

Finding your way in R and RStudio

Data Visualization: Session 2

Kieran Healy
Code Horizons, May 2022

We want to
draw graphs
reproducibly



Abstraction in software

Less

Easy things are awkward

Hard things are straightforward

Really hard things are possible

More

Easy things are trivial

Hard things are very awkward

Really hard things are impossible

Compare: D3, Grid, ggplot, Stata, Excel

Two ways to use R and ggplot

1. Do everything in R from start to finish

Raw data |> Read, Clean, Analyse |> Tidy table |> Make figures

Two ways to use R and ggplot

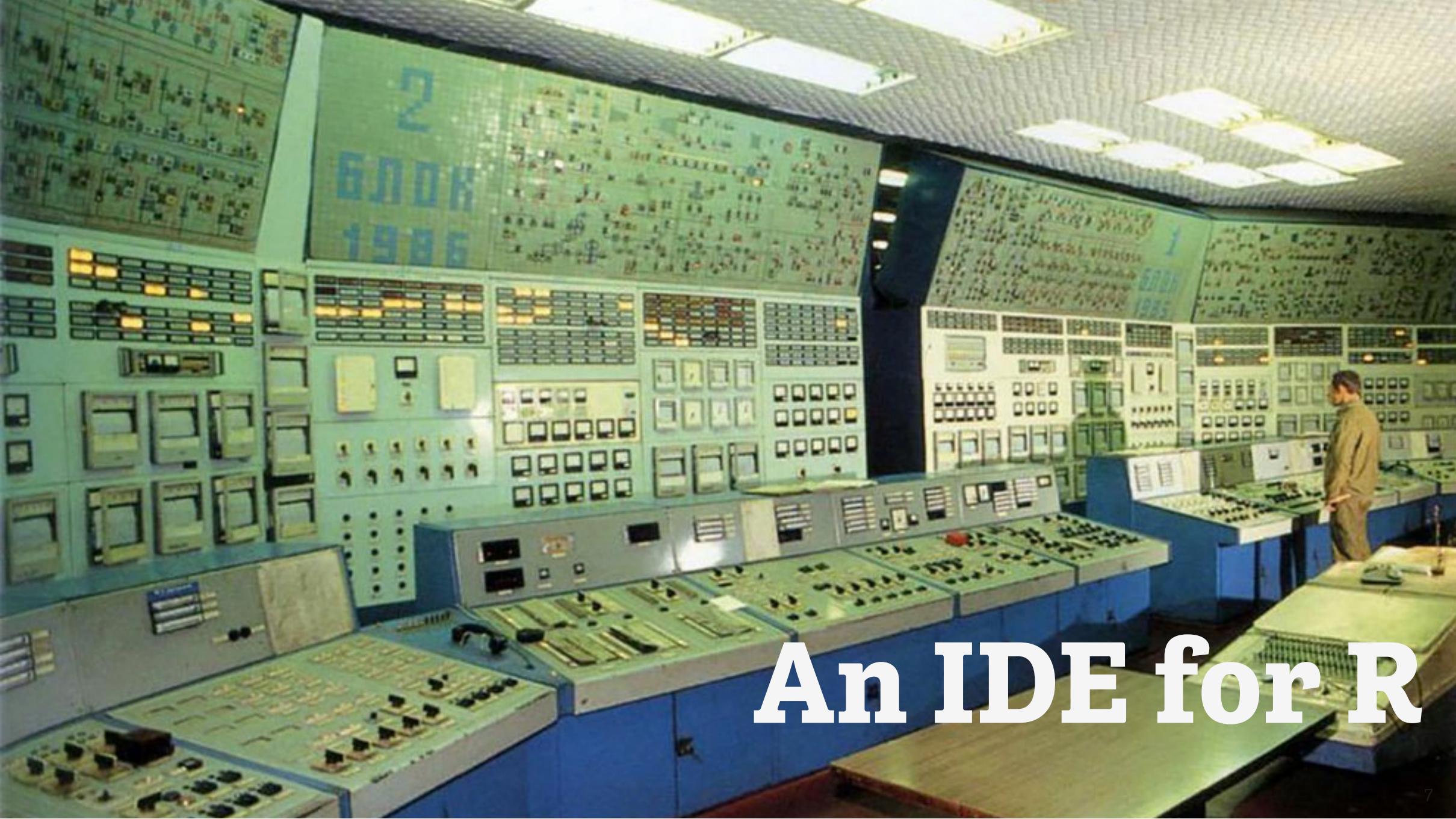
1. Do everything in R from start to finish

Raw data |> Read, Clean, Analyse |> Tidy table |> Make figures

2. Just hand ggplot a table of results

Stata/SAS/etc |> Tidy table |> Read in to R |> Make figures

The RStudio IDE



An IDE for R

A photograph of Jacques Pépin, a renowned chef, in his kitchen. He is wearing a blue apron with the text "Jacques Pépin Heart & Soul" on it. He is pouring olive oil from a glass bottle into a stainless steel bowl. The kitchen is well-stocked with various cooking utensils, ingredients like eggs and wine, and hanging copper pots. In the foreground, there's a wooden cutting board with a bowl of salad and some knives.

An IDE for Meals

The screenshot shows the RStudio interface at startup. The top menu bar includes Apple, RStudio, File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, Window, and Help. The title bar says "covdata - main - RStudio".

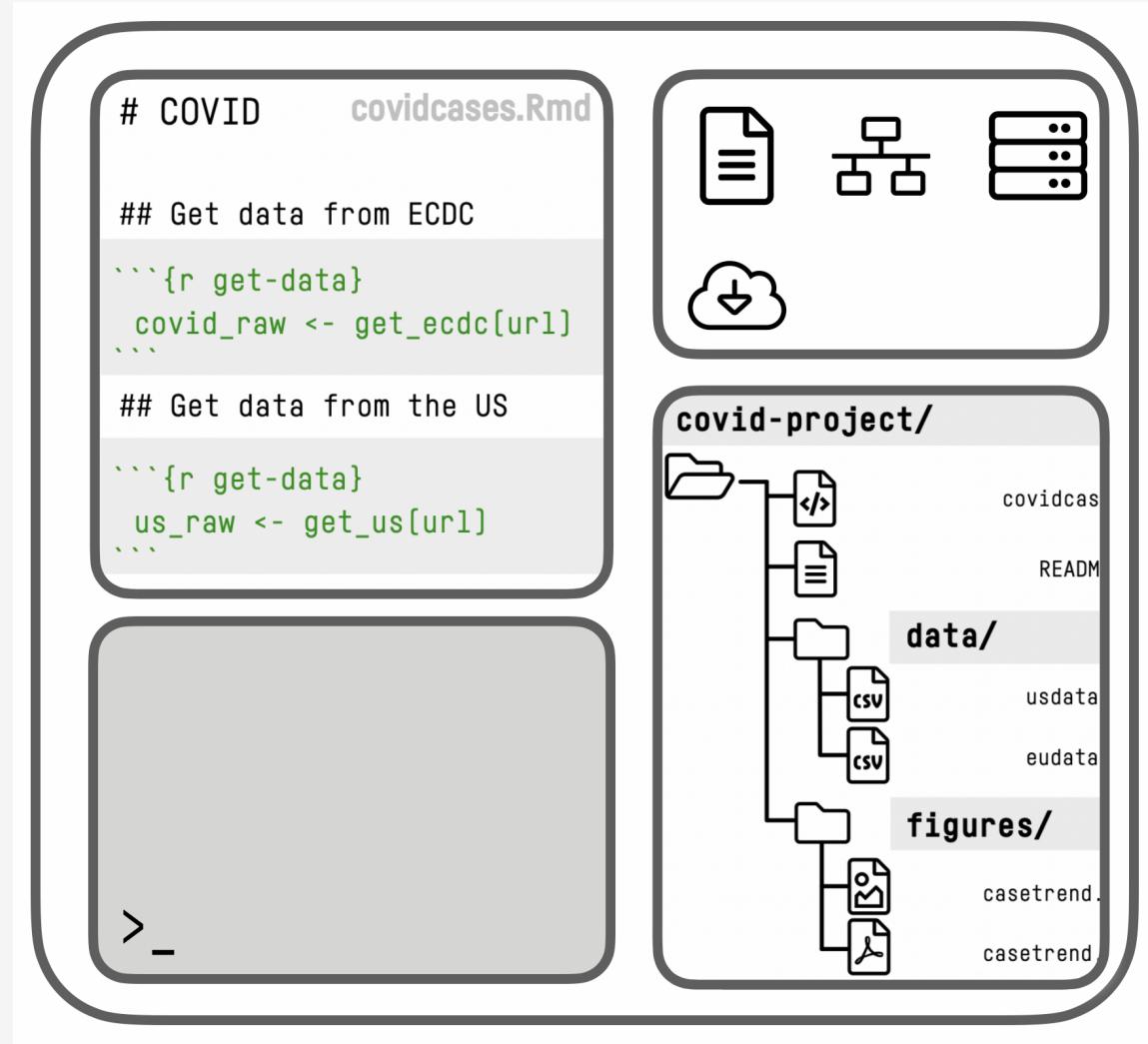
Code Editor: The left panel displays the file "covdata.Rmd" with R code. Lines 15-24 show the loading of the covdata package. Lines 26-28 describe the package's purpose and how it provides datasets in tibbles.

Environment: The top right pane shows the Global Environment and Functions. A single entry "set" is listed under Functions.

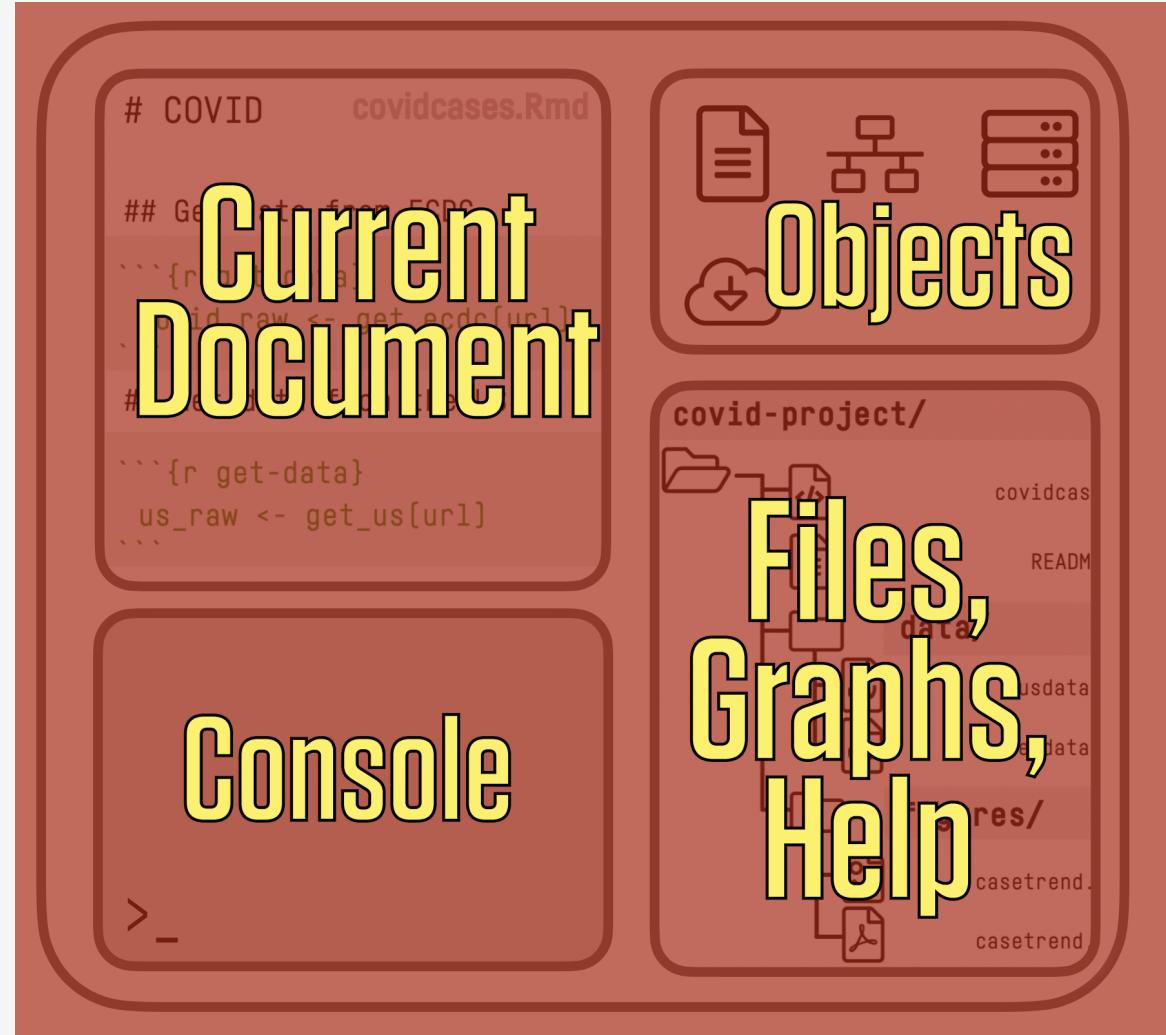
Console: The bottom left pane shows the R console output. It starts with a welcome message from R, followed by help information, the loading of the covdata package, and the attaching of the testthat package. It also lists masked objects from devtools.

File Browser: The bottom right pane shows the file structure of the covdata project. The contents include .github, .gitignore, .Rbuildignore, .Rhistory, _pkgdown.yml, _sinewconfig.yml, covdata.Rproj, data, data-raw, DESCRIPTION, inst, LICENSE, LICENSE.md, man, and NAMESPACE files.

RStudio at startup



RStudio schematic overview



RStudio schematic overview

**Think in terms of
Data + Transformations,
written out as code,
rather than a series of
point-and-click steps**

Our starting **data** + our
code is what's "real" in
our projects, not the
final output or any
intermediate objects

The screenshot shows the RStudio interface at startup. The top menu bar includes Apple, RStudio, File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, Window, and Help. The title bar says "covdata - main - RStudio".

Code Editor: The left pane displays the R Markdown file "covdata.Rmd". The code includes documentation for the "covdata" package and its setup section.

```
15
16
17 ## Loading the Package
18
19 The 'covdata' package aims to make data related to the COVID-19
  pandemic easily accessible to users of R. Once the package is
  installed, load it in the usual way:
20
21 ```{r setup}
22 library(tidyverse)
23 library(covdata)
24 ```
25
26 Loading the package makes a variety of datasets available for use.
  Because the data are in tibbles, the use of the 'tidyverse' suite of
```

Environment: The right pane shows the Global Environment and Functions sections. The "set" function is listed.

Console: The bottom-left pane shows the R console output.

```
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

Knitr hook "anchor" is now available
Loading required package: testthat

Attaching package: 'testthat'

The following object is masked from 'package:devtools':
  test_file
```

File Browser: The bottom-right pane shows the file structure of the "covdata" directory.

Name	Size	Modified
..		
.github	49 B	Apr 28, 2020, 2:03 PM
.gitignore	125 B	Jul 6, 2020, 9:00 AM
.Rbuildignore	20 KB	Aug 18, 2020, 12:18 PM
.Rhistory	2 KB	Jul 17, 2020, 1:04 PM
_pkgdown.yml	172 B	Apr 20, 2020, 2:57 PM
_sinewconfig.yml	395 B	Aug 21, 2020, 10:44 AM
covdata.Rproj	871 B	Aug 17, 2020, 12:59 PM
data		
data-raw		
DESCRIPTION	42 B	Apr 20, 2020, 2:57 PM
inst	1 KB	Apr 20, 2020, 2:57 PM
LICENSE		
LICENSE.md		
man		
NAMESPACE	129 B	Aug 17, 2020, 1:36 PM

RStudio at startup

Paper, Report,
Analysis, Notes, etc,
in RMarkdown

```
1 ---  
2 title: "Data Visualization"  
3 author: "Kieran Healy"  
4 date: "10-January-2020"  
5 output: html_document  
6 ---  
7  
8 ## Data Visualization Notes  
9  
10 This is a starter RMarkdown project template to accompany courses taught with *Data Visualization*. You can use it to take notes, write your code, and produce a good-looking, reproducible document that records the work you have done. At the very top of the file is a section of *metadata*, or information about what the file is and what it does. The metadata is delimited by three dashes at the start and another three at the end. You should change the title, author, and date to the values that suit you. Keep the 'output' line as it is for now, however. Each line in the metadata has a structure. First the *key* ("title", "author", etc), then a colon, and then the *value* associated with the key.  
11  
12 ## This Document is an RMarkdown File  
13  
14 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>.  
15  
16 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. A *code chunk* is
```

Console | Jobs

```
~/Documents/courses/stathorizons_0820/ ~  
Type 'demo()' for some demos, 'help()' for on-line help, or  
'help.start()' for an HTML browser interface to help.  
Type 'q()' to quit R.  
  
knitr hook "anchor" is now available  
Loading required package: testthat  
  
Attaching package: 'testthat'  
  
The following object is masked from 'package:devtools':  
  
 test_file  
  
> |
```

Environment | History | Connections | Git | Tutorial

Global Environment

Environment is empty

Files | Plots | Packages | Help | Viewer

New Folder | Delete | Rename | More

Name	Size	Modified
.gitignore	40 B	Jul 21, 2020, 11:16 AM
01_introduction.Rmd	4 KB	Jul 21, 2020, 11:16 AM
02_get_started.Rmd	3.3 KB	Jul 21, 2020, 11:16 AM
03_make_a_plot.Rmd	5.8 KB	Jul 21, 2020, 11:16 AM
04_show_the_right_numbers.Rmd	4.8 KB	Jul 21, 2020, 11:16 AM
05_tables_and_labels.Rmd	9.9 KB	Jul 21, 2020, 11:16 AM
06_models.Rmd	15 KB	Jul 21, 2020, 11:16 AM
07_maps.Rmd	12.5 KB	Jul 21, 2020, 11:16 AM
08_refine_plots.Rmd	21.7 KB	Jul 21, 2020, 11:16 AM
09_supplementary_material.Rmd	16.9 KB	Jul 21, 2020, 11:16 AM
assets		
data		
figures		
keynote		
LICENSE.md	18.1 KB	Jul 21, 2020, 11:16 AM
materials		
README.md	5.7 KB	Jul 21, 2020, 11:16 AM
slides		
stathorizons_0820.Rproj		Jul 22, 2020, 8:50 AM

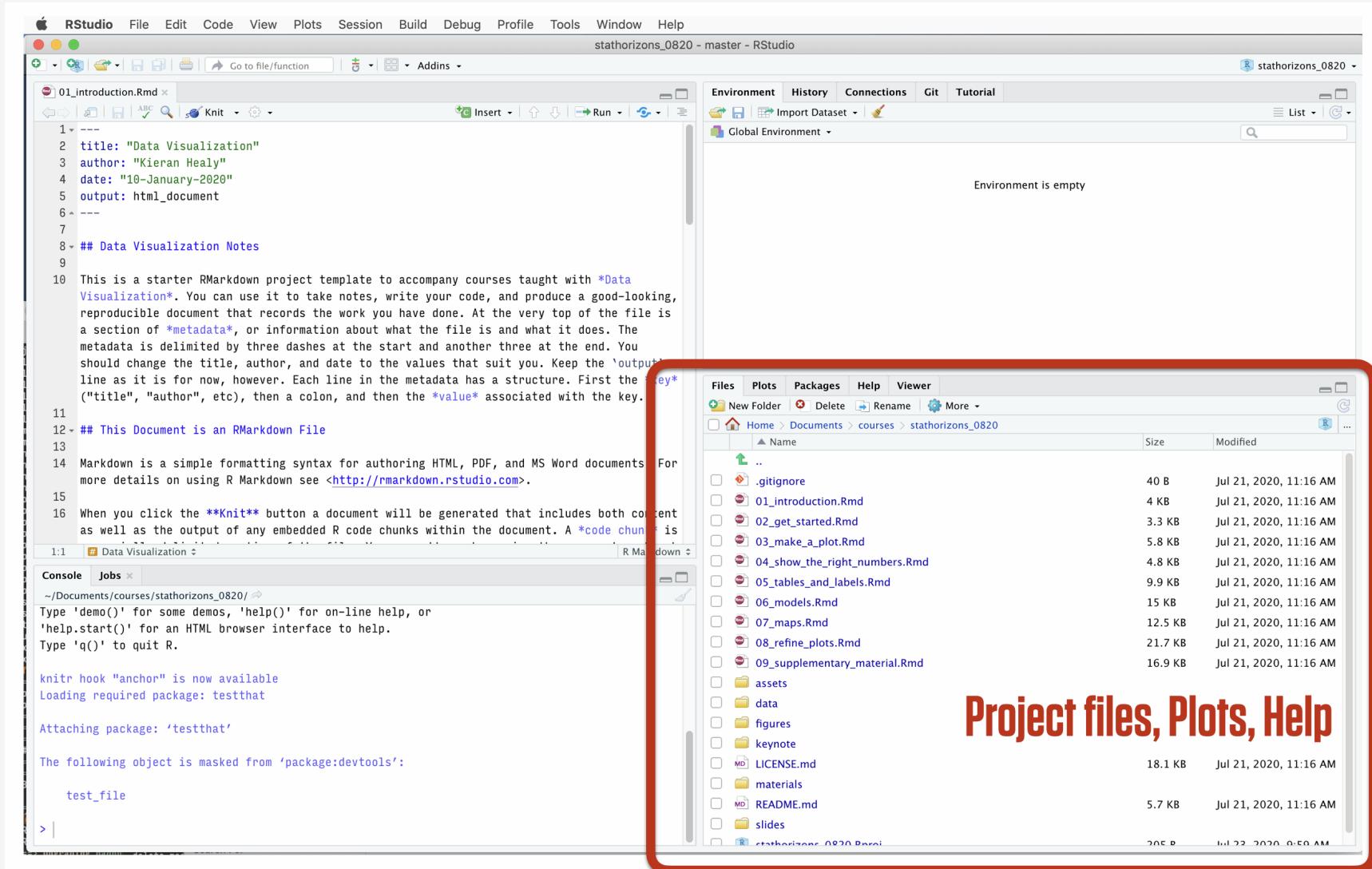
RStudio at startup

The screenshot shows the RStudio interface at startup. The main window title is "stathorizons_0820 - master - RStudio". The left pane displays the file "01_introduction.Rmd" with R Markdown code. The right pane shows the "Environment" tab with a message "Environment is empty". Below the environment is a file browser with a list of files in the directory "Home > Documents > courses > stathorizons_0820". The bottom-left pane is the "Console" tab, which is highlighted with a red border. It contains the following text:

```
~/Documents/courses/stathorizons_0820/ ~  
Type 'demo()' for some demos, 'help()' for on-line help, or  
'help.start()' for an HTML browser interface to help.  
Type 'q()' to quit R.  
  
knitr hook "anchor" is now available  
Loading required package: testthat  
  
Attaching package: 'testthat'  
  
The following object is masked from 'package:devtools':  
  
 test_file  
  
> |
```

Console: Type or send code here, see results

RStudio at startup



Project files, Plots, Help

RStudio at startup

The screenshot shows the RStudio interface at startup. The left pane contains an R Markdown file named '01_introduction.Rmd' with code and comments. The right pane shows the 'Environment' tab of the Global Environment viewer, which displays the message 'Environment is empty'. A large red box highlights the Environment tab. Overlaid on the right pane is the text 'Inspect objects you create' in a large, bold, red font.

Environment is empty

Inspect objects you create

Name	Size	Modified
..		
.gitignore	40 B	Jul 21, 2020, 11:16 AM
01_introduction.Rmd	4 KB	Jul 21, 2020, 11:16 AM
02_get_started.Rmd	3.3 KB	Jul 21, 2020, 11:16 AM
03_make_a_plot.Rmd	5.8 KB	Jul 21, 2020, 11:16 AM
04_show_the_right_numbers.Rmd	4.8 KB	Jul 21, 2020, 11:16 AM
05_tables_and_labels.Rmd	9.9 KB	Jul 21, 2020, 11:16 AM
06_models.Rmd	15 KB	Jul 21, 2020, 11:16 AM
07_maps.Rmd	12.5 KB	Jul 21, 2020, 11:16 AM
08_refine_plots.Rmd	21.7 KB	Jul 21, 2020, 11:16 AM
09_supplementary_material.Rmd	16.9 KB	Jul 21, 2020, 11:16 AM
assets		
data		
figures		
keynote		
LICENSE.md	18.1 KB	Jul 21, 2020, 11:16 AM
materials		
README.md	5.7 KB	Jul 21, 2020, 11:16 AM
slides		
stathorizons_0820.Rproj	205 B	Jul 22, 2020, 0:50 AM

RStudio at startup

Use RMarkdown
to produce and
reproduce work

Where we want to end up

Covid Cases

Kieran Healy

4/18/2021

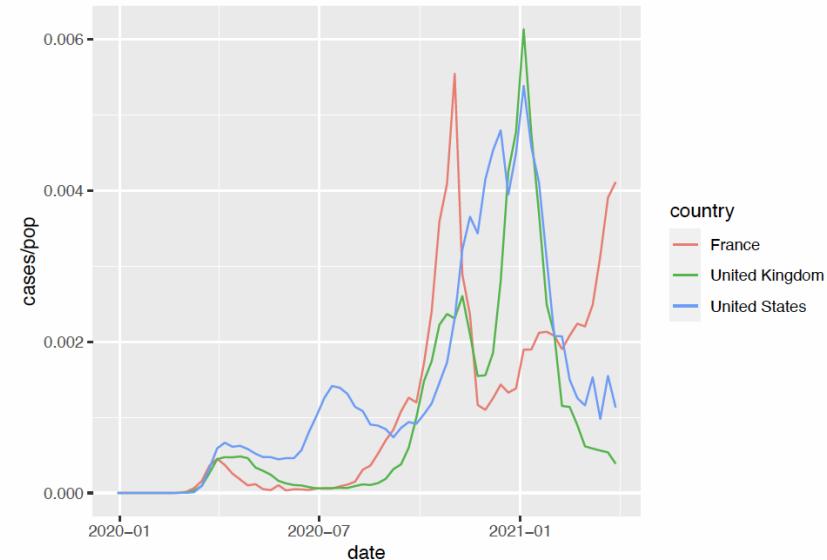
Placeholder Text

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Table 1: Total cases in three countries.

country	cases
United States	30706129
France	4822470
United Kingdom	4359388

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.



Where we want to end up

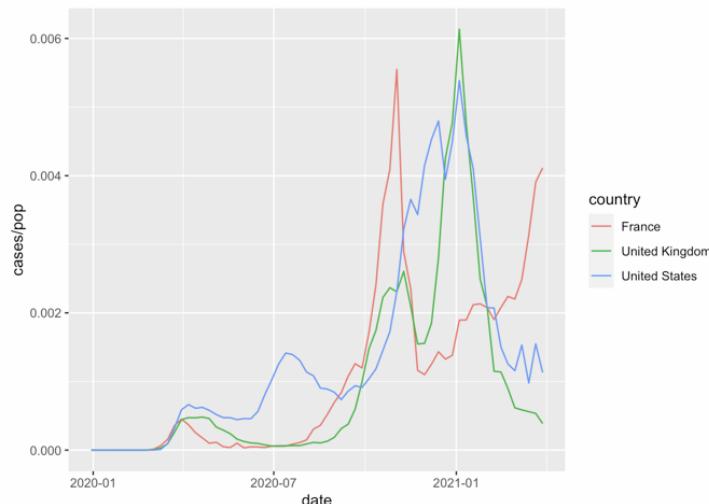
Placeholder Content

Placeholder text for the content area. This is a long paragraph of placeholder text.

Total cases in three countries.

country	cases
United States	30706129
France	4822470
United Kingdom	4359388

Placeholder text for the content area. This is a long paragraph of placeholder text.



Where we want to end up

Covid Cases

Kieran Healy

4/18/2021

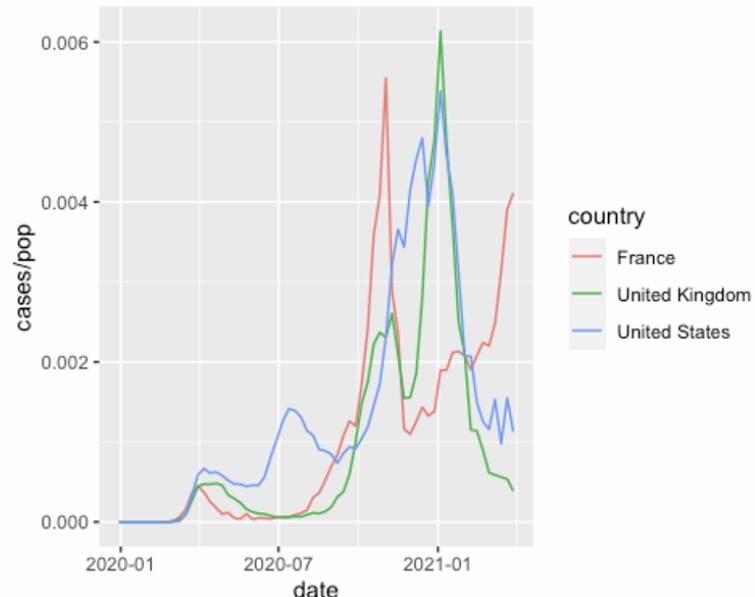
Lore ipsum

Loem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Total cases in three countries.

country	cases
United States	30706129
France	4822470
United Kingdom	4359388

Loem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.



How to get there?

```
# COVID      covidcases.R  
  
# Get data from ECDC  
# FIXME Write a fn to  
# do this  
data_raw <- read_csv(url)  
  
# Clean it  
# Notes on the cleaning  
# process.  
  
covid <- data_raw %>%  
  mutate(...) %>%  
  select(...)  
  
# Make some plots  
covid %>%  
  ggplot(...) +  
  geom_line(...)
```

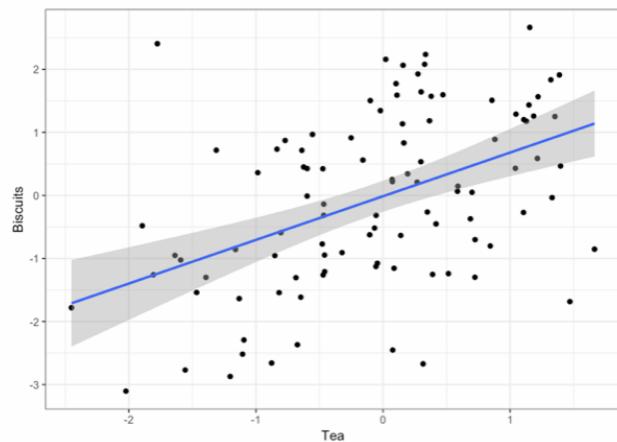
We could write an **R script** with some notes inside, using it to create some figures and tables, paste them into our document.

This will work, but we can do better.

We can **make** this ...

1. Lorem Ipsum

Lore ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enimad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.



Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

This is what we want to end up with. Nicely-formatted text, plots, and tables. In an "Office" approach we write the document and paste in the figures and tables.

... by writing this

Lorem Ipsum

 Lorem ipsum dolor sit amet, consectetur adipisicing elit,
 sed do *eiusmod tempor* incididunt ut labore et dolore magna
 aliqua. Ut enimad minim veniam, quis nostrud exercitation
 ullamco laboris nisi ut aliquip ex ea commodo consequat.

```
library(ggplot2)
tea <- rnorm(100)
biscuits <- tea + rnorm(100, 0, 1.3)
data <- data.frame(tea, biscuits)
p <- ggplot(data, aes(x = tea, y = biscuits)) +
  geom_point() +
  geom_smooth(method = "lm") +
  labs(x = "Tea", y = "Biscuits") + theme_bw()
print(p)
```

Duis aute irure dolor in reprehenderit in voluptate velit esse
cillum dolore eu fugiat nulla pariatur. Excepteur sint
occaecat cupidatat non proident, sunt in culpa qui officia
deserunt mollit anim id est laborum.

In a **literate programming** approach, chunks of code contained in documents are processed and then replaced with their output when the output document is produced.

The code gets replaced by its output

Lorem Ipsum

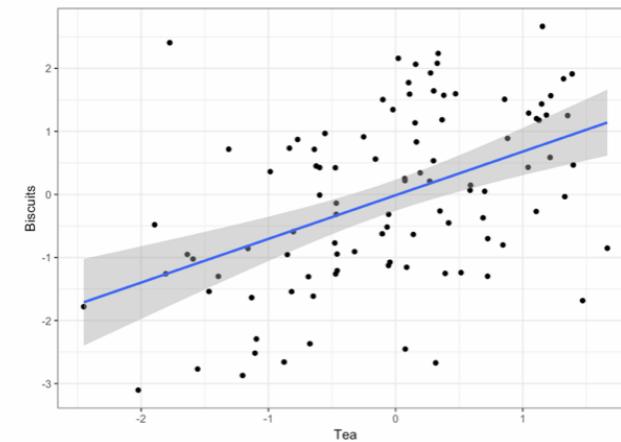
 Lorem ipsum dolor sit amet, consectetur adipisicing elit,
 sed do *eiusmod tempor* incididunt ut labore et dolore magna
 aliqua. Ut enimad minim veniam, quis nostrud exercititation
 ullamco laboris nisi ut aliquip ex ea commodo consequat.

```
library(ggplot2)
tea <- rnorm(100)
biscuits <- tea + rnorm(100, 0, 1.3)
data <- data.frame(tea, biscuits)
p <- ggplot(data, aes(x = tea, y = biscuits)) +
  geom_point() +
  geom_smooth(method = "lm") +
  labs(x = "Tea", y = "Biscuits") + theme_bw()
print(p)
```

Duis aute irure dolor in reprehenderit in voluptate velit esse
cillum dolore eu fugiat nulla pariatur. Excepteur sint
occaecat cupidatat non proident, sunt in culpa qui officia
deserunt mollit anim id est laborum.

1. Lorem Ipsum

 Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do
 eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut
 enimad minim veniam, quis nostrud exercititation ullamco laboris
 nisi ut aliquip ex ea commodo consequat.



Duis aute irure dolor in reprehenderit in voluptate velit esse cillum
dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non
proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

```
--  
title: "Covid Cases"  
author: "Kieran Healy"  
date: "4/18/2021"  
output: html_document  
--  
  
```{r setup, include=FALSE}  
knitr::opts_chunk$set(echo = FALSE)
...

Introduction

We'll be looking at some COVID case data.

```{r libraries, message = FALSE}  
  
library(tidyverse)  
library(here)  
library(janitor)  
library(socviz)  
...  
  
```{r load-the-data, message = FALSE}  
covid_cases <- read_csv("data/national_cases.csv")
...

Lorem Ipsum

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

```{r case-table}  
covid_cases %>%  
  filter(country %in% c("United States", "United Kingdom", "France")) %>%  
  group_by(country) %>%  
  summarize(cases = sum(cases)) %>%  
  arrange(desc(cases)) %>%  
  kable(caption = "Total cases in three countries.")  
...  
...
```

```
---
```

```
title: "Covid Cases"
author: "Kieran Healy"
date: "4/18/2021"
output: html_document
```

```
--
```

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

```

Introduction

We'll be looking at some COVID case data.

```
```{r libraries, message = FALSE}
```

```
library(tidyverse)
library(here)
library(janitor)
library(socviz)
```

```
```{r load-the-data, message = FALSE}
covid_cases <- read_csv("data/national_cases.csv")
```

```

## ## Lorem Ipsum

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

```
```{r case-table}
covid_cases %>%
  filter(country %in% c("United States", "United Kingdom", "France")) %>%
  group_by(country) %>%
  summarize(cases = sum(cases)) %>%
  arrange(desc(cases)) %>%
  kable(caption = "Total cases in three countries.")
```

```

# Header section with metadata

## Code chunk

In RStudio, code chunks can be "played" one at a time

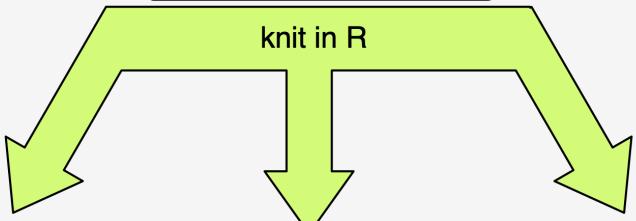
Code chunks can have their own labels and options

Text with Markdown formatting

Chunks are replaced by their output when the document is knitted

```
Report notes.Rmd
We can see this *relationship*
in a scatterplot.

```{r my-code}
p <- ggplot(data, mapping)
p + geom_point()
```
As you can see, this plot
looks pretty nice.
```



**Report notes.html**  
We can see this *relationship* in a scatterplot.

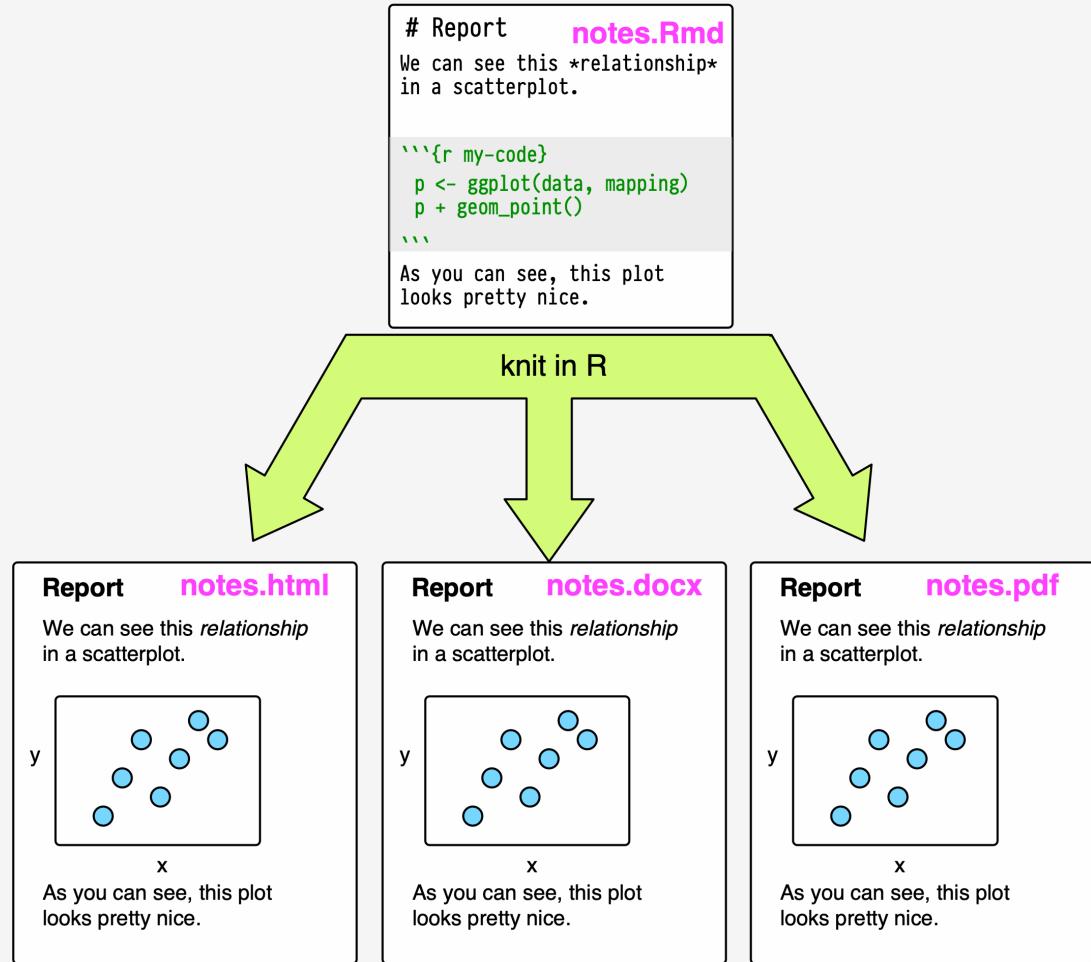
As you can see, this plot looks pretty nice.

**Report notes.docx**  
We can see this *relationship* in a scatterplot.

As you can see, this plot looks pretty nice.

**Report notes.pdf**  
We can see this *relationship* in a scatterplot.

As you can see, this plot looks pretty nice.



This approach has its limitations, but it's *very* useful and has many benefits.

## # Report notes.Rmd

We can see this \*relationship\* in a scatterplot.

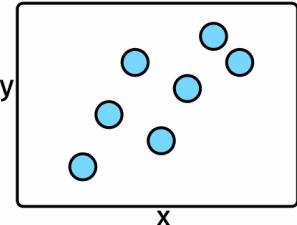
```
```{r my-code}
p <- ggplot(data, mapping)
p + geom_point()
```
```

As you can see, this plot looks pretty nice.

knit in R

## Report notes.html

We can see this *relationship* in a scatterplot.



As you can see, this plot looks pretty nice.

When learning these workflows, stick with the defaults at the beginning. Later, you can customize the look of the output in all kinds of ways.

The slides for this course were written in RMarkdown. Many of the resources for R that you'll find online (including websites and full-length books) were as well.

# The right frame of mind

This is like learning how to drive a car, or how to cook in a kitchen ... or learning to speak a language.

# The right frame of mind

This is like learning how to drive a car, or how to cook in a kitchen ... or learning to speak a language.

After some orientation to what's where, you will learn best by *doing*.

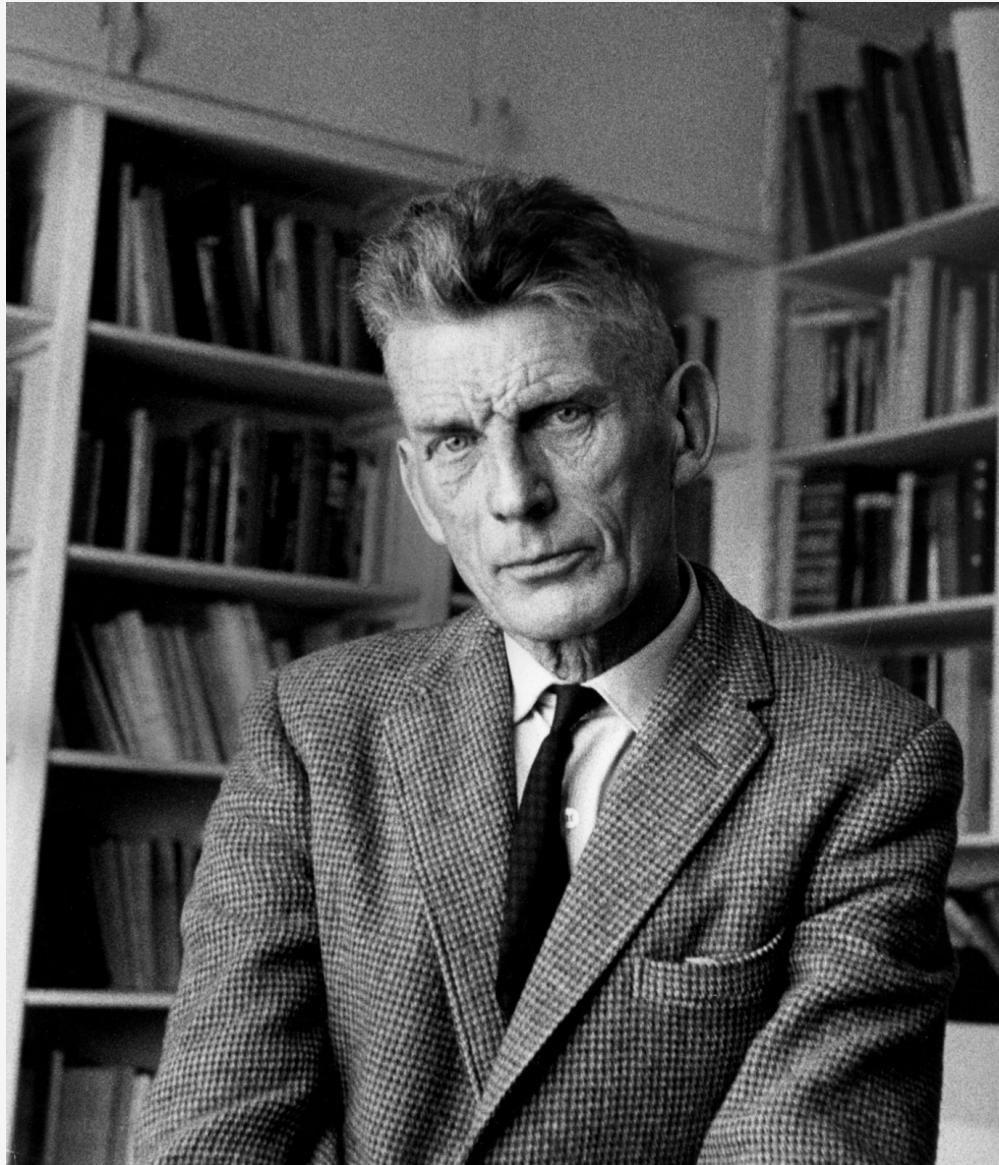
# The right frame of mind

This is like learning how to drive a car, or how to cook in a kitchen ... or learning to speak a language.

After some orientation to what's where, you will learn best by *doing*.

Software is a pain, but you won't crash the car or burn your house down.

**TYPE OUT  
YOUR CODE  
BY HAND**



Ever tried.  
Ever failed.  
No matter.  
Try again.  
Fail again.  
Fail better.

Samuel Beckett,  
early data analyst

# GETTING ORIENTED

# Loading the tidyverse libraries

```
library(tidyverse)
```

The tidyverse has several components.

We'll return to this message about Conflicts later.

Again, the code and messages you see here is actual R output, produced at the same time as the slide.

# Tidyverse components

```
library(tidyverse)
Loading tidyverse: ggplot2
Loading tidyverse: tibble
Loading tidyverse: tidyverse
Loading tidyverse: readr
Loading tidyverse: purrr
Loading tidyverse: dplyr
```

# Tidyverse components

```
library(tidyverse)
Loading tidyverse: ggplot2
Loading tidyverse: tibble
Loading tidyverse: tidyverse
Loading tidyverse: readr
Loading tidyverse: purrr
Loading tidyverse: dplyr
```

Load the package and ...  
< | Draw graphs  
< | Nicer data tables  
< | Tidy your data  
< | Get data into R  
< | Fancy Iteration  
< | Action verbs for tables

# What R looks like

Code you can type and run:

```
Inside code chunks, lines beginning with a # character are comments
Comments are ignored by R

my_numbers <- c(1, 1, 2, 4, 1, 3, 1, 5) # Anything after a # character is ignored as well
```

Output:

Equivalent to running the code above, typing my\_numbers at the console, and hitting enter.

```
my_numbers
[1] 1 1 2 4 1 3 1 5
```

# What R looks like

By convention, code output in documents is prefixed by ##

```
my_numbers
[1] 1 1 2 4 1 3 1 5
```

# What R looks like

By convention, code output in documents is prefixed by ##

```
my_numbers
[1] 1 1 2 4 1 3 1 5
```

Also by convention, outputting vectors, etc, gets a counter keeping track of the number of elements. For example,

```
letters
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
[20] "t" "u" "v" "w" "x" "y" "z"
```

# SOME THINGS TO KNOW ABOUT R

# 0. It's a calculator

## Arithmetic

```
(31 * 12) / 2^4
```

```
[1] 23.25
```

```
sqrt(25)
```

```
[1] 5
```

```
log(100)
```

```
[1] 4.60517
```

```
log10(100)
```

```
[1] 2
```

# 0. It's a calculator

## Arithmetic

```
(31 * 12) / 2^4
```

```
[1] 23.25
```

```
sqrt(25)
```

```
[1] 5
```

```
log(100)
```

```
[1] 4.60517
```

```
log10(100)
```

```
[1] 2
```

## Logic

```
4 < 10
```

```
[1] TRUE
```

```
4 > 2 & 1 > 0.5 # The "&" means "and"
```

```
[1] TRUE
```

```
4 < 2 | 1 > 0.5 # The " | " means "or"
```

```
[1] TRUE
```

```
4 < 2 | 1 < 0.5
```

```
[1] FALSE
```

# 0. It's a calculator

Logical equality and inequality (yielding a **TRUE** or **FALSE** result) is done with `==` and `!=`. Other logical operators include `<`, `>`, `<=`, `>=`, and `!` for negation. We'll use these in plots to filter data, test conditions, and so on.

```
A logical test
2 == 2 # Write `=` twice

[1] TRUE

This will cause an error, because R will think you are trying to assign a value
2 = 2

Error in 2 = 2 : invalid (do_set) left-hand side to assignment

3 != 7 # Write `!=` and then `=` to make `!=`

[1] TRUE
```

# 1. Everything in R has a name

```
my_numbers # We created this a few minutes ago
[1] 1 1 2 4 1 3 1 5

letters # This one is built-in
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
[20] "t" "u" "v" "w" "x" "y" "z"

pi # Also built-in
[1] 3.141593
```

# Some names are forbidden

Or it's a *really* bad idea to try to use them

```
Don't name objects with terms fo logical values,
or missing and null-value indicators

TRUE
FALSE
Inf
NaN
NA
NULL

Don't name objects with terms that are also
built-in functions for programming and flow-control

for
if
while
break
function
```

## 2. Everything is an object

There are a few built-in objects:

```
letters
```

```
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
[20] "t" "u" "v" "w" "x" "y" "z"
```

## 2. Everything is an object

There are a few built-in objects:

```
letters
```

```
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
[20] "t" "u" "v" "w" "x" "y" "z"
```

```
pi
```

```
[1] 3.141593
```

# 2. Everything is an object

There are a few built-in objects:

```
letters
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
[20] "t" "u" "v" "w" "x" "y" "z"

pi
[1] 3.141593

LETTERS
[1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"
[20] "T" "U" "V" "W" "X" "Y" "Z"
```

# 3. You can create objects

# 3. You can create objects

In fact, this is mostly what we will be doing.

# 3. You can create objects

In fact, this is mostly what we will be doing.

Objects are created by *assigning* a thing to a name:

```
name... gets ... this stuff
my_numbers <- c(1, 2, 3, 1, 3, 5, 25, 10)

name ... gets ... the output of the function `c()`
your_numbers <- c(5, 31, 71, 1, 3, 21, 6, 52)
```

# 3. You can create objects

In fact, this is mostly what we will be doing.

Objects are created by *assigning* a thing to a name:

```
name... gets ... this stuff
my_numbers <- c(1, 2, 3, 1, 3, 5, 25, 10)

name ... gets ... the output of the function `c()`
your_numbers <- c(5, 31, 71, 1, 3, 21, 6, 52)
```

The **c()** function *combines* or *concatenates* things

The **assignment operator**, **<-**, performs the action of creating objects.

# The assignment operator

The **assignment operator** performs the action of creating objects:

# The assignment operator

The **assignment operator** performs the action of creating objects:

Use a keyboard shortcut to write it:

Press **option** and **-** on a Mac

Press **alt** and **-** on Windows

# Assignment with =

You can use "==" as well as "<-" for assignment

```
my_numbers = c(1, 2, 3, 1, 3, 5, 25)
my_numbers
[1] 1 2 3 1 3 5 25
```

# Assignment with =

You can use "==" as well as "<->" for assignment

```
my_numbers = c(1, 2, 3, 1, 3, 5, 25)
my_numbers
[1] 1 2 3 1 3 5 25
```

On the other hand, "=" has a different meaning when used in functions.

# Assignment with =

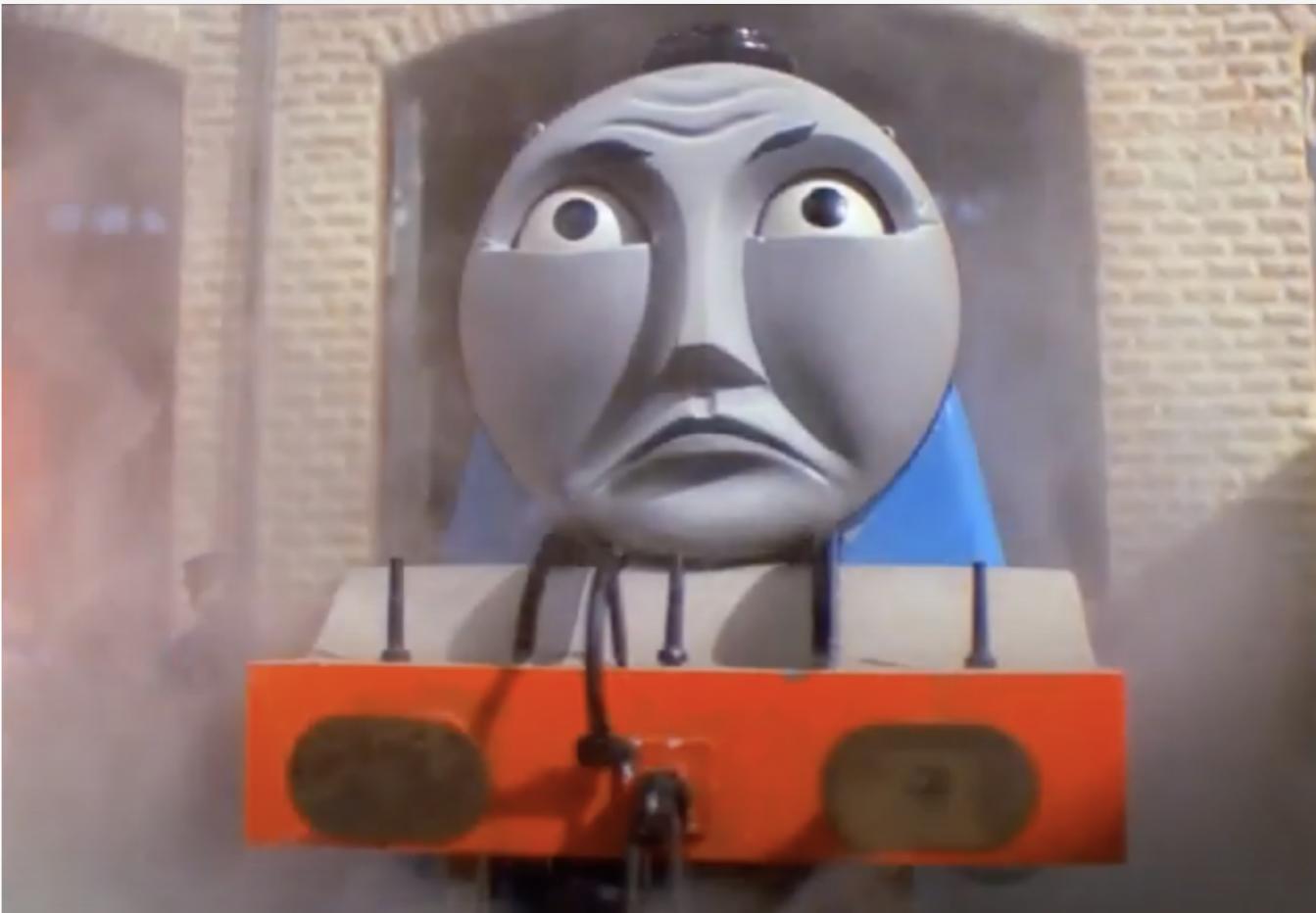
You can use "==" as well as "<-" for assignment

```
my_numbers = c(1, 2, 3, 1, 3, 5, 25)
my_numbers
[1] 1 2 3 1 3 5 25
```

On the other hand, "==" has a different meaning when used in functions.

I'm going to use "<-" for assignment throughout. Just be consistent either way.

# Assignment with =



It isn't  
*wrong* but  
we just  
don't *do* it

# 4. Do things to objects with **functions**

```
this object... gets ... the output of this function
my_numbers <- c(1, 2, 3, 1, 3, 5, 25, 10)

your_numbers <- c(5, 31, 71, 1, 3, 21, 6, 52)

my_numbers

[1] 1 2 3 1 3 5 25 10
```

Functions can be identified by the parentheses after their names.

```
my_numbers

[1] 1 2 3 1 3 5 25 10

If you run this you'll get an error
mean()
```

# What **functions** usually do

They take **inputs** to **arguments**

They perform **actions**

They produce, or return, **outputs**

# What **functions** usually do

They take **inputs** to **arguments**

They perform **actions**

They produce, or return, **outputs**

```
x <- c(1, 2, 3, 1, 3, 5, 25, 10)
```

```
x
```

```
[1] 1 2 3 1 3 5 25 10
```

# What **functions** usually do

They take **inputs** to **arguments**

They perform **actions**

They produce, or return, **outputs**

```
x <- c(1, 2, 3, 1, 3, 5, 25, 10)
```

```
mean(x = my_numbers)
```

```
x
```

```
[1] 6.25
```

```
[1] 1 2 3 1 3 5 25 10
```

# What **functions** usually do

```
Get the mean of what? Of x.
You need to tell the function what x is
mean(x = my_numbers)
```

```
[1] 6.25
```

```
mean(x = your_numbers)
```

```
[1] 23.75
```

# What **functions** usually do

```
Get the mean of what? Of x.
You need to tell the function what x is
mean(x = my_numbers)
```

```
[1] 6.25
```

```
mean(x = your_numbers)
```

```
[1] 23.75
```

If you don't *name* the arguments, R assumes you are providing them in the order the function expects.

```
mean(your_numbers)
```

```
[1] 23.75
```

# What **functions** usually do

What arguments? Which order? Read the function's help page

```
help(mean)
```

```
quicker
?mean
```

# What **functions** usually do

What arguments? Which order? Read the function's help page

```
help(mean)
```

```
quicker
?mean
```

How to read an R help page ...

# What **functions** usually do

Arguments often tell the function what to do in specific circumstances

```
missing_numbers <- c(1:10, NA, 20, 32, 50, 104, 32, 147, 99, NA, 45)
mean(missing_numbers)
[1] NA
mean(missing_numbers, na.rm = TRUE)
[1] 32.44444
```

# What **functions** usually do

Arguments often tell the function what to do in specific circumstances

```
missing_numbers <- c(1:10, NA, 20, 32, 50, 104, 32, 147, 99, NA, 45)
mean(missing_numbers)
[1] NA
mean(missing_numbers, na.rm = TRUE)
[1] 32.44444
```

Or select from one of several options

```
Look at ?mean to see what `trim` does
mean(missing_numbers, na.rm = TRUE, trim = 0.1)
[1] 27.25
```

# What **functions** usually do

There are all kinds of functions. They return different things.

```
summary(my_numbers)

Min. 1st Qu. Median Mean 3rd Qu. Max.
1.00 1.75 3.00 6.25 6.25 25.00
```

# What **functions** usually do

There are all kinds of functions. They return different things.

```
summary(my_numbers)

Min. 1st Qu. Median Mean 3rd Qu. Max.
1.00 1.75 3.00 6.25 6.25 25.00
```

You can assign the output of a function to a name, which turns it into an object. (Otherwise it'll send its output to the console.)

```
my_summary <- summary(my_numbers)

my_summary

Min. 1st Qu. Median Mean 3rd Qu. Max.
1.00 1.75 3.00 6.25 6.25 25.00
```

# What **functions** usually do

Objects hang around in your work environment until they are overwritten by you, or are deleted.

```
rm() function removes objects
rm(my_summary)

my_summary
Error: object 'my_summary' not found
```

# Functions can be nested

```
c(1:20)
```

```
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

# Functions can be nested

```
c(1:20)

[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

summary(c(1:20))

Min. 1st Qu. Median Mean 3rd Qu. Max.
1.00 5.75 10.50 10.50 15.25 20.00

names(summary(c(1:20)))

[1] "Min." "1st Qu." "Median" "Mean" "3rd Qu." "Max."
```

# Functions can be nested

```
c(1:20)

[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

summary(c(1:20))

Min. 1st Qu. Median Mean 3rd Qu. Max.
1.00 5.75 10.50 10.50 15.25 20.00

names(summary(c(1:20)))

[1] "Min." "1st Qu." "Median" "Mean" "3rd Qu." "Max."

length(names(summary(c(1:20)))))

[1] 6
```

# Functions can be nested

```
c(1:20)

[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

summary(c(1:20))

Min. 1st Qu. Median Mean 3rd Qu. Max.
1.00 5.75 10.50 10.50 15.25 20.00

names(summary(c(1:20)))

[1] "Min." "1st Qu." "Median" "Mean" "3rd Qu." "Max."

length(names(summary(c(1:20)))))

[1] 6
```

Nested functions are evaluated from the inside out.

# Use the pipe operator: | >

Instead of nesting functions in parentheses, we can use the *pipe operator* to join them together:

```
c(1:20) |> summary() |> names() |> length()
[1] 6
```

# Use the pipe operator: | >

Instead of nesting functions in parentheses, we can use the *pipe operator* to join them together:

```
c(1:20) |> summary() |> names() |> length()
[1] 6
```

Read this operator as "*and then*"

# Use the pipe operator: | >

Instead of nesting functions in parentheses, we can use the *pipe operator* to join them together:

```
c(1:20) |> summary() |> names() |> length()
[1] 6
```

Read this operator as "*and then*"

Better, vertical space is free in R:

```
c(1:20) |>
 summary() |>
 names() |>
 length()
[1] 6
```

# Pipelines make code more **readable**

Not great, Bob:

```
serve(stir(pour_in_pan(whisk(crack_eggs(get_from_fridge(eggs), into = "bowl"), len = 40), temp = "med-high")))
```

# Pipelines make code more **readable**

Not great, Bob:

```
serve(stir(pour_in_pan(whisk(crack_eggs(get_from_fridge(eggs), into = "bowl"), len = 40), temp = "med-high")))
```

Notice how the first thing you read is the last operation performed.

# Pipelines make code more **readable**

Not great, Bob:

```
serve(stir(pour_in_pan(whisk(crack_eggs(get_from_fridge(eggs), into = "bowl"), len = 40), temp = "med-high")))
```

Notice how the first thing you read is the last operation performed.

Really not much better:

```
serve(
 stir(
 pour_in_pan(
 whisk(
 crack_eggs(
 get_from_fridge(eggs),
 into = "bowl"),
 len = 40),
 temp = "med-high")
)
)
```

# Pipelines make code more **readable**

Much nicer:

```
eggs |>
 get_from_fridge() |>
 crack_eggs(into = "bowl") |>
 whisk(len = 40) |>
 pour_in_pan(temp = "med-high") |>
 stir() |>
 serve()
```

# Pipelines make code more **readable**

Much nicer:

```
eggs |>
 get_from_fridge() |>
 crack_eggs(into = "bowl") |>
 whisk(len = 40) |>
 pour_in_pan(temp = "med-high") |>
 stir() |>
 serve()
```

We'll still use nested parentheses quite a bit, often in the context of a function working inside a pipeline. But it's good not to have too many levels of nesting.

# The **magrittr** pipe: %>%

The pipe operator `|>` was not part of Base R until very recently. Piping was originally introduced in a package called `magrittr`, where it was written `%>%` and behaved in very nearly\* the same way as the base pipe now does.

# The **magrittr** pipe: %>%

The pipe operator `|>` was not part of Base R until very recently. Piping was originally introduced in a package called **magrittr**, where it was written `%>%` and behaved in very nearly\* the same way as the base pipe now does.

The **magrittr** pipe continues to work. A lot of existing code uses it (e.g., my book!).

# The magrittr pipe: %>%

The pipe operator `|>` was not part of Base R until very recently. Piping was originally introduced in a package called `magrittr`, where it was written `%>%` and behaved in very nearly\* the same way as the base pipe now does.

The `magrittr` pipe continues to work. A lot of existing code uses it (e.g., my book!).

*Sidenote:* There are bunch of special operators in R that have the naming convention `%something%`. For example `%^%` means "matrix multiply". We'll see more of them as we go. In this context the `%` is sometimes pronounced "grapes".

\*With the new pipe, you can only pass an object to the *first* argument in a function. This is fine for most tidyverse pipelines, where the first argument is usually (implicitly) the data. But it does mean that most Base R functions will continue not to be easily piped, as most of them do not follow the convention of passing the current data as the first argument

# Functions are bundled into packages

# Functions are bundled into packages

All programming languages gain power and convenience by having libraries or packages of functions that extend the core abilities of the language. In R, packages are loaded into your working environment using the `library()` function.

# Functions are bundled into packages

All programming languages gain power and convenience by having libraries or packages of functions that extend the core abilities of the language. In R, packages are loaded into your working environment using the **library()** function.

```
A package containing a dataset rather than functions
library(gapminder)

gapminder

A tibble: 1,704 × 6
country continent year lifeExp pop gdpPercap
<fct> <fct> <int> <dbl> <int> <dbl>
1 Afghanistan Asia 1952 28.8 8425333 779.
2 Afghanistan Asia 1957 30.3 9240934 821.
3 Afghanistan Asia 1962 32.0 10267083 853.
4 Afghanistan Asia 1967 34.0 11537966 836.
5 Afghanistan Asia 1972 36.1 13079460 740.
6 Afghanistan Asia 1977 38.4 14880372 786.
7 Afghanistan Asia 1982 39.9 12881816 978.
8 Afghanistan Asia 1987 40.8 13867957 852.
9 Afghanistan Asia 1992 41.7 16317921 649.
10 Afghanistan Asia 1997 41.8 22227415 635.
... with 1,694 more rows
```

# Functions are bundled into packages

# Functions are bundled into packages

You need only *install* a package once (and occasionally update it). But you must *load* the package in each R session before you can access its contents.

```
Do at least once for each package. Once done, not needed each time.
install.packages("palmerpenguins", repos = "http://cran.rstudio.com")

Needed sometimes, especially after an R major version upgrade.
update.packages(repos = "http://cran.rstudio.com")

To load a package, usually at the start of your RMarkdown document or script file
library(palmerpenguins)
penguins

A tibble: 344 × 8
species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
<fct> <fct> <dbl> <dbl> <int> <int>
1 Adelie Torgersen 39.1 18.7 181 3750
2 Adelie Torgersen 39.5 17.4 186 3800
3 Adelie Torgersen 40.3 18 195 3250
4 Adelie Torgersen NA NA NA NA
5 Adelie Torgersen 36.7 19.3 193 3450
6 Adelie Torgersen 39.3 20.6 190 3650
7 Adelie Torgersen 38.9 17.8 181 3625
8 Adelie Torgersen 39.2 19.6 195 4675
9 Adelie Torgersen 34.1 18.1 193 3475
10 Adelie Torgersen 42 20.2 190 4250
... with 334 more rows, and 2 more variables: sex <fct>, year <int>
```

# Grabbing a single function with ::

"Reach in" to an unloaded package and grab a function directly,  
using <package>::<function>

# Grabbing a single function with ::

"Reach in" to an unloaded package and grab a function directly,  
using <package>::<function>

```
A little glimpse of what we'll do soon
penguins |>
 select(species, body_mass_g, sex) |>
 gtsummary::tbl_summary(by = species)
```

| Characteristic            | Adelie, N = 152       | Chinstrap, N = 68     | Gentoo, N = 124       |
|---------------------------|-----------------------|-----------------------|-----------------------|
| body_mass_g, Median (IQR) | 3,700 (3,350 – 4,000) | 3,700 (3,488 – 3,950) | 5,000 (4,700 – 5,500) |
| Unknown                   | 1                     | 0                     | 1                     |
| sex, n (%)                |                       |                       |                       |
| female                    | 73 (50)               | 34 (50)               | 58 (49)               |
| male                      | 73 (50)               | 34 (50)               | 61 (51)               |
| Unknown                   | 6                     | 0                     | 5                     |

# Remember this warning about conflicts?

```
library(tidyverse)

─ Attaching packages ─────────────────────────────────── tidyverse 1.3.0 ─
✓ ggplot2 3.3.3 ✓ purrr 0.3.4
✓ tibble 3.1.0 ✓ dplyr 1.0.5
✓ tidyr 1.1.3 ✓ stringr 1.4.0
✓ readr 1.4.0 ✓ forcats 0.5.1

─ Conflicts ────────────────────────────────── tidyverse_conflicts() ─
x dplyr::filter() masks stats::filter()
x purrr::is_null() masks testthat::is_null()
x dplyr::lag() masks stats::lag()
x dplyr::matches() masks tidyrr::matches(), testthat::matches()
```

Notice how some functions in different packages have the same names.

# Remember this warning about conflicts?

```
library(tidyverse)

─ Attaching packages ━━━━━━━━━━━━━━ tidyverse 1.3.0 ━━━━━━━━

✓ ggplot2 3.3.3 ✓ purrr 0.3.4
✓ tibble 3.1.0 ✓ dplyr 1.0.5
✓ tidyr 1.1.3 ✓ stringr 1.4.0
✓ readr 1.4.0 ✓forcats 0.5.1

─ Conflicts ━━━━━━━━━━━━━━ tidyverse_conflicts() ━━━━━━━━

x dplyr::filter() masks stats::filter()
x purrr::is_null() masks testthat::is_null()
x dplyr::lag() masks stats::lag()
x dplyr::matches() masks tidyrr::matches(), testthat::matches()
```

Notice how some functions in different packages have the same names.

Related concepts of *namespaces* and *environments*.

# Scope of names

```
x <- c(1:10)
y <- c(90:100)

x
[1] 1 2 3 4 5 6 7 8 9 10

y
[1] 90 91 92 93 94 95 96 97 98 99 100
```

# Scope of names

```
x <- c(1:10)
y <- c(90:100)

x
[1] 1 2 3 4 5 6 7 8 9 10

y
[1] 90 91 92 93 94 95 96 97 98 99 100

mean()
Error in mean.default() : argument "x" is missing, with no default
```

# Scope of names

```
x <- c(1:10)
y <- c(90:100)

x
[1] 1 2 3 4 5 6 7 8 9 10

y
[1] 90 91 92 93 94 95 96 97 98 99 100

mean()
Error in mean.default() : argument "x" is missing, with no default

mean(x) # argument names are internal to functions
[1] 5.5

mean(x = x)
[1] 5.5

mean(x = y)
[1] 95

x
[1] 1 2 3 4 5 6 7 8 9 10

y
```

## 5. Objects come in **types** and **classes**

I'm going to speak somewhat loosely here for now, and gloss over some distinctions between object classes and data structures, as well as kinds of objects and their attributes.

## 5. Objects come in **types** and **classes**

I'm going to speak somewhat loosely here for now, and gloss over some distinctions between object classes and data structures, as well as kinds of objects and their attributes.

The object inspector in RStudio is your friend.

# 5. Objects come in **types** and **classes**

I'm going to speak somewhat loosely here for now, and gloss over some distinctions between object classes and data structures, as well as kinds of objects and their attributes.

The object inspector in RStudio is your friend.

You can ask an object what it is.

```
class(my_numbers)
[1] "numeric"

typeof(my_numbers)
[1] "double"
```

## 5. Objects come in **types** and **classes**

Objects can have more than one (nested) class:

# 5. Objects come in **types** and **classes**

Objects can have more than one (nested) class:

```
summary(my_numbers)

Min. 1st Qu. Median Mean 3rd Qu. Max.
1.00 1.75 3.00 6.25 6.25 25.00

my_smry <- summary(my_numbers) # remember, outputs can be assigned to a name, creating an object
class(summary(my_numbers)) # functions can be nested, and are evaluated from the inside out
[1] "summaryDefault" "table"

class(my_smry) # equivalent to the previous line
[1] "summaryDefault" "table"
```

# 5. Objects come in **types** and **classes**

```
typeof(my_smry)
[1] "double"

attributes(my_smry)

$names
[1] "Min." "1st Qu." "Median" "Mean" "3rd Qu." "Max."

$class
[1] "summaryDefault" "table"

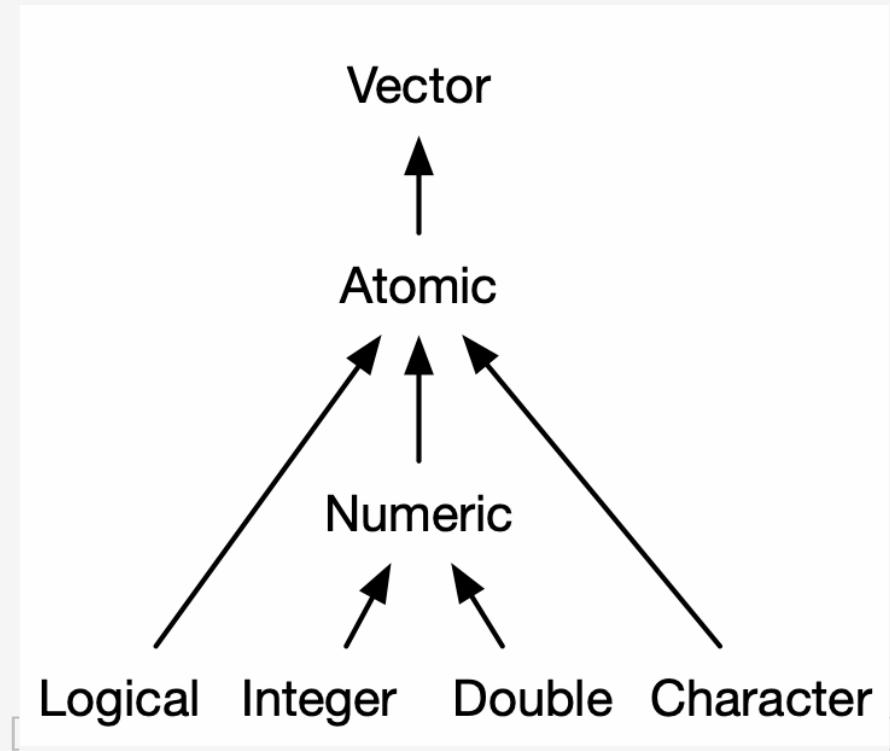
In this case, the functions extract the corresponding attribute
class(my_smry)

[1] "summaryDefault" "table"

names(my_smry)

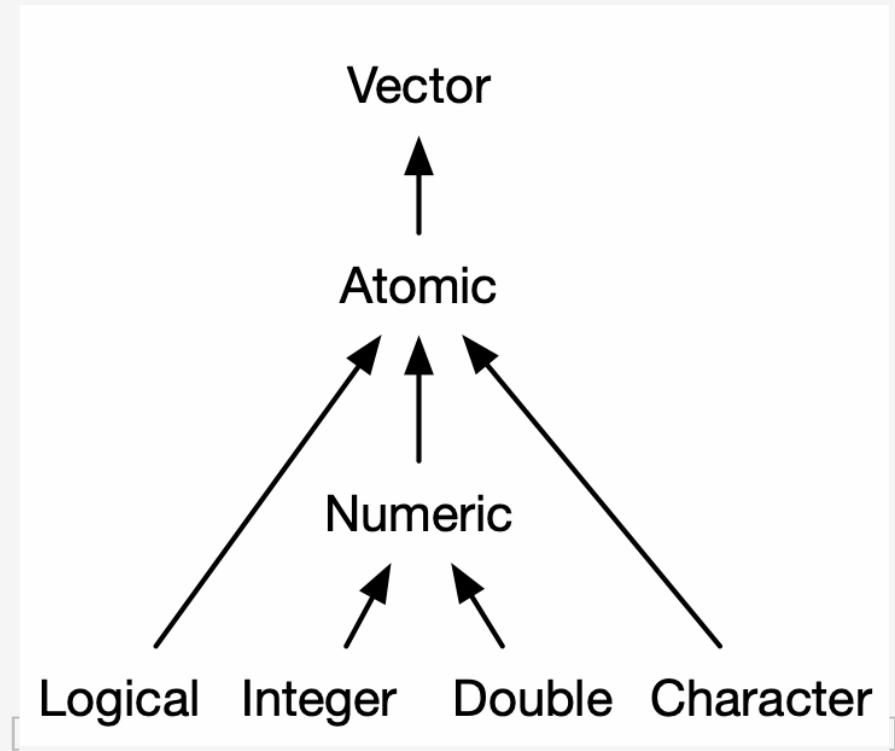
[1] "Min." "1st Qu." "Median" "Mean" "3rd Qu." "Max."
```

# A **vector** is a fundamental kind of object



From Hadley Wickham, *Advanced R*

# A **vector** is a fundamental kind of object



```
my_int <- c(1, 3, 5, 6, 10)
is.integer(my_int)
[1] FALSE

is.double(my_int)
[1] TRUE

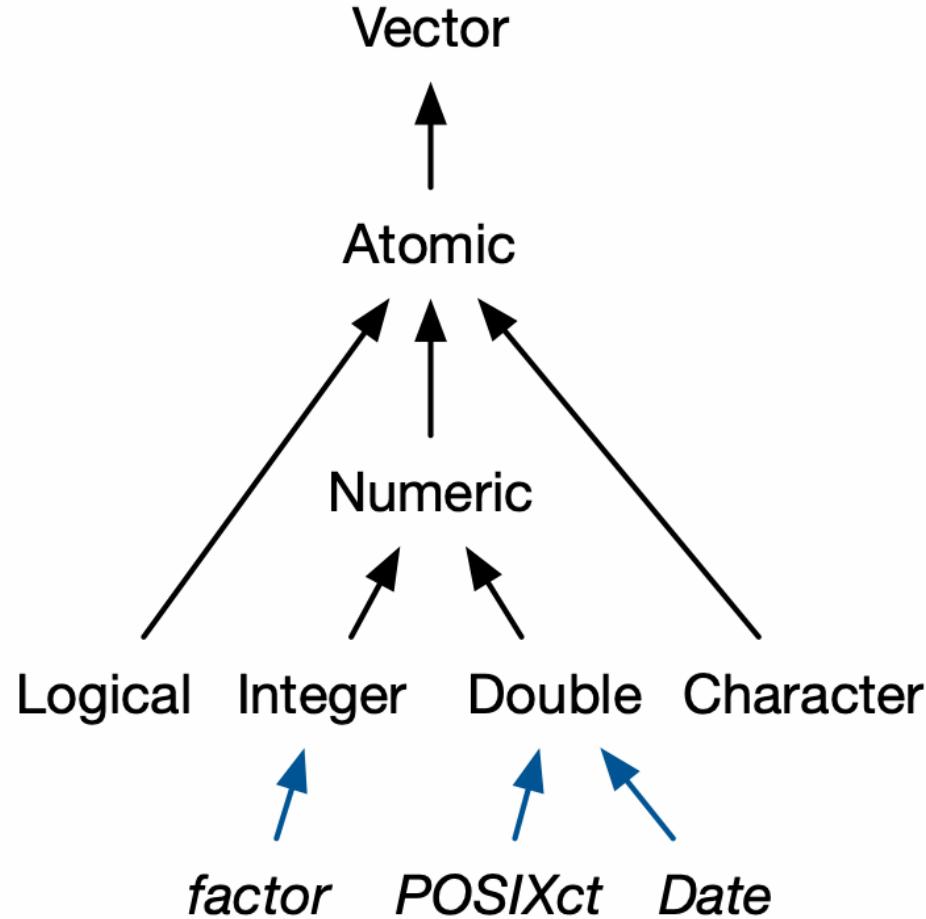
my_int <- as.integer(my_int)
is.integer(my_int)
[1] TRUE

my_chr <- c("Mary", "had", "a", "little", "lamb")
is.character(my_chr)
[1] TRUE

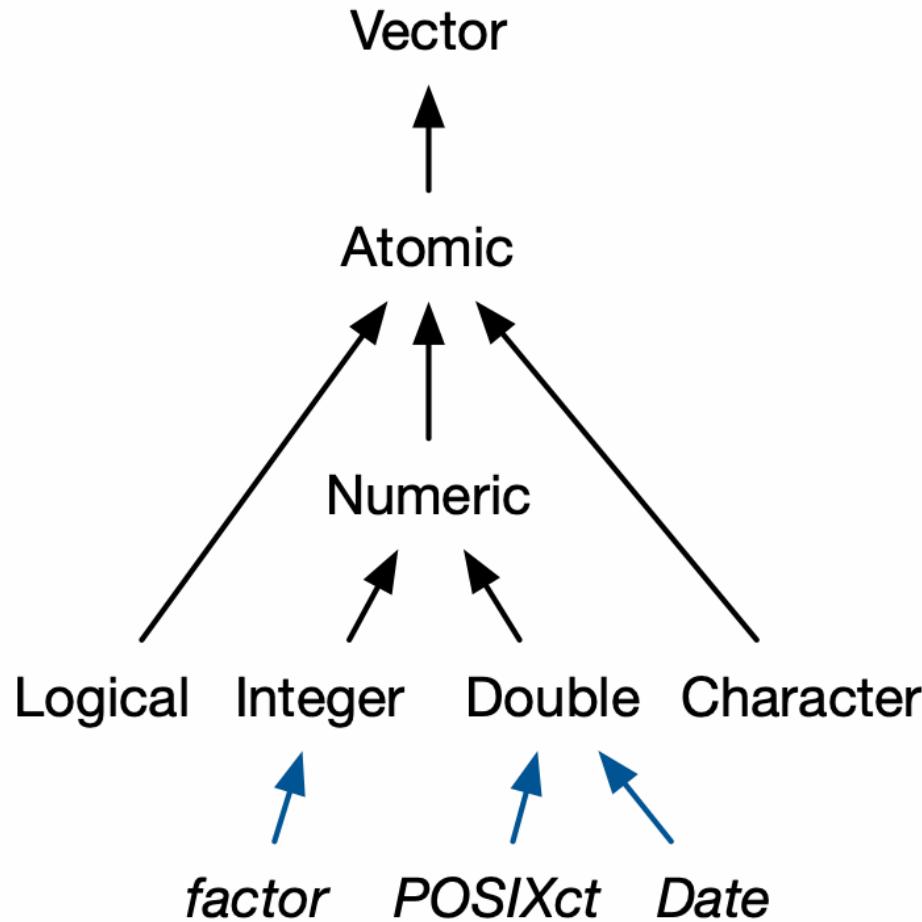
my_lgl <- c(TRUE, FALSE, TRUE)
is.logical(my_lgl)
[1] TRUE
```

From Hadley Wickham, *Advanced R*

# The most common types of **vector**



# The most common types of **vector**



```
Factors are for storing categorical variables
x <- factor(c("Yes", "No", "No", "Maybe", "Yes", "Yes"))
x
[1] Yes No No Maybe Yes Yes
Levels: Maybe No Yes

summary(x) # Alphabetical order by default
Maybe No Yes
1 2 3

typeof(x) # A factor is a vector of integers
[1] "integer"

attributes(x) # ... with labels for its "levels"
$levels
[1] "Maybe" "No" "Yes"
##
$class
[1] "factor"

levels(x)
[1] "Maybe" "No" "Yes"

is.ordered(x)
[1] FALSE
```

# Individual vectors can't be heterogenous

Objects can be manually or automatically coerced from one class to another. Take care!

# Individual vectors can't be heterogenous

Objects can be manually or automatically coerced from one class to another. Take care!

```
class(my_numbers)
[1] "numeric"

my_new_vector <- c(my_numbers, "Apple")
my_new_vector # vectors are homogeneous/atomic
[1] "1" "2" "3" "1" "3" "5" "25" "10" "Apple"

class(my_new_vector)
[1] "character"
```

# Individual vectors can't be heterogenous

Objects can be manually or automatically coerced from one class to another. Take care!

```
class(my_numbers)
[1] "numeric"

my_new_vector <- c(my_numbers, "Apple")
my_new_vector # vectors are homogeneous/atomic
[1] "1" "2" "3" "1" "3" "5" "25" "10" "Apple"

class(my_new_vector)
[1] "character"

my_dbl <- c(2.1, 4.77, 30.111, 3.14519)
is.double(my_dbl)
[1] TRUE

my_dbl <- as.integer(my_dbl)

my_dbl
[1] 2 4 30 3
```

# Tibbles are a **list of vectors** of various **types**

```
gapminder # tibbles and data frames can contain vectors of different types

A tibble: 1,704 × 6
country continent year lifeExp pop gdpPercap
<fct> <fct> <int> <dbl> <int> <dbl>
1 Afghanistan Asia 1952 28.8 8425333 779.
2 Afghanistan Asia 1957 30.3 9240934 821.
3 Afghanistan Asia 1962 32.0 10267083 853.
4 Afghanistan Asia 1967 34.0 11537966 836.
5 Afghanistan Asia 1972 36.1 13079460 740.
6 Afghanistan Asia 1977 38.4 14880372 786.
7 Afghanistan Asia 1982 39.9 12881816 978.
8 Afghanistan Asia 1987 40.8 13867957 852.
9 Afghanistan Asia 1992 41.7 16317921 649.
10 Afghanistan Asia 1997 41.8 22227415 635.
... with 1,694 more rows

class(gapminder)

[1] "tbl_df" "tbl" "data.frame"

typeof(gapminder) # hmm

[1] "list"
```

Underneath, most complex R objects are some kind of list with different components.

# Classes can be nested

Some classes build on and enhance the properties of simpler classes.

Base R's trusty **data.frame**

```
library(socviz)
titanic

fate sex n percent
1 perished male 1364 62.0
2 perished female 126 5.7
3 survived male 367 16.7
4 survived female 344 15.6

class(titanic)

[1] "data.frame"

The `\$` idiom picks out a named column here;
more generally, the named element of a list
titanic$percent

[1] 62.0 5.7 16.7 15.6
```

# Classes can be nested

Some classes build on and enhance the properties of simpler classes.

## Base R's trusty `data.frame`

```
library(socviz)
titanic

fate sex n percent
1 perished male 1364 62.0
2 perished female 126 5.7
3 survived male 367 16.7
4 survived female 344 15.6

class(titanic)

[1] "data.frame"

The `\$` idiom picks out a named column here;
more generally, the named element of a list
titanic$percent

[1] 62.0 5.7 16.7 15.6
```

## The Tidyverse's enhanced `tibble`

```
tibbles are build on data frames
class(titanic)

[1] "data.frame"

titanic_tb <- as_tibble(titanic)

titanic_tb

A tibble: 4 × 4
fate sex n percent
<fct> <fct> <dbl> <dbl>
1 perished male 1364 62
2 perished female 126 5.7
3 survived male 367 16.7
4 survived female 344 15.6

class(titanic_tb)

[1] "tbl_df" "tbl" "data.frame"
```

# All of this will matter later on

gss\_sm

```
A tibble: 2,867 × 32
year id ballot age childs sibs degree race sex region income16
<dbl> <dbl> <labelled> <dbl> <dbl> <labe> <fct> <fct> <fct> <fct>
1 2016 1 1 47 3 2 Bach... White Male New E... $170000...
2 2016 2 2 61 0 3 High ... White Male New E... $50000 ...
3 2016 3 3 72 2 3 Bach... White Male New E... $75000 ...
4 2016 4 1 43 4 3 High ... White Fema... New E... $170000...
5 2016 5 3 55 2 2 Gradu... White Fema... New E... $170000...
6 2016 6 2 53 2 2 Junio... White Fema... New E... $60000 ...
7 2016 7 1 50 2 2 High ... White Male New E... $170000...
8 2016 8 3 23 3 6 High ... Other Fema... Middl... $30000 ...
9 2016 9 1 45 3 5 High ... Black Male Middl... $60000 ...
10 2016 10 3 71 4 1 Junio... White Male Middl... $60000 ...
... with 2,857 more rows, and 21 more variables: relig <fct>, marital <fct>,
padeg <fct>, madeg <fct>, partyid <fct>, polviews <fct>, happy <fct>,
partners <fct>, grass <fct>, zodiac <fct>, pres12 <labelled>,
wtssall <dbl>, income_rc <fct>, agegrp <fct>, ageq <fct>, siblings <fct>,
kids <fct>, religion <fct>, bigregion <fct>, partners_rc <fct>, obama <dbl>
```

# All of this will matter later on

gss\_sm

```
A tibble: 2,867 × 32
year id ballot age child� sibs degree race sex region income16
<dbl> <dbl> <labelled> <dbl> <dbl> <labe> <fct> <fct> <fct> <fct> <fct>
1 2016 1 1 47 3 2 Bach... White Male New E... $170000...
2 2016 2 2 61 0 3 High ... White Male New E... $50000 ...
3 2016 3 3 72 2 3 Bach... White Male New E... $75000 ...
4 2016 4 1 43 4 3 High ... White Fema... New E... $170000...
5 2016 5 3 55 2 2 Gradu... White Fema... New E... $170000...
6 2016 6 2 53 2 2 Junio... White Fema... New E... $60000 ...
7 2016 7 1 50 2 2 High ... White Male New E... $170000...
8 2016 8 3 23 3 6 High ... Other Fema... Middl... $30000 ...
9 2016 9 1 45 3 5 High ... Black Male Middl... $60000 ...
10 2016 10 3 71 4 1 Junio... White Male Middl... $60000 ...
... with 2,857 more rows, and 21 more variables: relig <fct>, marital <fct>,
padeg <fct>, madeg <fct>, partyid <fct>, polviews <fct>, happy <fct>,
partners <fct>, grass <fct>, zodiac <fct>, pres12 <labelled>,
wtssall <dbl>, income_rc <fct>, agegrp <fct>, ageq <fct>, siblings <fct>,
kids <fct>, religion <fct>, bigregion <fct>, partners_rc <fct>, obama <dbl>
```

Tidyverse tools are generally *type safe*, meaning their functions return the same type of thing every time, or fail if they cannot do this. So it's good to know about the various data types.

# 6. Arithmetic on vectors

In R, all numbers are vectors of different sorts. Even single numbers ("scalars") are conceptually vectors of length 1.

# 6. Arithmetic on vectors

In R, all numbers are vectors of different sorts. Even single numbers ("scalars") are conceptually vectors of length 1.

Arithmetic on vectors\* follows a series of *recycling rules* that favor ease of expression of vectorized, "elementwise" operations.

\*And arrays, too.

# 6. Arithmetic on vectors

See if you can predict what the following operations do:

```
my_numbers
[1] 1 2 3 1 3 5 25 10

result1 <- my_numbers + 1
```

# 6. Arithmetic on vectors

See if you can predict what the following operations do:

```
my_numbers
[1] 1 2 3 1 3 5 25 10

result1 <- my_numbers + 1

result1
[1] 2 3 4 2 4 6 26 11
```

# 6. Arithmetic on vectors

See if you can predict what the following operations do:

```
my_numbers
[1] 1 2 3 1 3 5 25 10

result1 <- my_numbers + 1

result1
[1] 2 3 4 2 4 6 26 11

result2 <- my_numbers + my_numbers
```

# 6. Arithmetic on vectors

See if you can predict what the following operations do:

```
my_numbers
[1] 1 2 3 1 3 5 25 10

result1 <- my_numbers + 1

result1
[1] 2 3 4 2 4 6 26 11

result2 <- my_numbers + my_numbers

result2
[1] 2 4 6 2 6 10 50 20
```

# 6. Arithmetic on vectors

See if you can predict what the following operations do:

```
my_numbers
[1] 1 2 3 1 3 5 25 10

result1 <- my_numbers + 1

result1
[1] 2 3 4 2 4 6 26 11

result2 <- my_numbers + my_numbers

result2
[1] 2 4 6 2 6 10 50 20

two_nums <- c(5, 10)

result3 <- my_numbers + two_nums
```

# 6. Arithmetic on vectors

See if you can predict what the following operations do:

```
my_numbers
[1] 1 2 3 1 3 5 25 10

result1 <- my_numbers + 1

result1
[1] 2 3 4 2 4 6 26 11

result2 <- my_numbers + my_numbers

result2
[1] 2 4 6 2 6 10 50 20

two_nums <- c(5, 10)

result3 <- my_numbers + two_nums

result3
[1] 6 12 8 11 8 15 30 20
```

# 6. Arithmetic on vectors

```
three_nums <- c(1, 5, 10)

result4 <- my_numbers + three_nums

Warning in my_numbers + three_nums: longer object length is not a multiple of
shorter object length
```

# 6. Arithmetic on vectors

```
three_nums <- c(1, 5, 10)

result4 <- my_numbers + three_nums

Warning in my_numbers + three_nums: longer object length is not a multiple of
shorter object length

result4

[1] 2 7 13 2 8 15 26 15
```

Note that you got a **warning** here. R will still do what you told it do, though! Don't ignore warnings until you understand what they mean.

# 7. R will be frustrating

# 7. R will be frustrating

The IDE tries its best to help you. Learn to attend to what it is trying to say.

Warning message:

```
In my_numbers + two_nums :
longer object length is not a multiple of shorter object length
```

# 7. R will be frustrating

The IDE tries its best to help you. Learn to attend to what it is trying to say.

Warning message:

```
In my_numbers + two_nums :
 longer object length is not a multiple of shorter object length
```

```
38 #> #> make a plot is
✖ 39 p <- ggplot(data = gapminder
 expected ',' after expression| x = gdpPercap,
 41 y = lifeExp))
42 |
```

# 7. R will be frustrating

The IDE tries its best to help you. Learn to attend to what it is trying to say.

Warning message:

```
In my_numbers + two_nums :
 longer object length is not a multiple of shorter object length
```

```
38 #> 39 p <- ggplot(data = gapminder
x 40 expected ',' after expression| x = gdpPercap,
41 y = lifeExp))
42 |
```

```
39 p <- ggplot(data = gapminder,
40 mapping = aes(x = gdpPercap,
x 41 y = lifeExp)))
42 | unexpected token ')'
43 | ^ |
```

# Let's Go!

# Time to make a plot

Like before:

```
gapminder

A tibble: 1,704 × 6
country continent year lifeExp pop gdpPercap
<fct> <fct> <int> <dbl> <int> <dbl>
1 Afghanistan Asia 1952 28.8 8425333 779.
2 Afghanistan Asia 1957 30.3 9240934 821.
3 Afghanistan Asia 1962 32.0 10267083 853.
4 Afghanistan Asia 1967 34.0 11537966 836.
5 Afghanistan Asia 1972 36.1 13079460 740.
6 Afghanistan Asia 1977 38.4 14880372 786.
7 Afghanistan Asia 1982 39.9 12881816 978.
8 Afghanistan Asia 1987 40.8 13867957 852.
9 Afghanistan Asia 1992 41.7 16317921 649.
10 Afghanistan Asia 1997 41.8 22227415 635.
... with 1,694 more rows
```

# Like before

```
library(tidyverse)
library(gapminder)

p <- ggplot(data = gapminder,
 mapping = aes(x = gdpPercap,
 y = lifeExp))

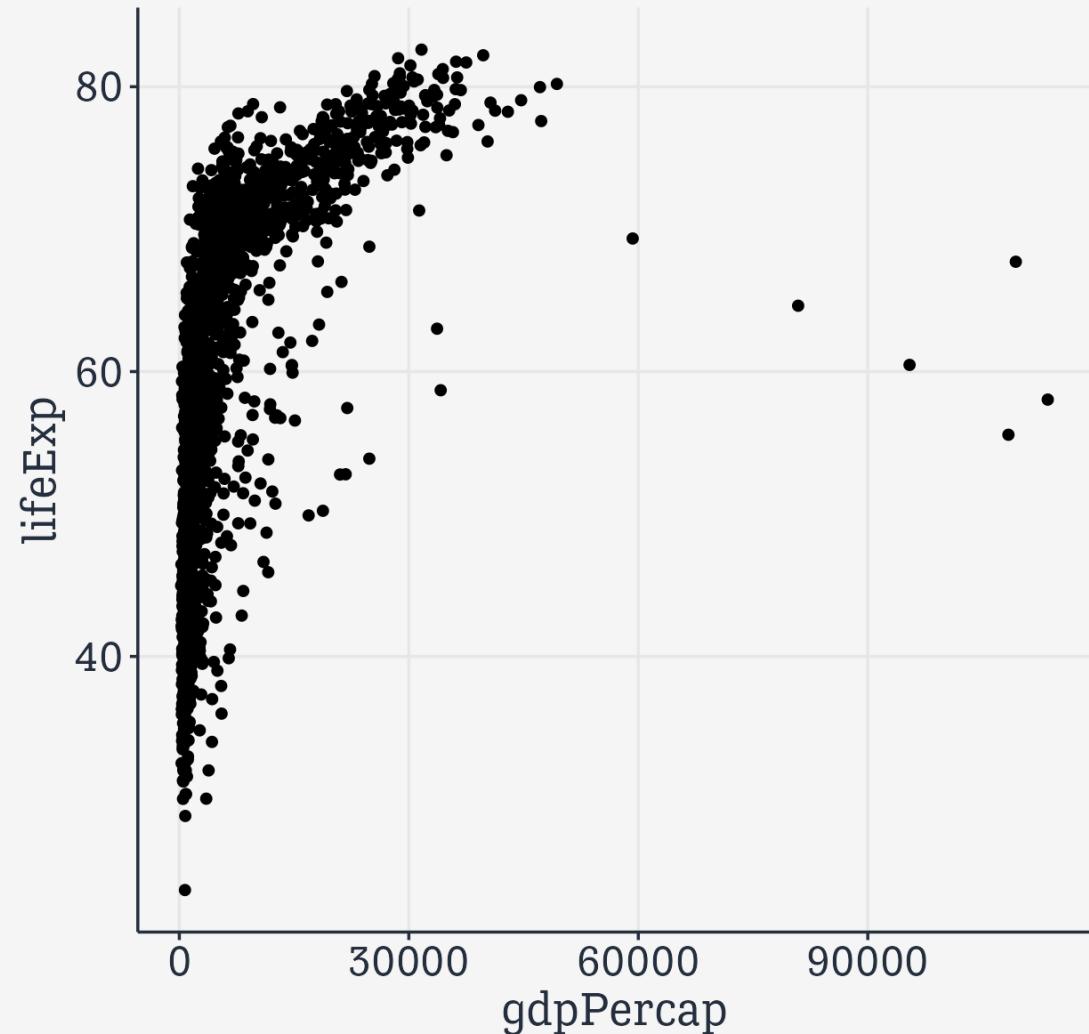
p + geom_point()
```

# Like before

```
library(tidyverse)
library(gapminder)

p <- ggplot(data = gapminder,
 mapping = aes(x = gdpPercap,
 y = lifeExp))

p + geom_point()
```



# What we did

```
library(tidyverse)
library(gapminder)
```

Load the packages we need: `tidyverse` and `gapminder`

# What we did

```
library(tidyverse)
library(gapminder)
```

```
p <- ggplot(data = gapminder,
 mapping = aes(x = gdpPercap,
 y = lifeExp))
```

Load the packages we need: `tidyverse` and `gapminder`

New object named **p gets** the output of the **ggplot() function**, given these **arguments**  
Notice how one of the arguments, **mapping**, is itself taking the output of a function named **aes()**

# What we did

```
library(tidyverse)
library(gapminder)
```

Load the packages we need: `tidyverse` and `gapminder`

```
p <- ggplot(data = gapminder,
 mapping = aes(x = gdpPercap,
 y = lifeExp))
```

New object named **p gets** the output of the **ggplot() function**, given these **arguments**  
Notice how one of the arguments, **mapping**, is itself taking the output of a function named **aes()**

```
p + geom_point()
```

Show me the output of the **p** object and the **geom\_point()** function.  
The **+** here acts just like the **|>** pipe, but for ggplot functions only. (This is an accident of history.)

# And what is R doing?

R objects are just lists of stuff to use or things to do

Objects are like Bento Boxes



**Data**

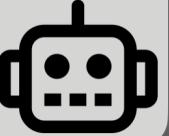
```
A tibble: 1,704 x 6
 country continent year lifeExp
 <fctr> <fctr> <int> <dbl> <i>
1 Afghanistan Asia 1952 28.801 8425
2 Afghanistan Asia 1957 30.332 9240
3 Afghanistan Asia 1962 31.997 10267
4 Afghanistan Asia 1967 34.020 11537
5 Afghanistan Asia 1972 36.088 13079
6 Afghanistan Asia 1977 38.438 14880
7 Afghanistan Asia 1982 39.854 12881
8 Afghanistan Asia 1987 40.822 13867
9 Afghanistan Asia 1992 41.674 16317
10 Afghanistan Asia 1997 41.763 22227
```

**Mappings**

- **Represent or Map**  
“**lifeExp**” using the x axis
- **Represent or Map**  
“**gdpPercap**” using the y axis
- **Represent or Map**  
“**continent**” using colors

Just deal with these for me automatically for now, robot

|                                  |                                    |
|----------------------------------|------------------------------------|
| <b>scales</b><br><b>plot_env</b> | <b>coordinates</b><br><b>theme</b> |
|----------------------------------|------------------------------------|

bleep bloop


**Labels**

- **Label** the x axis “GDP per Capita”
- **Label** the y axis “Life Expectancy”
- **Label** the color key “Continent”

# The **p** object

Environment

History

Connections

Git

Tutorial



Import Dataset ▾



297 MiB ▾



R ▾



Global Environment ▾

Data



p

List of 9

Functions

# Peek in with the object inspector

The screenshot shows the RStudio object inspector for an object named 'p'. The top bar includes search, previous/next, and show attributes buttons. The main table lists the components of 'p':

| Name        | Type                                      | Value                                 |
|-------------|-------------------------------------------|---------------------------------------|
| p           | list [9] (S3: gg, ggplot)                 | List of length 9                      |
| data        | list [1704 x 6] (S3: tbl_df, tbl, tibble) | A tibble with 1704 rows and 6 columns |
| layers      | list [0]                                  | List of length 0                      |
| scales      | environment [1] (S3: ScalesList)          | <environment: 0x11f8106b0>            |
| mapping     | list [2] (S3: uneval)                     | List of length 2                      |
| theme       | list [0]                                  | List of length 0                      |
| coordinates | environment [5] (S3: CoordCanvas)         | <environment: 0x11f8150c8>            |
| facet       | environment [2] (S3: FacetNull)           | <environment: 0x12a81ab00>            |
| plot_env    | environment [6]                           | <environment: R_GlobalEnv>            |
| labels      | list [2]                                  | List of length 2                      |

# Peek in with the object inspector