Welcome to the fall rstats peer-led graduate seminar

1. Go to https://ccbbatut.github.io/rstats_fall2016/, and download the materials for week 1
2. Open up RStudio
3. Take our survey:

   https://www.surveymonkey.com/r/RCCK76V

spncrfx@gmail.com

@foxandtheflu

Introduction to R, RStudio, and R Markdown

Spencer Fox
09 September 2016

Week 1 goals

Week 1 goals

- Run code in R using RStudio and R Markdown

Week 1 goals

- Run code in R using RStudio and R Markdown
- Install and use R packages

Week 1 goals

- Run code in R using RStudio and R Markdown
- Install and use R packages
- Play around with R variables

Week 1 goals

- Run code in R using RStudio and R Markdown
- Install and use R packages
- Play around with R variables
- Read data in R

Why program?

Why program?

- Simulation

Why program?

- Simulation
- Automation

Why program?

- Simulation
- Automation

- Reproducibility

When not to use R (or when to integrate R with other languages)

When not to use R (or when to integrate R with other languages)

- Bioinformatics (though it's getting better)

When not to use R (or when to integrate R with other languages)

- Bioinformatics (though it's getting better)
  - Focus of Spring seminar

When not to use R (or when to integrate R with other languages)

- Bioinformatics (though it's getting better)
    - Focus of Spring seminar
- String manipulation

When not to use R (or when to integrate R with other languages)

- Bioinformatics (though it's getting better)
    - Focus of Spring seminar
- String manipulation
- Your code takes days to run (integrate with Rcpp)

Why use R?

Why use R?

- Free

Why use R?

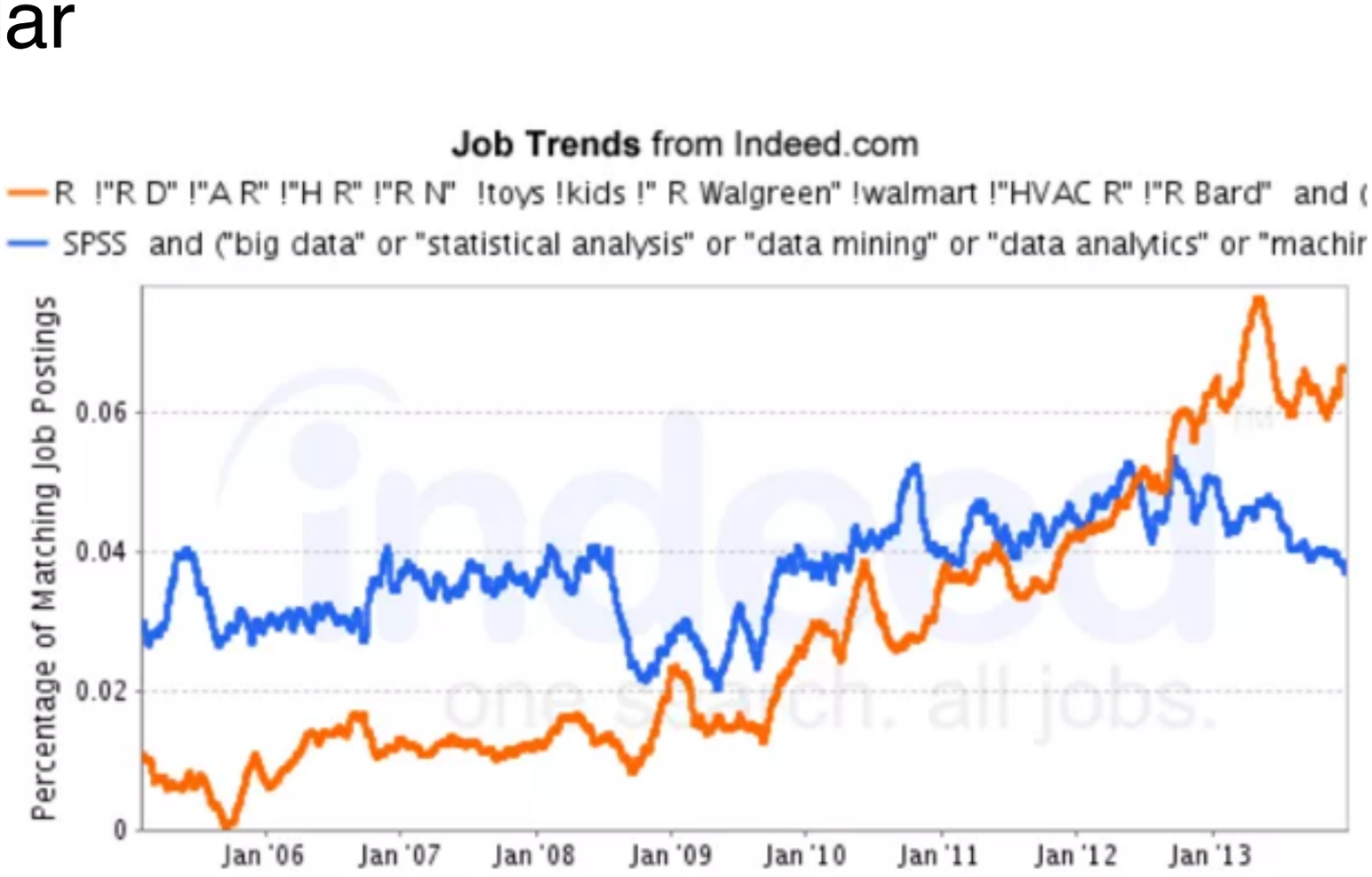
- Free
- Powerful Statistics

Why use R?

- Free
- Powerful Statistics
- Packages!

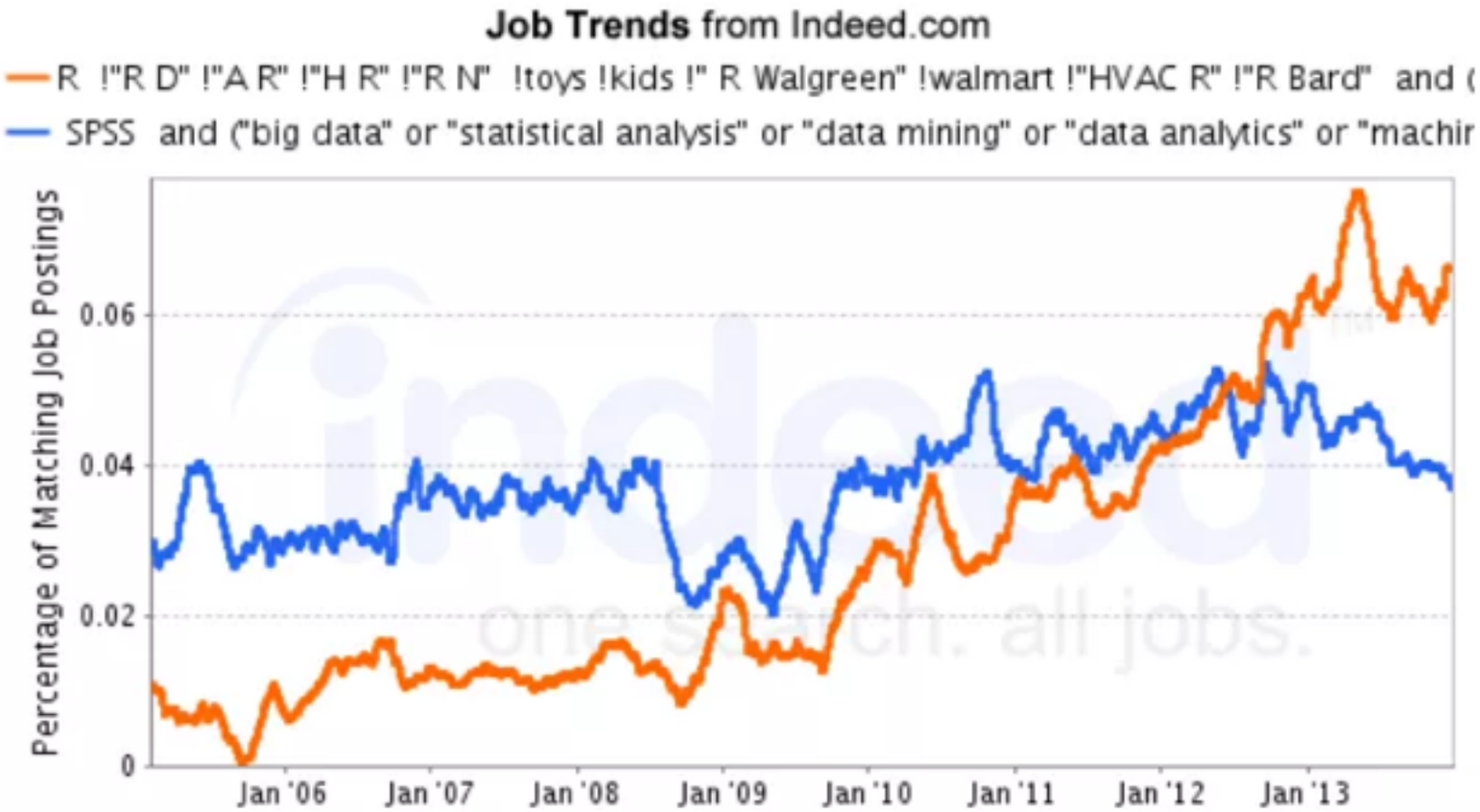
Why use R?

- Free
- Powerful Statistics
- Packages!
- Increasingly popular

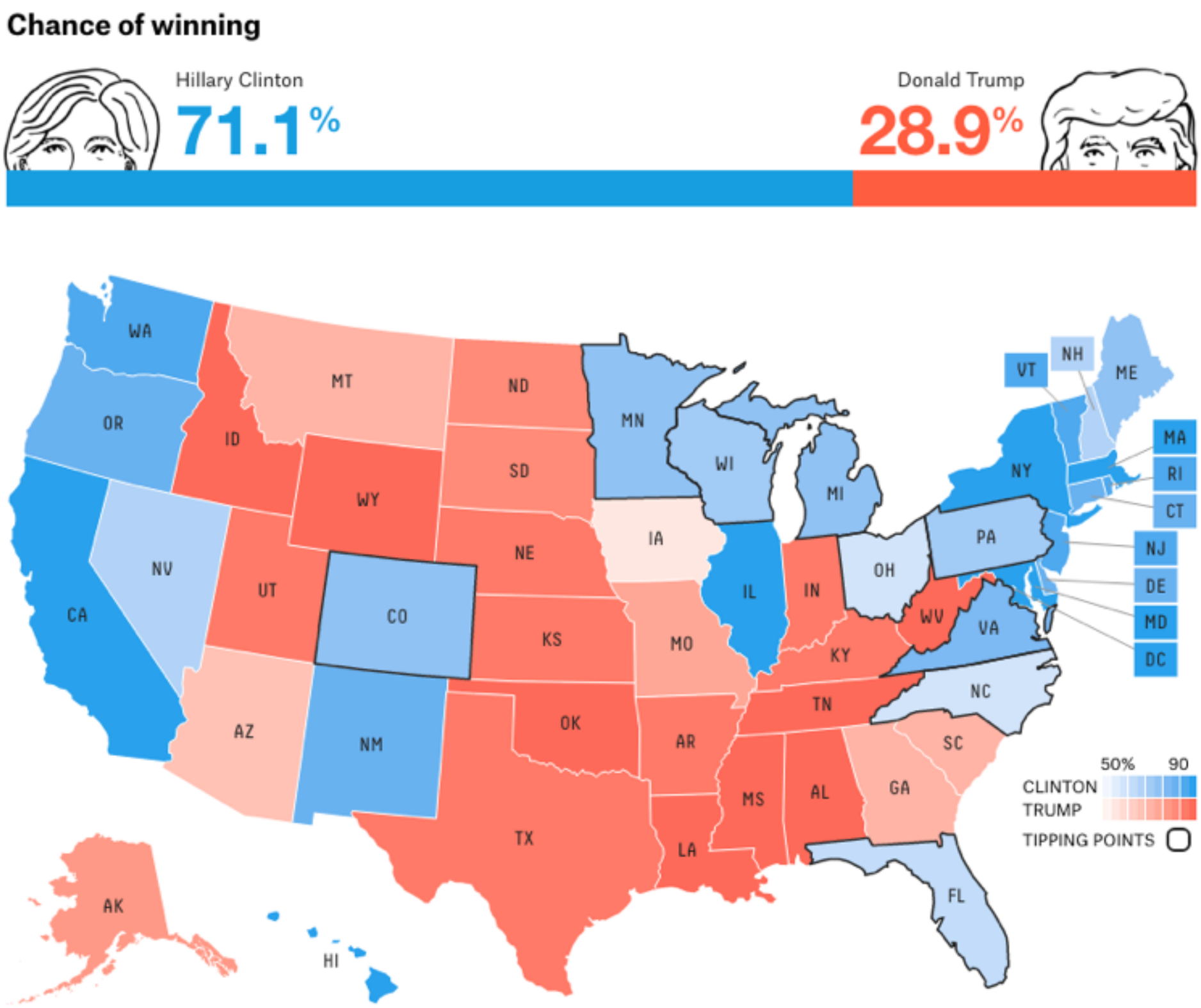


Why use R?

- Free
- Powerful Statistics
- Packages!
- Increasingly popular
- Visualization



Visualizing data



fivethirtyeight

Example R “Pipeline”

Example R “Pipeline”

1. Generate data

Example R “Pipeline”

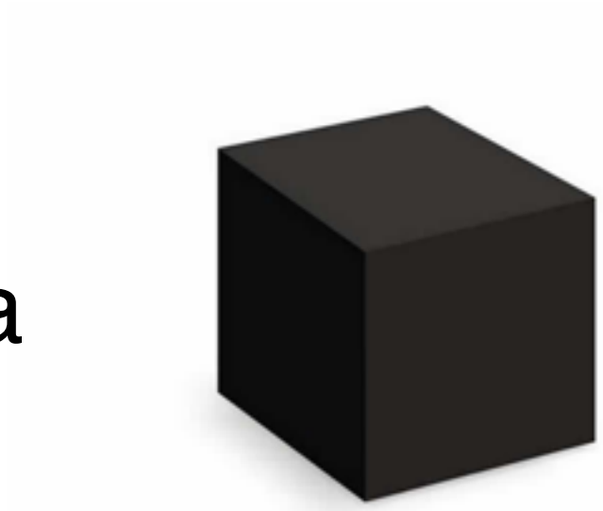
1. Generate data

2. Analyze data

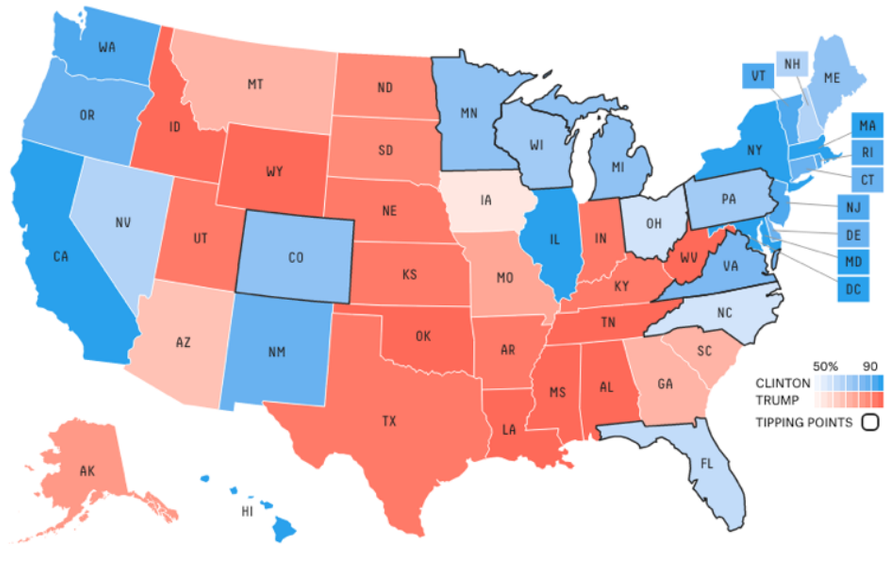
Example R “Pipeline”

1. Generate data

2. Analyze data



3. Show analysis



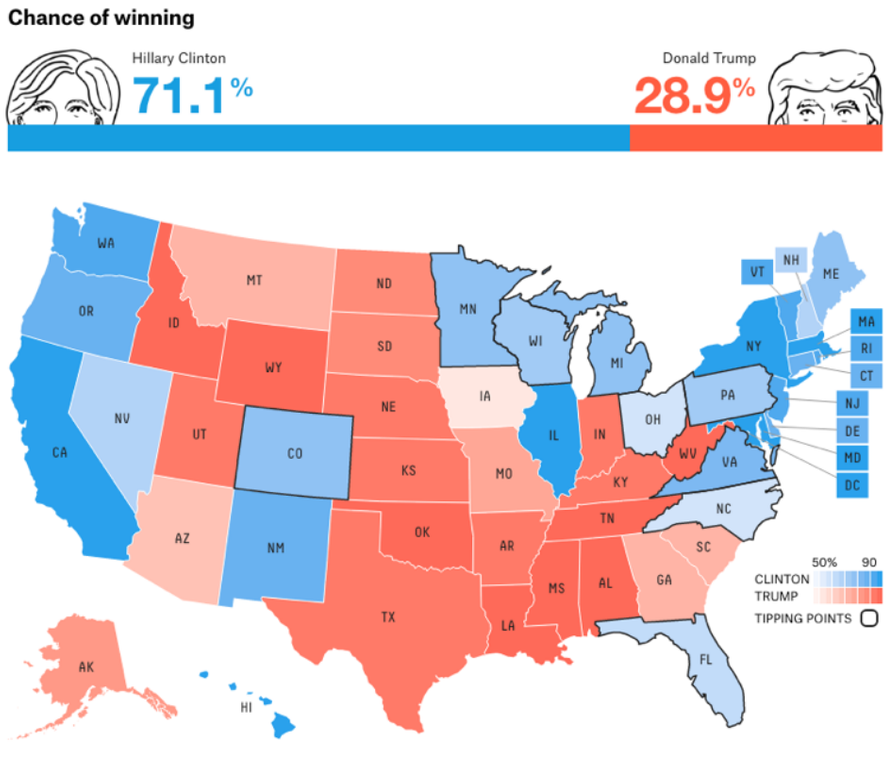
Example R “Pipeline”

1. Generate data

in R

2. Analyze data

3. Show analysis



Cooking a meal with R

Cooking a meal with R

What is the desired end product?

Cooking a meal with R

What is the desired end product?

What do we need to get it?

- Turkey
- Stuffing
- Green Beans
- Mashed Potatoes
- Gravy

Every dish has a function

Every dish has a function

Convert data to usable format

Every dish has a function

Convert data to usable format

Run a statistical analysis

Every dish has a function

Convert data to usable format

Run a statistical analysis

Run a 2nd stat analysis

Every dish has a function

Convert data to usable format

Make a summary plot

Run a statistical analysis

Run a 2nd stat analysis

Every function has a recipe

Every function has a recipe

1. Ingredients
    1. Green beans
    2. Creamy mushroom soup
    3. French fried onions

Every function has a recipe

1. Ingredients
    1. Green beans
    2. Creamy mushroom soup
    3. French fried onions
2. Instructions
    1. Heat oven to 350
    2. etc.

Every function has a recipe

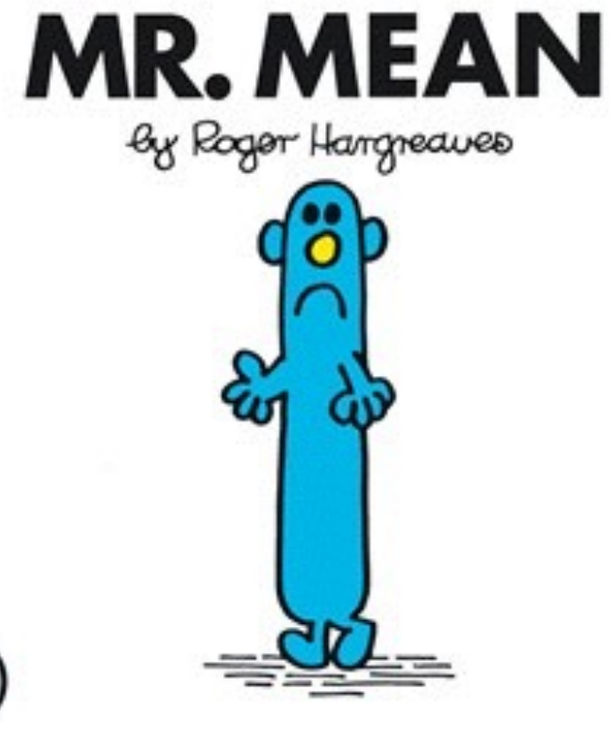
1. Ingredients
    1. Green beans
    2. Creamy mushroom soup
    3. French fried onions
2. Instructions
    1. Heat oven to 350
    2. etc.
3. Output
    1. Green bean casserole!

I thought I was learning R!!!!

Every function has a “recipe”: Calculating a mean

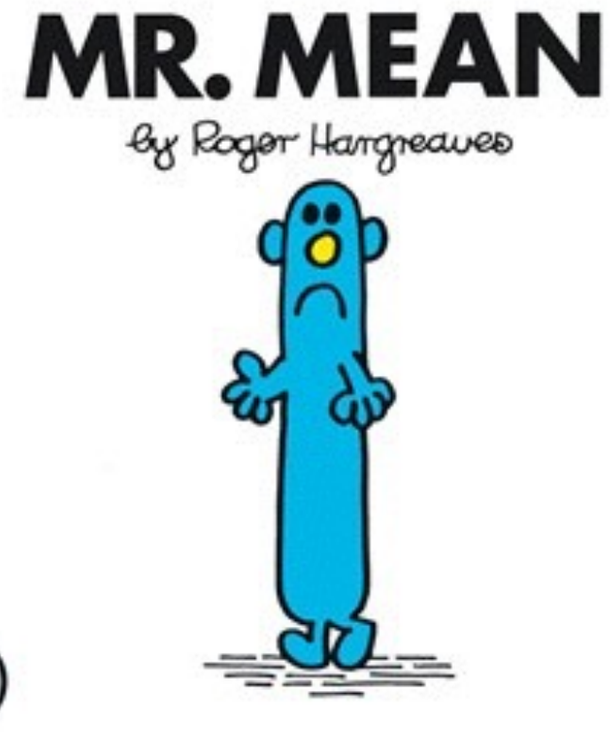
Every function has a “recipe”: Calculating a mean

1. Ingredients



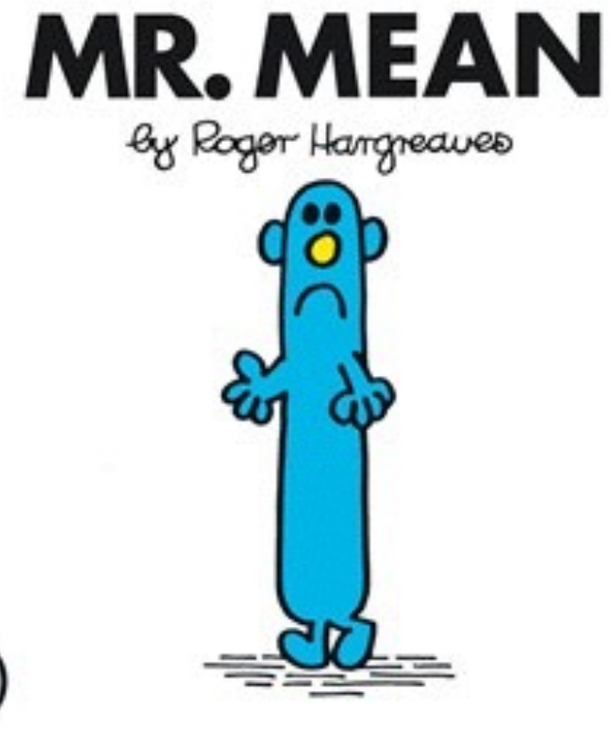
Every function has a “recipe”: Calculating a mean

1. Ingredients
    1. List of numbers



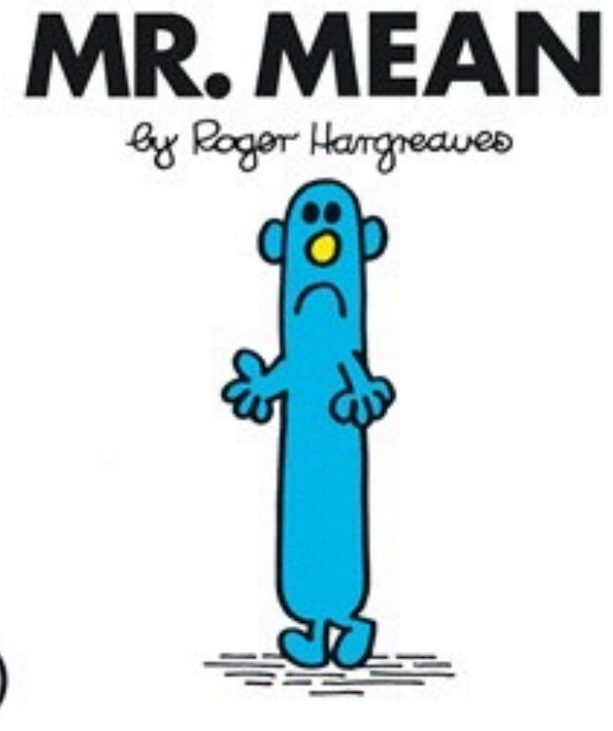
Every function has a “recipe”: Calculating a mean

1. Ingredients
    1. List of numbers
2. Instructions



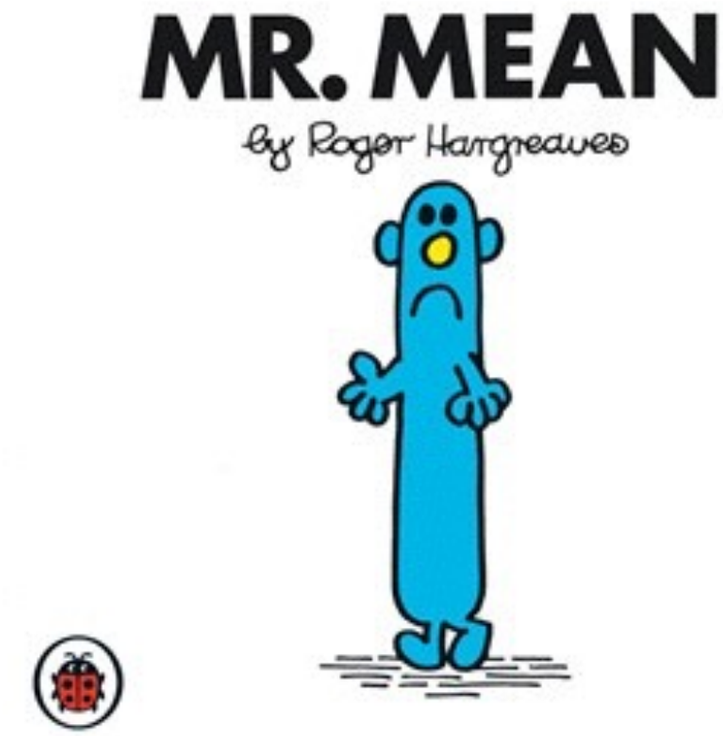
Every function has a “recipe”: Calculating a mean

1. Ingredients
    1. List of numbers
2. Instructions
    1. Count numbers



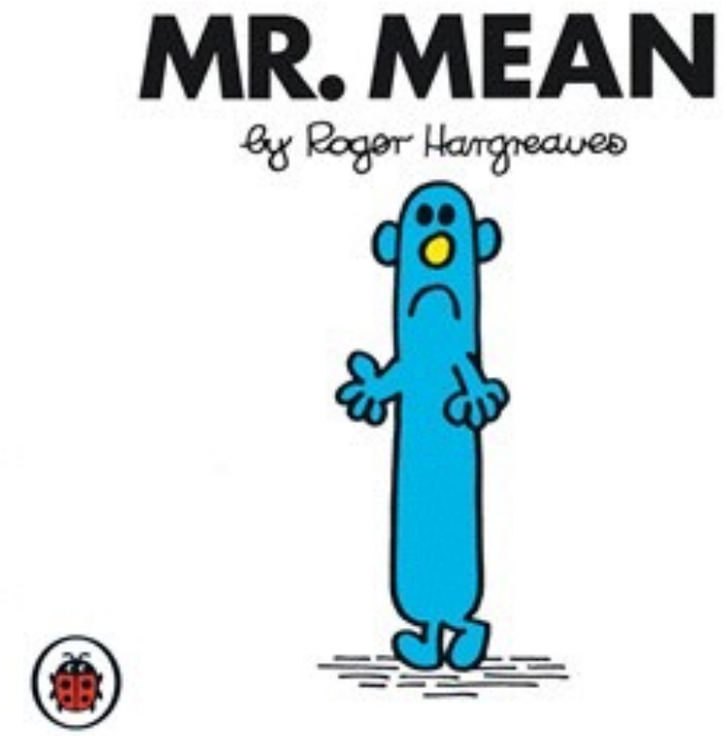
Every function has a “recipe”: Calculating a mean

1. Ingredients
    1. List of numbers
2. Instructions
    1. Count numbers
    2. Sum all of the numbers



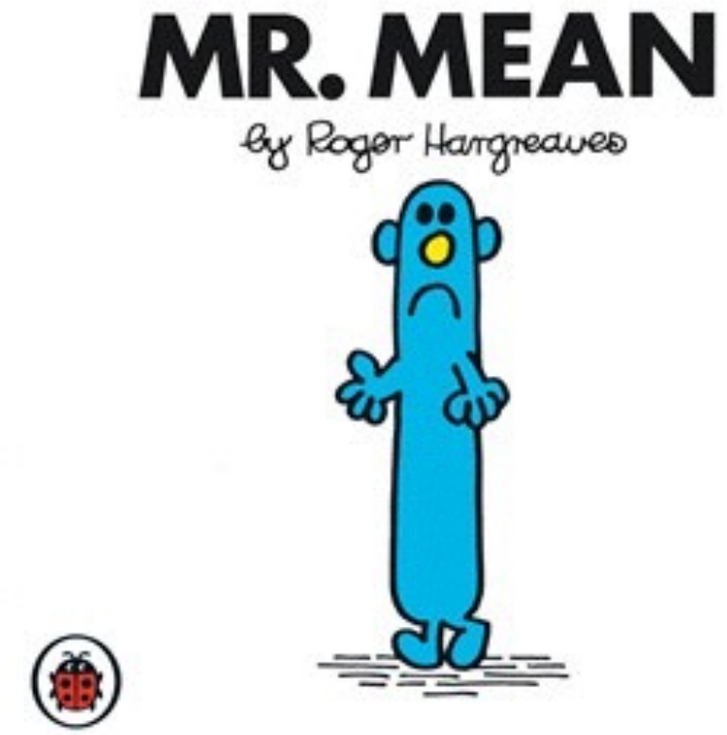
Every function has a “recipe”: Calculating a mean

1. Ingredients
    1. List of numbers
2. Instructions
    1. Count numbers
    2. Sum all of the numbers
3. Output



Every function has a “recipe”: Calculating a mean

1. Ingredients
    1. List of numbers
2. Instructions
    1. Count numbers
    2. Sum all of the numbers
3. Output
    1. Return Sum/Count



Every function has a “recipe”: Calculating a mean

1. Ingredients
    1. List of numbers
2. Instructions
    1. Count numbers
    2. Sum all of the numbers
3. Output
    1. Return Sum/Count


```
> mean(c(2,4,6,8,10))
[1] 6
```

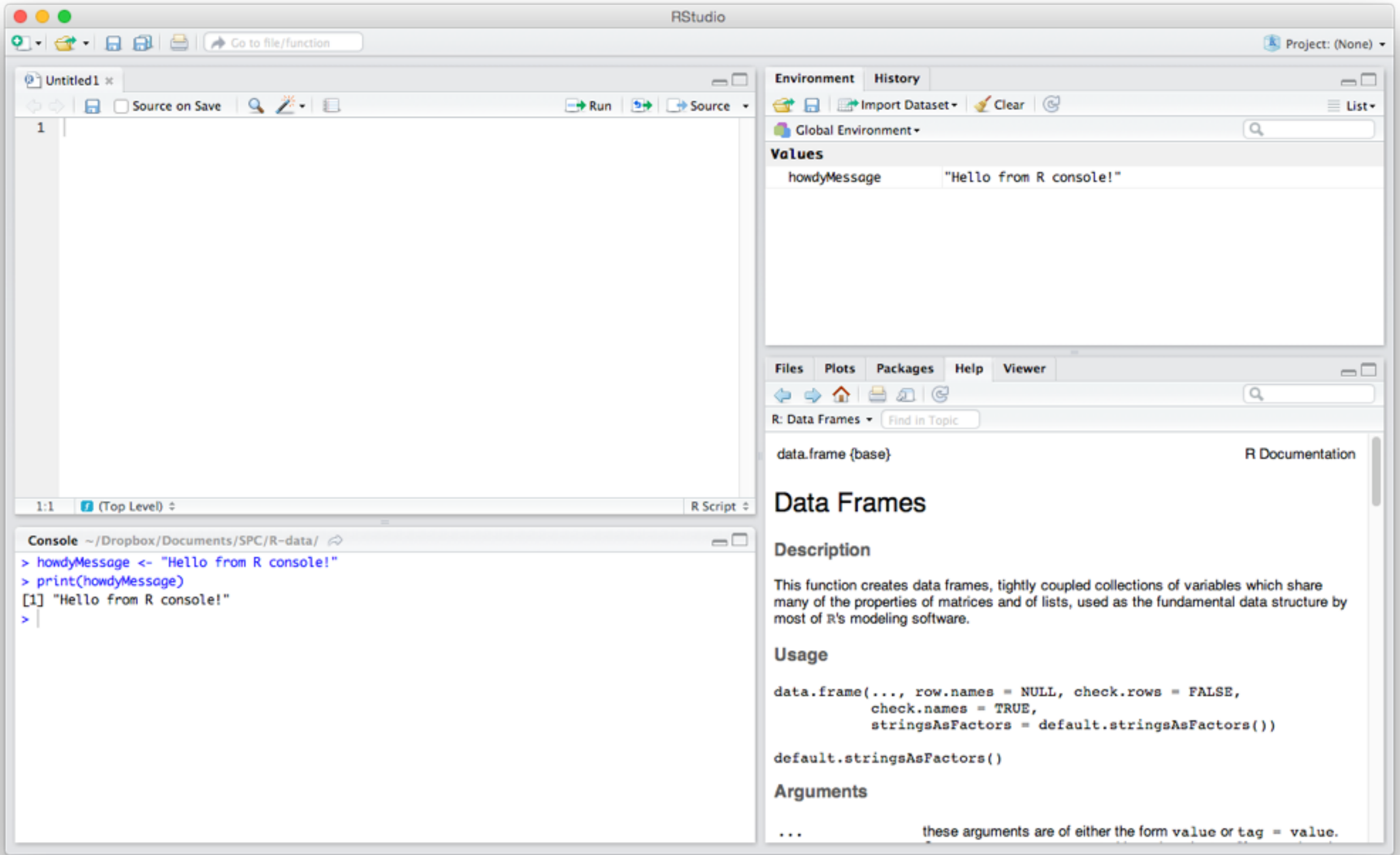
Why are packages so great?

Why are packages so great?

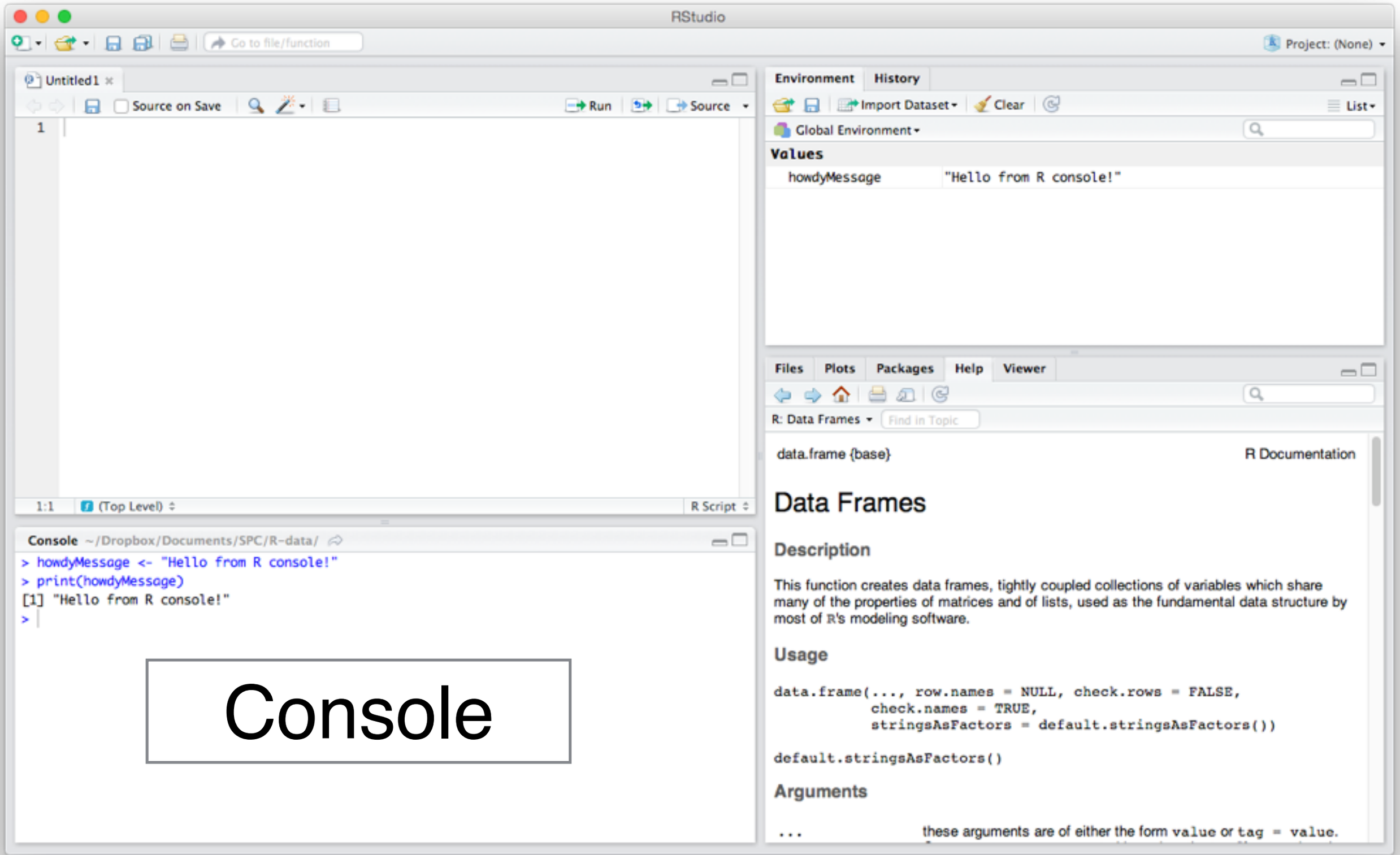
Why are packages so great?

Just need to understand inputs and outputs, and the rest is done for you.

Using R (RStudio)

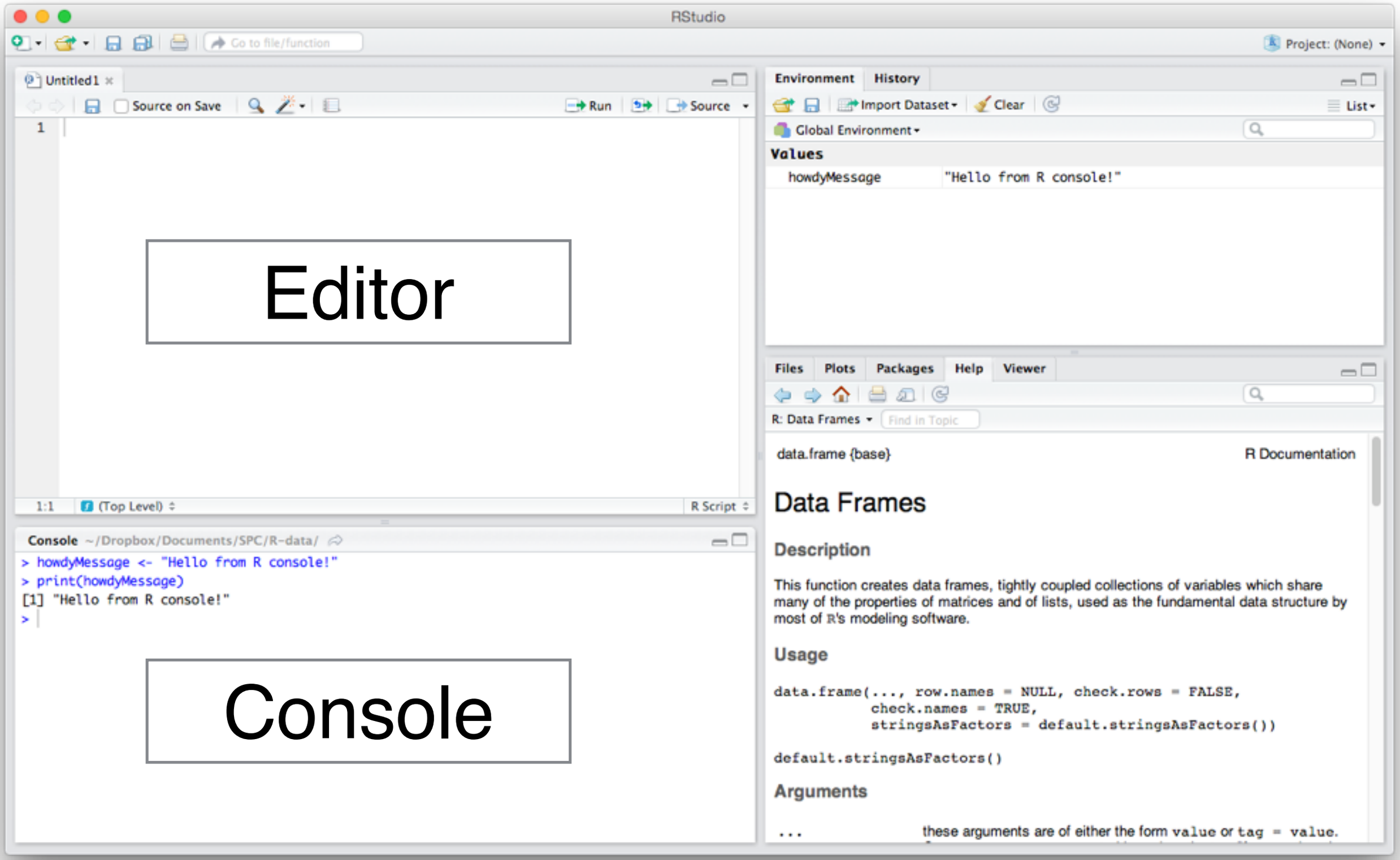


Using R (RStudio)

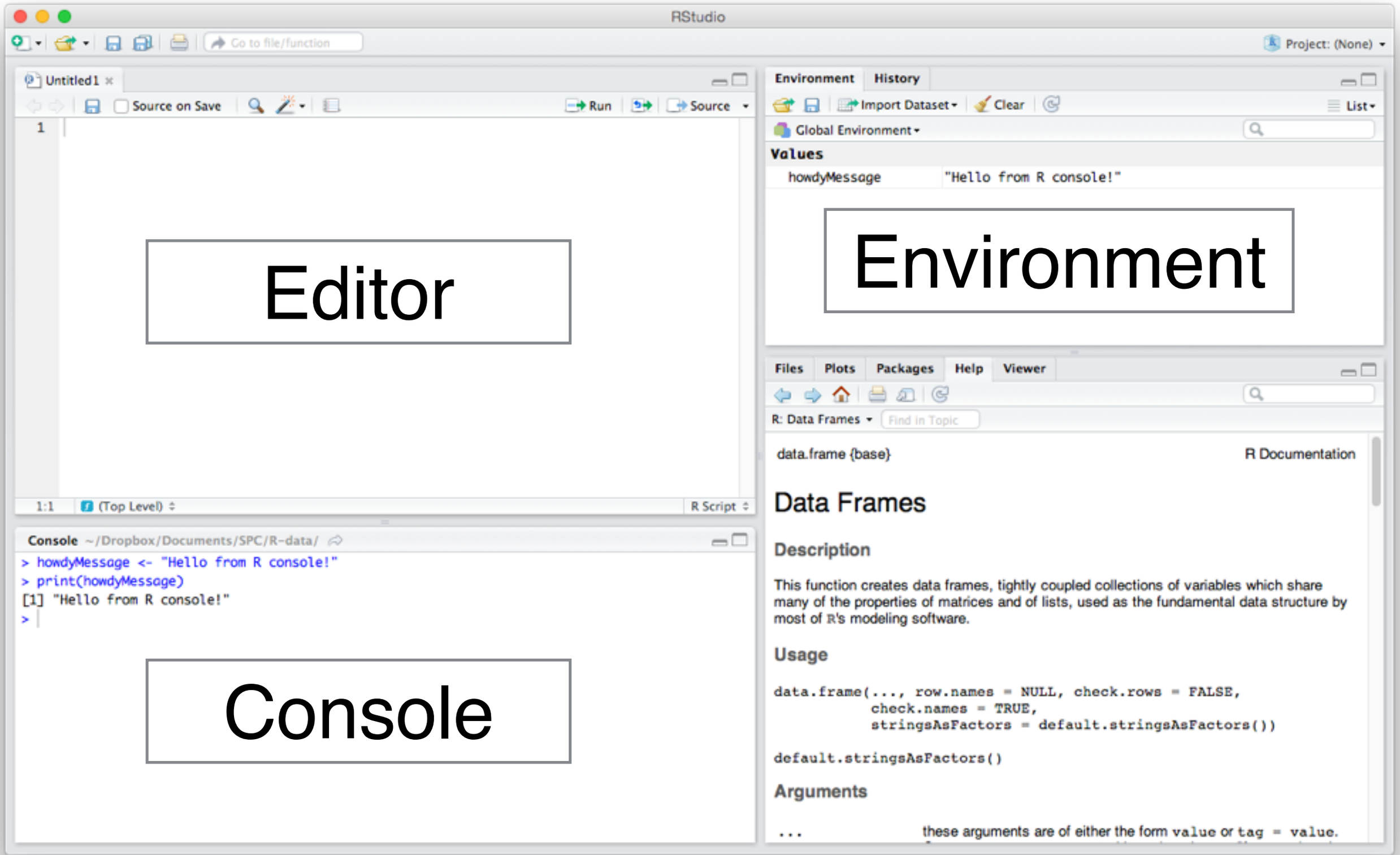


Console

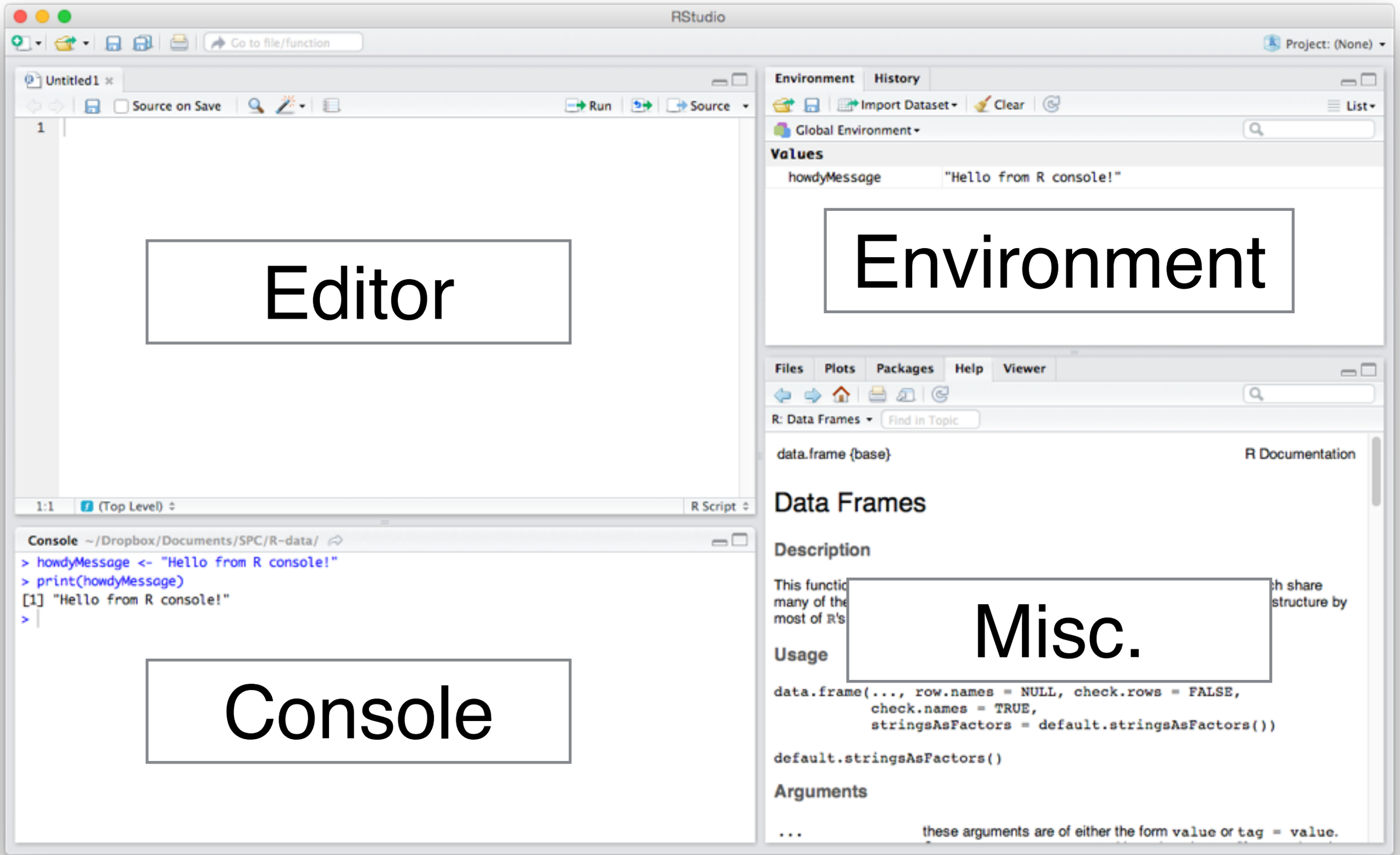
Using R (RStudio)



Using R (RStudio)



Using R (RStudio)



1st Programming Exercise

1. open up intro_rmd.Rmd in Rstudio
2. Start playing with code
    1. Do: **Ask Questions**, run code, change things and see what happens
    2. Don't: change code until it works and move on without understanding why

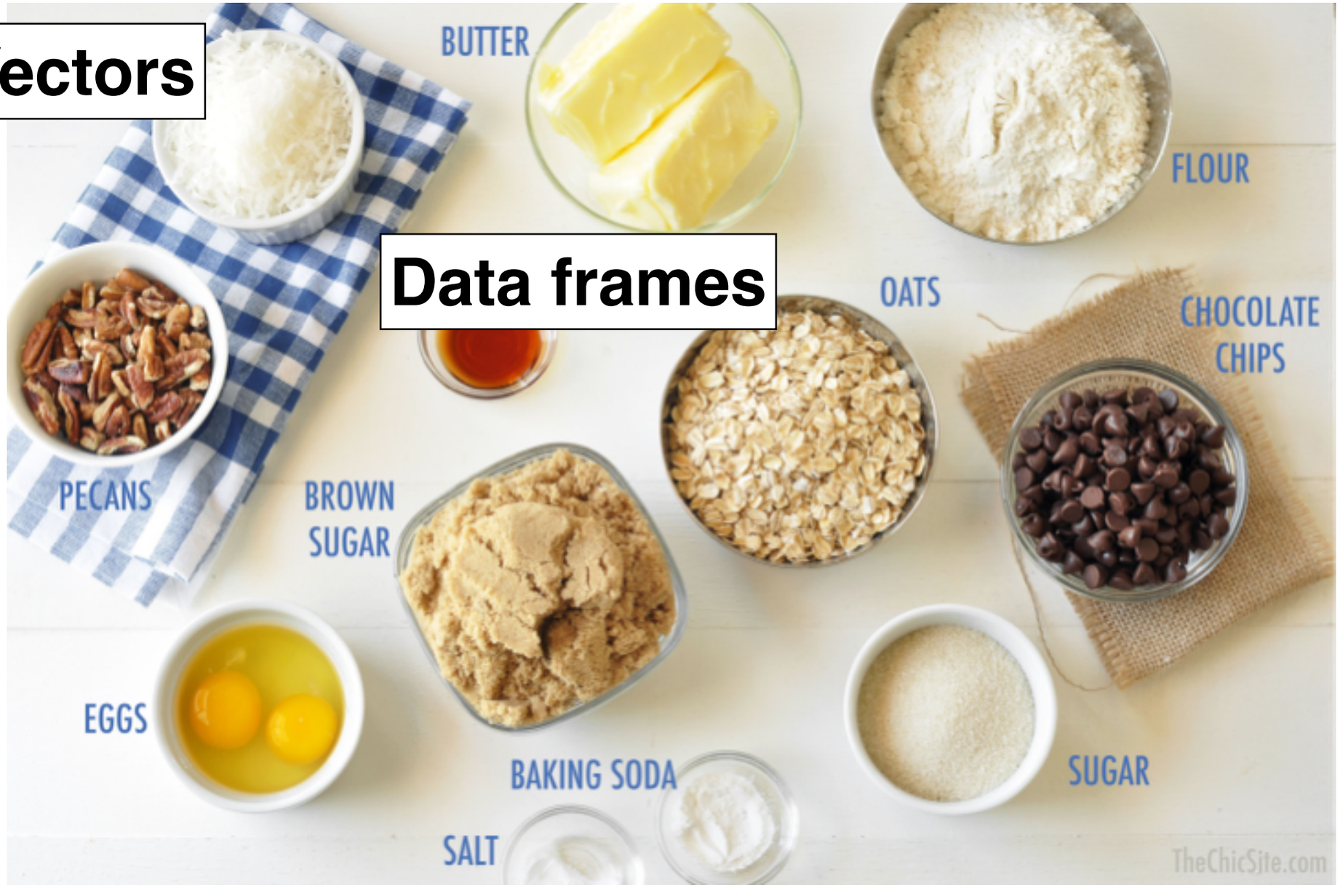
How to run code:

1. Move cursor to a linecoding block
2. Highlight line(s) of code
3. Type, ctrl+enter (windows) or cmd+enter (mac)
4. See code running in console
5. View output/figures

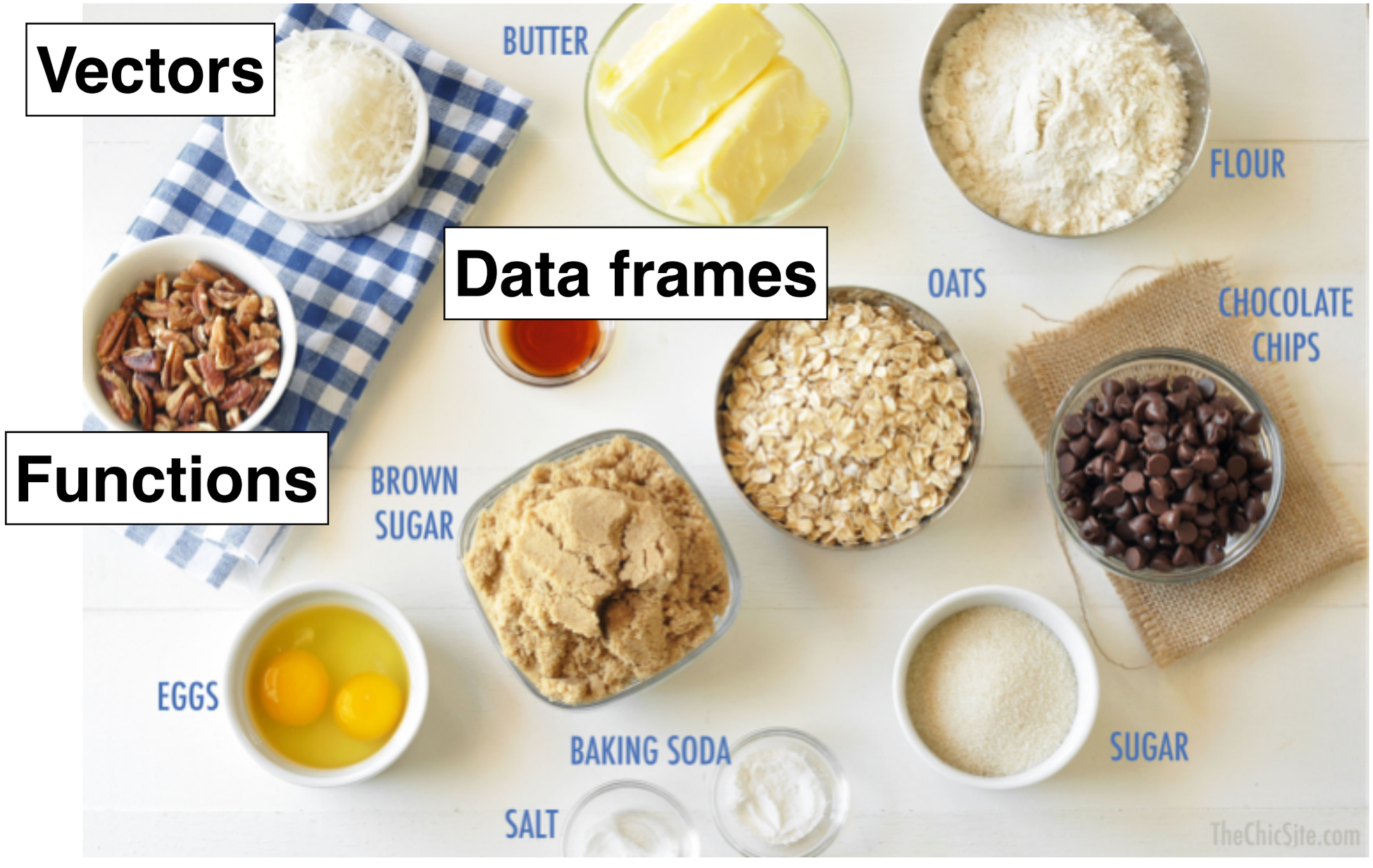
Now you can run code in R, so just need ingredients for your recipe

Now you can run code in R, so just need ingredients for your recipe

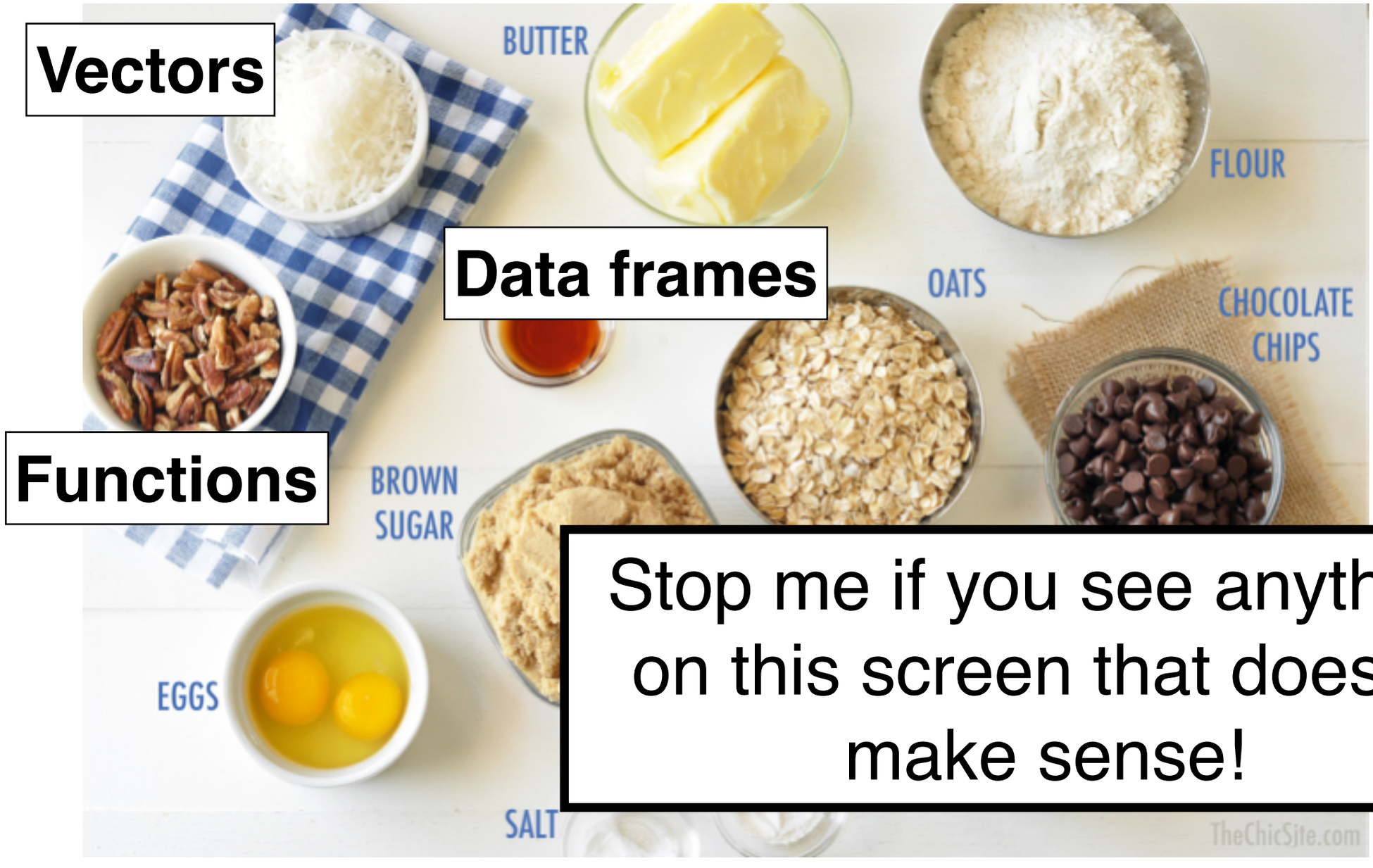
Now you can run code in R, so just need ingredients for your recipe



Now you can run code in R, so just need ingredients for your recipe



Now you can run code in R, so just need ingredients for your recipe



Stop me if you see anything on this screen that doesn't make sense!

R building blocks

At a basic level, R is a calculator

```
> 5+5
[1] 10
> 7*3
[1] 21
```

R building blocks

At a basic level, R is a calculator

```
> 5+5
[1] 10
> 7*3
[1] 21
```

But it's a smart calculator

```
> x = 5
> y = 3
> x*y
[1] 15
> x+y
[1] 8
```

R building blocks

At a basic level, R is a calculator

```
> 5+5
[1] 10
> 7*3
[1] 21
```

But it's a smart calculator

```
> x = 5
> y = 3
> x*y
[1] 15
> x+y
[1] 8
```

```
> x <- 5
> y <- 3
> x*y
[1] 15
> x+y
[1] 8
```

R building blocks

At a basic level, R is a calculator

```
> 5+5
[1] 10
> 7*3
[1] 21
```

No difference!

But it's a smart calculator

```
> x = 5
> y = 3
> x*y
[1] 15
> x+y
[1] 8
```

```
> x <- 5
> y <- 3
> x*y
[1] 15
> x+y
[1] 8
```

R building blocks

A very smart calculator

```
> x = 1:10
> x
 [1]  1  2  3  4  5  6  7  8  9 10
> z = x*3
> z
```

R building blocks

A very smart calculator

```
> x = 1:10
> x
 [1]  1  2  3  4  5  6  7  8  9 10
> z = x*3
> z
 [1]  3  6  9 12 15 18 21 24 27 30
```

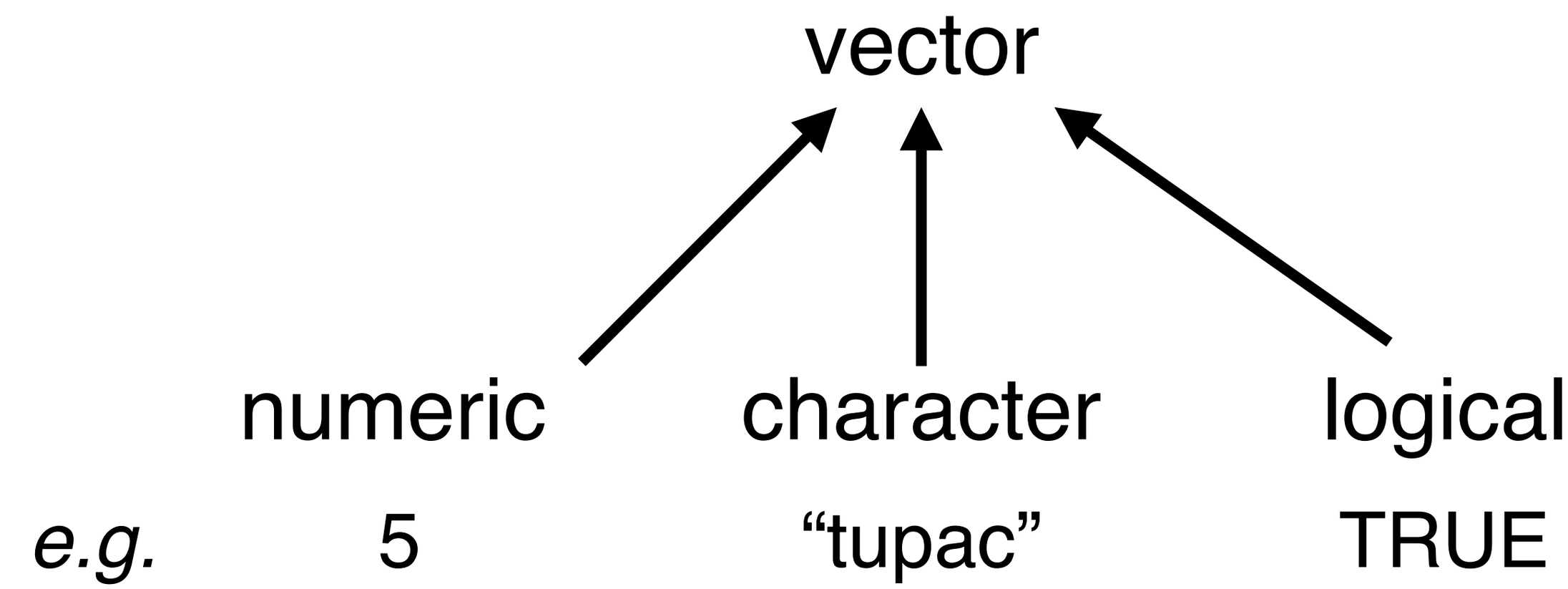
R data structure flowchart

numeric character logical

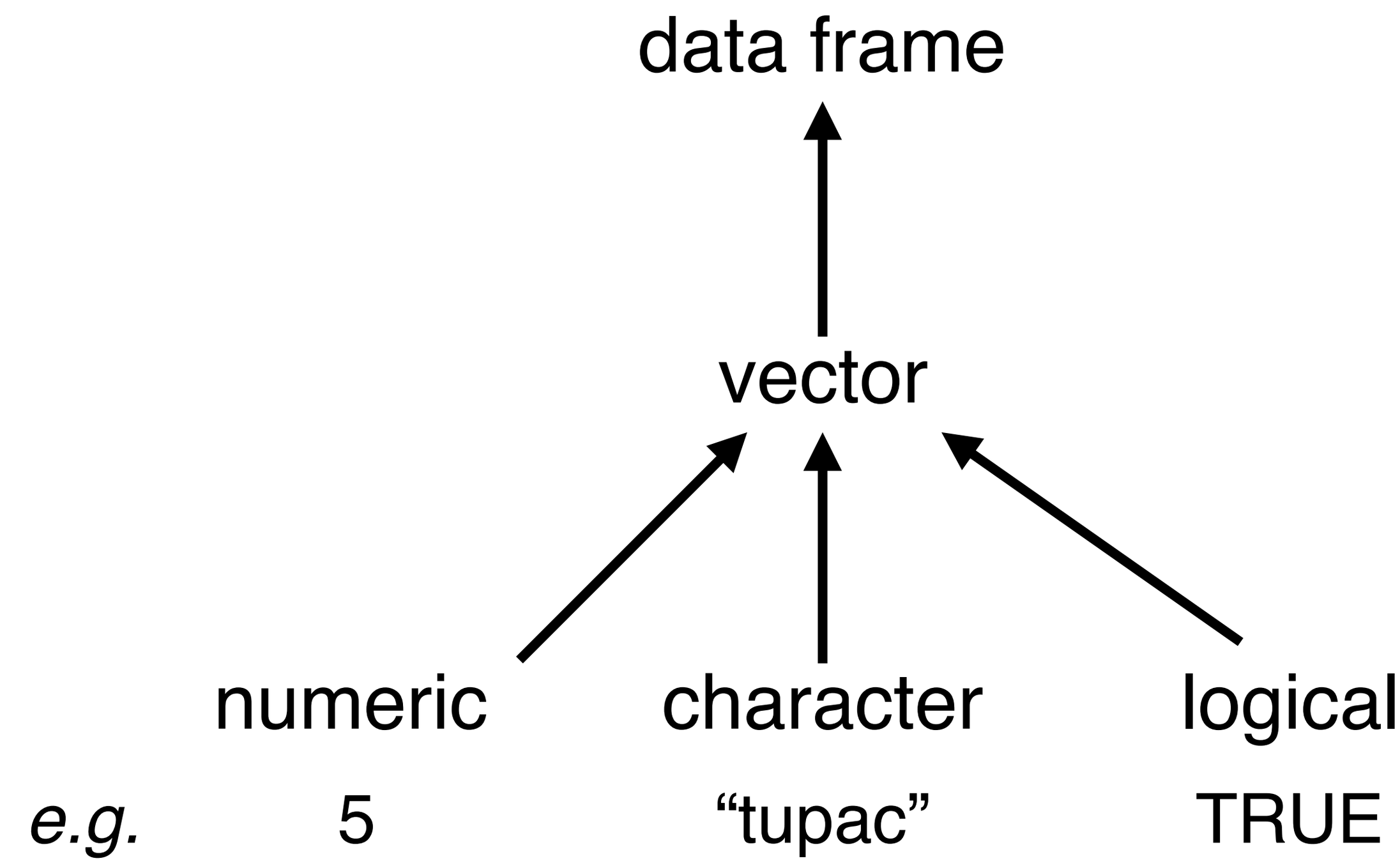
R data structure flowchart

	numeric	character	logical
e.g.	5	“tupac”	TRUE

R data structure flowchart



R data structure flowchart



R data structure flowchart

data frame

vector

numeric character logical

e.g. 5 “tupac” TRUE

R data structures

```
> vec <- c(5, 3, 2, 3, 5, 6, 7, 8)
> vec
[1] 5 3 2 3 5 6 7 8
```

R data structures

```
> vec <- c(5, 3, 2, 3, 5, 6, 7, 8)
> vec
[1] 5 3 2 3 5 6 7 8
> df <- data.frame(col1=vec, col2=vec*2)
> df
  col1 col2
1    5   10
2    3    6
3    2    4
4    3    6
5    5   10
6    6   12
7    7   14
8    8   16
```

R data structures

```
> vec <- c(5, 3, 2, 3, 5, 6, 7, 8)
> vec
[1] 5 3 2 3 5 6 7 8
> vec[3]
[1] 2
```

R data structures

```
> vec <- c(5, 3, 2, 3, 5, 6, 7, 8)
> vec
[1] 5 3 2 3 5 6 7 8
> vec[3]
[1] 2
> vec > 4
[1]  TRUE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
```

R data structures

```
> vec <- c(5, 3, 2, 3, 5, 6, 7, 8)
> vec
[1] 5 3 2 3 5 6 7 8
> vec[3]
[1] 2
> vec > 4
[1]  TRUE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
> vec[vec > 4]
[1] 5 5 6 7 8
```

Using functions

```
> vec <- c(5, 3, 2, 3, 5, 6, 7, 8)
> vec
[1] 5 3 2 3 5 6 7 8
```

```
> df <- data.frame(col1=vec, col2=vec*2)
> df
  col1 col2
1    5   10
2    3    6
3    2    4
4    3    6
5    5   10
6    6   12
7    7   14
8    8   16
```

Using functions

```
> vec <- c(5, 3, 2, 3, 5, 6, 7, 8)
> vec
[1] 5 3 2 3 5 6 7 8
```

```
> mean(vec)
[1] 4.875
> mean(df$col1)
[1] 4.875
```

```
> df <- data.frame(col1=vec, col2=vec*2)
> df
  col1 col2
1    5   10
2    3    6
3    2    4
4    3    6
5    5   10
6    6   12
7    7   14
8    8   16
```

Using functions

```
> vec <- c(5, 3, 2, 3, 5, 6, 7, 8)
> vec
[1] 5 3 2 3 5 6 7 8
```

```
> mean(vec)
[1] 4.875
> mean(df$col1)
[1] 4.875
```

```
> mean(vec[vec>5])
[1] 7
```

```
> df <- data.frame(col1=vec, col2=vec*2)
> df
  col1 col2
1    5   10
2    3    6
3    2    4
4    3    6
5    5   10
6    6   12
7    7   14
8    8   16
```

Using functions

```
> vec <- c(5, 3, 2, 3, 5, 6, 7, 8)
> vec
[1] 5 3 2 3 5 6 7 8
```

```
> mean(vec)
[1] 4.875
> mean(df$col1)
[1] 4.875
```

```
> mean(vec[vec>5])
[1] 7
```

```
> summary(vec)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  2.000   3.000   5.000   4.875   6.250   8.000
```

```
> df <- data.frame(col1=vec, col2=vec*2)
> df
  col1 col2
1    5   10
2    3    6
3    2    4
4    3    6
5    5   10
6    6   12
7    7   14
8    8   16
```

Using functions

```
> vec <- c(5, 3, 2, 3, 5, 6, 7, 8)
> vec
[1] 5 3 2 3 5 6 7 8
```

```
> mean(vec)
[1] 4.875
> mean(df$col1)
[1] 4.875
> mean(vec[vec>5])
[1] 7
> summary(vec)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  2.000   3.000   5.000   4.875   6.250   8.000
```

```
> df <- data.frame(col1=vec, col2=vec*2)
> df
  col1 col2
1    5   10
2    3    6
3    2    4
4    3    6
5    5   10
6    6   12
7    7   14
8    8   16
```

There are a million useful functions, Google is your friend: “How to find number of rows in data frame in R?”

Using functions

```
> vec <- c(5, 3, 2, 3, 5, 6, 7, 8)
> vec
[1] 5 3 2 3 5 6 7 8
```

```
> mean(vec)
[1] 4.875
> mean(df$col1)
[1] 4.875
```

```
> mean(vec[vec>5])
[1] 7
```

```
> summary(vec)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  2.000   3.000   5.000   4.875   6.250   8.000
```

```
> df <- data.frame(col1=vec, col2=vec*2)
> df
  col1 col2
1    5   10
2    3    6
3    2    4
4    3    6
5    5   10
6    6   12
7    7   14
8    8   16
```

There are a million useful functions, Google is your friend: “How to find number of rows in data frame in R?”

```
nrow()
```

Getting help with functions

Input and output

```
> ?sum
```

Getting help with functions

Input and output

```
> ?sum
```

sum {base}

Sum of Vector Elements

Description

`sum` returns the sum of all the values present in its arguments.

Usage

```
sum(..., na.rm = FALSE)
```

Arguments

`...`	numeric or complex or logical vectors.
`na.rm`	logical. Should missing values (including `NaN`) be removed?

2nd Programming Exercise

The excel “black box” strikes

Mistaken Identifiers: Gene name errors can be introduced inadvertently when using Excel in bioinformatics

Barry R Zeeberg[†], Joseph Riss[†], David W Kane, Kimberly J Bussey, Edward Uchio, W Marston Linehan, J Carl Barrett and John N Weinstein

[†] Contributed equally

BMC Bioinformatics 2004 **5**:80 | **DOI:** 10.1186/1471-2105-5-80

Received: 05 March 2004 | **Accepted:** 23 June 2004 | **Published:** 23 June 2004

The excel “black box” strikes

Gene name errors are widespread in the scientific literature

Mark Ziemann, Yotam Eren and Assam El-Osta

Genome Biology 2016 **17**:177 | **DOI:** 10.1186/s13059-016-1044-7 | © The Author(s). 2016

Published: 23 August 2016

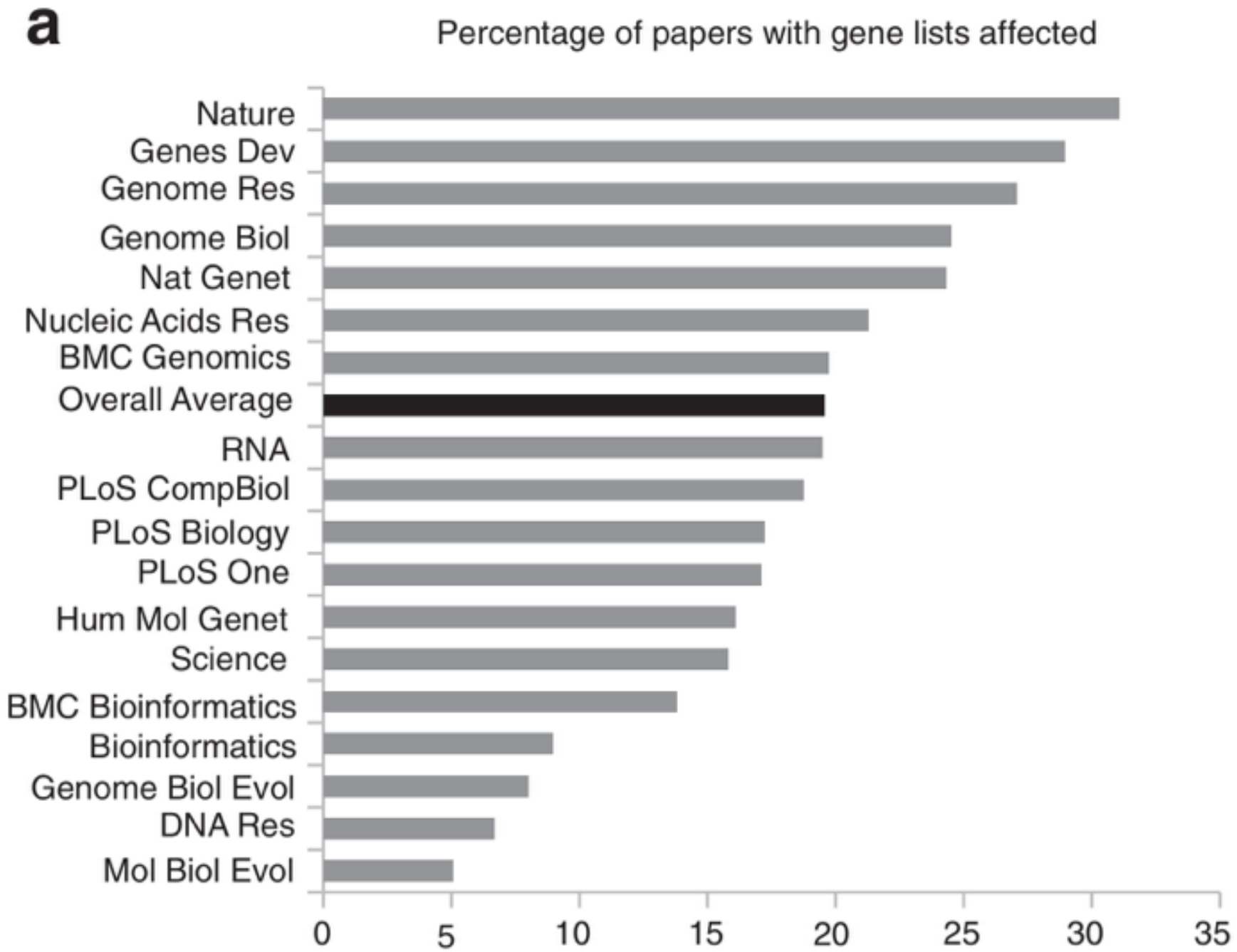
The excel “black box” strikes

Gene name errors are widespread in the scientific literature

Mark Ziemann, Yotam Eren and Assam El-Osta

Genome Bic

Published: 2



R has it's own problems

```
> bad_read <- read.csv("files/1_intro_r/ex1_calories.csv")
```

R has it's own problems

```
> bad_read <- read.csv("files/1_intro_r/ex1_calories.csv")
> bad_read
         date calories
1  2015-12-08     3500
2  2015-12-09     2150
3  2015-12-10     2250
4  2015-12-11     1900
5  2015-12-12     2500
6  2015-12-13        X
7  2015-12-14     2000
8  2015-12-15     1850
9  2015-12-16     1900
10 2015-12-17     1800
```

R has it's own problems

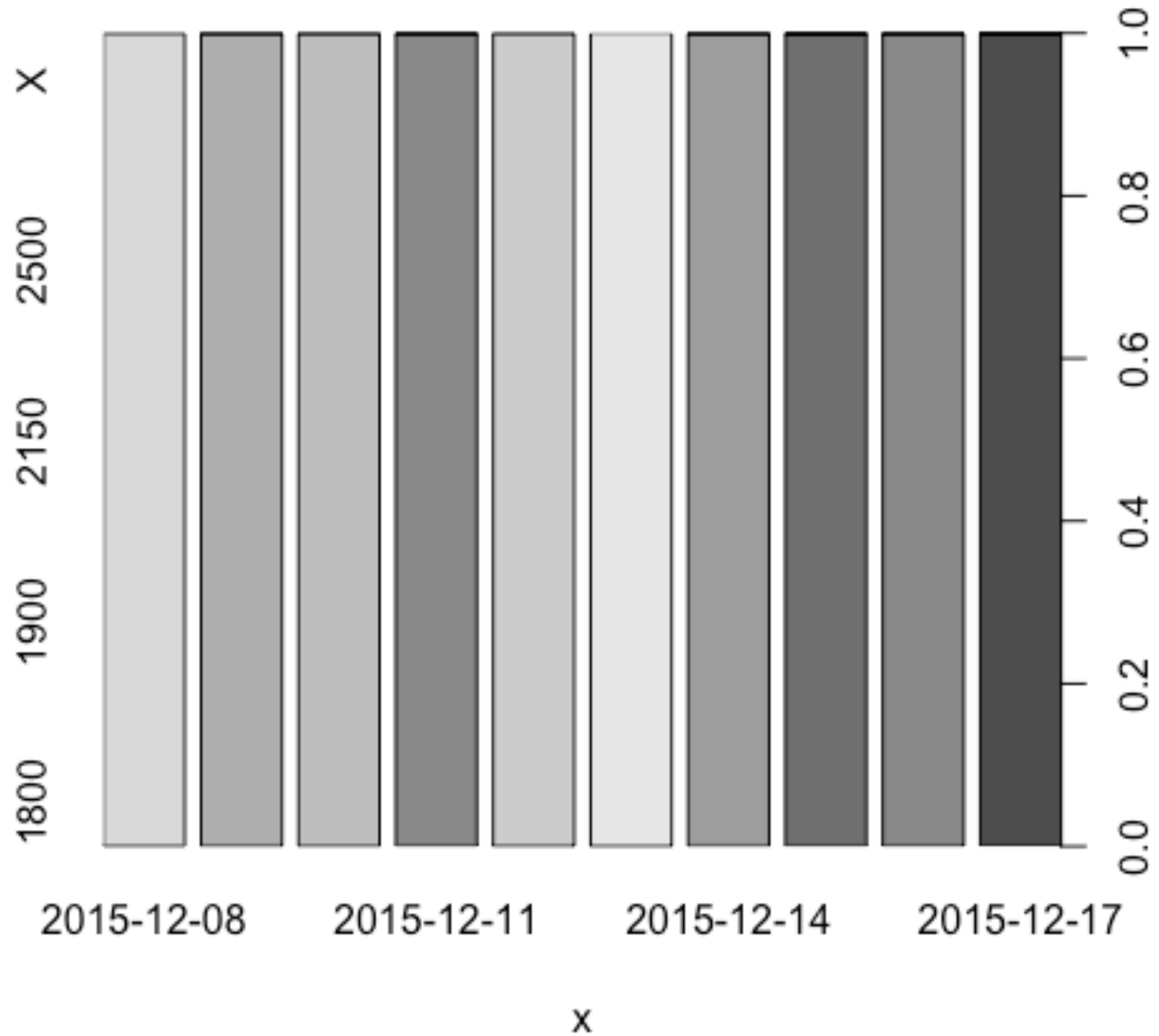
```
> bad_read <- read.csv("files/1_intro_r/ex1_calories.csv")
> bad_read
         date calories
1  2015-12-08     3500
2  2015-12-09     2150
3  2015-12-10     2250
4  2015-12-11     1900
5  2015-12-12     2500
6  2015-12-13        X
7  2015-12-14     2000
8  2015-12-15     1850
9  2015-12-16     1900
10 2015-12-17     1800
```

```
> plot(bad_read$date, bad_read$calories)
```

R has it's own problems

```
> bad_read <- read.csv("files/1_intro_r/ex1_calories.csv")
> bad_read
         date calories
1  2015-12-08     3500
2  2015-12-09     2150
3  2015-12-10     2250
4  2015-12-11     1900
5  2015-12-12     2500
6  2015-12-13        X
7  2015-12-14     2000
8  2015-12-15     1850
9  2015-12-16     1900
10 2015-12-17     1800
```

```
> plot(bad_read$date, bad_read$calories)
```



R has it’s own problems

R has it's own problems

```
> bad_read$calories
 [1] 3500 2150 2250 1900 2500 X    2000 1850 1900 1800
Levels: 1800 1850 1900 2000 2150 2250 2500 3500 X
```

R has it's own problems

```
> bad_read$calories
 [1] 3500 2150 2250 1900 2500 X    2000 1850 1900 1800
Levels: 1800 1850 1900 2000 2150 2250 2500 3500 X

> class(bad_read$calories)
[1] "factor"
```

R has it's own problems

```
> bad_read$calories
 [1] 3500 2150 2250 1900 2500 X    2000 1850 1900 1800
Levels: 1800 1850 1900 2000 2150 2250 2500 3500 X

> class(bad_read$calories)
[1] "factor"
```

```
> as.numeric(bad_read$calories)
 [1] 8 5 6 3 7 9 4 2 3 1
```

R has it's own problems

```
> bad_read$calories
 [1] 3500 2150 2250 1900 2500 X    2000 1850 1900 1800
Levels: 1800 1850 1900 2000 2150 2250 2500 3500 X

> class(bad_read$calories)
[1] "factor"
```

```
> as.numeric(bad_read$calories)
 [1] 8 5 6 3 7 9 4 2 3 1
```

```
read.csv("files/1_intro_r/ex1_calories.csv", stringsAsFactors = FALSE)
```

Hadley to the rescue

Hadley to the rescue

Hadley to the rescue

Hadley Wickham
@hadleywickham

Following

How do I hate thee stringsAsFactors = TRUE?
Let me count the ways #rstats

RETWEETS 50 LIKES 97

5:43 PM - 23 Jul 2015

The readr package

```
> library(readr)
```

The readr package

```
> library(readr)
> good_read <- read_csv("files/1_intro_r/ex1_calories.csv")
Parsed with column specification:
cols(
  date = col_date(format = ""),
  calories = col_character()
)
```

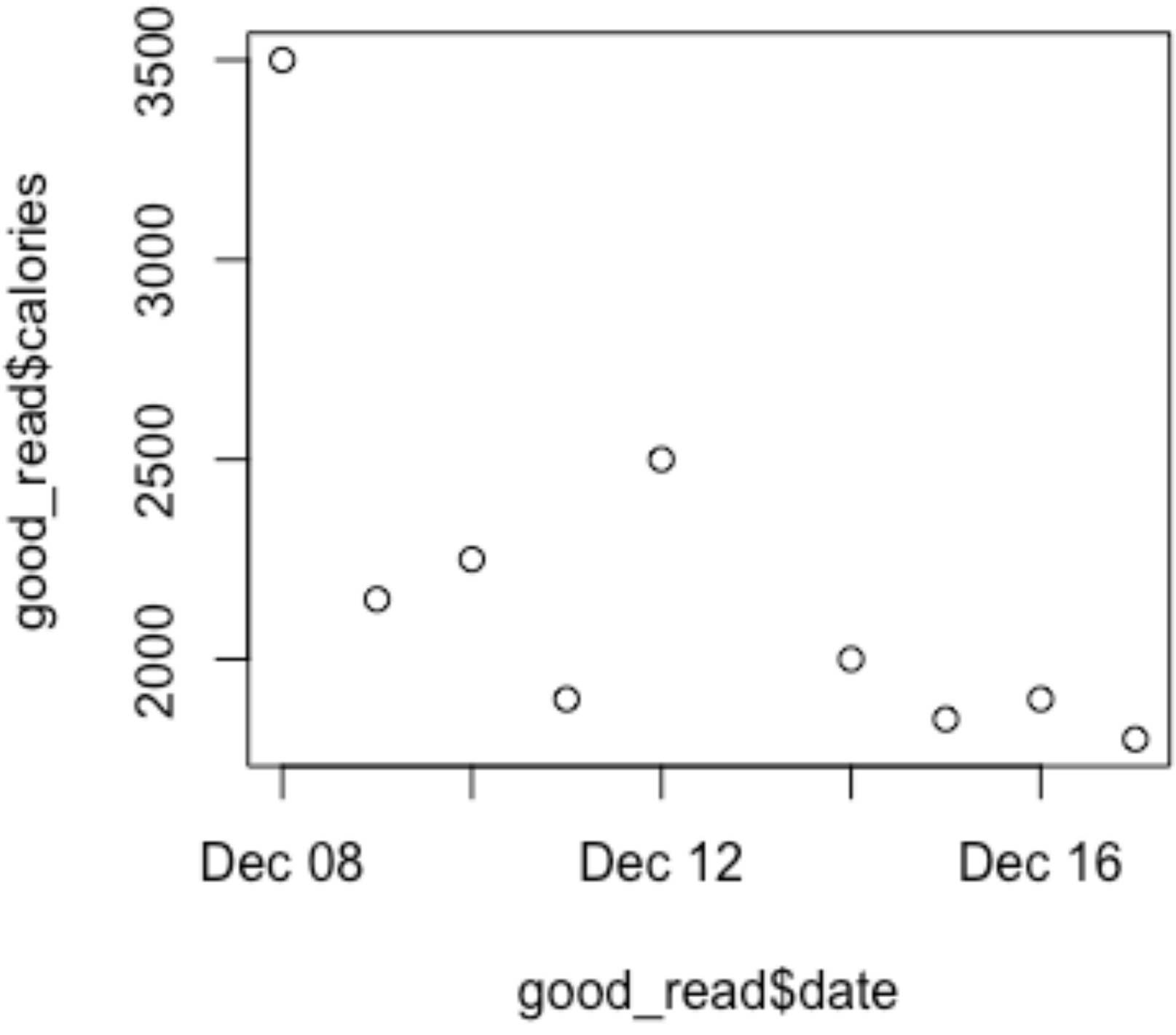
The readr package

```
> library(readr)
> good_read <- read_csv("files/1_intro_r/ex1_calories.csv")
Parsed with column specification:
cols(
  date = col_date(format = ""),
  calories = col_character()
)
```

```
> good_read
# A tibble: 10 x 2
         date calories
       <date>    <chr>
1  2015-12-08     3500
2  2015-12-09     2150
3  2015-12-10     2250
4  2015-12-11     1900
5  2015-12-12     2500
6  2015-12-13        X
7  2015-12-14     2000
8  2015-12-15     1850
9  2015-12-16     1900
10 2015-12-17     1800
```

The readr package

```
> plot(good_read$date, good_read$calories)
```



3rd Programming Exercise

Tips for getting the file path of your data:

1. type: > getwd(), which gives you your current working directory; my output looks like this:
   [1] “/Users/spencerfox/projects/rstats_fall2016”

2. type read.csv(“”), and then start typing the beginning of your wd path within the quotation marks, and hit tab to autocomplete folder names

3. Navigate folders to your data, and then select your .csv when you see it