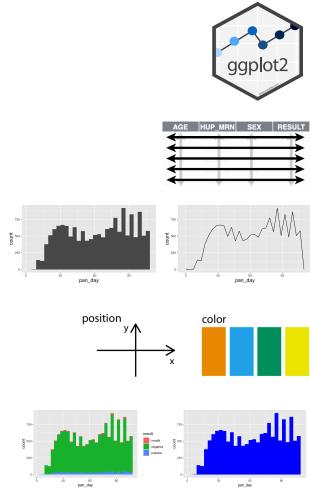
## Your Turn #6

Return to [02 - Visualize.Rmd](#). Work through the exercises of the section titled “Your Turn 6.”

Click “yes” when you’re done!

05:00

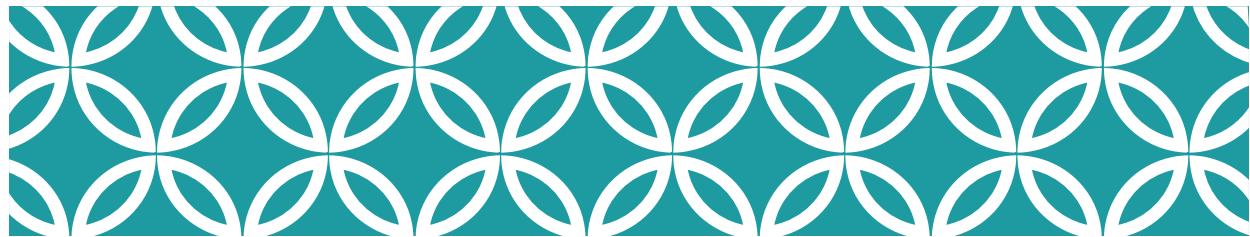
## Recap



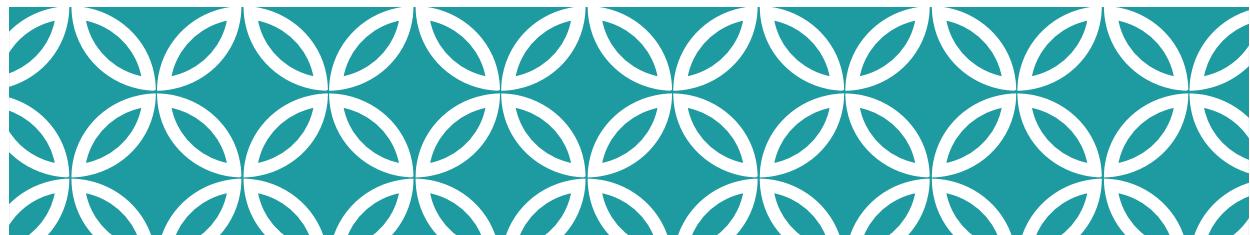
**ggplot2** is a package that provides a **grammar of graphics**. You can create **any type of plot** using a simple template to which you provide:

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.

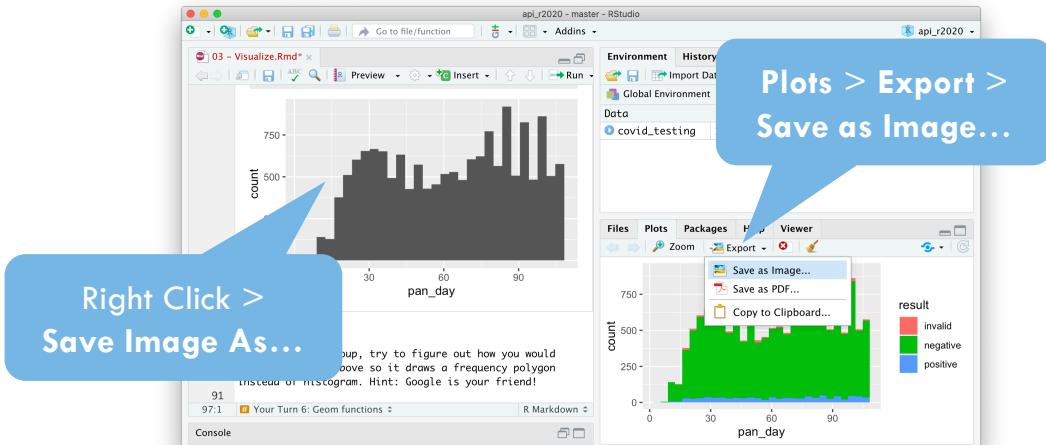
Aesthetics can be **mapped** to a variable or **set** to a constant value.



## What Else?

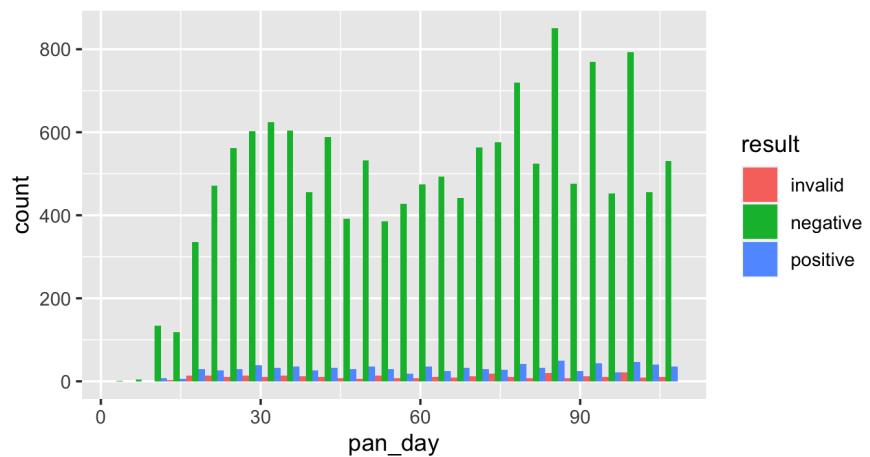


# Manually saving plots



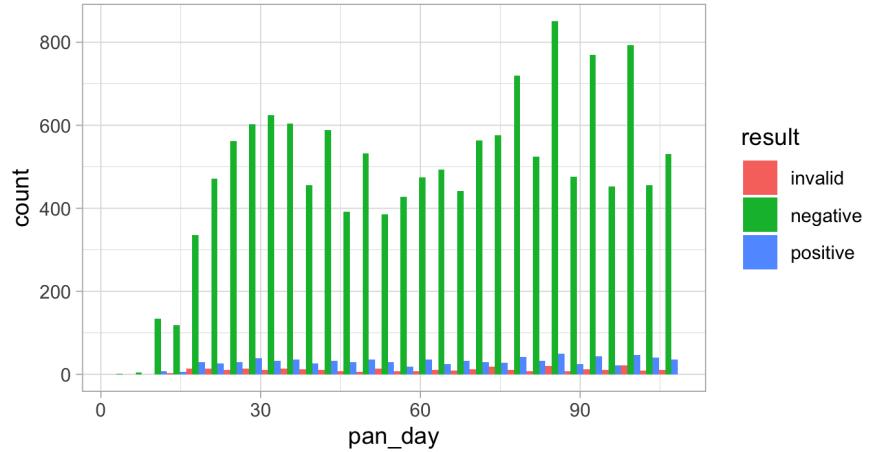
## Position adjustments

How overlapping objects are arranged



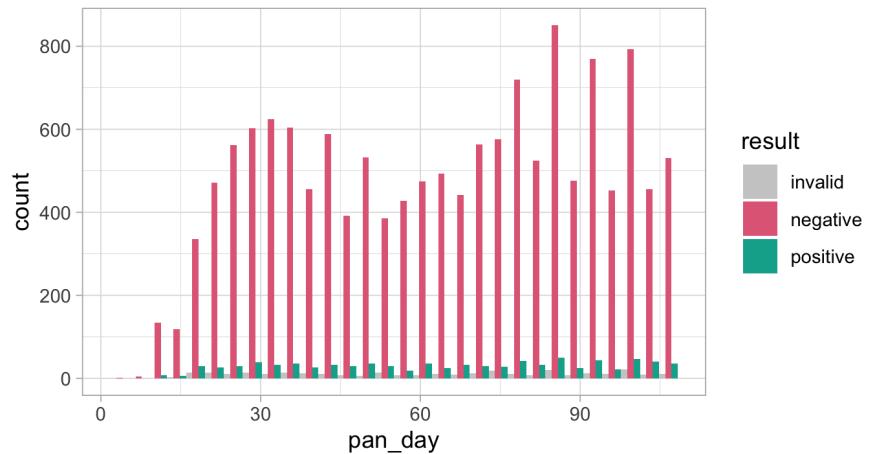
# Themes

## Visual appearance of non-data elements



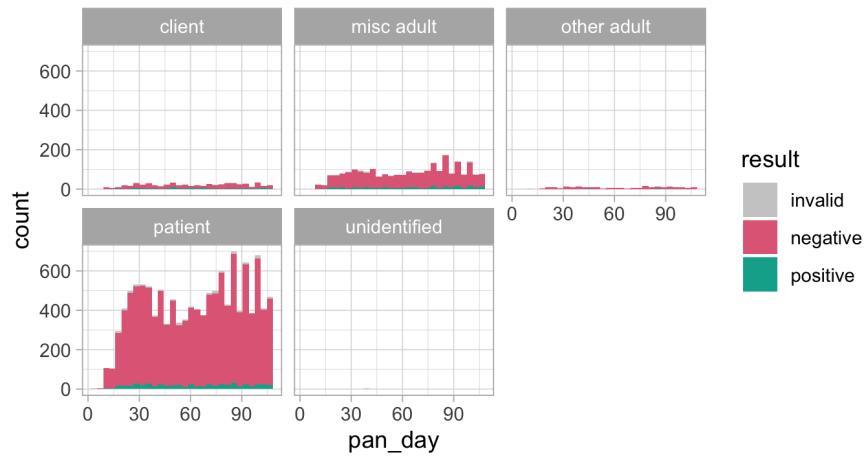
## Scales

## Customize color scales and other mappings

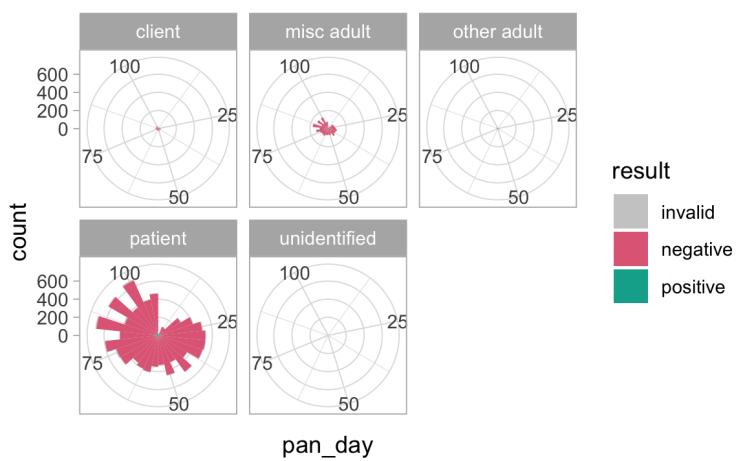


# Facets

Subplots that display subsets of the data



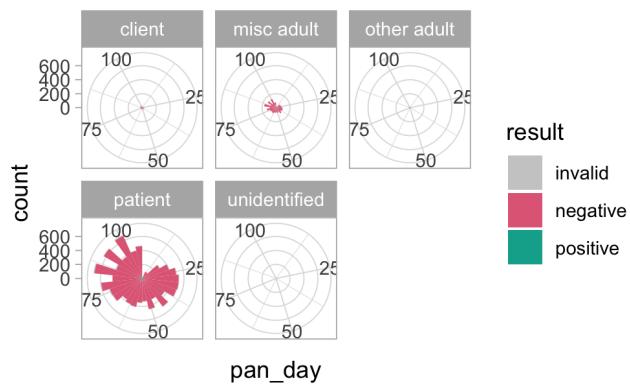
# Coordinate systems



# Titles and captions

COVID19 test volume

Displayed in polar coordinates, mostly to show off ggplot2



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

Required

Optional

# Data Visualization with ggplot2 :: CHEAT SHEET

## Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph by combining three main 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>) +
  geom_function(mapping = aes(<MAPPINGS>),
  stat = <STAT>, position = <POSITION>) +
  <COORDINATE_FUNCTION>+
  <FACET_FUNCTION>+
  <SCALE_FUNCTION>+
  <THEME_FUNCTION>
```

required

Not required, optional parameters and defaults supplied

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

`geom_point(aes(x = cyl, y = hwy, data = mpg, geom = "point"))`

## Geoms

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

### GRAPHICAL PRIMITIVES

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

a + geom\_blank()  
(useful for expanding limits)

b + geom\_curve(aes(latency = lat + 1, x = long, xend = long, y = yend, alpha, angle, color, curvature, linetype, size, weight))

a + geom\_path(linend = "butt", linejoin = "round", lineorder = 1)

a + geom\_polygon(aes(group = group))  
x, y, alpha, color, group, linetype, size

b + geom\_rect(aes(xmin = x, xmax = x + 1, ymin = y, ymax = y + 1), xname = xmin, yname = ymin, alpha, 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\_vline(aes(intercept = lat))

b + geom\_segment(aes(yend = lat + 1, xend = long + 1))

b + geom\_spoke(aes(angle = 1:115, radius = 1))

### ONE VARIABLE continuous

c < ggplot(mpg, aes(hwy)); c + geom\_line()

c + geom\_area(stat = "bin")

c + geom\_density(kernel = "gaussian")

c + geom\_dotplot(binaxis = "y", stackdir = "center")

c + geom\_hex()

c + geom\_linerange()

c + geom\_pointrange()

c + geom\_rect()

c + geom\_rug()

c + geom\_smooth(method = lm)

c + geom\_text(aes(label = cyl))

c + geom\_violin(scale = "area")

c + geom\_density()

c + geom\_hex()

c + geom\_linerange()

c + geom\_pointrange()

c + geom\_rect()

c + geom\_rug()

c + geom\_smooth()

c + geom\_text()

c + geom\_violin()

c + geom\_hex()

c + geom\_linerange()

c + geom\_pointrange()

c + geom\_rect()

c + geom\_rug()

c + geom\_smooth()

c + geom\_text()

c + geom\_violin()

c + geom\_hex()

c + geom\_linerange()

c + geom\_pointrange()

c + geom\_rect()

c + geom\_rug()

c + geom\_smooth()

c + geom\_text()

c + geom\_violin()

c + geom\_hex()

c + geom\_linerange()

c + geom\_pointrange()

c + geom\_rect()

c + geom\_rug()

c + geom\_smooth()

c + geom\_text()

c + geom\_violin()

c + geom\_hex()

c + geom\_linerange()

c + geom\_pointrange()

c + geom\_rect()

c + geom\_rug()

c + geom\_smooth()

c + geom\_text()

c + geom\_violin()

c + geom\_hex()

c + geom\_linerange()

c + geom\_pointrange()

c + geom\_rect()

c + geom\_rug()

c + geom\_smooth()

c + geom\_text()

c + geom\_violin()

c + geom\_hex()

c + geom\_linerange()

c + geom\_pointrange()

c + geom\_rect()

c + geom\_rug()

c + geom\_smooth()

c + geom\_text()

c + geom\_violin()

c + geom\_hex()

c + geom\_linerange()

c + geom\_pointrange()

c + geom\_rect()

c + geom\_rug()

c + geom\_smooth()

c + geom\_text()

c + geom\_violin()

c + geom\_hex()

c + geom\_linerange()

c + geom\_pointrange()

c + geom\_rect()

c + geom\_rug()

c + geom\_smooth()

c + geom\_text()

c + geom\_violin()

c + geom\_hex()

c + geom\_linerange()

c + geom\_pointrange()

c + geom\_rect()

c + geom\_rug()

c + geom\_smooth()

c + geom\_text()

c + geom\_violin()

c + geom\_hex()

c + geom\_linerange()

c + geom\_pointrange()

c + geom\_rect()

c + geom\_rug()

c + geom\_smooth()

c + geom\_text()

c + geom\_violin()

c + geom\_hex()

c + geom\_linerange()

c + geom\_pointrange()

c + geom\_rect()

c + geom\_rug()

c + geom\_smooth()

c + geom\_text()

c + geom\_violin()

c + geom\_hex()

c + geom\_linerange()

c + geom\_pointrange()

c + geom\_rect()

c + geom\_rug()

c + geom\_smooth()

c + geom\_text()

c + geom\_violin()

c + geom\_hex()

c + geom\_linerange()

c + geom\_pointrange()

c + geom\_rect()

c + geom\_rug()

c + geom\_smooth()

c + geom\_text()

c + geom\_violin()

c + geom\_hex()

c + geom\_linerange()

c + geom\_pointrange()

c + geom\_rect()

c + geom\_rug()

c + geom\_smooth()

c + geom\_text()

c + geom\_violin()

c + geom\_hex()

c + geom\_linerange()

c + geom\_pointrange()

c + geom\_rect()

c + geom\_rug()

c + geom\_smooth()

c + geom\_text()

c + geom\_violin()

c + geom\_hex()

c + geom\_linerange()

c + geom\_pointrange()

c + geom\_rect()

c + geom\_rug()

c + geom\_smooth()

c + geom\_text()

c + geom\_violin()

c + geom\_hex()

c + geom\_linerange()

c + geom\_pointrange()

c + geom\_rect()

c + geom\_rug()

c + geom\_smooth()

c + geom\_text()

c + geom\_violin()

c + geom\_hex()

c + geom\_linerange()

c + geom\_pointrange()

c + geom\_rect()

c + geom\_rug()

c + geom\_smooth()

c + geom\_text()

c + geom\_violin()

c + geom\_hex()

c + geom\_linerange()

c + geom\_pointrange()

c + geom\_rect()

c + geom\_rug()

c + geom\_smooth()

c + geom\_text()

c + geom\_violin()

c + geom\_hex()

c + geom\_linerange()

c + geom\_pointrange()

c + geom\_rect()

c + geom\_rug()

c + geom\_smooth()

c + geom\_text()

c + geom\_violin()

c + geom\_hex()

c + geom\_linerange()

c + geom\_pointrange()

c + geom\_rect()

c + geom\_rug()

c + geom\_smooth()

c + geom\_text()

c + geom\_violin()

c + geom\_hex()

c + geom\_linerange()

c + geom\_pointrange()

c + geom\_rect()

c + geom\_rug()

c + geom\_smooth()

c + geom\_text()

c + geom\_violin()

c + geom\_hex()

c + geom\_linerange()

c + geom\_pointrange()

c + geom\_rect()

c + geom\_rug()

c + geom\_smooth()

c + geom\_text()

c + geom\_violin()

c + geom\_hex()

c + geom\_linerange()

c + geom\_pointrange()

c + geom\_rect()

c + geom\_rug()

c + geom\_smooth()

c + geom\_text()

c + geom\_violin()

c + geom\_hex()

c + geom\_linerange()

c + geom\_pointrange()

c + geom\_rect()

c + geom\_rug()

c + geom\_smooth()

c + geom\_text()

c + geom\_violin()

c + geom\_hex()

c + geom\_linerange()

c + geom\_pointrange()

c + geom\_rect()

c + geom\_rug()

c + geom\_smooth()

c + geom\_text()

c + geom\_violin()

c + geom\_hex()

c + geom\_linerange()

c + geom\_pointrange()

c + geom\_rect()

c + geom\_rug()

c + geom\_smooth()

c + geom\_text()

c + geom\_violin()

c + geom\_hex()

c + geom\_linerange()

c + geom\_pointrange()

c + geom\_rect()

c + geom\_rug()

c + geom\_smooth()

c + geom\_text()

c + geom\_violin()

c + geom\_hex()

## 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.

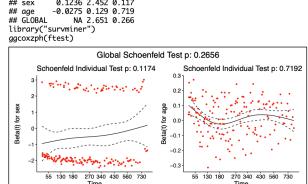
```
ggsurvplot(fit, data = lung, fun = "event")
ggsurvplot(fit, data = lung, fun = "cumhaz")
```

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 `gcooxzph()` 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)
fit <- cox.zph(fit, rho = "chisq")
fit
```



The function `gcooxdiagnostic()` 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 `"marginal"`, `"deviance"`, `"score"`, `"schoenfeld"`, `"dbeta"`, `"dfbetas"`, and `"scaledchi"`.

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)
```

```
ggcoxdiagnostic(fit,
```

```
type = "deviance",
ox.scale = "linear.predictions")
```

```
ggcoxdiagnostic(fit,
type = "schoenfeld")
```

```
ggcoxadjustedcurves(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 plotted for each variable.

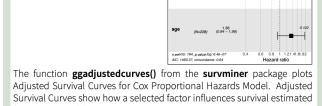
```
library("survival")
library("survminer")
lung <- filter(lung, (age > 70) & (age < 70))
fit <- coxph(Surv(time, status) ~ sex + ph.ecog + age, data = lung)
```

```
fit
```

```
# Call:
#> coxph(formula = Surv(time, status) ~ sex + ph.ecog + age, data = lung)
#>
#> coef exp(coef) se(coef)    z    p
#> sex   -0.0275  0.979  0.0263  0.266
#> ph.ecog  0.478   1.600  0.113  4.15 3.1e-05
#> age    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 Curve for Cox Proportional Hazards Model. Adjusted Survival Curves show how a selected factor influences survival estimated from a Cox model.

Note that adjusted curves differ from Kaplan Meier estimates since they present expected survival based on given Cox model.

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

```
lungSex <- filter(lung, sex == 1,
                   "Male", "Female")
```

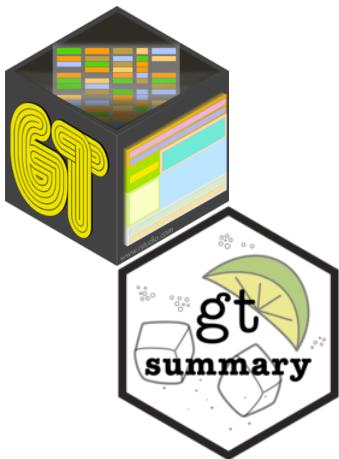
```
fit <- coxph(Surv(time, status) ~ sex + ph.ecog + age +
               sex:age, data = lung)
```

```
ggcoxadjustedcurves(fit, data = lung)
```

```
ggcoxadjustedcurves(fit, data = lung)
```



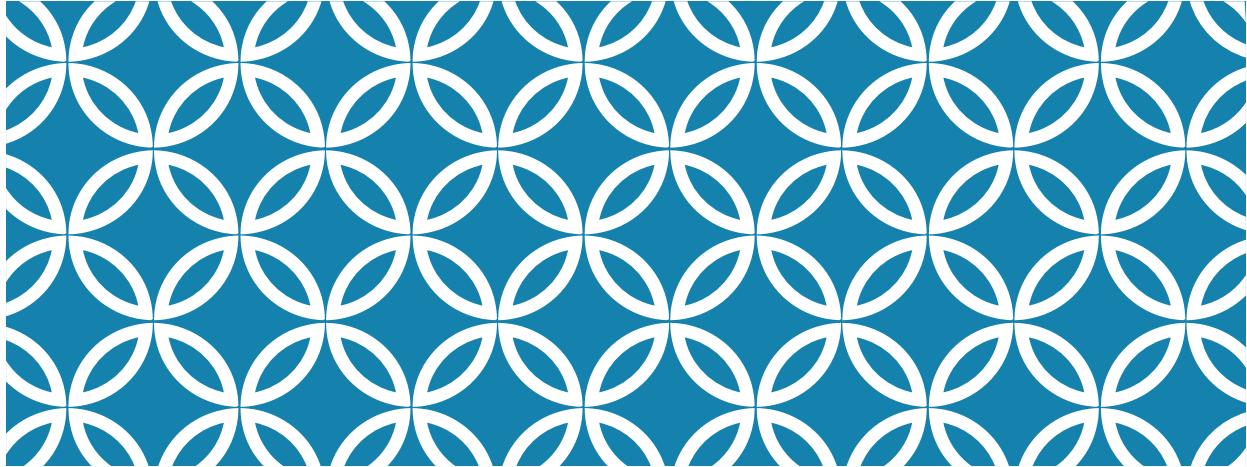
# A grammar for tables



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

<sup>1</sup> Statistics presented: median (IQR); n (%)

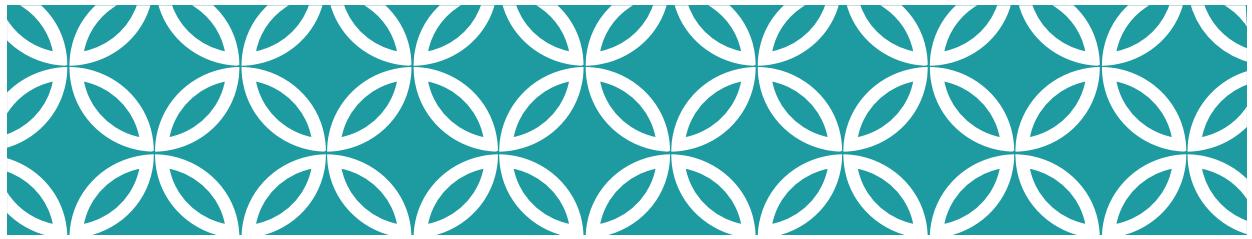
<sup>2</sup> Statistical tests performed: Wilcoxon rank-sum test; chi-square test of independence



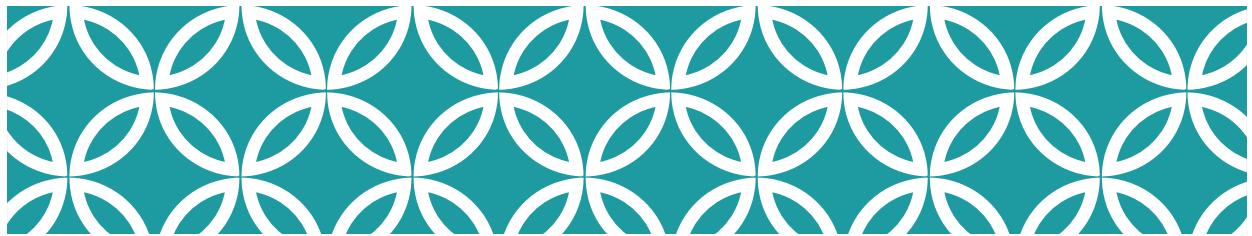
## CHOP R 101: Intro to R for Clinical Data | **Transform**



a grammar for transforming  
data frames

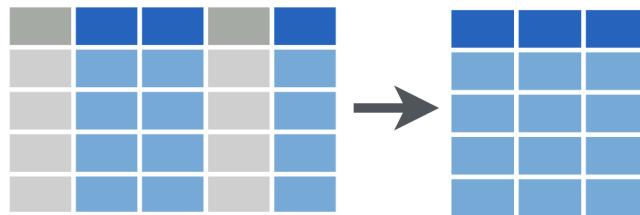


## Subsetting data

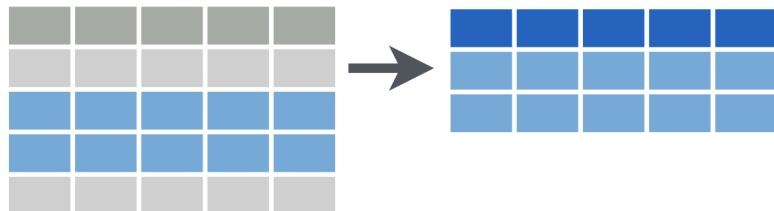


### Subsetting columns vs rows

`select()`



`filter()`



# select()

Extract columns by name.

```
select(data_frame, ...)
```

names of columns  
to extract

# select()

Extract columns 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  |

# Your Turn #1

Which of the following will select the `first_name` column from the `covid_testing` data frame and capture the result in a data frame named `newdata`?

- A) `newdata = select(first_name, covid_testing)`
- B) `newdata <- select(covid_testing, first_name)`
- C) `select(newdata, covid_testing, first_name)`
- D) `newdata <- select(covid_testing, First_Name)`
- E) Both B and D

Type your response in the chat!

01:00

## filter()

Extract rows that meet logical criteria.

```
filter(data_frame, condition)
```

logical test

(return each row for which  
the test is `TRUE`)

# filter()

= sets  
== compares

Extract rows that meet logical criteria.

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

| covid_testing |         |            |            |
|---------------|---------|------------|------------|
|               | mrn     | first_name | last_name  |
| FALSE         | 5000876 | sarella    | stark      |
| FALSE         | 5006017 | alester    | stark      |
| FALSE         | 5001412 | jhezane    | westerling |
| TRUE          | 5000083 | lollys     | cleagane   |

→

| mrn     | first_name | last_name |
|---------|------------|-----------|
| 5000083 | lollys     | cleagane  |

## Logical Operators

|          |                          |
|----------|--------------------------|
| 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             |
| is.na(x) | a missing value          |

## Your Turn #2

Write a `filter()` statement that returns a data frame containing only the rows from `covid_testing` in which the `last_name` column is NOT equal to "stark".

(You don't have to capture the returned data frame)

Type your response in the chat!

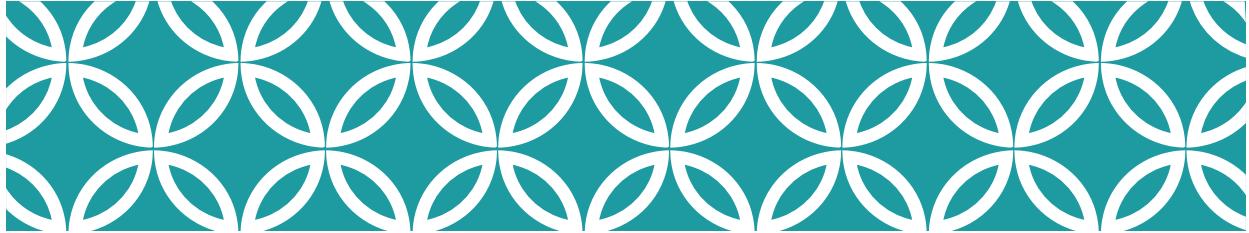
01:00

## Your Turn #3

Which of these would successfully filter the `covid_testing` data frame to only tests with positive results?

- A) `filter(covid_testing, result == positive)`
- B) `filter(covid_testing, result = "positive")`
- C) `filter(covid_testing, result == "positive")`
- D) `filter(covid_testing, positive == "result")`

01:00



## The pipe operator %>%



## The Pipe Operator %>%

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

Passes the **object on the left** as **first argument** to the **function on the right**.

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

"then"

```
covid_testing %>%
  select(last_name, result) %>%
  filter(result == "positive")
```

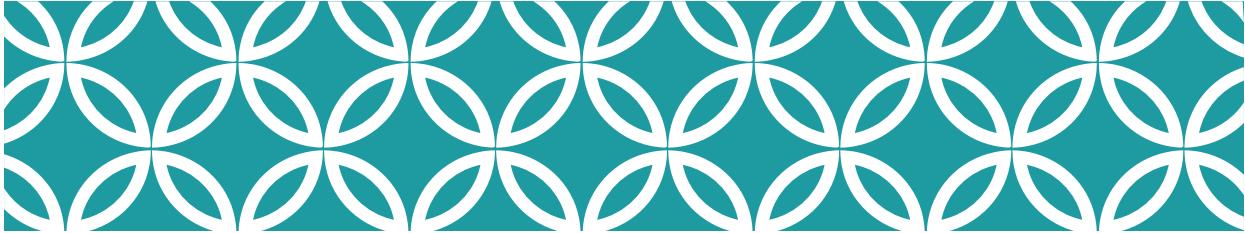
## Your Turn #4

Rewrite the following statement with a pipe:

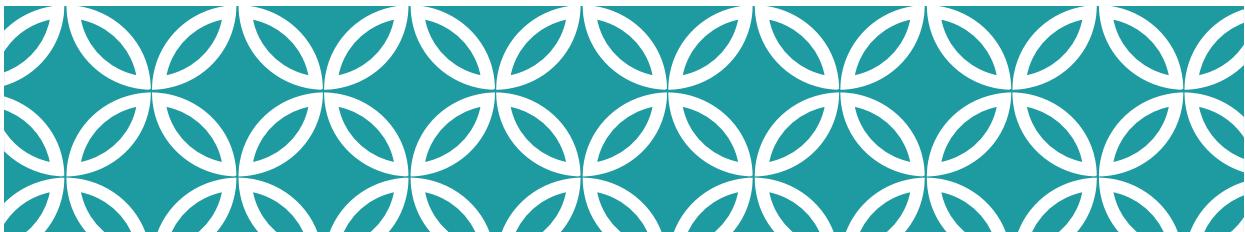
```
select(mydata, first_name, last_name)
```

Type the answer in the chat!

01:00



## Create new columns



### mutate()

Create new calculated columns.

```
mutate(data_frame, name = value)
```

new column name

single =

calculation  
(can include data from  
other columns)

# mutate()

Create new calculated columns.

```
mutate(covid_testing,  
       col_rec_tat_mins = col_rec_tat * 60)
```

| mrn     | col_rec_tat |
|---------|-------------|
| 5000876 | 29.5        |
| 5006017 | 3.6         |
| 5001412 | 1.4         |
| 5000533 | 2.3         |



| mrn     | col_rec_tat | col_rec_tat_mins |
|---------|-------------|------------------|
| 5000876 | 29.5        | 1770             |
| 5006017 | 3.6         | 216              |
| 5001412 | 1.4         | 84               |
| 5000533 | 2.3         | 138              |

## Your Turn #5

Open 03 - Transform.Rmd and work through the exercises.

Click “yes” when you are finished.

05:00

## Recap



`select()` subsets columns by name



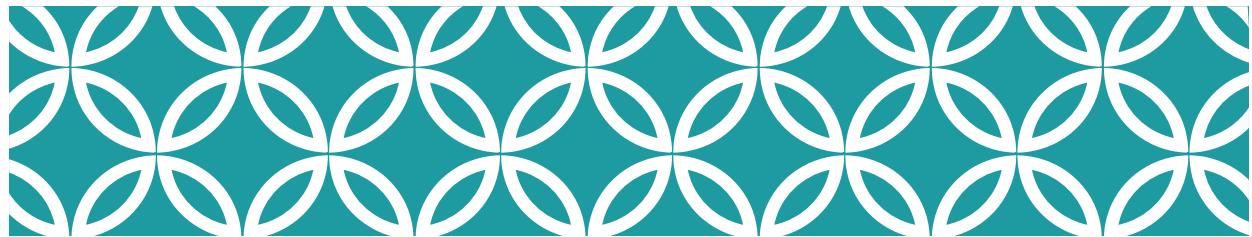
`filter()` subsets rows by a logical condition



`mutate()` creates new calculated columns

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

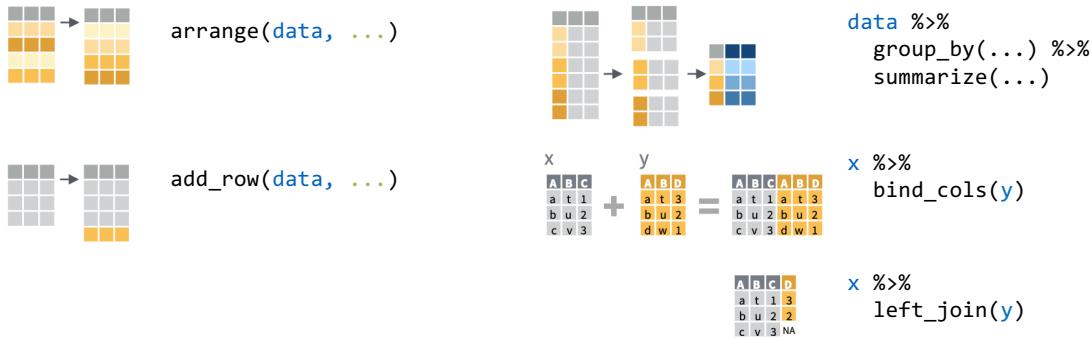
Use the **pipe operator** `%>%` to combine dplyr functions into a pipeline



What else?



# More dplyr functions



## Data Transformation with dplyr :: CHEAT SHEET

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



### 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(..., wt = NULL, sort = FALSE)`

Count number of rows in each group defined by the variables in ... Also `tally()`.

`count(mtcars, Species)`

`count_if(..., ...)`

Count number of rows in each group defined by the variables in ... Also `tally_if()`.

`count_if(mtcars, is.numeric)`

`summarise_all(...)`

Apply funs to every column.

`summarise_at(..., ..., .by_group = TRUE)`

Apply funs to specific columns.

`summarise_if(..., ..., .by_group = TRUE)`

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 %>%`

`arrange(desc(mpg))`

### Manipulate Cases

**EXTRACT CASES**  
Row functions return a subset of rows as a new table.

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

`distinct(data, ..., keep = 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(mtcars, 0.5, replace = FALSE)`

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

`slice(data, ...)` Select rows by position. `slice(mtcars, 1:10)`

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

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

**Logical and boolean operators to use with filter()**

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

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.

### 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(mtcars, Sepal.Length)`

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

Use these helpers with `select()`, e.g. `select_if(is.na)`

`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.

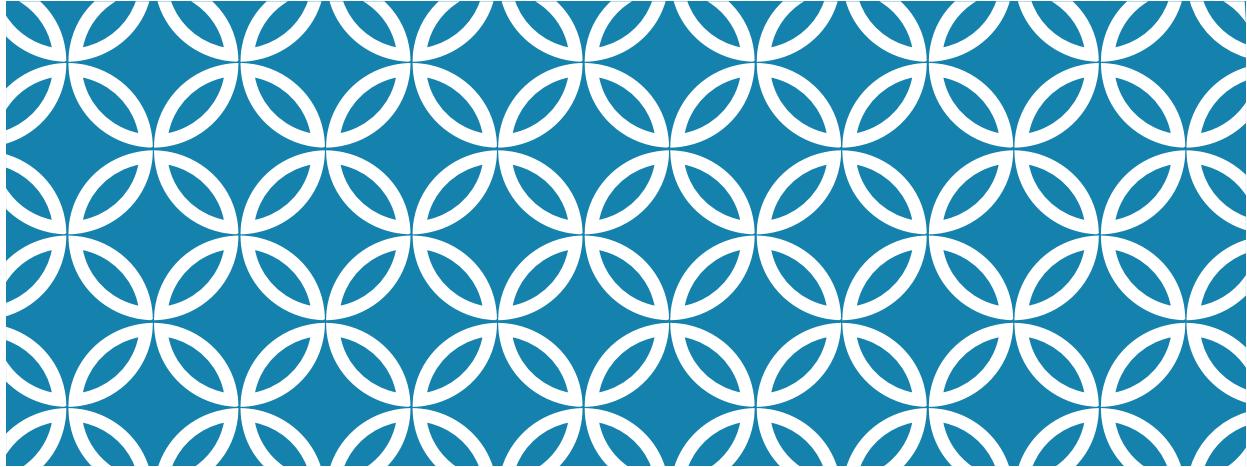
`transmute(mtcars, gpm = 1/mpg)`

`mutate_all(tbl, funs(...))` Apply funs to every column. Use with `funs()`. Also `mutate_if()`.

`mutate_if(mtcars, is.numeric, funs(log10))`

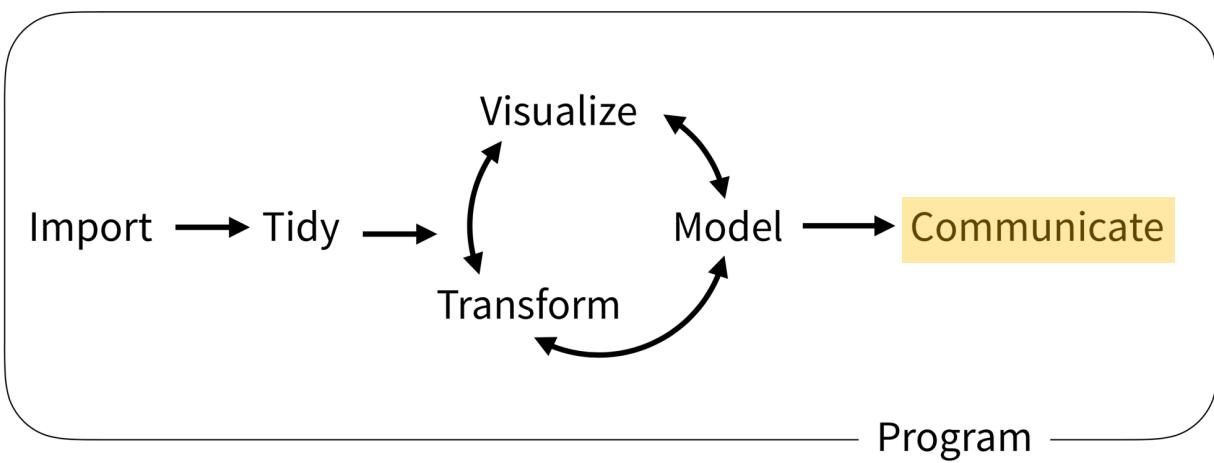
`mutate_if(mtcars, is.factor, funs(as.character))`





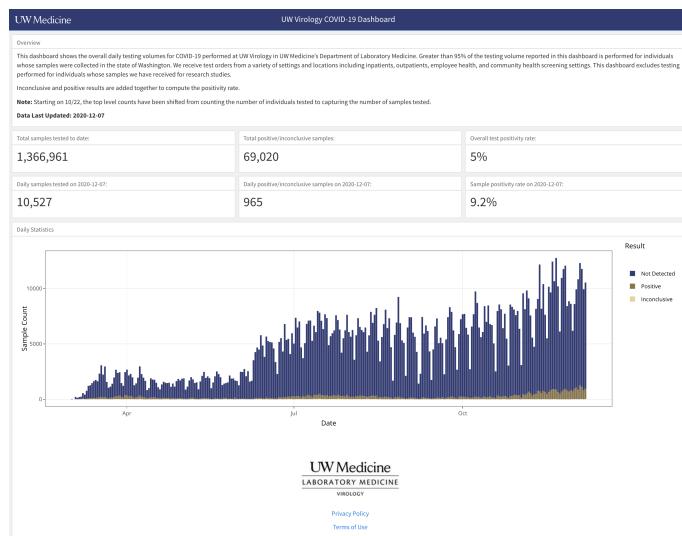
## CHOP R 101: Intro to R for Clinical Data | **Dashboard**

### The Data Analysis Pipeline



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

## Dashboards visualize key data on demand



Turn an R Markdown document  
into an interactive dashboard

# Recap: Anatomy of an R Markdown Document

```
1 --
2 title: 'My Markdown Document'
3 output: html_document
4 ---
5
6 # One Hashtag = Large Header
7
8 ## Two Hashtags = Smaller Header
9
10 Here is some text.
11
12 * It's easy to make a list
13 * Here is how you style text *cursive* or **bold**
14
15
16 `r
17 x <- rnorm(100)
18 summary(x)
19 `r
20
```

Header

Text  
(with marks)

Code chunk

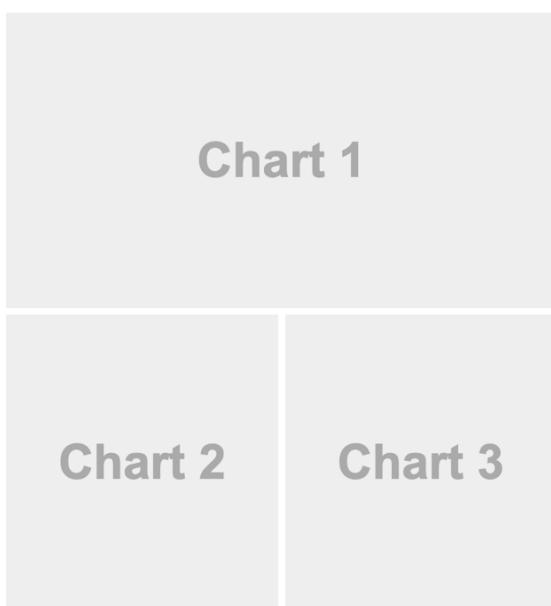
```
1 --
2 output:
3   flexdashboard::flex_dashboard:
4     orientation: columns
5 ---
6
7 Column
8 -----
9
10 ### Chart 1
11
12 `r
13 ...
14
15 Column
16 -----
17
18 ### Chart 2
19
20 `r
21 ...
22
23 ### Chart C
24
25 `r
26 ...
27
```

Chart 1

Chart 2

Chart 3

```
1 ---  
2 output:  
3   flexdashboard::flex_dashboard:  
4     orientation: rows  
5 ---  
6  
7 Row  
8 -----  
9  
10 ### Chart 1  
11  
12 `r`  
13 `r`  
14  
15 Row  
16 -----  
17  
18 ### Chart 2  
19  
20 `r`  
21 `r`  
22  
23 ### Chart C  
24  
25 `r`  
26 `r`  
27
```



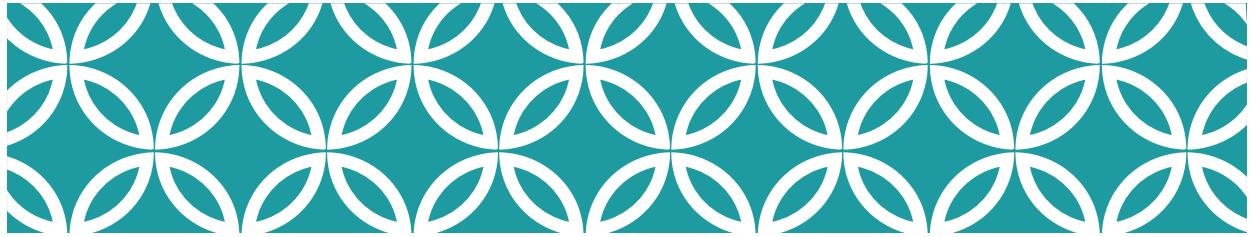
## Your Turn #1

Open 04 – Dashboard.Rmd to work with a draft COVID19 flexdashboard.  
Do the following:

1. Knit the document to see what the dashboard looks like.
2. Change the **Test Volumes Over Time** plot so that it visualizes the fraction of positive tests on a given day. (Hint: map the `fill` aesthetic to `result`). Knit the document again and note the change.
3. Change the **layout** from `columns` to a `row` orientation. Knit the document again and note the change.

Click “yes” when you are finished!





## Making plots interactive



### Plotly makes ggplots interactive

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

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



## Your Turn #2

Return to `04 - Dashboard.Rmd`.

Make the **Test Volumes Over Time** plot interactive.

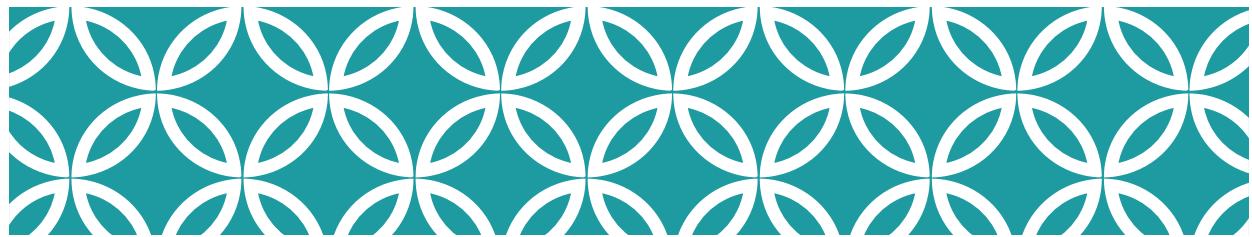
Knit the document again to note the change. Hover over the interactive plots. Note what happens when you click on the various buttons that appear above the plot.

Click “yes” when you are finished.

03:00



## Interactive tables



## DT makes data frames interactive

```
covid_testing
```

```
covid_testing %>%  
  datatable()
```



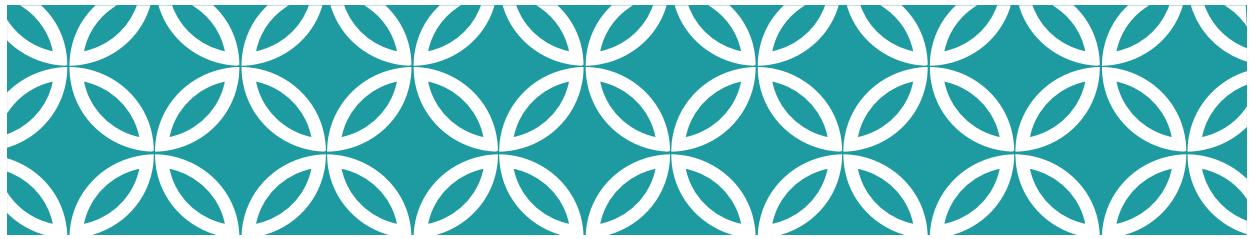
## Your Turn #3

Return to `04 - Dashboard.Rmd`. Let's try to display a list of positive results in the **Details of Positive Test Results** area of the plot.

1. Note that this area currently shows the contents of the `covid_testing` data frame but doesn't actually filter for positive results. Add to the pipeline a `filter()` statement that selects only those rows where the `result` value is "positive" (Hint: "positive" must be in quotes). Knit the dashboard and note the change.
2. In order to use the `datatable()` function, we first need to load the `DT` package. Add a line to the `setup` chunk (all the way on top!) to load the `DT` package.
3. Now make the table interactive! Knit the dashboard and note the change.

Click "yes" when you are finished!

03:00



## Add styling



|          |          |
|----------|----------|
| Cerulean | Flatly   |
| Chart    | Chart    |
| Cosmo    | Readable |
| Chart    | Chart    |

## Adding a theme

Header

```
---
```

```
title: "My Dashboard"
```

```
output:
```

```
  flexdashboard::flex_dashboard:
```

```
    orientation: rows
```

```
    theme: cerulean
```

```
---
```

<https://rmarkdown.rstudio.com/flexdashboard/using.html#themes>

COVID19 Dashboard Hackathon!

## Recap



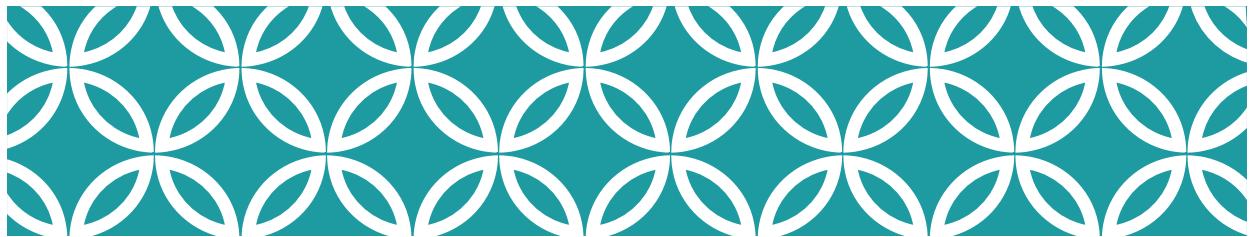
Turn R Markdown documents into interactive dashboards with [flexdashboard](#)



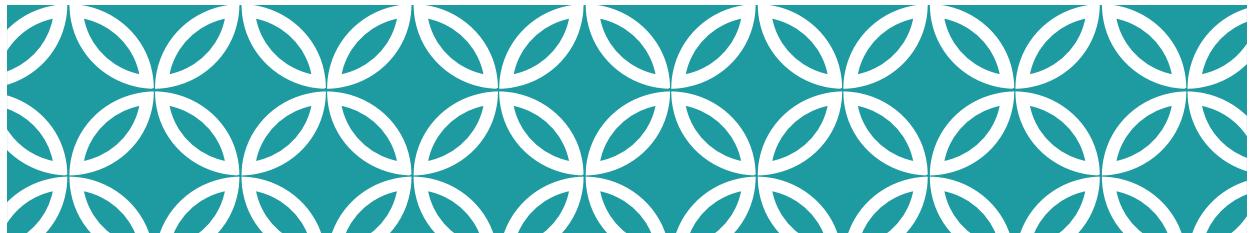
Turn ggplot objects into interactive plots with [plotly](#)



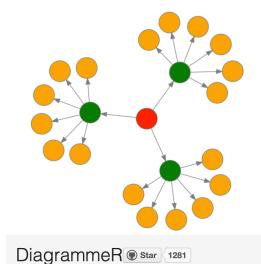
Turn data frames into interactive tables with [DT](#)



What else?



## More interactive charts

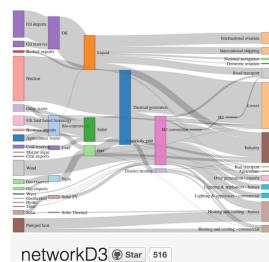


DiagrammeR  Star 1281

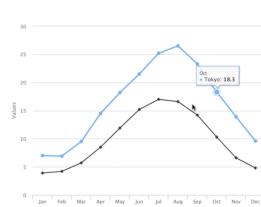


leaflet  Star 578

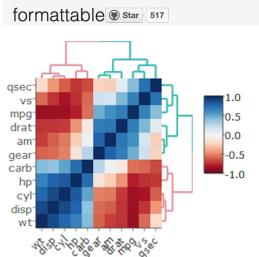
| id | name   | age | grade | test1_score | test2_score |
|----|--------|-----|-------|-------------|-------------|
| 1  | bob    | 26  | C     | 8.9         | 9.1         |
| 2  | Ashley | 27  | A     | 9.5         | 9.1         |
| 3  | Jenny  | 26  | A     | 9.6         | 9.2         |
| 4  | David  | 26  | C     | 8.9         | 9.1         |
| 5  | Jenny  | 29  | B     | 9.1         | 8.9         |
| 6  | Hans   | 28  | B     | 9.3         | 8.6         |
| 7  | Leo    | 27  | B     | 9.3         | 9.2         |
| 8  | John   | 27  | A     | 9.9         | 9.3         |
| 9  | Emily  | 26  | C     | 8.5         | 9.1         |
| 10 | Lee    | 30  | C     | 8.6         | 8.8         |



networkD3  516



highcharter  Star 507



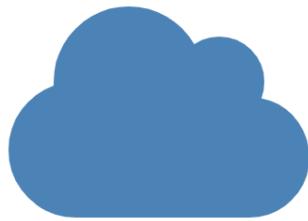
[heatmaply](#)  Star 230

<https://gallery.htmlwidgets.org/>



# A framework for developing interactive web applications in R

# Deploying flexdashboards and Shiny apps



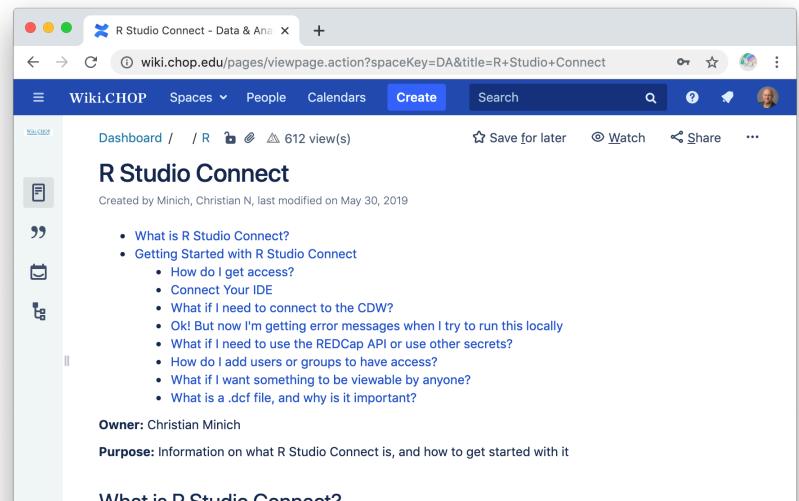
Shinyapps.io (free)



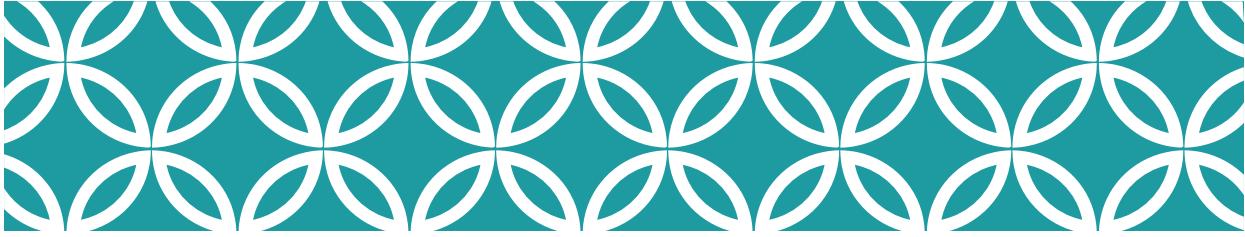
Shiny Server (free)  
RStudio Connect (\$)

## CHOP RStudio Connect

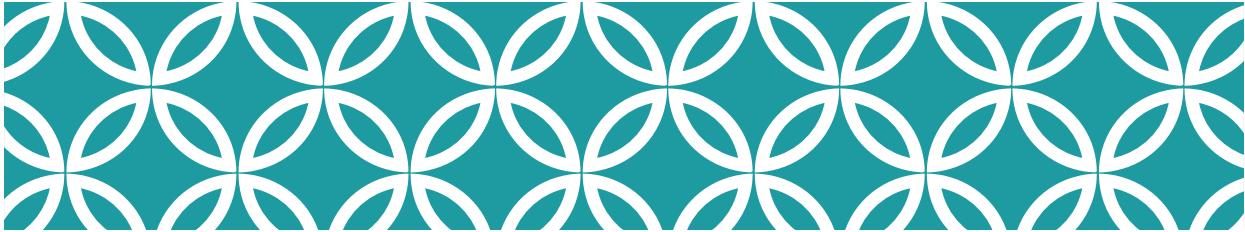
[wiki.chop.edu/display/DA/R+Studio+Connect](http://wiki.chop.edu/display/DA/R+Studio+Connect)



The screenshot shows a web browser window displaying a wiki page titled "R Studio Connect". The URL in the address bar is [wiki.chop.edu/pages/viewpage.action?spaceKey=DA&title=R+Studio+Connect](http://wiki.chop.edu/pages/viewpage.action?spaceKey=DA&title=R+Studio+Connect). The page content includes a sidebar with icons for Dashboard, Spaces, People, Calendars, Create, Search, and more. The main content area has a title "R Studio Connect" and a subtitle "Created by Minich, Christian N, last modified on May 30, 2019". Below the title is a bulleted list of questions: "What is R Studio Connect?", "Getting Started with R Studio Connect", "How do I get access?", "Connect Your IDE", "What if I need to connect to the CDW?", "Ok! But now I'm getting error messages when I try to run this locally", "What if I need to use the REDCap API or use other secrets?", "How do I add users or groups to have access?", "What if I want something to be viewable by anyone?", and "What is a .dct file, and why is it important?". At the bottom of the page, there is a section for "Owner: Christian Minich" and "Purpose: Information on what R Studio Connect is, and how to get started with it".



## What next?



Install R and RStudio on your computer!



### After the course

If you would like to practice with the materials from the course, go to the [GitHub repository](#), and click the green button labeled "Code" to download the repository as a .ZIP file.

To install all the packages we used in the training environment, open RStudio and run the following command in the Console:

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

<https://skadauke.github.io/intro-to-r-for-clinicians-chop/>

## “Come” to a CHOP R User Group Meeting!



### CHOP R User Group Meeting 🎙

Thursday, Dec 10 at 2 PM

Meredith Lee and Cass Wilkinson Saldaña:  
*How to solve small, everyday work problems  
with R and R Markdown*

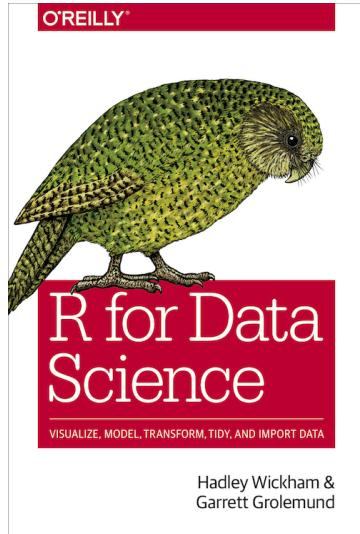
<https://bluejeans.com/849314871>

## Do a **Mini** Project in R!

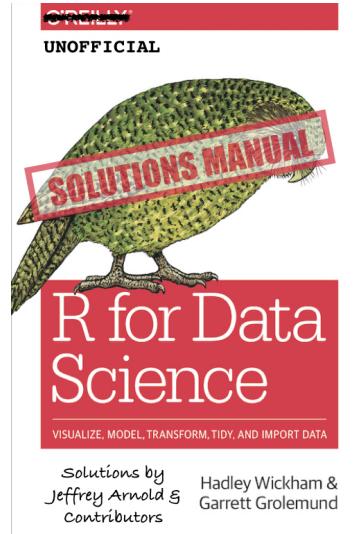
The only way to learn R is to actually spend some time writing R code.

- Identify a work problem that can be solved with a **Mini** R Markdown document
- See tomorrow's CHOP R User Group meeting for sample **Mini** projects
- Kick-off meeting next week (time TBD)
- Get lots of support via CHOP R User Group and Arcus Education
- Optional: Give a **Mini** presentation at an upcoming CHOP R User Group meeting

<https://bit.ly/chopr101-mini-project>



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



<https://jrnold.github.io/r4ds-exercise-solutions/>