Finding your way in R

Data Wrangling: Session 2

Kieran Healy

Statistical Horizons, December 2022

Using RMarkdown to produce and reproduce your work

Where we want to end up

Covid Cases

Kieran Healy

4/18/2021

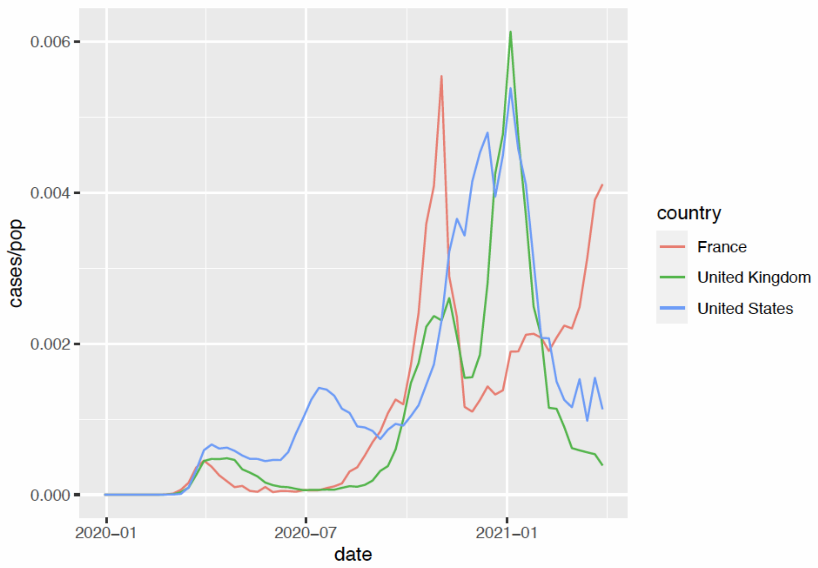
Lorem Ipsum

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.

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

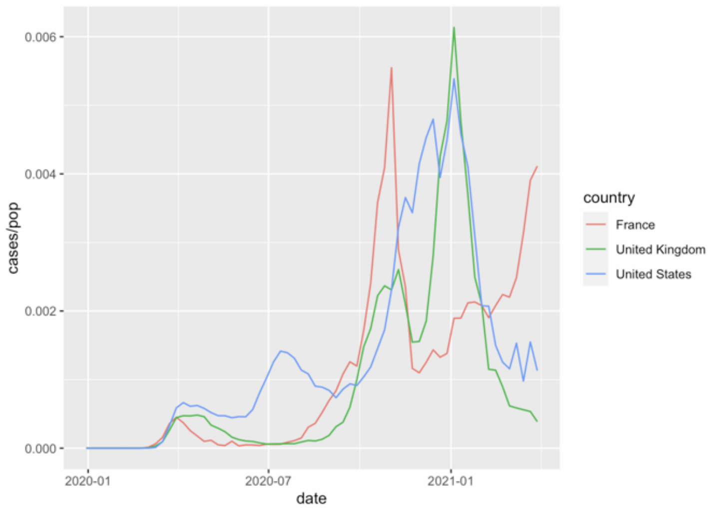
Lorem Ipsum

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.

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

Covid Cases

Kieran Healy

4/18/2021

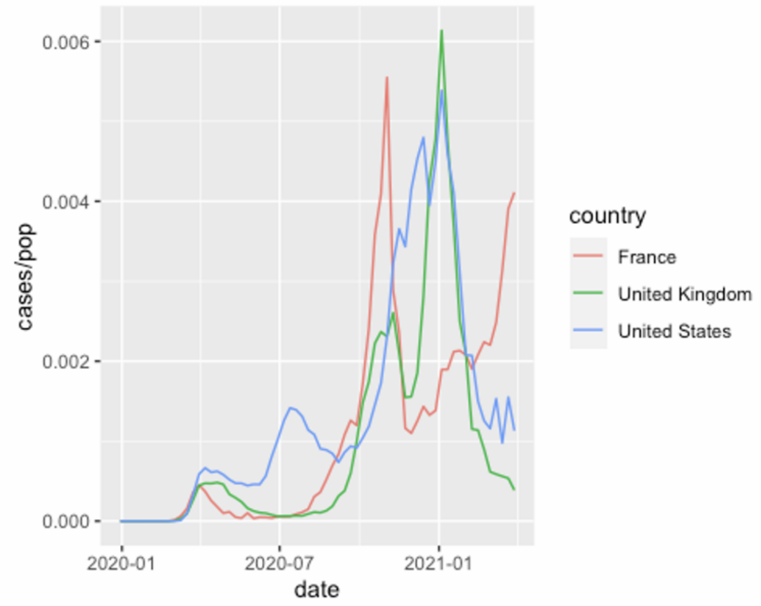
Lorem Ipsum

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.

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.



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(…)
```

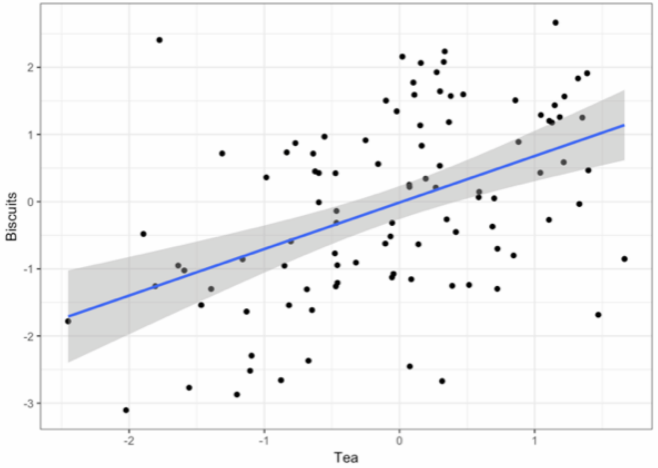
Write an R script with some notes inside. Create some figures and tables, paste them into our document.

This will work, but we can do better.

We can make this ...

1. Lorem Ipsum

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.



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.

... 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.
```

```r
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.
```

The code gets replaced by its output

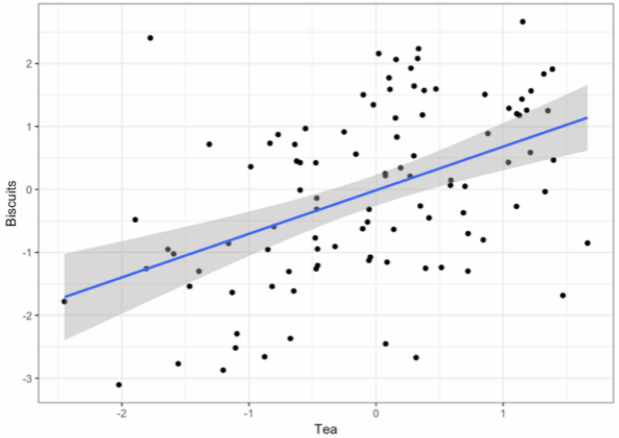
```
# Lorem Ipsum

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 exercitation
ullamco laboris nisi ut aliquip ex ea commodo consequat.
```

```r
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 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.

```
---
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

Covid Cases

Kieran Healy

4/18/2021

Introduction

We'll be looking at some COVID case data.

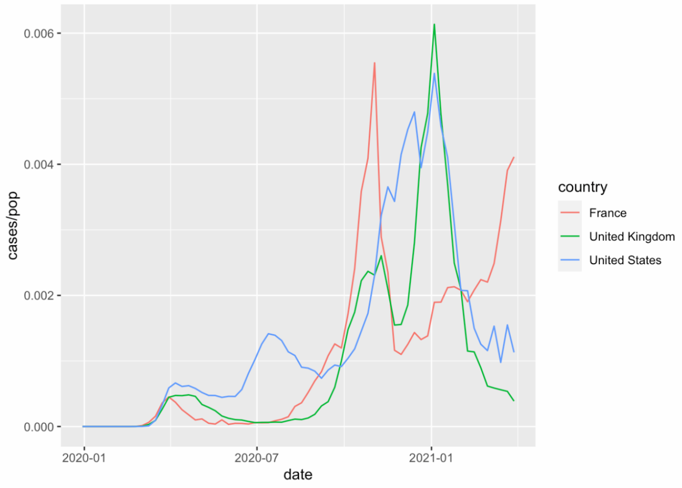
Lorem Ipsum

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.

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.



Covid Cases

Kieran Healy

4/18/2021

Introduction

We'll be looking at some COVID case data.

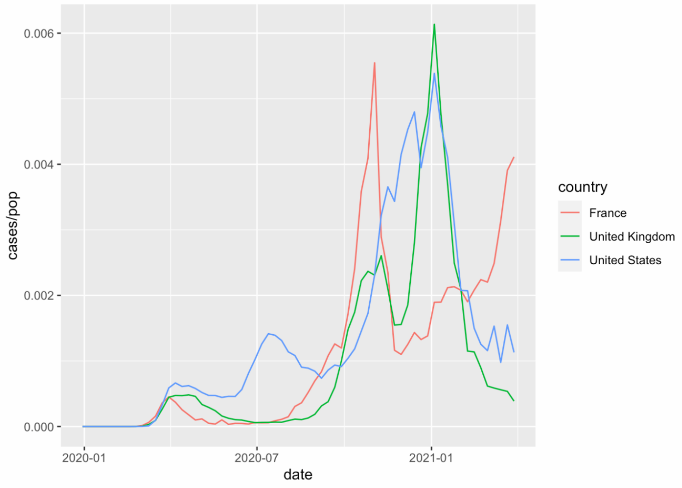
Lorem Ipsum

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.

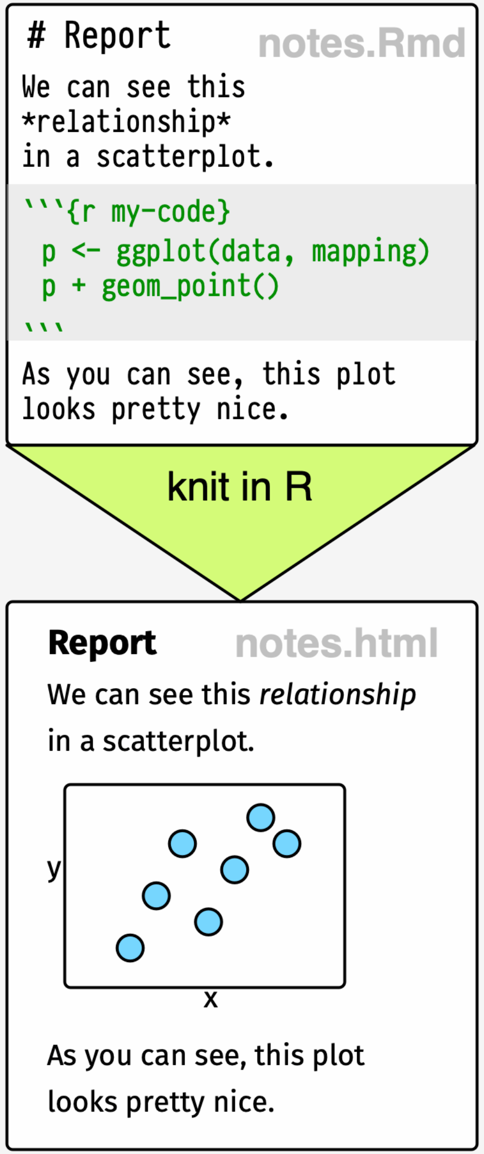
Total cases in three countries.

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.



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



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 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.

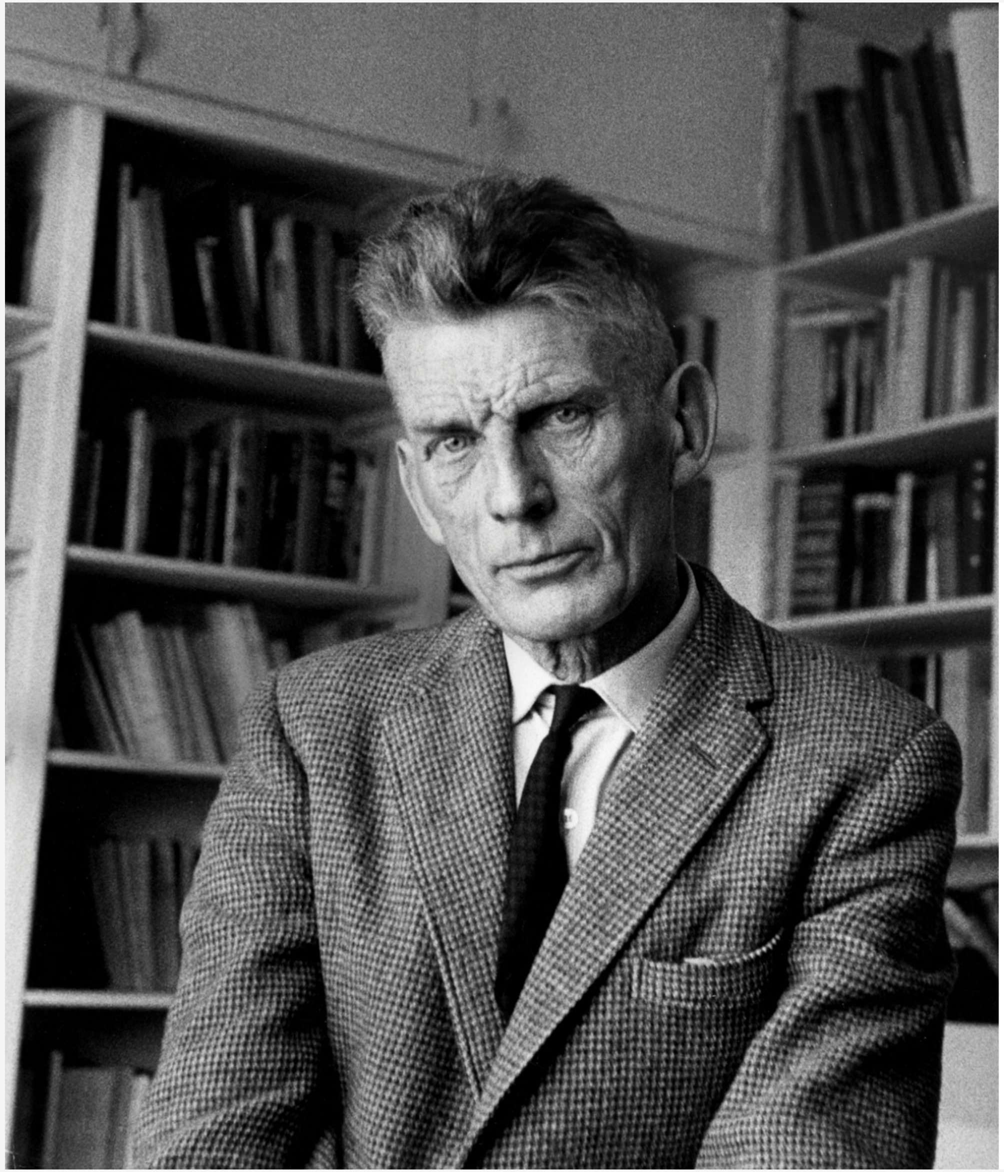
After some orientation to what's where, you will learn best by *doing*.

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

TYPE OUT YOUR CODE **BY HAND**

TYPE OUT YOUR CODE **BY HAND**

By far the best way to get a feel for how R works, and to get comfortable with the cycle of try-fail-cry-fix-wail-retry that is a permanent part of writing any code.



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

Samuel Beckett,
early data analyst

GETTING ORIENTED

Loading the tidyverse libraries

```r
library(tidyverse)
```

The tidyverse has several components.

We'll return to this message about Conflicts later.

Tidyverse components

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

Tidyverse components

`library(tidyverse)`	Call the package and ...
`Loading tidyverse: ggplot2`	<| **Draw graphs**
`Loading tidyverse: tibble`	<| **Nicer data tables**
`Loading tidyverse: tidyr`	<| **Tidy your data**
`Loading tidyverse: readr`	<| **Get data into R**
`Loading tidyverse: purrr`	<| **Fancy Iteration**
`Loading tidyverse: dplyr`	<| **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

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

```
## [1] 23.25
```

```r
sqrt(25)
```

```
## [1] 5
```

```r
log(100)
```

```
## [1] 4.60517
```

```r
log10(100)
```

```
## [1] 2
```

0. It's a calculator

Arithmetic

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

```
## [1] 23.25
```

```r
sqrt(25)
```

```
## [1] 5
```

```r
log(100)
```

```
## [1] 4.60517
```

```r
log10(100)
```

```
## [1] 2
```

Logic

```r
4 < 10
```

```
## [1] TRUE
```

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

```
## [1] TRUE
```

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

```
## [1] TRUE
```

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

```
## [1] FALSE
```

Boolean and Logical operators

Logical equality and inequality (yielding a `TRUE` or `FALSE` result) is done with == and ! =. Other logical operators include <, >, <=, >=, and ! for negation.

```
## 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
```

Watch out!

Here's a gotcha. You might think you could write `3 < 5 & 7` and have it be interpreted as "Three is less than five and also less than seven [True or False?]":

```
3 < 5 & 7
```

```
## [1] TRUE
```

Watch out!

Here's a gotcha. You might think you could write `3 < 5 & 7` and have it be interpreted as "Three is less than five and also less than seven [True or False?]":

```
3 < 5 & 7
```

```
## [1] TRUE
```

It seems to work!

Watch out!

But now try `3 < 5 & 1`, where your intention is "Three is less than five and also less than one [True or False?]"

```
3 < 5 & 1
```

```
## [1] TRUE
```

What's happening is that `3 < 5` is evaluated first, and resolves to `TRUE`, leaving us with the expression `TRUE & 1`.

R interprets this as `TRUE & as.logical(1)`.

In Boolean algebra, 1 resolves to `TRUE`. Any other number is `FALSE`. So,

```
TRUE & as.logical(1)
```

```
## [1] TRUE
```

Watch out!

But now try `3 < 5 & 1`, where your intention is "Three is less than five and also less than one [True or False?]"

```
3 < 5 & 1
```

```
## [1] TRUE
```

What's happening is that `3 < 5` is evaluated first, and resolves to `TRUE`, leaving us with the expression `TRUE & 1`.

R interprets this as `TRUE & as.logical(1)`.

In Boolean algebra, 1 resolves to `TRUE`. Any other number is `FALSE`. So,

```
TRUE & as.logical(1)
```

```
## [1] TRUE
```

```
3 < 5 & 3 < 1
```

```
## [1] FALSE
```

You have to make your comparisons explicit.

Logic and floating point arithmetic

Let's evaluate `0.6 + 0.2 == 0.8`

Logic and floating point arithmetic

Let's evaluate `0.6 + 0.2 == 0.8`

```
0.6 + 0.2 == 0.8
```

```
## [1] TRUE
```

Logic and floating point arithmetic

Let's evaluate `0.6 + 0.2 == 0.8`

```
0.6 + 0.2 == 0.8
```

```
## [1] TRUE
```

Now let's try `0.6 + 0.3 == 0.9`

Logic and floating point arithmetic

Let's evaluate `0.6 + 0.2 == 0.8`

```
0.6 + 0.2 == 0.8
```

```
## [1] TRUE
```

Now let's try `0.6 + 0.3 == 0.9`

```
0.6 + 0.3 == 0.9
```

```
## [1] FALSE
```

Logic and floating point arithmetic

Let's evaluate `0.6 + 0.2 == 0.8`

```
0.6 + 0.2 == 0.8
```

```
## [1] TRUE
```

Now let's try `0.6 + 0.3 == 0.9`

```
0.6 + 0.3 == 0.9
```

```
## [1] FALSE
```

Er. That's not right.

Logic and floating point arithmetic

Let's evaluate `0.6 + 0.2 == 0.8`

```r
0.6 + 0.2 == 0.8
```

```
## [1] TRUE
```

Now let's try `0.6 + 0.3 == 0.9`

```r
0.6 + 0.3 == 0.9
```

```
## [1] FALSE
```

Er. That's not right.

Welcome to floating point math!

Logic and floating point arithmetic

In Base 10, you can't precisely express fractions like $\frac{1}{3}$ and $\frac{1}{9}$. They come out as repeating decimals: 0.3333... or 0.1111... You *can* cleanly represent fractions that use a prime factor of the base, which in the case of Base 10 are 2 and 5.

Logic and floating point arithmetic

In Base 10, you can't precisely express fractions like $\frac{1}{3}$ and $\frac{1}{9}$. They come out as repeating decimals: 0.3333... or 0.1111... You *can* cleanly represent fractions that use a prime factor of the base, which in the case of Base 10 are 2 and 5.

Computers represent numbers as binary (i.e. Base 2) floating-points. In Base 2, the only prime factor is 2. So $\frac{1}{5}$ or $\frac{1}{10}$ in binary would be repeating.

Logic and floating point arithmetic

When you do binary math on repeating numbers and convert back to decimals you get tiny leftovers, and this can mess up *logical* comparisons of equality. The `all.equal()` function exists for this purpose.

Logic and floating point arithmetic

When you do binary math on repeating numbers and convert back to decimals you get tiny leftovers, and this can mess up *logical* comparisons of equality. The `all.equal()` function exists for this purpose.

```
print(.1 + .2)
```

```
## [1] 0.3
```

```
print(.1 + .2, digits=18)
```

```
## [1] 0.300000000000000044
```

```
all.equal(.1 + .2, 0.3)
```

```
## [1] TRUE
```

Logic and floating point arithmetic

When you do binary math on repeating numbers and convert back to decimals you get tiny leftovers, and this can mess up *logical* comparisons of equality. The `all.equal()` function exists for this purpose.

```
print(.1 + .2)
```

```
## [1] 0.3
```

```
print(.1 + .2, digits=18)
```

```
## [1] 0.300000000000000044
```

```
all.equal(.1 + .2, 0.3)
```

```
## [1] TRUE
```

See e.g. https://0.30000000000000004.com

More later on why this might bite you, and how to deal with it

More later on why this might bite you, and how to deal with it

For now, "Be very careful about doing logical comparisons on floating-point numbers" is not a bad rule.

1. Everything in R has a name

```r
my_numbers # We created this a few minutes ago
```

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

```r
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"
```

```r
pi  # Also built-in
```

```
## [1] 3.141593
```

Some names are forbidden

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

```
TRUE
FALSE
Inf
NaN
NA
NULL

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

```r
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"
```

```r
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

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
```

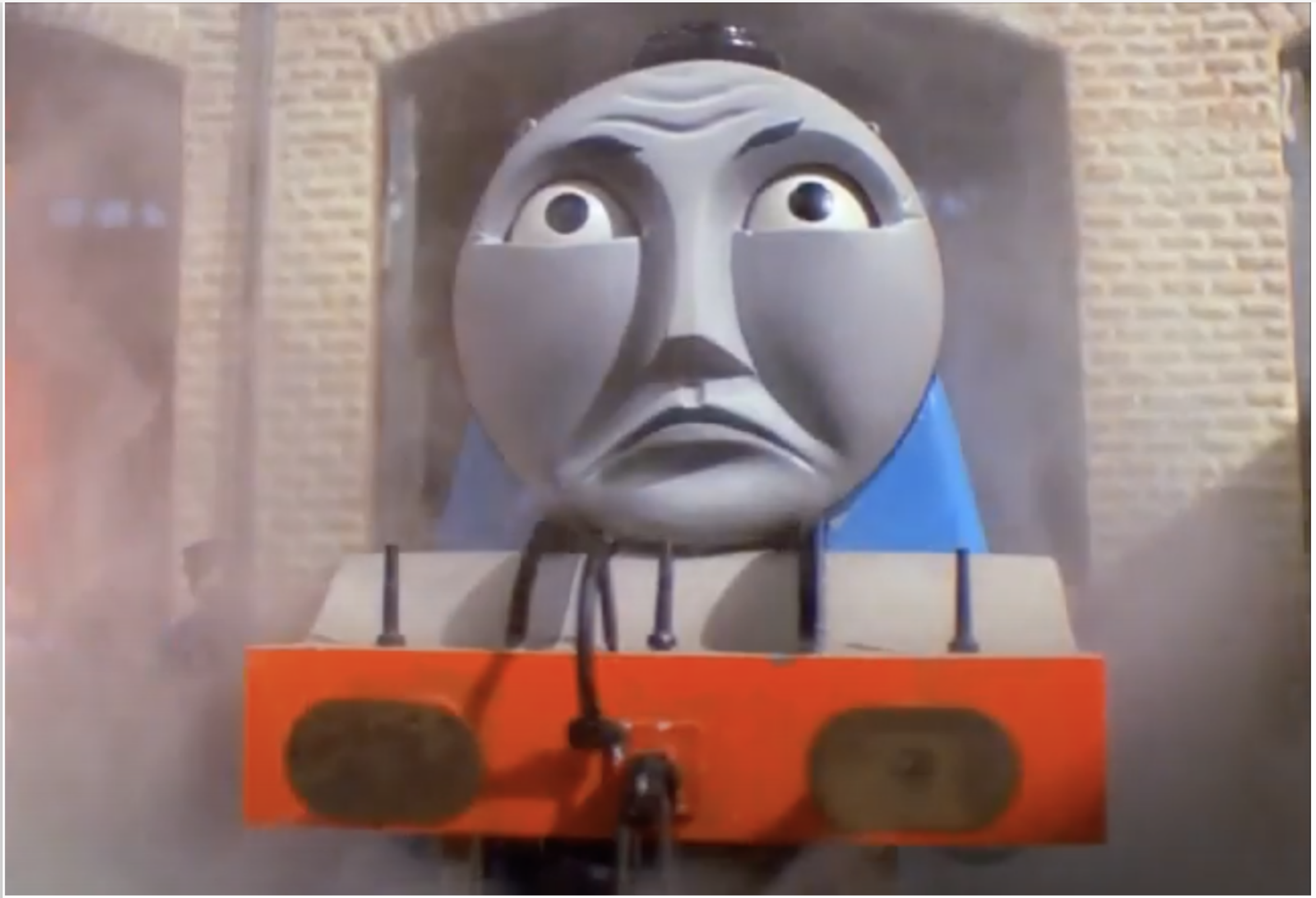
```
## [1]  1  2  3  1  3  5 25
```

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

I'm going to use `<-` for assignment throughout.

Be consistent either way.

Assignment with =



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

4. Do things to objects with functions

```r
## 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)
```

```r
my_numbers
```

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

Functions can be identified by the parentheses after their names.

```r
my_numbers
```

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

```r
## 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**

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

What functions usually do

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

They perform **actions**

They produce, or return, **outputs**

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

```
[1] 6.25
```

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

```r
help(mean)
```

```r
## quicker
?mean
```

What functions usually do

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

```r
help(mean)
```

```r
## quicker
?mean
```

> How to read an R help page ...

What functions usually do

Arguments often tell the function what to do in specific circumstances

```r
missing_numbers <- c(1:10, NA, 20, 32, 50, 104, 32, 147, 99, NA, 45)

mean(missing_numbers)
```

```
## [1] NA
```

```r
mean(missing_numbers, na.rm = TRUE)
```

```
## [1] 32.44444
```

What functions usually do

Arguments often tell the function what to do in specific circumstances

```r
missing_numbers <- c(1:10, NA, 20, 32, 50, 104, 32, 147, 99, NA, 45)

mean(missing_numbers)
```

```
## [1] NA
```

```r
mean(missing_numbers, na.rm = TRUE)
```

```
## [1] 32.44444
```

Or select from one of several options

```r
## 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.

```r
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.)

```r
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**

```r
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

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

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

```r
mean(c(1:20))
```

```
## [1] 10.5
```

Functions can be nested

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

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

```r
mean(c(1:20))
```

```
## [1] 10.5
```

```r
summary(mean(c(1:20)))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##    10.5    10.5    10.5    10.5    10.5    10.5
```

```r
names(summary(mean(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
```

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

```
## [1] 10.5
```

```
summary(mean(c(1:20)))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##    10.5    10.5    10.5    10.5    10.5    10.5
```

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

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

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

```
## [1] 6
```

Functions can be nested

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

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

```r
mean(c(1:20))
```

```
## [1] 10.5
```

```r
summary(mean(c(1:20)))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##    10.5    10.5    10.5    10.5    10.5    10.5
```

```r
names(summary(mean(c(1:20))))
```

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

```r
length(names(summary(mean(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*:

```
c(1:20) |> mean() |> summary() |> names() |>  length()
```

```
## [1] 6
```

Use the pipe operator: |>

Instead of nesting functions in parentheses, we can use the *pipe operator*:

```
c(1:20) |> mean() |> 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*:

```r
c(1:20) |> mean() |> summary() |> names() |>  length()
```

```
## [1] 6
```

Read this operator as "***and then***" Better, vertical space is free in R:

```r
c(1:20) |>
  mean() |>
  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 other pipe: `%>%`

The Base R pipe operator, `|>` is a relatively recent addition to R.

Piping operations were originally introduced in a package called called `magrittr`, where it took the form `%>%`

The other pipe: `%>%`

The Base R pipe operator, `|>` is a relatively recent addition to R.

Piping operations were originally introduced in a package called called `magrittr`, where it took the form `%>%`

It's been so successful, a version of it has been incorporated into Base R.

The other pipe: `%>%`

The Base R pipe operator, **`|>`** is a relatively recent addition to R.

Piping operations were originally introduced in a package called called `magrittr`, where it took the form **`%>%`**

It's been so successful, a version of it has been incorporated into Base R.

The Base R pipe *mostly* but does not *quite* work the same way as **`%>%`** in every case.

We'll use the Base R pipe in this course, but you'll see the Magrittr pipe a lot out in the world.

*With the Base R 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

Packages are loaded into your working environment using the `library()` function.

Functions are bundled into packages

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_…¹ body_…² sex    year
##    <fct>   <fct>              <dbl>         <dbl>      <int>   <int> <fct> <int>
##  1 Adelie  Torgersen           39.1          18.7        181    3750 male   2007
##  2 Adelie  Torgersen           39.5          17.4        186    3800 fema…  2007
##  3 Adelie  Torgersen           40.3          18          195    3250 fema…  2007
##  4 Adelie  Torgersen           NA            NA           NA      NA <NA>   2007
##  5 Adelie  Torgersen           36.7          19.3        193    3450 fema…  2007
##  6 Adelie  Torgersen           39.3          20.6        190    3650 male   2007
##  7 Adelie  Torgersen           38.9          17.8        181    3625 fema…  2007
##  8 Adelie  Torgersen           39.2          19.6        195    4675 male   2007
##  9 Adelie  Torgersen           34.1          18.1        193    3475 <NA>   2007
## 10 Adelie  Torgersen           42            20.2        190    4250 <NA>   2007
## # … with 334 more rows, and abbreviated variable names ¹flipper_length_mm,
## #   ²body_mass_g
```

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?

```r
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 tidyr::matches(), testthat::matches()
```

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

Remember this warning about conflicts?

```r
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 tidyr::matches(), testthat::matches()
```

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

Related concepts of *namespaces* and *environments*.

Scope of names

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

x
```

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

```r
y
```

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

Scope of names

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

x
```

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

```r
y
```

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

```r
mean()

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

Scope of names

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

x
```

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

```r
y
```

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

```r
mean()

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

```r
mean(x) # argument names are internal to functions
```

```
## [1] 5.5
```

```r
mean(x = x)
```

```
## [1] 5.5
```

```r
mean(x = y)
```

```
## [1] 95
```

```r
x
```

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

```r
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

```r
typeof(my_smry)
```

```
## [1] "double"
```

```r
attributes(my_smry)
```

```
## $names
## [1] "Min."    "1st Qu." "Median"  "Mean"    "3rd Qu." "Max."
##
## $class
## [1] "summaryDefault" "table"
```

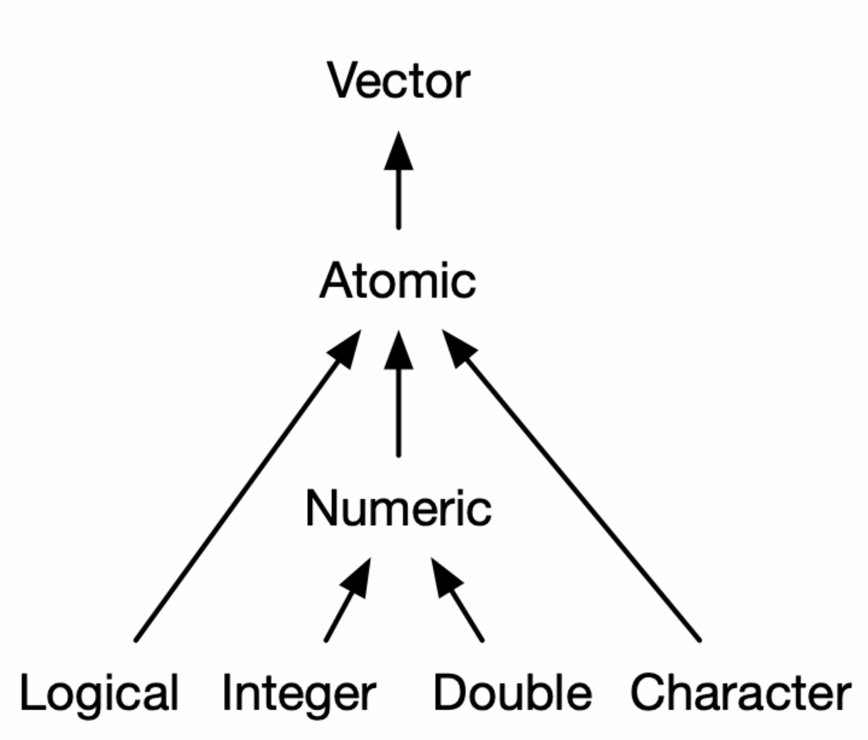
```r
## In this case, the functions extract the corresponding attribute
class(my_smry)
```

```
## [1] "summaryDefault" "table"
```

```r
names(my_smry)
```

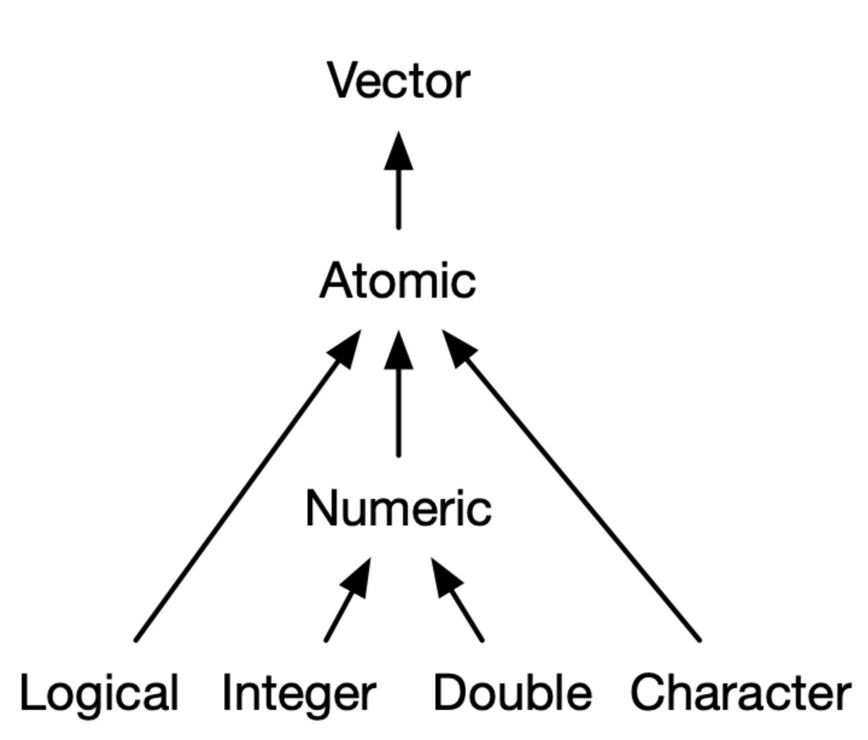
```
## [1] "Min."    "1st Qu." "Median"  "Mean"    "3rd Qu." "Max."
```

The most common types of **vector**



From Hadley Wickham, *Advanced R*

The most common types of vector



From Hadley Wickham, *Advanced R*

```
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
```

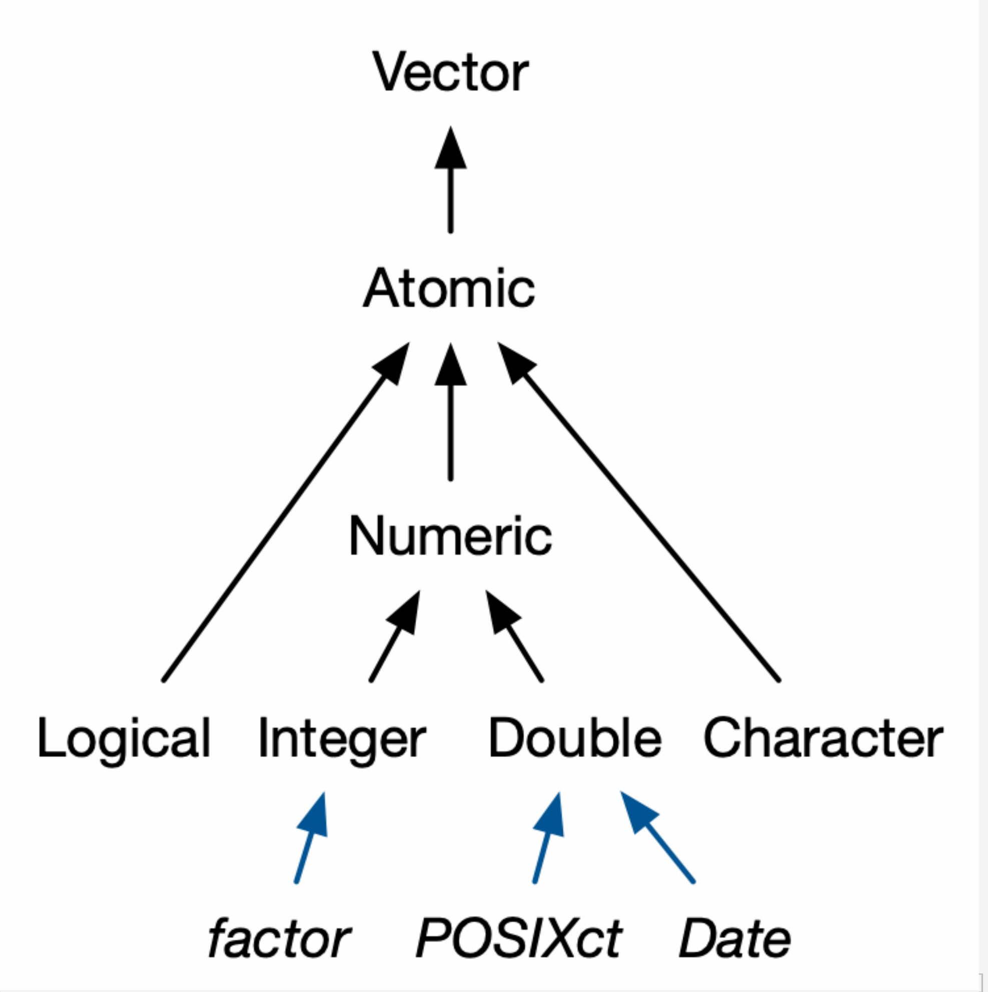
```
my_chr <- c("Mary", "had", "a", "little", "lamb")
is.character(my_chr)
```

```
## [1] TRUE
```

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

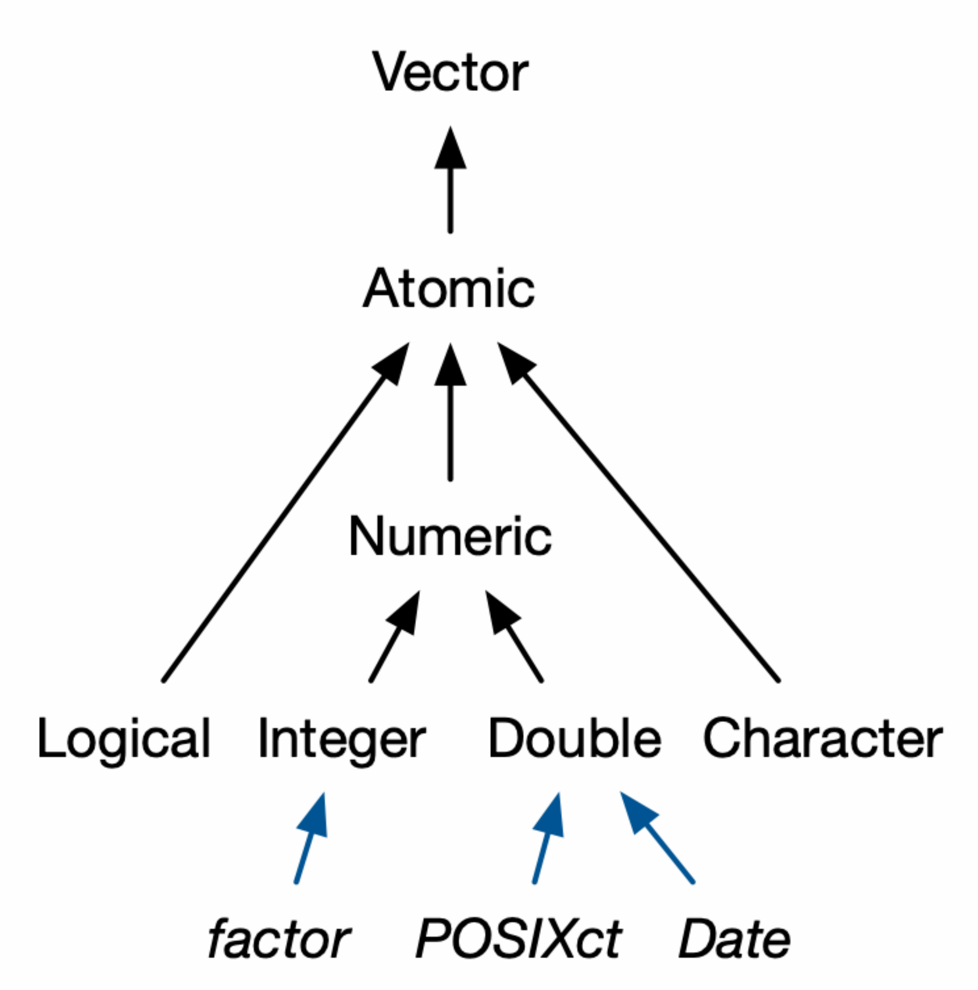
```
## [1] TRUE
```

The most common types of **vector**



From Hadley Wickham, *Advanced R*

The most common types of vector



From Hadley Wickham, *Advanced R*

```r
## Factors are for storing undordered or ordered categorical
x <- factor(c("Yes", "No", "No", "Maybe", "Yes", "Yes", "Yes
x
```

```
## [1] Yes   No    No    Maybe Yes   Yes   Yes   No
## Levels: Maybe No Yes
```

```r
summary(x) # Alphabetical order by default
```

```
## Maybe    No   Yes
##     1     3     4
```

```r
typeof(x)       # Underneath, a factor is a type of integer
```

```
## [1] "integer"
```

```r
attributes(x)   # ... with labels for its numbers, or "level
```

```
## $levels
## [1] "Maybe" "No"    "Yes"
##
## $class
## [1] "factor"
```

```r
class(x)
```

```
## [1] "factor"
```

```r
levels(x)
```

```
## [1] "Maybe" "No"    "Yes"
```

Vector types can't be heterogenous

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

Vector types 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"
```

Vector types 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
```

A table of data is a kind of **list**

```r
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
```

```r
class(gapminder)
```

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

```r
typeof(gapminder) # hmm
```

```
## [1] "list"
```

Lists *can* be heterogenous. Underneath, most complex R objects are some kind of list with different components.

Some classes are nested

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
```

Some classes are nested

Base R's trusty `data.frame`

```r
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
```

```r
class(titanic)
```

```
## [1] "data.frame"
```

```r
## 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`

```r
## tibbles are build on data frames
titanic_tb <- as_tibble(titanic)
class(titanic)
```

```
## [1] "data.frame"
```

```r
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
```

```r
class(titanic_tb)
```

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

Some classes are nested

Base R's trusty `data.frame`

```r
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
```

```r
class(titanic)
```

```
## [1] "data.frame"
```

```r
## 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`

```r
## tibbles are build on data frames
titanic_tb <- as_tibble(titanic)
class(titanic)
```

```
## [1] "data.frame"
```

```r
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
```

```r
class(titanic_tb)
```

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

A data frame and a tibble are both fundamentally a list of vectors of the same length, where the vectors can be of different types (e.g. numeric, character, logical, etc)

All of this will matter later on

```r
gss_sm
```

```
## # A tibble: 2,867 × 32
##     year    id ballot   age childs sibs  degree race  sex   region incom…¹ relig
##    <dbl> <dbl> <labe> <dbl>  <dbl> <lab> <fct>  <fct> <fct> <fct>  <fct>   <fct>
##  1  2016     1 1         47      3 2     Bache… White Male  New E… $17000… None
##  2  2016     2 2         61      0 3     High … White Male  New E… $50000… None
##  3  2016     3 3         72      2 3     Bache… White Male  New E… $75000… Cath…
##  4  2016     4 1         43      4 3     High … White Fema… New E… $17000… Cath…
##  5  2016     5 3         55      2 2     Gradu… White Fema… New E… $17000… None
##  6  2016     6 2         53      2 2     Junio… White Fema… New E… $60000… None
##  7  2016     7 1         50      2 2     High … White Male  New E… $17000… None
##  8  2016     8 3         23      3 6     High … Other Fema… Middl… $30000… Cath…
##  9  2016     9 1         45      3 5     High … Black Male  Middl… $60000… Prot…
## 10  2016    10 3         71      4 1     Junio… White Male  Middl… $60000… None
## # … with 2,857 more rows, 20 more variables: 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>, and
## #   abbreviated variable name ¹income16
```

All of this will matter later on

```r
gss_sm
```

```
## # A tibble: 2,867 × 32
##     year    id ballot   age childs sibs  degree race  sex   region incom…¹ relig
##    <dbl> <dbl> <labe> <dbl>  <dbl> <lab> <fct>  <fct> <fct> <fct>  <fct>   <fct>
##  1  2016     1 1         47      3 2     Bache… White Male  New E… $17000… None
##  2  2016     2 2         61      0 3     High … White Male  New E… $50000… None
##  3  2016     3 3         72      2 3     Bache… White Male  New E… $75000… Cath…
##  4  2016     4 1         43      4 3     High … White Fema… New E… $17000… Cath…
##  5  2016     5 3         55      2 2     Gradu… White Fema… New E… $17000… None
##  6  2016     6 2         53      2 2     Junio… White Fema… New E… $60000… None
##  7  2016     7 1         50      2 2     High … White Male  New E… $17000… None
##  8  2016     8 3         23      3 6     High … Other Fema… Middl… $30000… Cath…
##  9  2016     9 1         45      3 5     High … Black Male  Middl… $60000… Prot…
## 10  2016    10 3         71      4 1     Junio… White Male  Middl… $60000… None
## # … with 2,857 more rows, 20 more variables: 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>, and
## #   abbreviated variable name ¹income16
```

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.

See if you can predict what the following operations do:

*And arrays, too.

6. Arithmetic on vectors

```r
my_numbers
```

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

```r
result1 <- my_numbers + 1
```

6. Arithmetic on vectors

```
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

```r
my_numbers
```

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

```r
result1 <- my_numbers + 1
```

```r
result1
```

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

```r
result2 <- my_numbers + my_numbers
```

6. Arithmetic on vectors

```
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

```r
my_numbers
```

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

```r
result1 <- my_numbers + 1
```

```r
result1
```

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

```r
result2 <- my_numbers + my_numbers
```

```r
result2
```

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

```r
two_nums <- c(5, 10)

result3 <- my_numbers + two_nums
```

6. Arithmetic on vectors

```r
my_numbers
```

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

```r
result1 <- my_numbers + 1
```

```r
result1
```

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

```r
result2 <- my_numbers + my_numbers
```

```r
result2
```

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

```r
two_nums <- c(5, 10)

result3 <- my_numbers + two_nums
```

```r
result3
```

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

6. Arithmetic on vectors

```r
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

```r
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
```

```r
result4
```

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

Note that you get a *warning* here. It'll still do it, though! Don't ignore warnings until you understand what they mean.

7. R will be frustrating

7. R will be frustrating

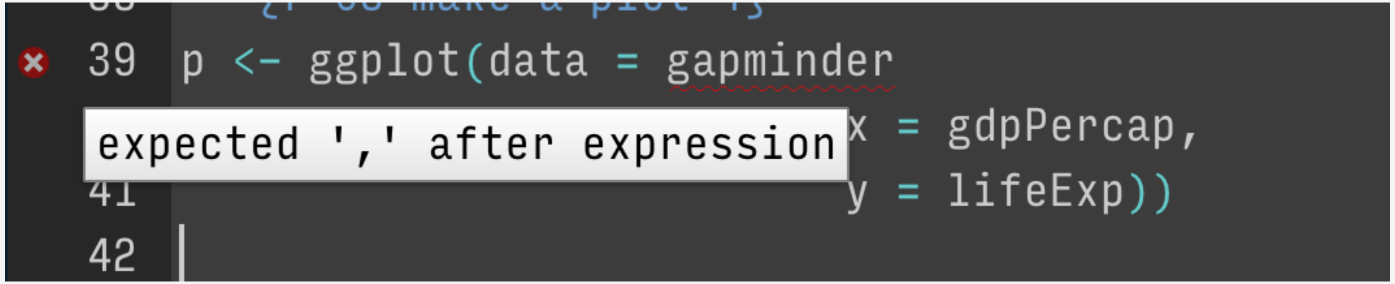
The IDE tries its best to help you. Learn to attend to what it is trying to say.

```
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
```



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
```