Tidybiology +DS: Session 1

An Introduction to Biological Data Science in R

Matthew Hirschey, Ph.D.

April 8-9, 2020

Doctors make decisions based on symptoms

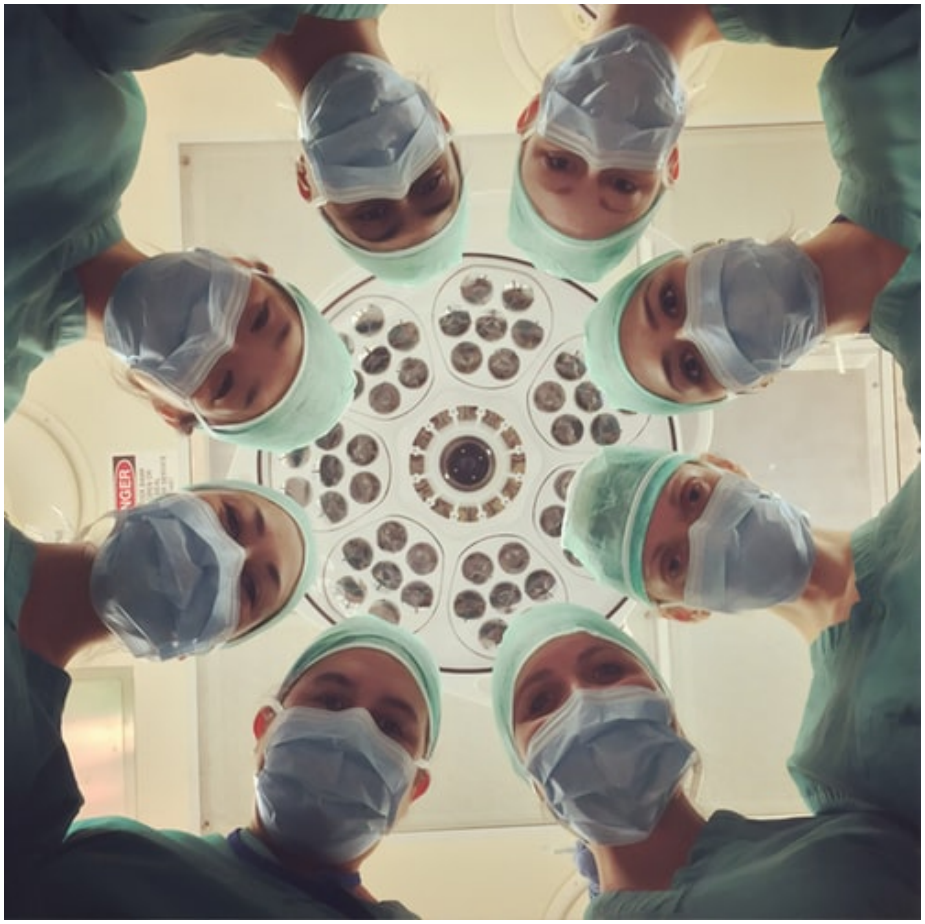


Photo by National Cancer Institute on Unsplash (https://unsplash.com/@nci)

New digital healthcare era introduces new decision-making challenges

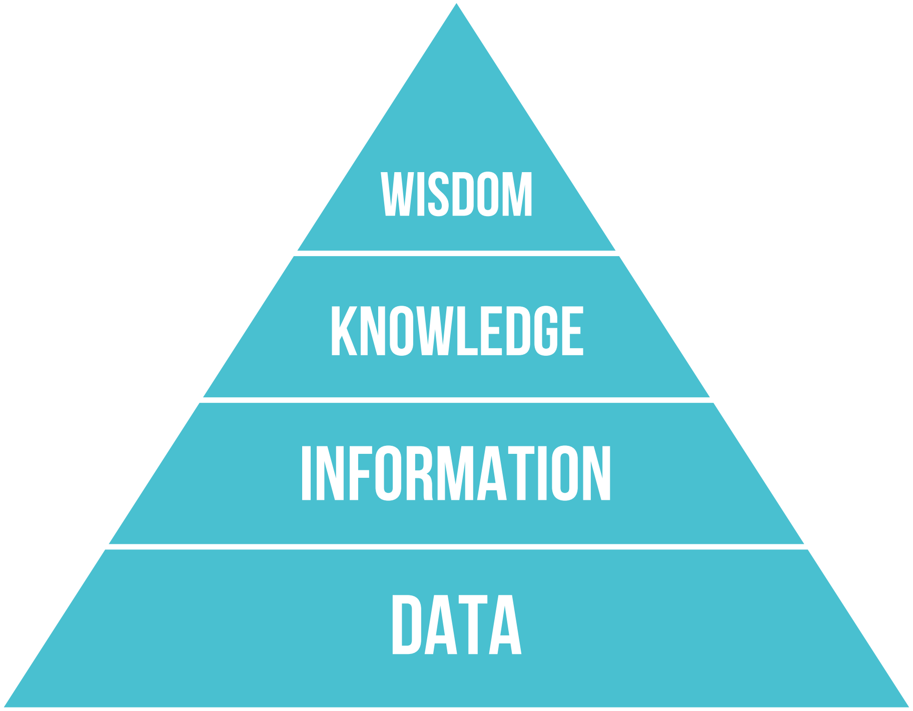
Volume
- Data collection & storage allows access to huge amounts of medical information

Ubiquity
- Data are available anywhere across geography, social, and economic classes

Latency
- Technology facilitates no delay in access to data

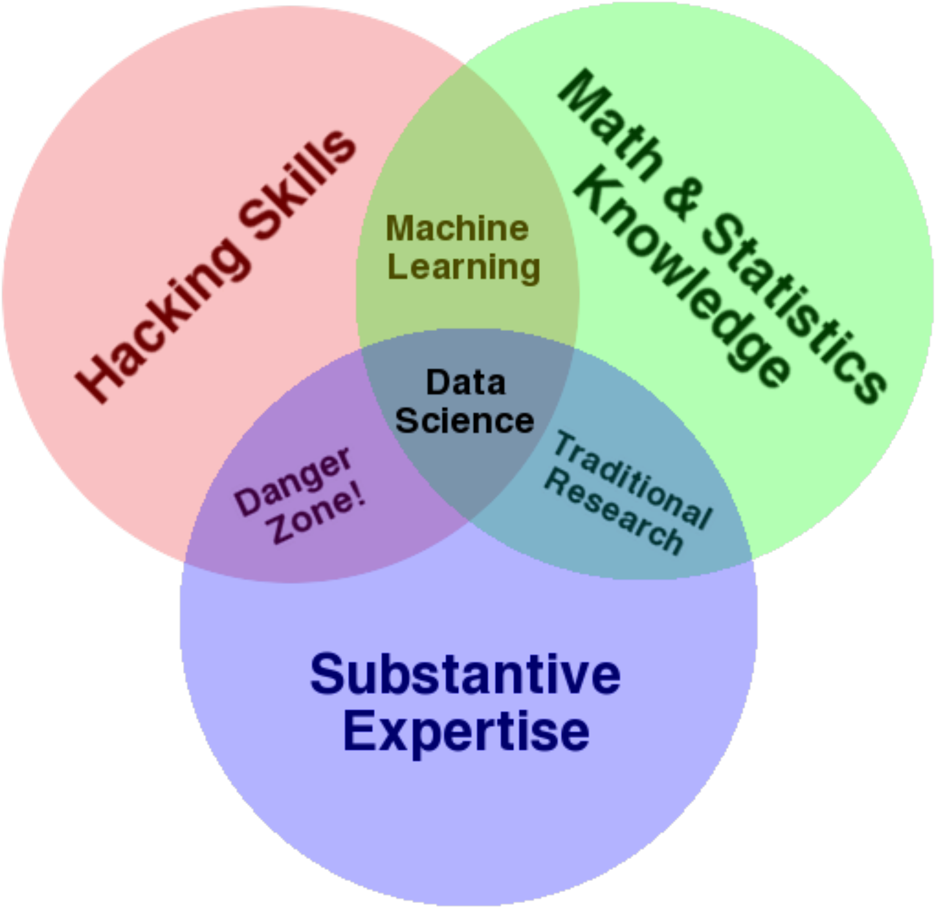
How do you make better health care decisions?

Data-driven decision making!



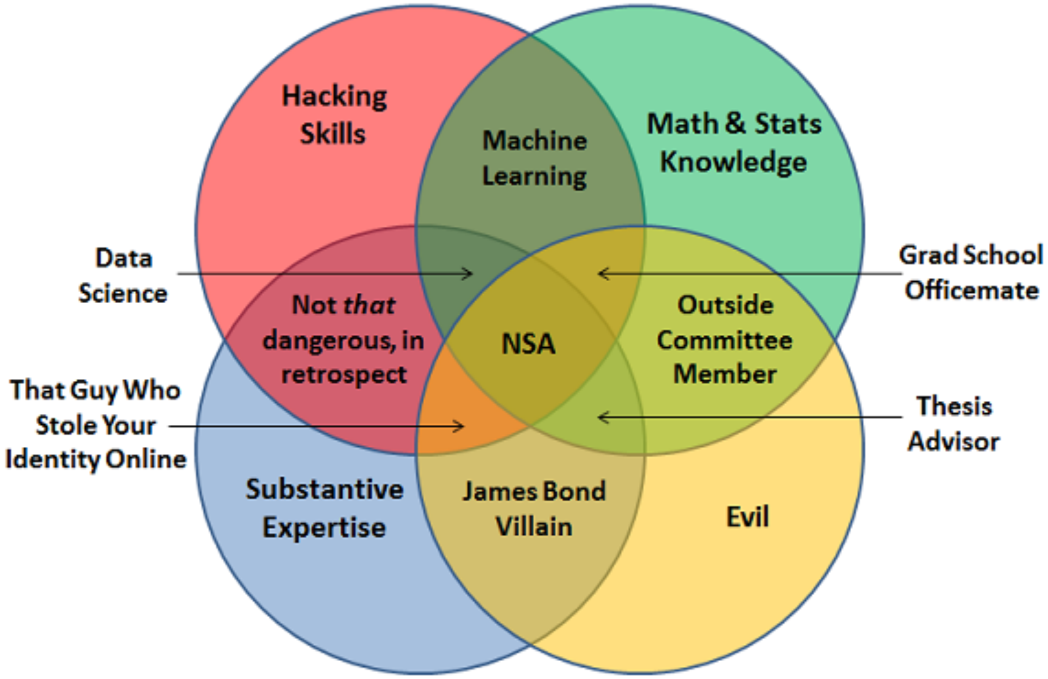
https://en.wikipedia.org/wiki/DIKW_pyramid (https://en.wikipedia.org/wiki/DIKW_pyramid)

Emerging field of Data Science



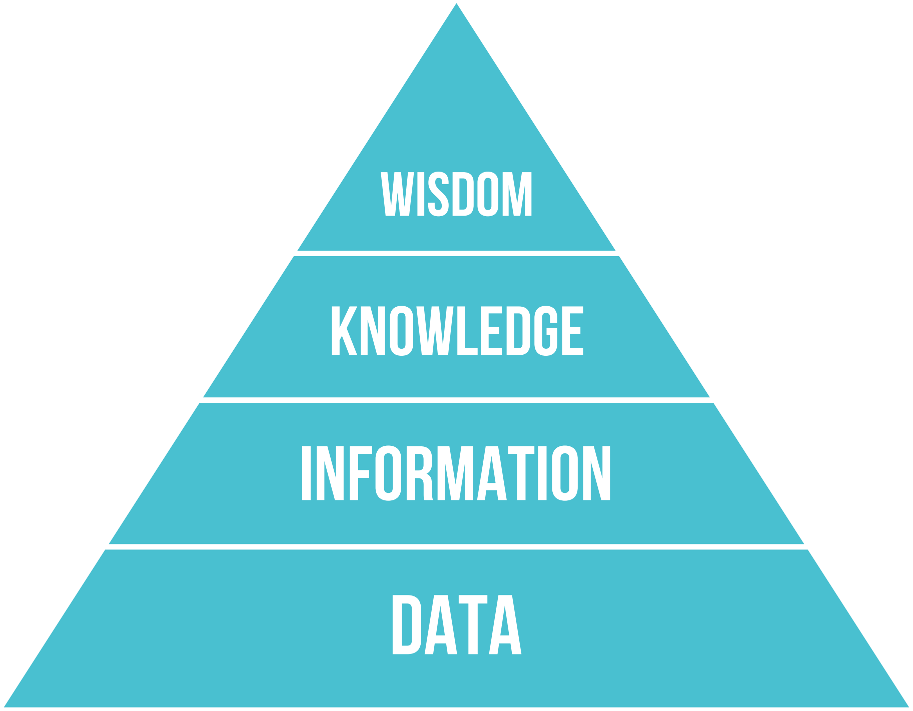
http://drewconway.com/zia/2013/3/26/the-data-science-venn-diagram (http://drewconway.com/zia/2013/3/26/the-data-science-venn-diagram)

Venn Diagram of Data Science v2.0



Joel Grus via KDnuggets

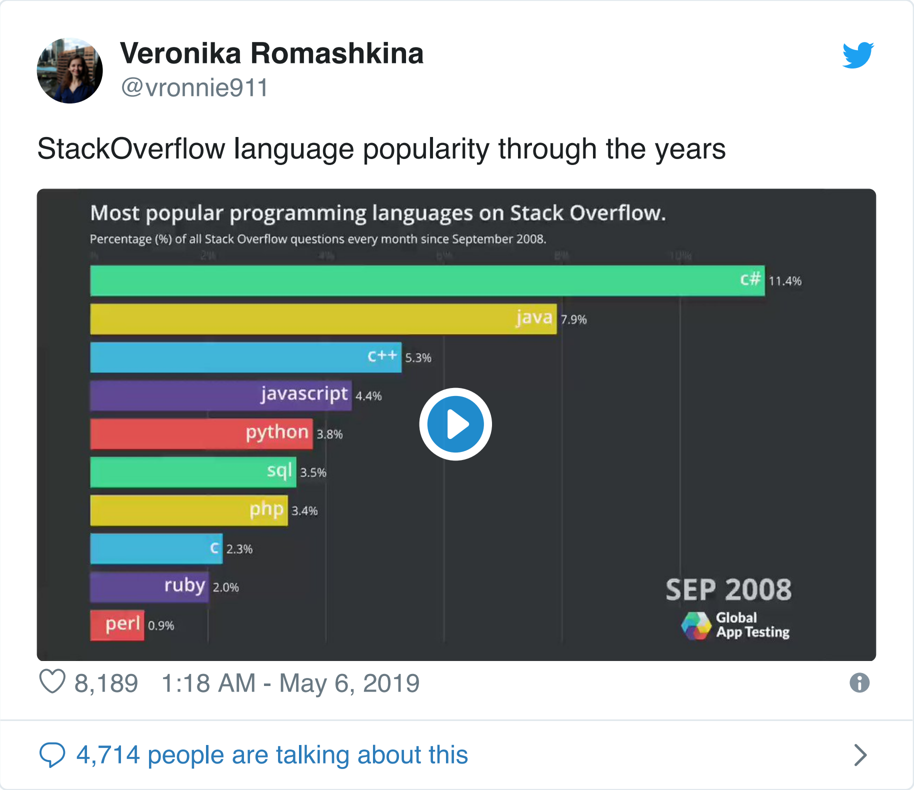
Overall goal is Knowledge Generation



https://en.wikipedia.org/wiki/DIKW_pyramid (https://en.wikipedia.org/wiki/DIKW_pyramid)

Several Approaches to Knowledge Generation

- New tools allow data interrogation more easily than ever before



Veronika Romashkina
@vronnie911

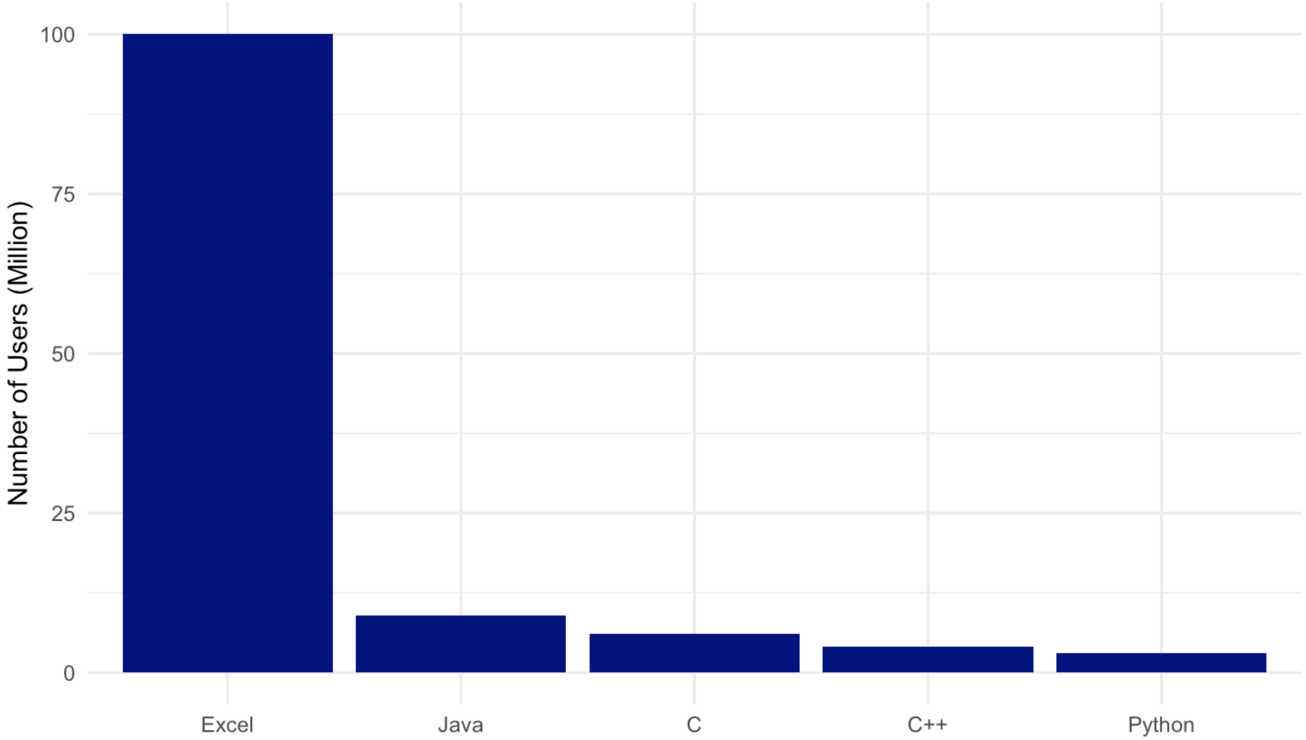
StackOverflow language popularity through the years

Most popular programming languages on Stack Overflow.
Percentage (%) of all Stack Overflow questions every month since September 2008.
c# 11.4%
java 7.9%
c++ 5.3%
javascript 4.4%
python 3.8%
sql 3.5%
php 3.4%
c 2.3%
ruby 2.0%
perl 0.9%
SEP 2008
Global App Testing

♡ 8,189 1:18 AM - May 6, 2019

4,714 people are talking about this

World’s most popular programming languages



Any questions?

R language

R is a language

Photo by Hannah Wright on Unsplash

R has values

- 1
- “North Carolina”
- “2020-04-08”

R has objects

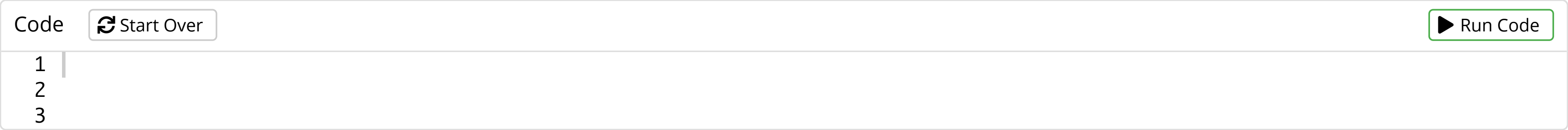
- A name without quotes
- Assigned using <- (looks like an arrow pointing left)
- Can be a value, object, or function result

Try assigning an object

1. Assign an object; **remember, no quotes on name**
   `name <- 4`

2. Return that object by typing its name
   `name`

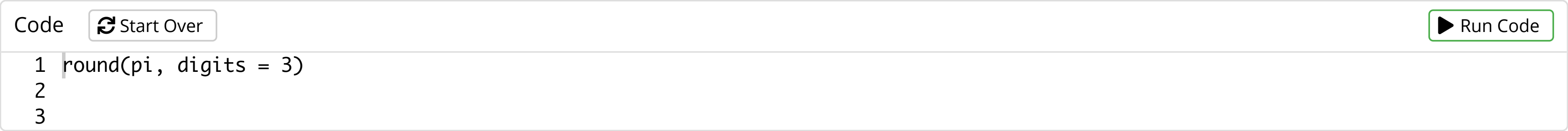
Try this in the code chunk below, then hit “Run Code”



R has functions

- A name without quotes
- Followed by () to run the function
- Optional arguements: values, objects, or function results
- `round(x, digits = 3)`

Try this in the code chunk below, then hit “Run Code”



```r
round(pi, digits = 3)


```

Which one of these are numbers?

Which of these are numbers?

○ 1

○ "1"

○ "one"

○ one

Submit Answer

Which of these will work?

Suppose `one <- 1`

Which of these will work?

□ log(one)

□ log("one")

□ log(1)

□ log("1")

Submit Answer

Try it for yourself!

Code Start Over Run Code

```
one <- 1


```

Data are stored in tables and dataframes

Data stored in a dataframe are conceptually equivalent to a spreadsheet with rows and columns

This is a sample from the `heart` dataset

patient_id	age	sex	cp	trestbps
1	52	male	0	125
2	53	male	0	140
3	70	male	0	145
4	61	male	0	148
5	62	female	0	138

Data are stored in tables and dataframes

Data stored in a dataframe are conceptually equivalent to a spreadsheet with rows and columns

This is a sample from the `heart` dataset

```
## # A tibble: 5 x 5
##   patient_id   age sex       cp trestbps
##        <int> <dbl> <chr>  <dbl>    <dbl>
## 1          1    52 male       0      125
## 2          2    53 male       0      140
## 3          3    70 male       0      145
## 4          4    61 male       0      148
## 5          5    62 female     0      138
```

Extract or create new objects

You can call a single part of the data frame

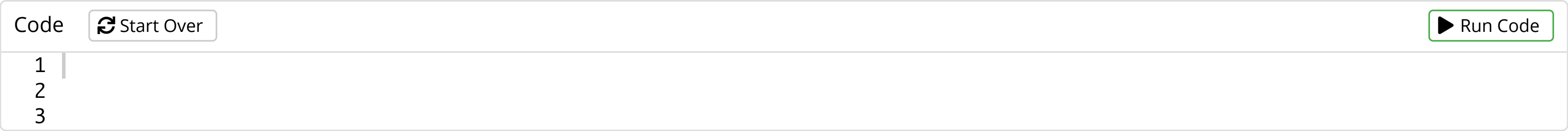
heart$target

```
## # A tibble: 1,025 x 1
##    target
##     <dbl>
##  1      0
##  2      0
##  3      0
##  4      0
##  5      0
##  6      1
##  7      0
##  8      0
##  9      0
## 10      0
## # … with 1,015 more rows
```

Extract or create new objects

Write the R code required to extract a variable from the heart dataset:

Remember, the format is: heart$target



Extract or create new objects

You can also save a part of the dataframe as an object for later use

```
target <- heart$target
```

In the code chunk below:

1. On the first line, write the R code to save a single column to a new object
2. On the second line, type the object name - this will print out the new object
3. Run the code

Code Start Over ▶ Run Code

```

```

Any questions?

R
Integrated Development Environment

R

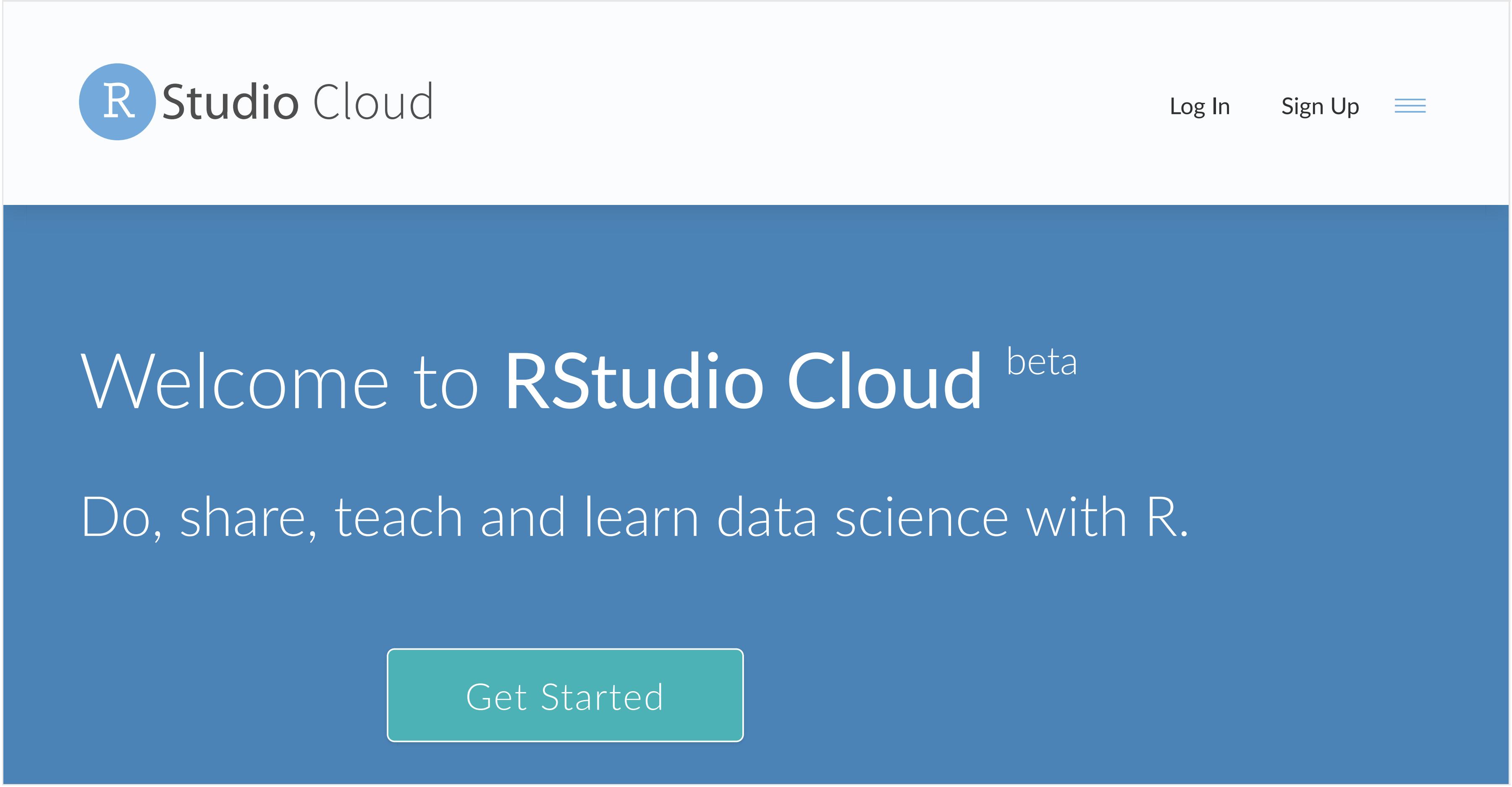
Statistical Programming Language

Code Start Over Run Code

Rstudio

Integrated Development Environment

Open Rstudio (http://www.rstudio.cloud)



Welcome to **RStudio Cloud** beta

Do, share, teach and learn data science with R.

Rstudio Demonstration

Go to code/
Open 01_r_demo.Rmd
Follow along.

Any questions?

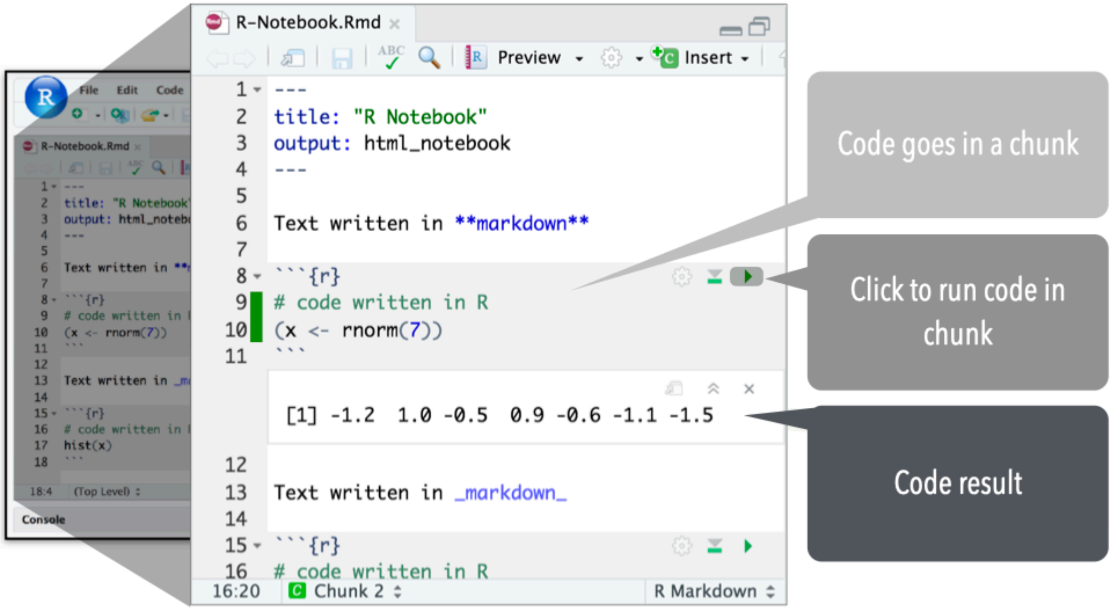
R Markdown

R Markdown Introduction

Go to code/
Open 02_rmd_exercise.Rmd
Read through the file and do everything it tells you to do.

R Markdown

An authoring format for Data Science.



~~~
---
title: "R Notebook"
output: html_notebook
---

Text written in **markdown**

```{r}
# code written in R
(x <- rnorm(7))
```
~~~

```
[1] -1.2  1.0 -0.5  0.9 -0.6 -1.1 -1.5
```

~~~
Text written in _markdown_

```{r}
# code written in R
~~~

Code goes in a chunk

Click to run code in chunk

Code result

Any questions?

R
Packages

R Packages

The R language contains thousands of functions, data sets, and help pages.
- but only a few hundred are included when you download R

This is called ‘Base R’

The other functions, data sets, and help pages are grouped into collections known as packages that you can choose to download or not download.

“Verbs” (i.e. functions) act on data

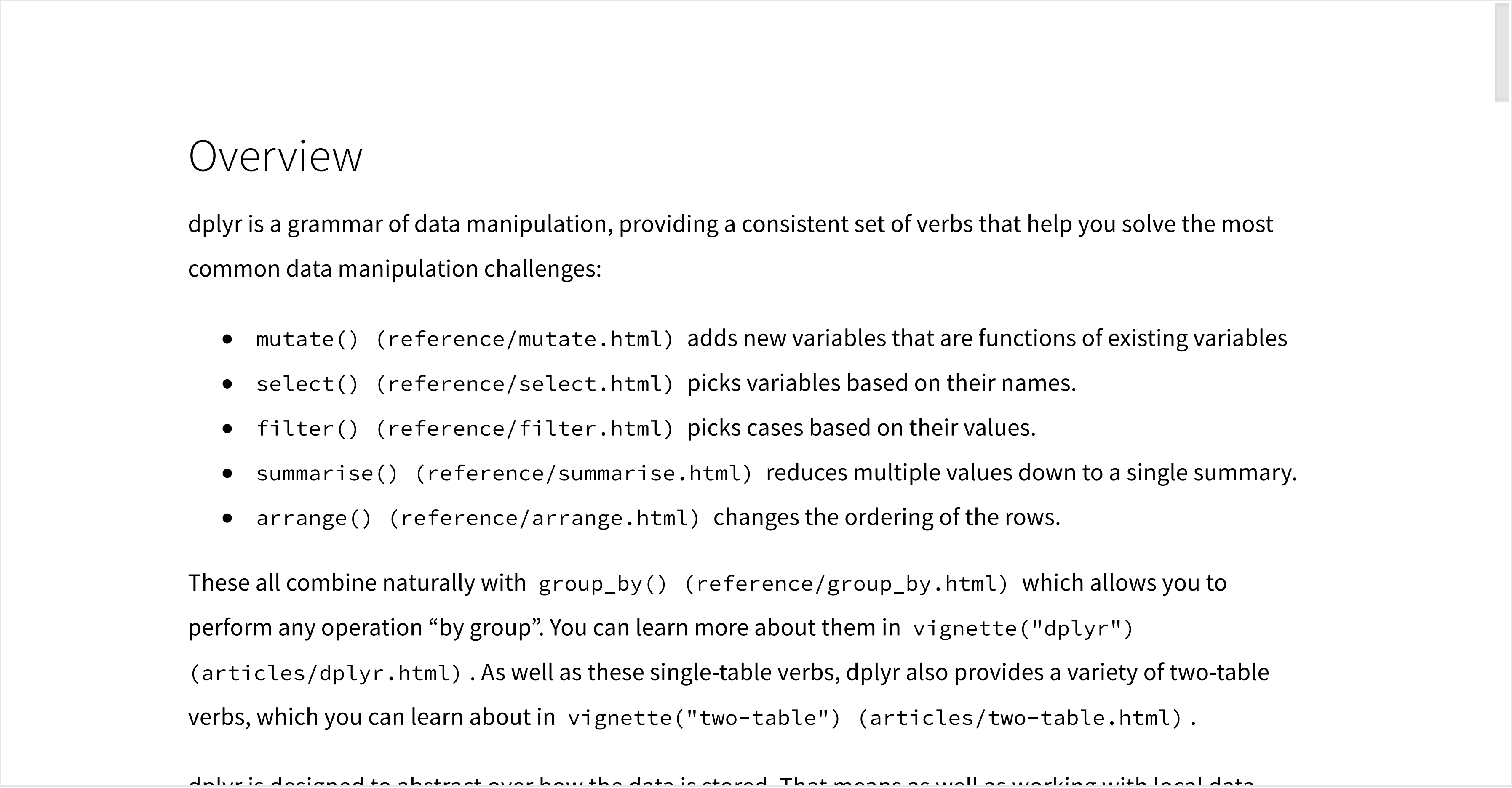
```
do_this(to_that)
```

```
do_this(to_that, using_these)
```

We talked about functions before (e.g. `round(pi, 3)`*)*

Functions are the power of using R

Packages contain functions, documentation, data



Overview

dplyr is a grammar of data manipulation, providing a consistent set of verbs that help you solve the most common data manipulation challenges:

- `mutate() (reference/mutate.html)` adds new variables that are functions of existing variables
- `select() (reference/select.html)` picks variables based on their names.
- `filter() (reference/filter.html)` picks cases based on their values.
- `summarise() (reference/summarise.html)` reduces multiple values down to a single summary.
- `arrange() (reference/arrange.html)` changes the ordering of the rows.

These all combine naturally with `group_by() (reference/group_by.html)` which allows you to perform any operation "by group". You can learn more about them in `vignette("dplyr") (articles/dplyr.html)`. As well as these single-table verbs, dplyr also provides a variety of two-table verbs, which you can learn about in `vignette("two-table") (articles/two-table.html)`.

dplyr is designed to abstract over how the data is stored. That means as well as working with local data

CRAN

- Most R packages are stored on CRAN, alongside R.
- Think of them as optional extensions of the R language.

Image by daroczig (https://gist.github.com/daroczig/3cf06d6db4be2bbe3368)

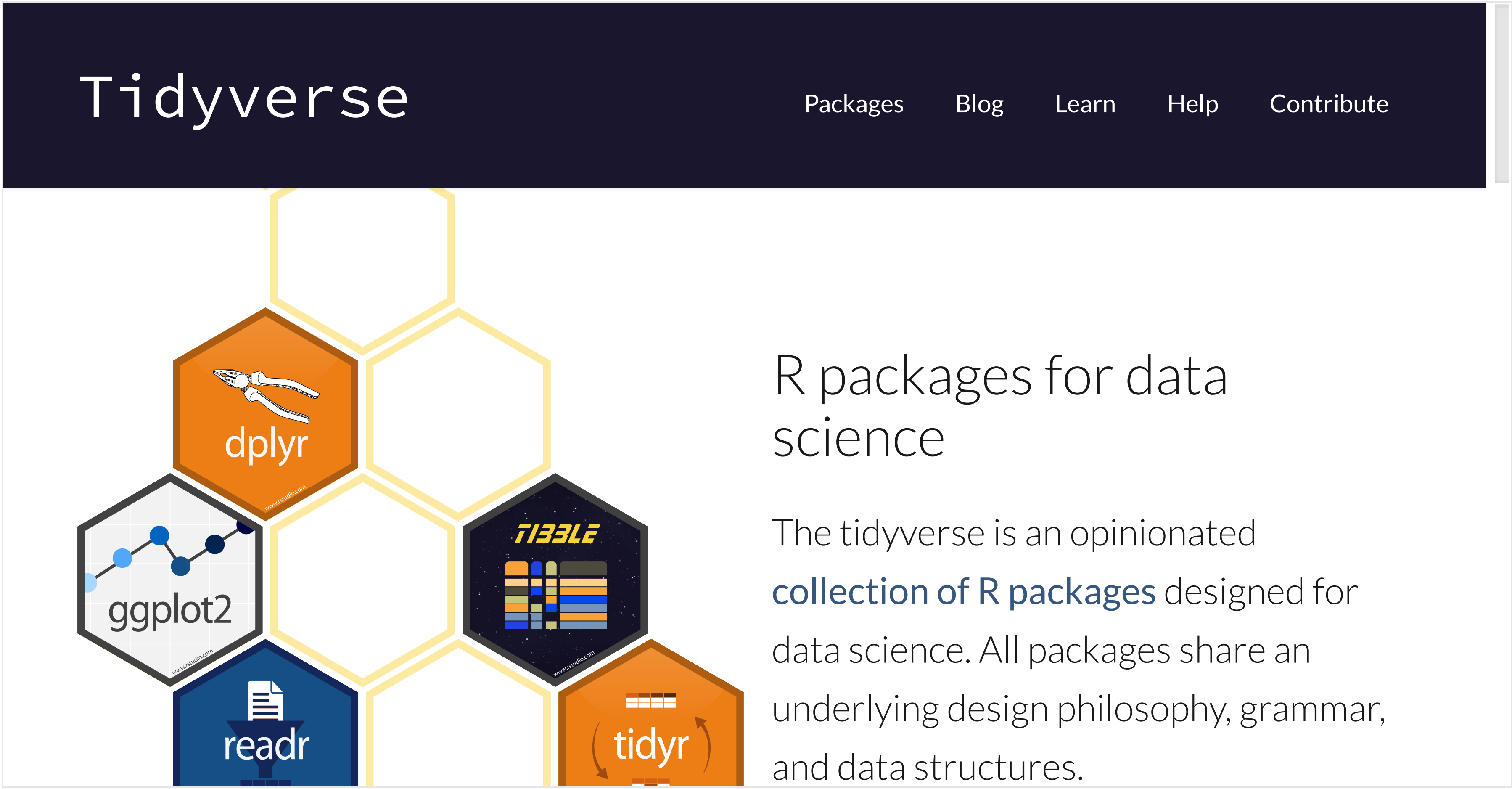
Using Packages (Part I)

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

Do this 1 time per computer.

This command will install the package into your instance of R, whether it is local, on a server, or in the cloud. This is required to use the functions in a package.

Tidyverse



Tidyverse

R packages for data science

The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

Tidyverse Pop Quiz!

The tidyverse contains the following packages (ggplot2, dplyr, tidyr, readr, purrr, tibble, hms, stringr, lubridate, forcats, DBI, haven, httr, jsonlite, readxl, rvest, xml2, modelr, tidyverse).

How would you install them?

Tidyverse Pop Quiz!

```
install.packages("ggplot2") install.packages("dplyr")
install.packages("tidyr") install.packages("readr")
install.packages("purrr") install.packages("tibble")
install.packages("hms") install.packages("stringr")
install.packages("lubridate") install.packages("forcats")
install.packages("DBI") install.packages("haven")
install.packages("httr") install.packages("jsonlite")
install.packages("readxl") install.packages("rvest")
install.packages("xml2") install.packages("modelr")
install.packages("broom")
```

Better:

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

An R package that serves as a short cut for installing and loading the components of the tidyverse.

Using Packages (Part II)

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

Do this 1 time per *computer*.

2. `library(tidyverse)`

Do this 1 time per *session*

Using Packages (Part II)

Downloading a package isn't the same as using it.

If you'd like to use an R package, you need to tell R.
You do that by running the command `library`, again followed by parentheses and the package name.

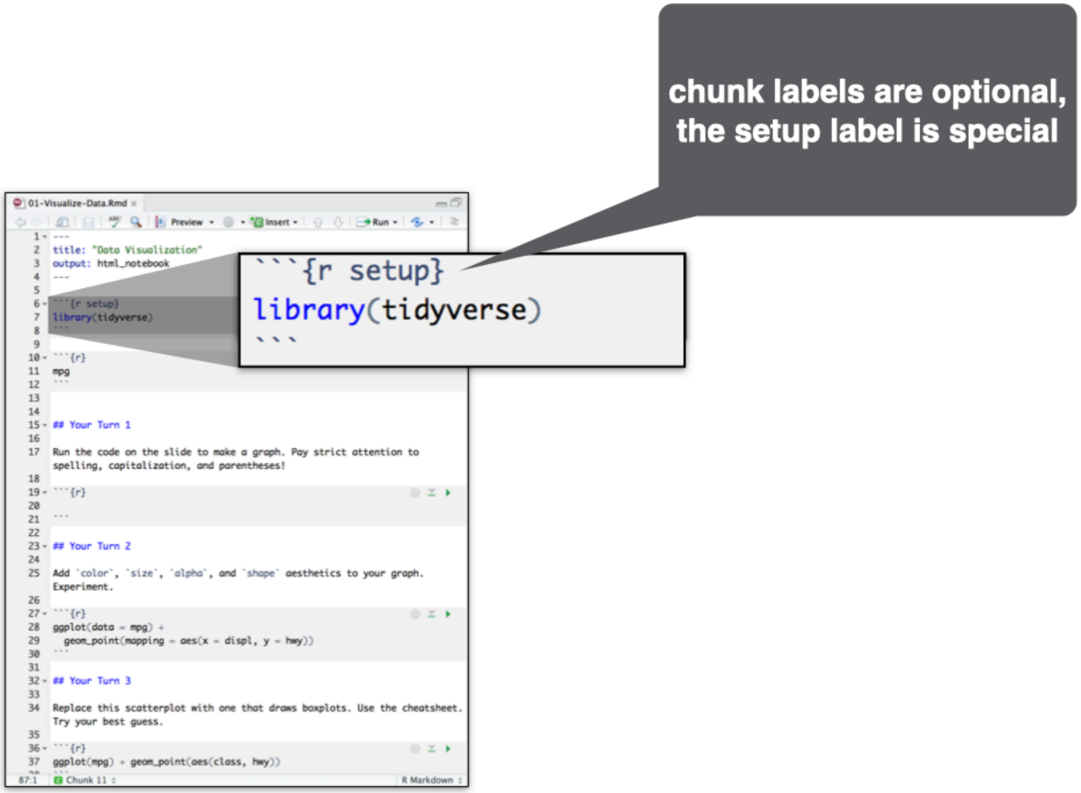
```
library(package_name)
```

This command loads all of the functions, data sets, and help pages that come with the package into your R session, where you can use them.

If you close R, you'll need to reload the package with library() if you want to use it again.

Using Packages (Part III)

The setup chunk is always run once before anything else



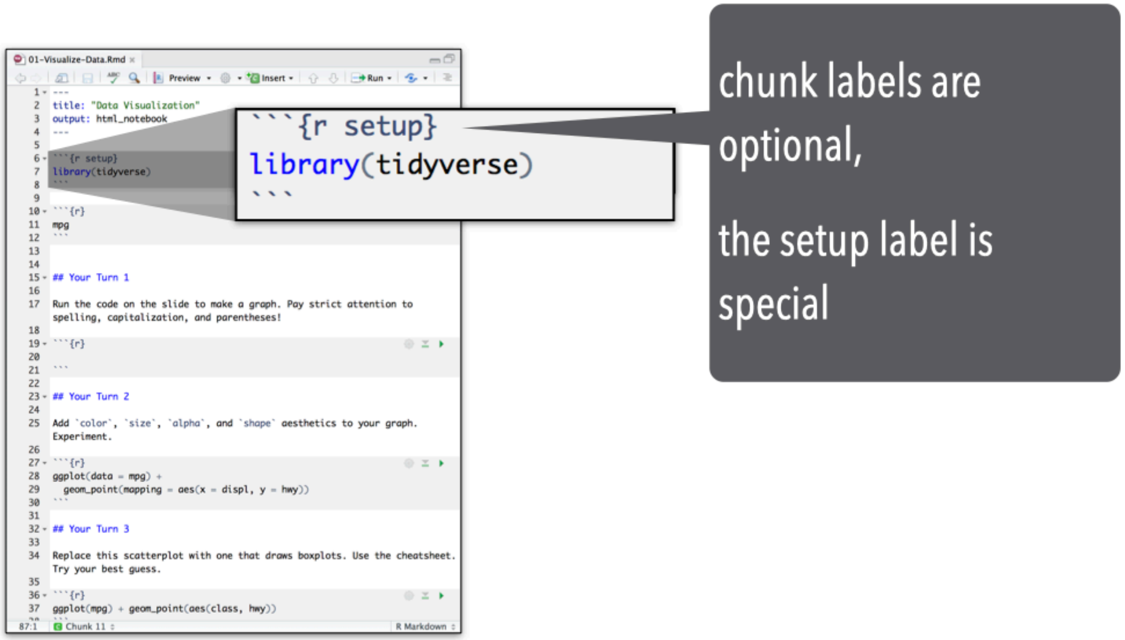
~~~
---
title: "Data Visualization"
output: html_notebook
---

```{r setup}
library(tidyverse)
```

```{r}
mpg
```


## Your Turn 1

Run the code on the slide to make a graph. Pay strict attention to
spelling, capitalization, and parentheses!

```{r}

```

## Your Turn 2

Add `color`, `size`, `alpha`, and `shape` aesthetics to your graph.
Experiment.

```{r}
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy))
```

## Your Turn 3

Replace this scatterplot with one that draws boxplots. Use the cheatsheet.
Try your best guess.

```{r}
ggplot(mpg) + geom_point(aes(class, hwy))
```
~~~

~~~
```{r setup}
library(tidyverse)
```
~~~

chunk labels are optional, the setup label is special

Set-up Chunk Exercise

Add a setup chunk (as shown below) to the top of `02_rmd_exercise.Rmd`.
Use it to load the tidyverse package (*remember to run this chunk*)
Then uncomment and run the final code chunk at the bottom of your file.



~~~
---
title: "Data Visualization"
output: html_notebook
---

```{r setup}
library(tidyverse)
```

```{r}
mpg
```


## Your Turn 1

Run the code on the slide to make a graph. Pay strict attention to
spelling, capitalization, and parentheses!

```{r}

```

## Your Turn 2

Add `color`, `size`, `alpha`, and `shape` aesthetics to your graph.
Experiment.

```{r}
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy))
```

## Your Turn 3

Replace this scatterplot with one that draws boxplots. Use the cheatsheet.
Try your best guess.

```{r}
ggplot(mpg) + geom_point(aes(class, hwy))
~~~

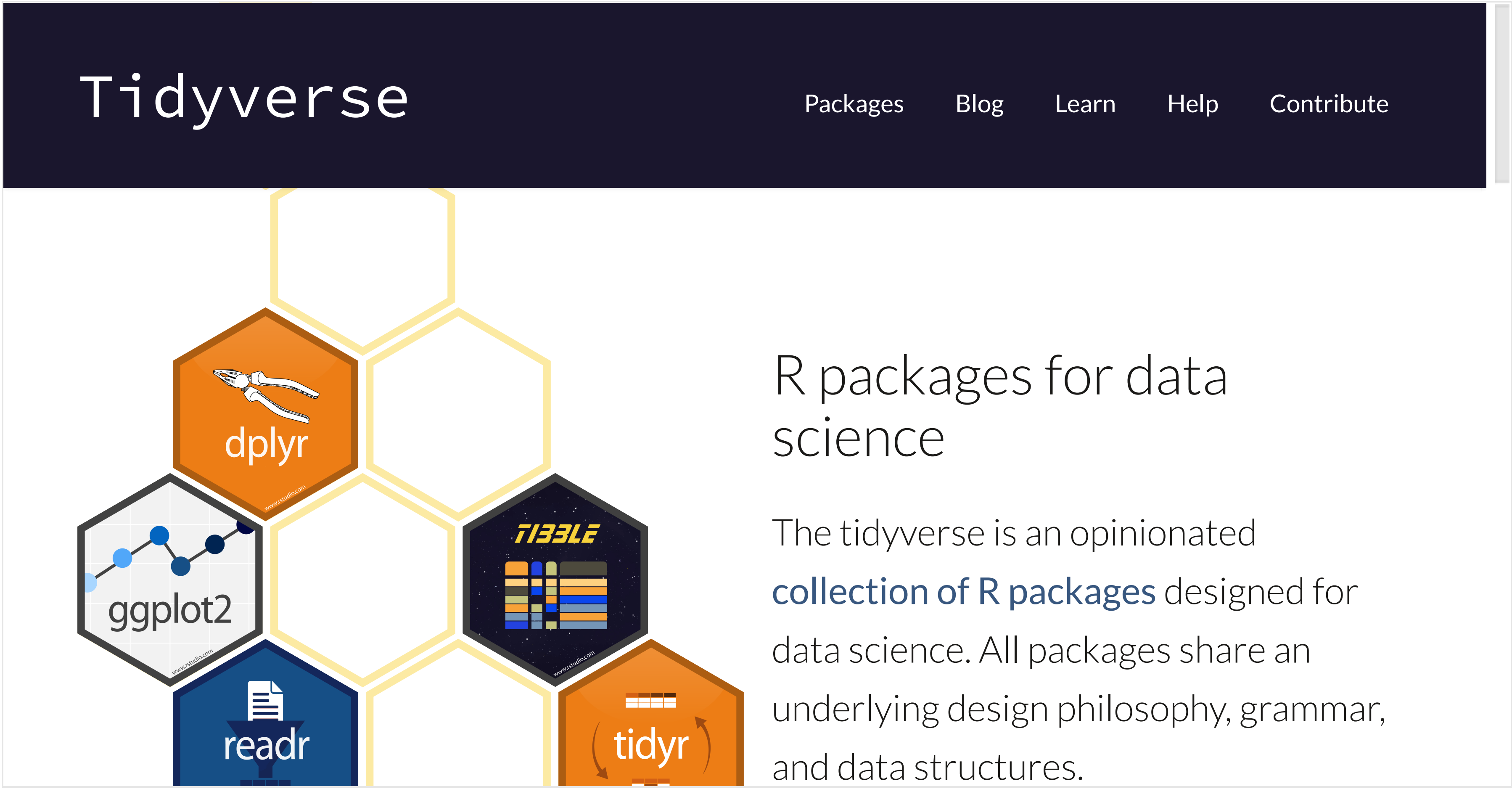
~~~
```{r setup}
library(tidyverse)
```
~~~

chunk labels are optional,

the setup label is special

Tidyverse

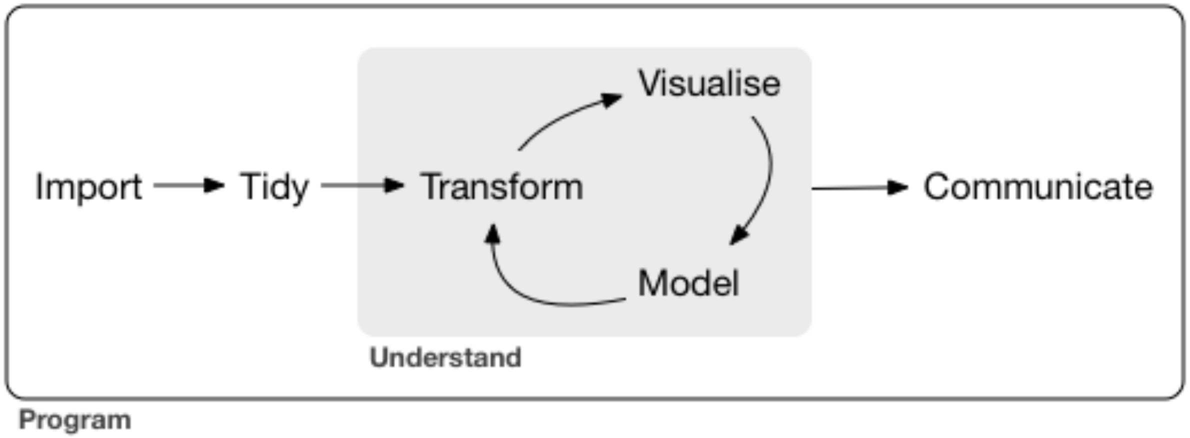
Tidyverse is one suite of tools for data science



R packages for data science

The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

Exploratory Data Analysis



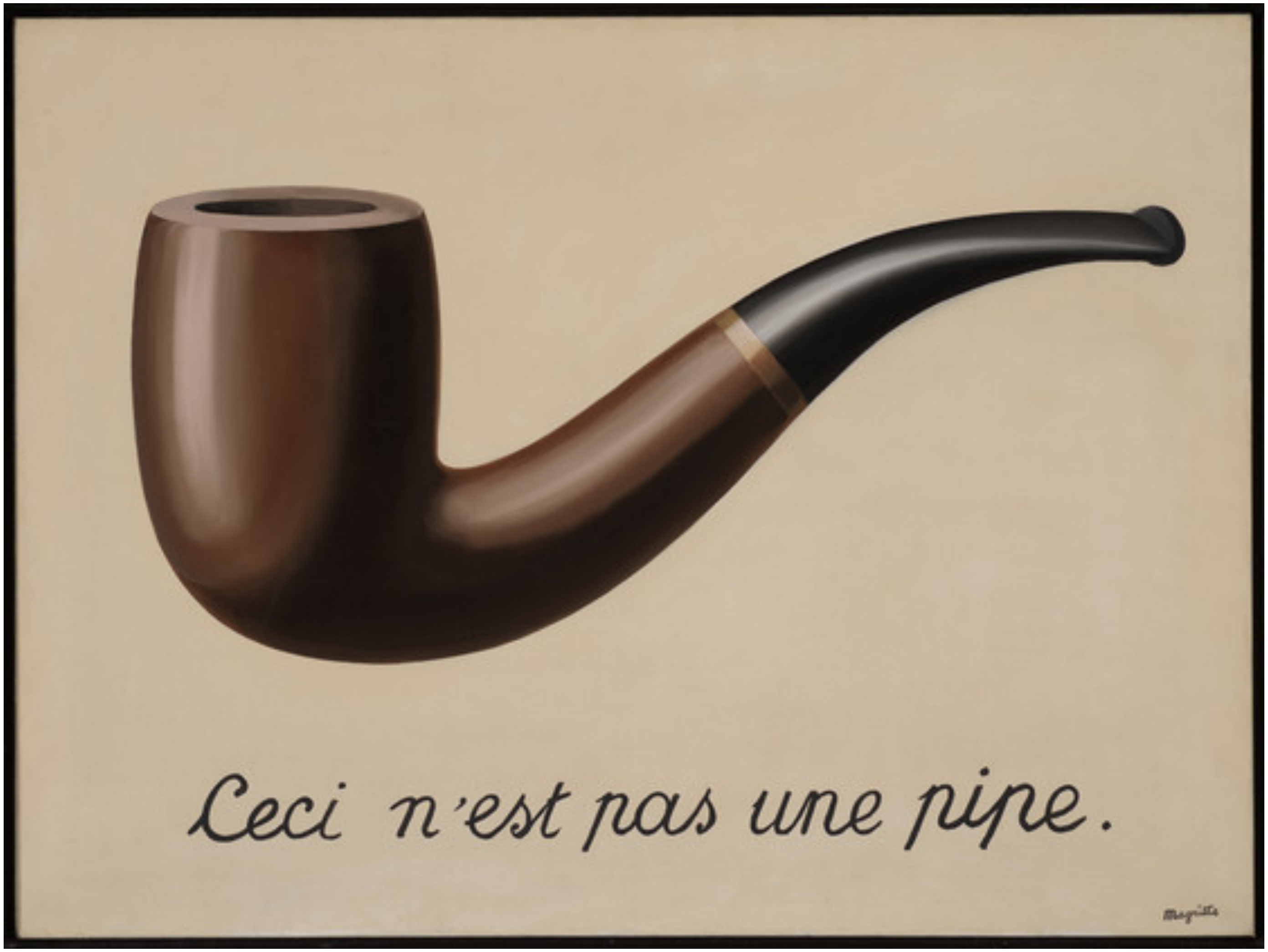
Tidyverse Basic Principles

IMPORT (readr):
- `read_csv()`
- `read_delim()`

TIDY & TRANSFORM (dplyr):
- `mutate()` adds new variables that are functions of existing variables
- `select()` picks variables based on their names.
- `filter()` picks cases based on their values.
- `summarise()` reduces multiple values down to a single summary.
- `arrange()` changes the ordering of the rows.

VISUALISE (ggplot): creating graphics, based on ‘The Grammar of Graphics’
- `aes()`
- `geom_x()` + layers

MODEL (broom):
- `tidy()`, `glance()`, `augment()`


Ceci n'est pas une pipe.
Magritte

`magrittr` package

`magrittr` package

`magrittr` package by Stefan Milton Bache developed the concept of the pipe, which is used heavily in the `tidyverse`

“and then”

The Pipe

The “pipe” is a sequence of functions, that are sequentially applied to an object

```
wakeup(self) %>%
  put_on("clothes") %>%
  eat("breakfast") %>%
  go(to = "work")
```

Alternative nested code:

```
go(eat(put_on(wakeup(self), "clothes"), "breakfast"), to = "work")
```

The Pipe (Quiz I)

What does this code do?

```
wakeup(self) %>%
  put_on("clothes") %>%
  eat("breakfast") %>%
  fmk() %>%
  go(to = "work")
```

The Pipe (Quiz II)

What does this code do?

```
morning_routine <- wakeup(self) %>%
  put_on("clothes") %>%
  eat("breakfast") %>%
  fmk() %>%
  go(to = "work")
```

The Pipe (Quiz III)

What does this code do?

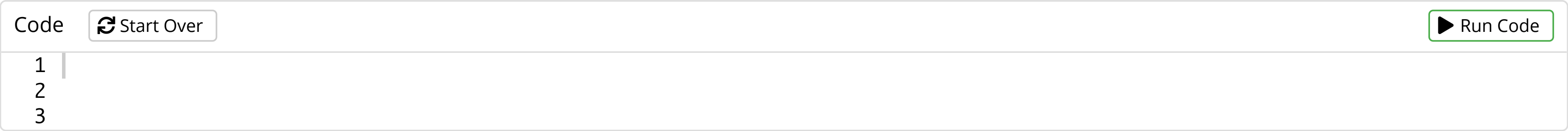
```
morning_routine <- wakeup(self) %>%
  put_on("clothes") %>%
  eat("breakfast") %>%
  fmk() %>%
  go(to = "work")
```

The Pipe (Quiz IV)

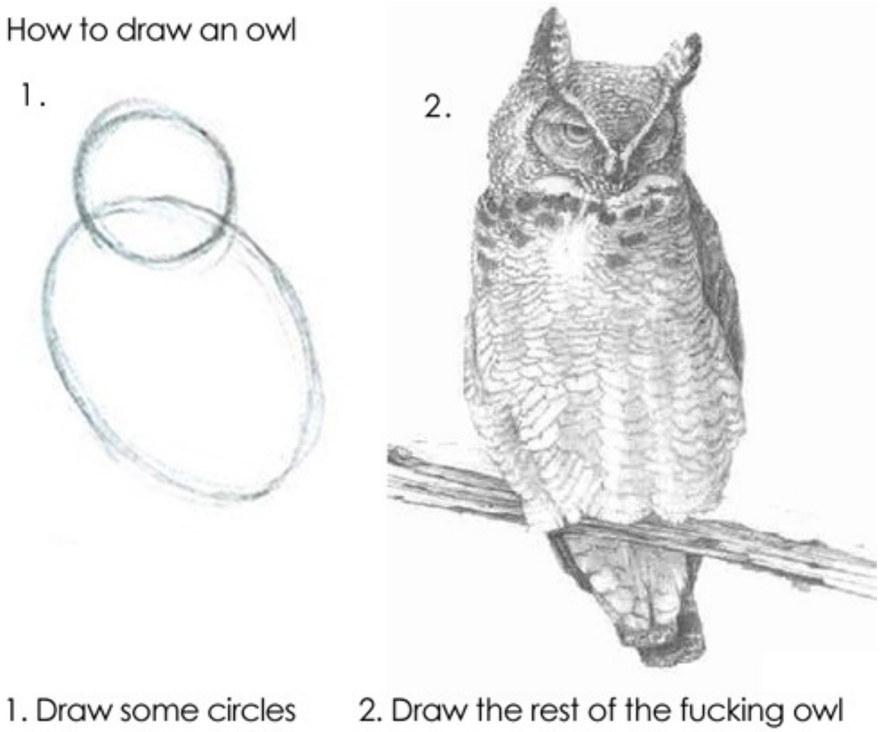
What does this code do?

```
heart %>%
select(age, sex) %>%
group_by(sex) %>%
summarize(mean(age))
```

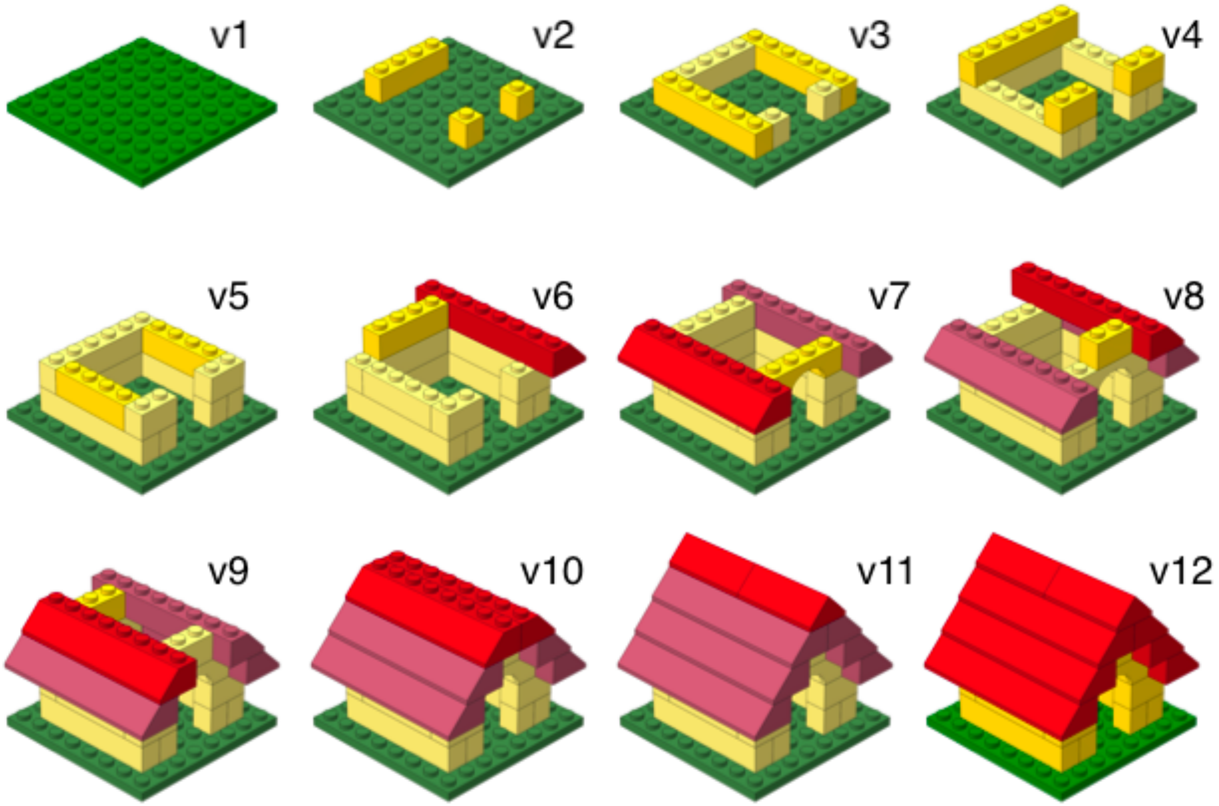
Try it out here!



Writing code IS NOT like drawing an owl



Writing code IS a step-wise process



https://datasciencebox.org (https://datasciencebox.org)

Any questions?

R
Getting to know your data

`heart` dataset

Heart Disease Dataset

- For this class, we will use a dataset called `heart`.
- This data set dates from 1988 and consists of four databases: Cleveland, Hungary, Switzerland, and Long Beach V.
- The complete data set contains 76 attributes, including a predicted attribute generated by ML models to predict heart failure. However, all published experiments refer to using a subset of 14 attributes, which we will use for this class

Inspecting your dataframe: dimensions

Use the `dim()` function to see how many rows (observations) and columns (variables) are in `heart`

```
dim(heart)
```

Enter your function here:

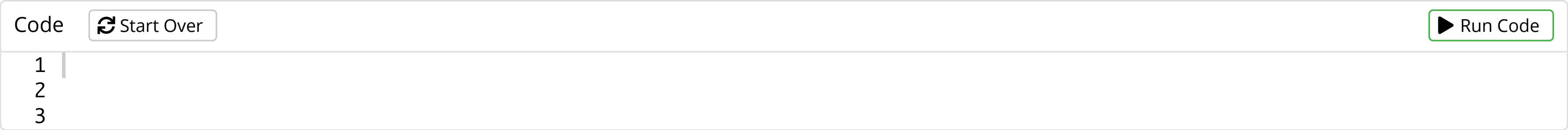
Code Start Over ▶ Run Code

Inspecting your dataframe: glimpse

Use the `glimpse()` function to see what kinds of variables the `heart` dataset contains

```
glimpse(heart)
```

Enter your function here:



Basic Data Types in R

Recall that R has different data types -

character - `"a"`, `"tidyverse"`

numeric - `2`, `11.5`

integer - `2L` (the `L` tells R to store this as an integer)

logical - `TRUE`, `FALSE`

complex - `1+4i`

(**raw**)

You will also come across the **double** datatype. It is the same as **numeric**

factor. A **factor** is a collection of *ordered* character variables

Basic Data Types in R

In addition to the `glimpse()` function, you can use the `class()` function to determine the data type of a specific column

```
class(heart$sex)
```

```
## [1] "character"
```

Try getting the `class` of a variable:

Code   Start Over   ▶ Run Code

(Re)Introducing `%>%`

The `%>%` operator is a way of “chaining” together strings of commands that make reading your code easy.
The following code chunk illustrates how `%>%` works:

```
heart %>%
select(sex, age) %>%
filter(sex == "male") %>%
head()
```

```
## # A tibble: 6 x 2
##   sex     age
##   <chr> <dbl>
## 1 male     52
## 2 male     53
## 3 male     70
## 4 male     61
## 5 male     58
## 6 male     55
```

(Re)Introducing `%>%`

The previous code chunk does the following - it takes your dataset and then "pipes" it into `select()`, and then applies a `filter()` to the data.

```
heart %>%
select(sex, age) %>%
filter(sex == "male") %>%
head()
```

the `head` function lists only the top n results – convenient for long variables

When you see `%>%`, think "and then"

(Re)Introducing `%>%`

The alternative to using `%>%` is running the following code

filter(select(df_input, sex, age), sex == “male”)

Although this is only one line as opposed to three, it’s both more difficult to write and more difficult to read

Any questions?

R
Manipulating your data

Introducing `dplyr`

dplyr is a package that contains a suite of functions that allow you to easily manipulate a dataset

Some of the things you can do are -

- select rows and columns that match specific criteria
- create new variables (columns)
- obtain summary statistics on individual groups within your datsets

The main verbs we will cover are `select()`, `filter()`, `arrange()`, `mutate()`, and `summarise()`. These all combine naturally with `group_by()` which allows you to perform any operation “by group”

`select()` specific columns from your dataset

The most basic `select()` is one where you comma separate a list of columns you want included

For example, if you only want to select the sex and age columns, run the following code chunk

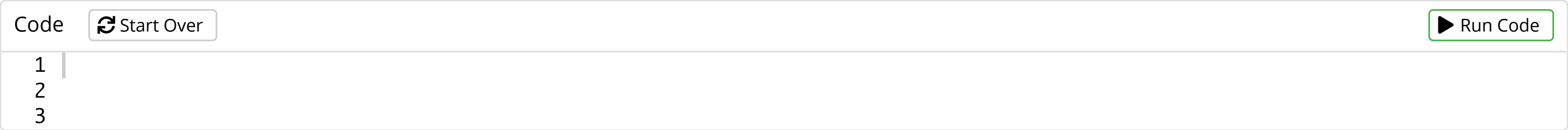
```
heart %>%
select(sex, age) %>%
head()
```

Code Start Over ▶ Run Code

`select()`

If you want to select all columns *except* sex, run the following

```
heart %>%
select(-sex) %>%
head(5)
```



select()

Finally, you can provide a range of columns to return two columns and everything in between. For example

```
heart %>%
select(sex:age) %>%
head(5)
```

Code ⟳ Start Over ▶ Run Code

`filter()` rows based on certain condition(s)

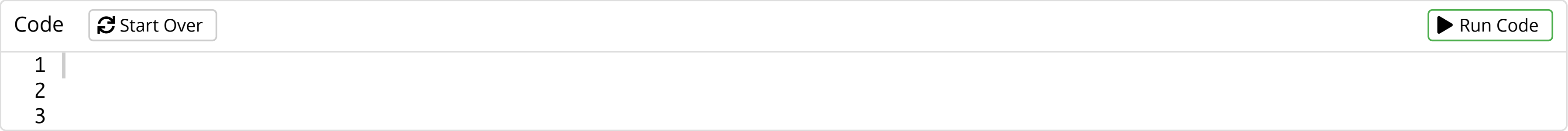
The `filter()` verb evalutes a logical statement, and if a row meets the condition of this statement (i.e. is true) then it gets chosen (or “filtered”).

All other rows are discarded

`filter()`

Filtering can be performed on **categorical** data

```
heart %>%
filter(sex == "male") %>%
head(3)
```



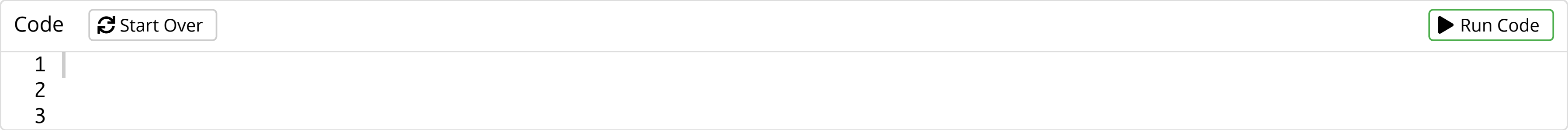
Note that `filter()` only applies to rows, and has no effect on columns

`filter()`

Filtering can also be performed on **numeric** data

For example, if you wanted to choose age with a value greater than 53, you would run the following.

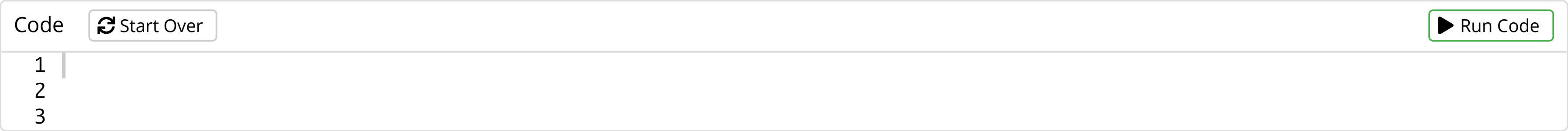
```
heart %>%
filter(age > 53) %>%
head(3)
```



`filter()`

To filter on multiple conditions, you can write a sequence of `filter()` commands

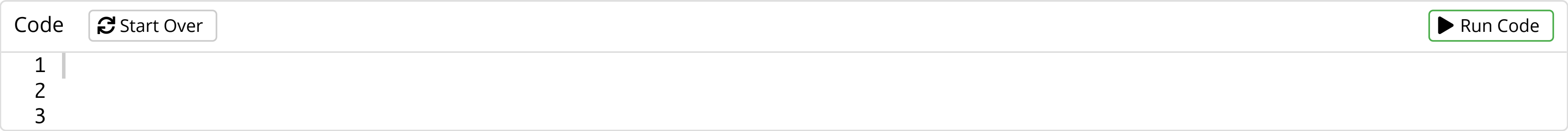
```
heart %>%
filter(sex == "male") %>%
filter(age > 53) %>%
head(3)
```



filter()

To avoid writing multiple `filter()` commands, multiple logical statements can be put inside a single `filter()` command, separated by commas

```
heart %>%
filter(sex == "male",
age > 53) %>%
head(3)
```

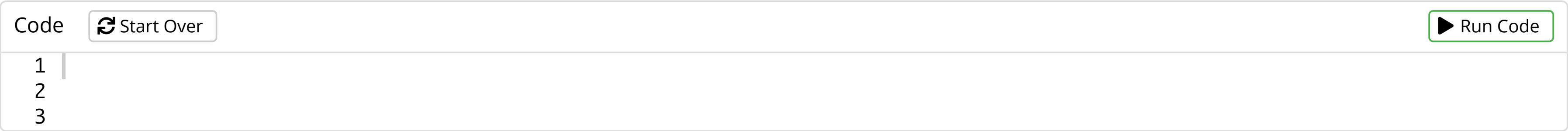


`arrange()` sorts rows

The input for arrange is one or many columns, and `arrange()` sorts the rows in ascending order i.e. from smallest to largest

For example, to sort rows from smallest to largest age, run the following

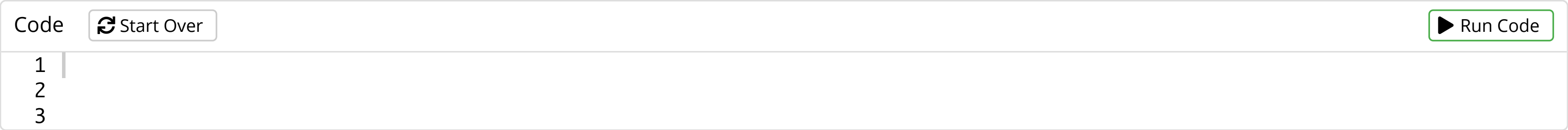
```
heart %>%
arrange(age) %>%
head(3)
```



arrange()

To reverse this order, use the `desc()` function within `arrange()`

```
heart %>%
arrange(desc(age)) %>%
head(3)
```



mutate()

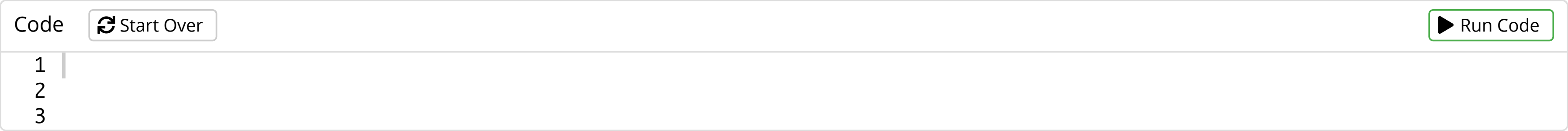
The `mutate()` verb, unlike the ones covered so far, creates new variable(s) i.e. new column(s). For example

```
heart %>%
mutate(new_col = sqrt(age)) %>%
head(1)
```

The code chunk above takes all the elements of the column age, evaluates the square root of each element, and populates a new column called `new_col` with these results

Try `mutate()` to make a new column

```
heart %>%
mutate(new_col = sqrt(age)) %>%
head(3)
```



`summarise()` and `summarize()`

`summarize()` is one of the key functions in `dplyr`. It produces a new dataframe that aggregates that values of a column based on a certain condition.

For example, to calculate the mean age, run the following

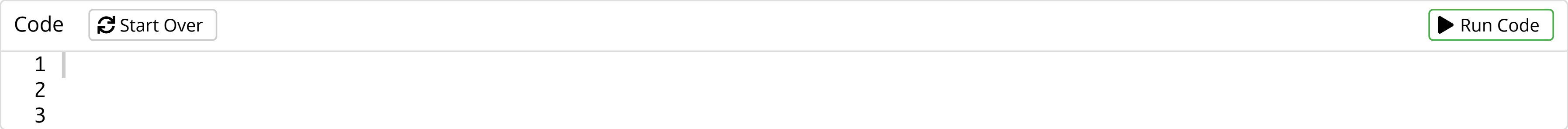
```
heart %>%
summarise(mean(age))
```

Code Start Over Run Code

group_by()

`group_by()` and `summarize()` are a powerful combination of functions to summarize by groups

```
heart %>%
group_by(sex) %>%
summarise(mean(age))
```



*remember this pair! ## Saving a new dataset as an object

If you'd like to save the output of your wrangling as an object, you will need to use the <- operator

```
heart_new <- heart %>%
group_by(sex) %>%
summarise(mean(age))
```

*Assigning the object the same name (i.e. `heart`) will overwrite the object

Saving a new dataset as a file

To save `heart_new` as a new file (e.g. csv), run the following:

```
write_csv(heart_new, "heart_new.csv")
```

Any questions?

R
Data Science Workflow

Visualizing our dataset

Go to code/
Open 03_rmd_practice.Rmd
Complete the activity.

Any questions?

R
Help

Resources

- Rstudio package ‘cheatsheets’
- Package function help pages: `?mean`, or navigate to the Help tab and search there
- Run the following to access the Dplyr vignette: `browseVignettes("dplyr")`
- Stackoverflow: https://stackoverflow.com (https://stackoverflow.com)
- R for Data Science, by Grolemund & Wickham https://r4ds.had.co.nz/index.html (https://r4ds.had.co.nz/index.html)

Acknowledgements

Teaching Assistants
- Allie Mills, Ph.D.
- Akshay Bareja, D.Phil.

Inspiration, ideas, packages, code
- R4DS (Garrett Grolemund and Hadley Wickham)
- Mine Çetinkaya-Rundel (datasciencebox.org)
- Chester Ismay and Albert Y. Kim (Modern Dive)
- Garrett Grolemund (Remastering the Tidyverse)
- Tidyverse devs and community
- Rstudio

Any questions?

Thank you