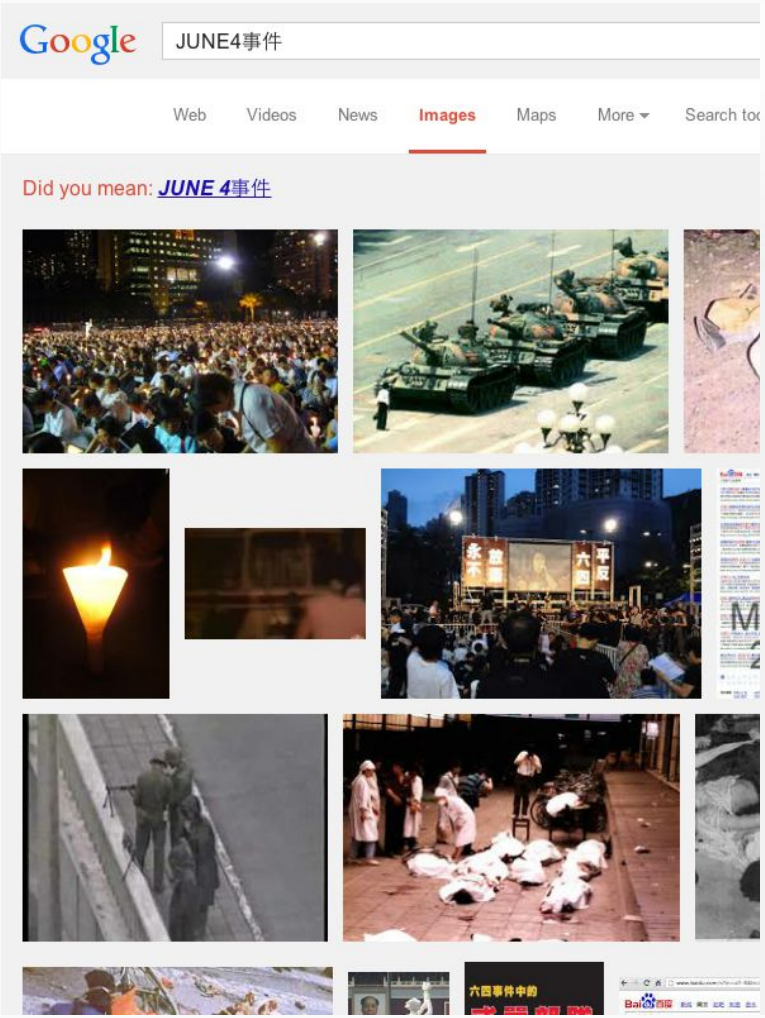
Getting Started with Statistical Computing

Prof. Greg Distelhorst
R Statistical Computing Short Course for Political Scientists

Nondemocracies usually control the media

Soviet Newspaper

Nondemocracies usually control the media

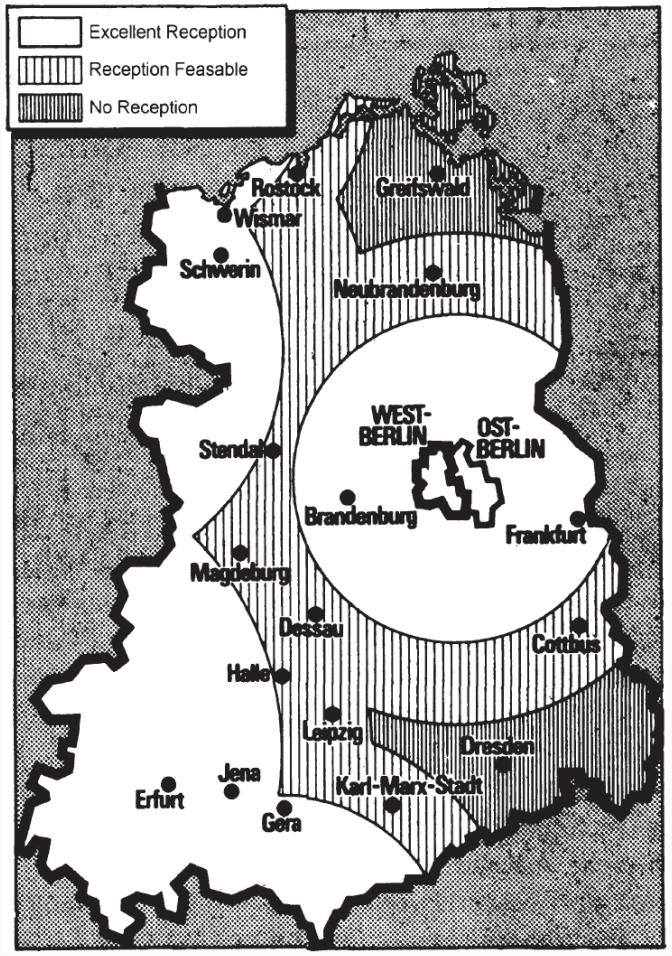


Searching for the date of the Tiananmen massacre on Google and Baidu

Nondemocracies usually control the media

Would access to **more free, foreign media**
reduce support for nondemocracy?

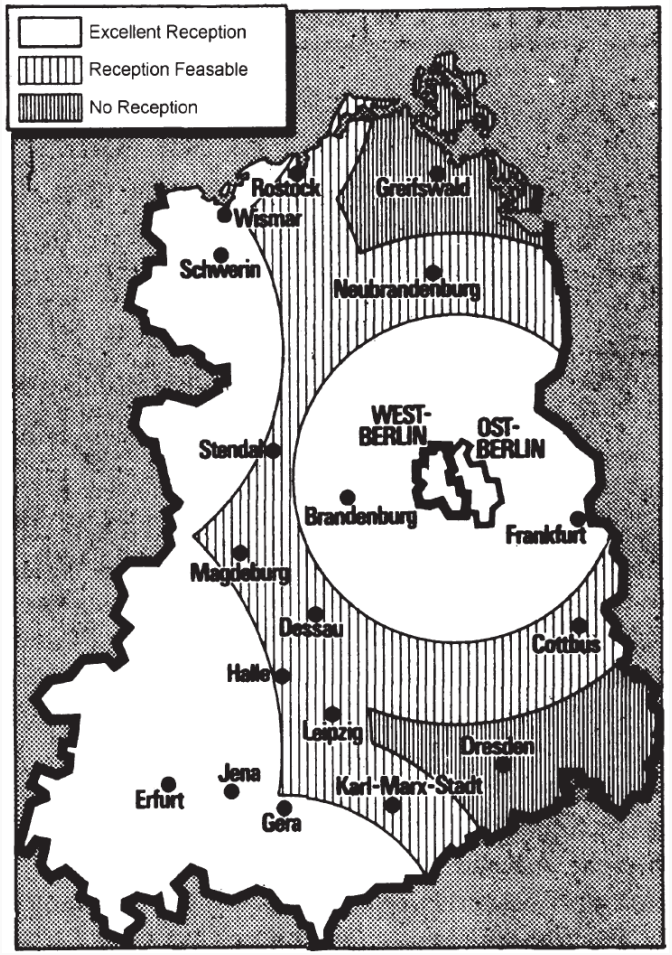
Does free media reduce support for autocracy?



- Throughout the Cold War, West Germany broadcast TV into East Germany.
- ...but only some cities got reception
- Kern and Hainmueller (2009) analyze archival surveys and exit visa applications from East Germany published after the fall of communism.
- Use city-level variation in exposure to help identify the causal effect

Expectations?

Does free media reduce support for autocracy?



No. It actually increased support.

Overall, we find that exposure to West German television had a substantial positive effect on regime support. Evaluated at the means of the three outcome variables, **the average increase in regime support due to West German television exposure was between 11% and 15%.** (p389)

Why?

Deterrence? Exposure to bad parts of western life.

Escapism? Markets create better entertainment.

What goes into a persuasive political science study?

- Theory (current state of the field in comparative politics)
- History
- Geography
- Language skills
- Data collection plan (here: collection/merging of archival data)
- Theoretical knowledge of statistics and causal inference
- **Practical coding skills to execute and report analyses**
- Clear writing
- A resilent spirit in the face of negative feedback

This course covers one of these skills: **statistical computing in R.**

Course summary

End of week goals

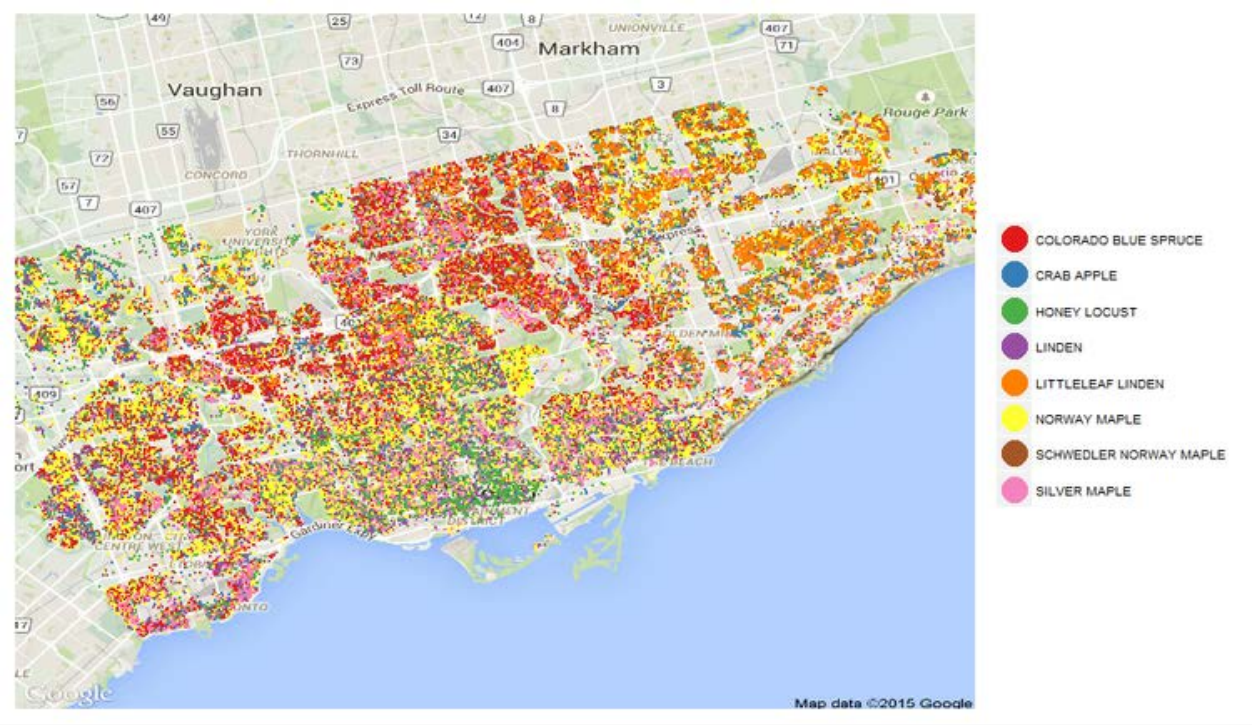
- Read, clean, merge, and visualize data of various types and arbitrary size.
- Foundation allows you to teach yourself new skills in R.

Mechanics

- Four sessions this week
- In-class exercises
- Slides posted online after class

Why do statistical computing in R?

- Free
- Versatile: basic stats to machine learning
- RStudio (also free) is a decent development environment
- R's capabilities grow through its user community



Before we start

A quick unscientific survey

Gameplan

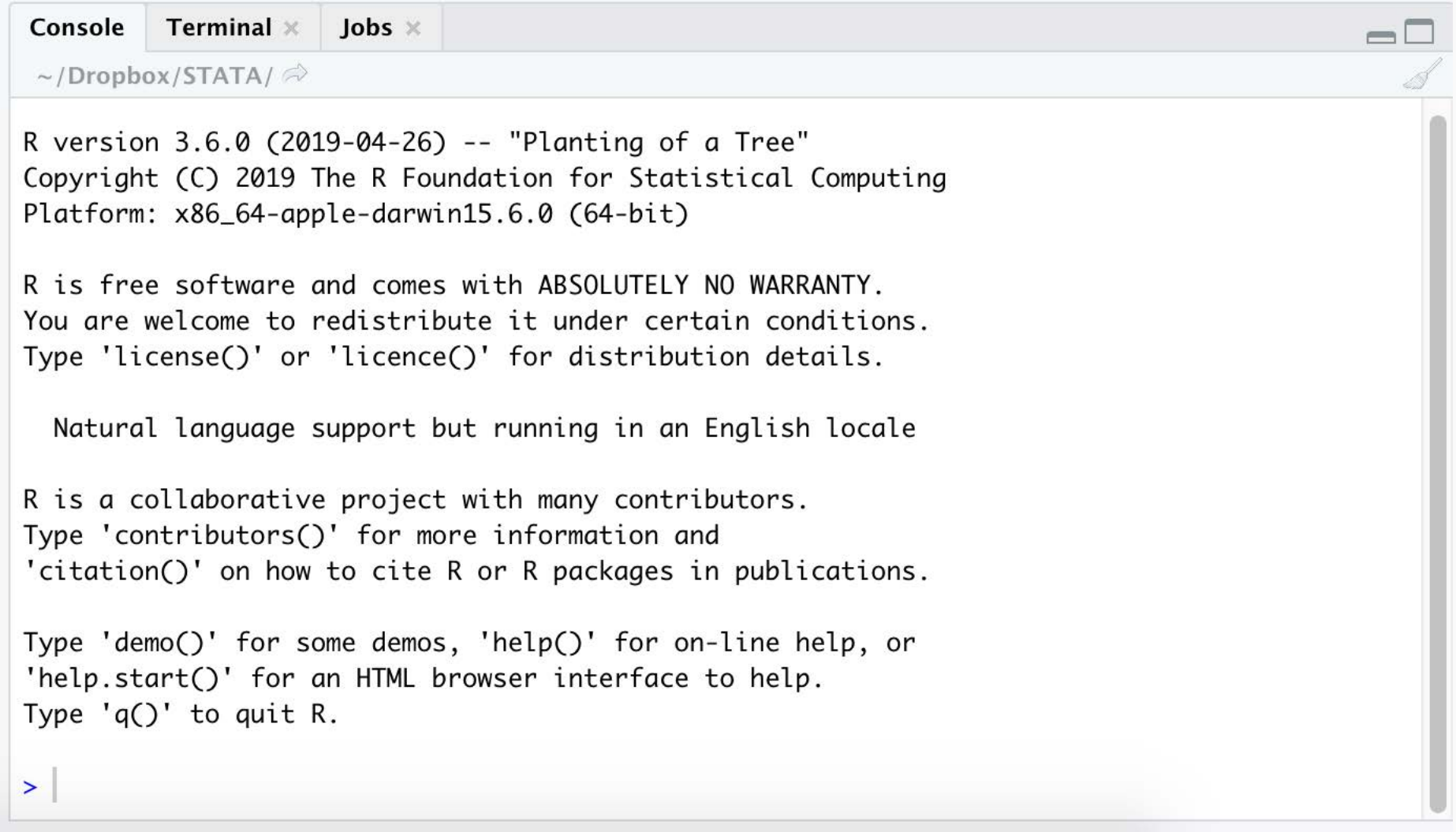
- Basic math in the R Console
- Working in R's memory
  - Exercises: Luigi and ScamBank
- First taste of data: Vectors
- Logicals: `TRUE` and `FALSE`
  - Exercises: Maple Leafs' performance
- Summary
- Extending R's functionality with packages

- Scripting and the `projects/` folder

- Installing the Tidyverse

Basic Math in the R Console

The Console


Console
Terminal
Jobs
~/Dropbox/STATA/
R version 3.6.0 (2019-04-26) -- "Planting of a Tree"
Copyright (C) 2019 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin15.6.0 (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
Natural language support but running in an English locale
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.
>

The Console

```
> [type a command here and hit 'enter']
```

Examples

```
2 * 2
```

```
[1] 4
```

```
42 + 77
```

```
[1] 119
```

```
7^2
```

```
[1] 49
```

Parentheses control evaluation

```
4 * 6 - 3
```

```
[1] 21
```

```
4 * (6 - 3)
```

```
[1] 12
```

Talking about R

Operators

- Symbols like `+ - * / ^` are **operators**
- What R gives back is the **result**, **return value**, or **output**. `99 / 11` returns 9.

Functions

- In addition to operators, R also has **functions**, for example:
  - Square root: $\sqrt{9} \rightarrow$ `sqrt(9)`, returns 3
  - Log base 10: $\log_{10} 100 \rightarrow$ `log10(100)`, returns 2
  - Natural log: $\ln 40 \rightarrow$ `log(40)`, returns 3.688879
- We pass **arguments** to functions in parentheses.
  - Above, 9 is an argument passed to the function `sqrt()`
- Use `?` to get help for functions: `?log`

Saving data in memory

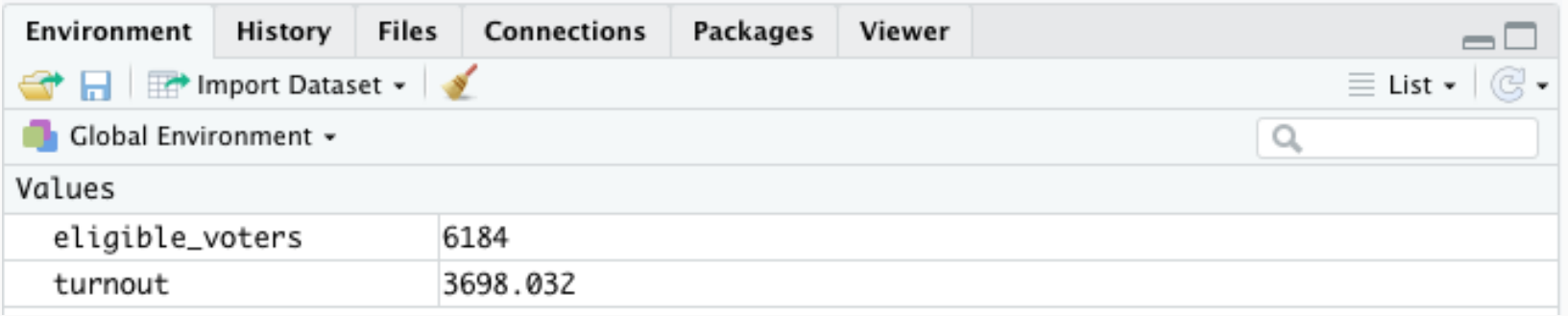
In an electoral district with 9,432 individuals, 3,248 were not eligible to vote. Turnout among the eligible was 59.8%. The Party of Luigi was victorious, receiving 52.3% of the votes cast. How many votes were cast for the Party of Luigi?

```
eligible_voters = 9432 - 3248
```

Saving data in memory

In an electoral district with 9,432 individuals, 3,248 were not eligible to vote. Turnout among the eligible was 59.8%. The Party of Luigi was victorious, receiving 52.3% of the votes cast. How many votes were cast for the Party of Luigi?

```
eligible_voters = 9432 - 3248
turnout = eligible_voters * .598
```



Saving data in memory

In an electoral district with 9,432 individuals, 3,248 were not eligible to vote. Turnout among the eligible was 59.8%. The Party of Luigi was victorious, receiving 52.3% of the votes cast. How many votes were cast for the Party of Luigi?

```
eligible_voters = 9432 - 3248
turnout = eligible_voters * .598
turnout = round(eligible_voters * .598)
```

Saving data in memory

In an electoral district with 9,432 individuals, 3,248 were not eligible to vote. Turnout among the eligible was 59.8%. The Party of Luigi was victorious, receiving 52.3% of the votes cast. How many votes were cast for the Party of Luigi?

```
eligible_voters = 9432 - 3248
turnout = eligible_voters * .598
turnout = round(eligible_voters * .598)
turnout * .523
```

```
[1] 1934.054
```

```
round(turnout * .523)
```

```
[1] 1934
```

Exercises: Basic Math

In an electoral district with 9,432 individuals, 3,248 were not eligible to vote. Turnout among the eligible was 59.8%. The Party of Luigi was victorious, receiving 52.3% of the votes cast. How many votes were cast for the Party of Luigi?

1. Compute the votes cast for the Party of Luigi **using one line** of R code, using parentheses to control the order of evaluation. (No use of R's memory.)

2. *ScamBank* offers a savings account with a guaranteed 11.5% in annual interest. If you deposited the base funding level of $17,000 for UofT political science doctoral students in the account on January 1, 2020 (and made no deposits or withdrawals) how much would you have on January 1, 2024?

Solutions

1. Compute the votes cast for the Party of Luigi **using one line** of R code, using parentheses to control the order of evaluation. (No use of R's memory.)

```
round(round((9432 - 3248) * .598) * .523)
```

```
[1] 1934
```

2. *ScamBank* offers a savings account with a guaranteed 11.5% in annual interest. If you deposited $17,000 on January 1, 2020 (and made no deposits or withdrawals) how much would you have on January 1, 2024?

```
# Four full years at 11.5% per year
((((17000 * 1.115) * 1.115) * 1.115) * 1.115)
```

```
[1] 26275.34
```

```
17000 * 1.115^4
```

```
[1] 26275.34
```

Two ways to save variables to memory

R has two ways to assign variables: `=` and `<-`. The following expressions are equivalent:

```
turnout_rate = .598
```

```
turnout_rate <- .598
```

Your instructor prefers `=` because it is easier to type and we can use it both to save variable to memory and to set argument values when we call functions. However, this is a matter of taste.

RStudio's keyboard shortcut for `<-` is *alt* -.

Clearing R's memory

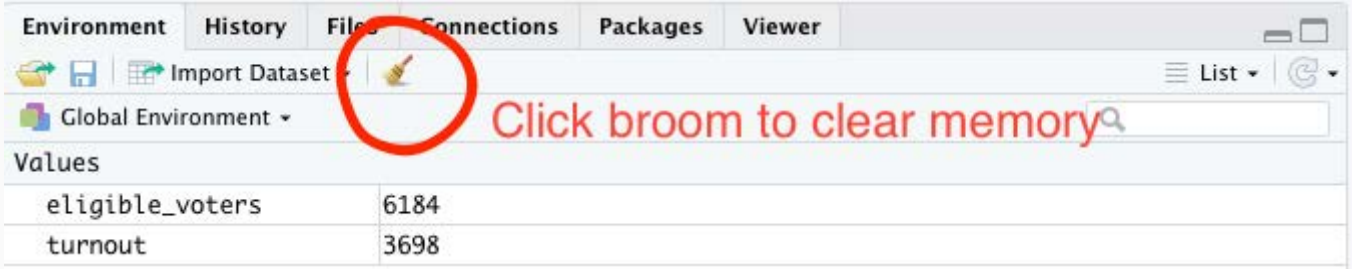
Above we saved `eligible_voters` and `turnout` to memory. We can delete them from memory using the `rm()` function.

```
rm(eligible_voters, turnout)
```

The following command clears R's memory:

```
rm(list=ls())
```

...or you can click the little broom icon in the **Environment** tab

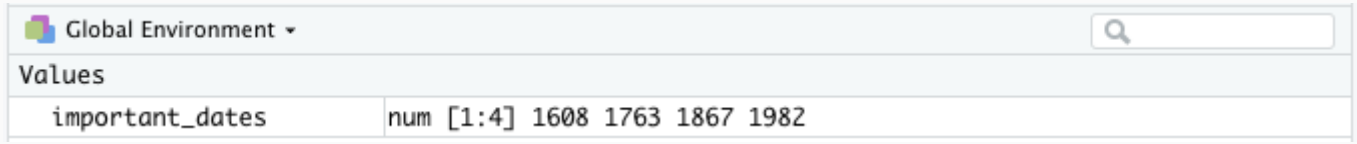


Vectors: A First Taste of Data

Introduction to Vectors

- **Vectors** are sequences of numbers (or other things)
- Use the function `c()` to create vectors

```
important_dates = c(1608, 1763, 1867, 1982)
```



- Many R functions and operators also work on vectors

```
2019 - important_dates
```

```
[1] 411 256 152  37
```

Introduction to Vectors

A regional parliament has 32 legislators. There are four districts in the region with populations of 6,182, 12,024, 4,508, and 2,390. Each district needs to be assigned MPs proportional to its population.

How many MPs should represent each of the four localities?

```
local_pops = c(6182, 12024, 4508, 2390)
```

`sum()` adds all values in a vector. To compute the total population:

```
total_pop = sum(local_pops)
total_pop
```

```
[1] 25104
```

The population shares of each district:

```
pop_shares = local_pops / total_pop
pop_shares
```

```
[1] 0.24625558 0.47896750 0.17957298 0.09520395
```

Population-based allocation of 32 MPs:

```
round(32 * pop_shares)
```

```
[1]  8 15  6  3
```

Vector Basics

Most functions and operators work on vectors as expected

```
myvec = c(4, 9, 1, 121)
```

```
2 * myvec
```

```
[1]   8  18   2 242
```

```
myvec - 4
```

```
[1]   0   5  -3 117
```

```
sqrt(myvec)
```

```
[1]  2  3  1 11
```

```
myvec^2
```

```
[1]    16    81     1 14641
```

Vector Basics

`length()` returns the number of elements in a vector:

```
myvec = c(4, 9, 1, 121)
length(myvec)
```

```
[1] 4
```

Combine `sum()` and `length()` to compute the average:

```
sum(myvec) / length(myvec)
```

```
[1] 33.75
```

...or use R's built-in function.

```
mean(myvec)
```

```
[1] 33.75
```

Extracting elements from vectors

Subscripting with `[]`

```
myvec = c(4, 9, 1, 121)
myvec[1]
```

```
[1] 4
```

```
myvec[4]
```

```
[1] 121
```

Can pass a vector of locations too:

```
myvec[c(2, 3)]
```

```
[1] 9 1
```

Sequences of integers

`startnum:endnum` returns a vector containing the sequence of integers between `startnum` and `endnum`.

```
2:4
```

```
[1] 2 3 4
```

```
myvec[2:4]
```

```
[1]   9   1 121
```

Logicals: What's TRUE?

Introduction to Logicals

- R has many data **types** that serve different purposes. So far we've used only **numeric** types.
- Another important data type is a **logical**.
- Logicals can be either `TRUE` or `FALSE` (**capital letters only**)
  - Equivalently: `T` or `F`
- Logicals are often generated by **relational operators**:

`<`, `<=`, `>`, `>=`, `==`, `!=`

Introduction to Logicals

Relational operator examples

```
5 == 3   # Equal to
```

```
[1] FALSE
```

```
5 > 3    # Greater than
```

```
[1] TRUE
```

```
5 < 3    # Less than
```

```
[1] FALSE
```

```
32 >  32 # Greater than
```

```
[1] FALSE
```

```
32 >= 32 # Greater than or equal to
```

```
[1] TRUE
```

Careful: the equality operator `==` is very similar to how we assign values to variables using a single `=`.

The test for inequality `!=` (not equal to) uses an exclamation point or a "bang". `!` also reverses a logical.

```
5 != 3
```

```
[1] TRUE
```

```
!FALSE
```

```
[1] TRUE
```

```
!TRUE
```

```
[1] FALSE
```

Relational Operators and Vectors

We can apply relational operators to entire vectors:

```
highs = c(34, 33, 30, 33, 33, 24, 25) # Toronto high temps, July 1-7 2019
highs >= 30
```

```
[1]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE
```

`table()` counts all the unique values of a vector:

```
table(highs >= 30)
```

```
FALSE  TRUE
    2     5
```

There were 5 days with high temperatures of at least 30 degrees C.

Exercises: Vectors, Relational Operators, and Logicals

Obtain the goal data (two vectors) for the 2018-2019 Toronto Maple Leafs here: http://www.gregdistelhorst.com/bootcamp/

1. Compute the Leafs' **total goal differential** for the season in two ways:
    - First computing total goals for and goals against, then computing the difference.
    - First computing **the difference in each game**, then summing the within-game differences.
2. Using relational operators and `table()`, how many games did the Maple Leafs win?
3. In how many games did the Leafs **score** at least 6 goals?
4. How many games did the Leafs **win by** at least 3 goals?
5. What was the average goals scored against the Leafs for the whole season?
6. Was the Leafs' defensive performance (limiting goals against) better in the first half of the season or the second half of the season?

Solutions

1. Total goal differential in two ways.

```
sum(goals_for) - sum(goals_against)
```

```
[1] 35
```

```
diff_by_game = goals_for - goals_against
diff_by_game[1:10] # First ten elements
```

```
 [1]  1 -2  1  3  2  2  3 -3 -3  2
```

```
sum(diff_by_game)
```

```
[1] 35
```

2. How many games did the Leafs win?

```
table(diff_by_game >= 1)
```

```
FALSE  TRUE
   36    46
```

3. **Scored** 6+ goals?

```
table(goals_for >= 6)
```

```
FALSE  TRUE
   68    14
```

4. **Won by** 3+ goals?

```
table(diff_by_game >= 3)
```

```
FALSE  TRUE
   64    18
```

5. Average goals against (whole season)

```
mean(goals_against)
```

```
[1] 3.060976
```

Solutions

6. Was the Leafs' defensive performance (limiting goals against) better in the first half of the season or the second half of the season?

```
length(goals_against)
```

```
[1] 82
```

There are 82 games in a season, 41 in a half-season.

```
mean(goals_against[1:41])
```

```
[1] 2.756098
```

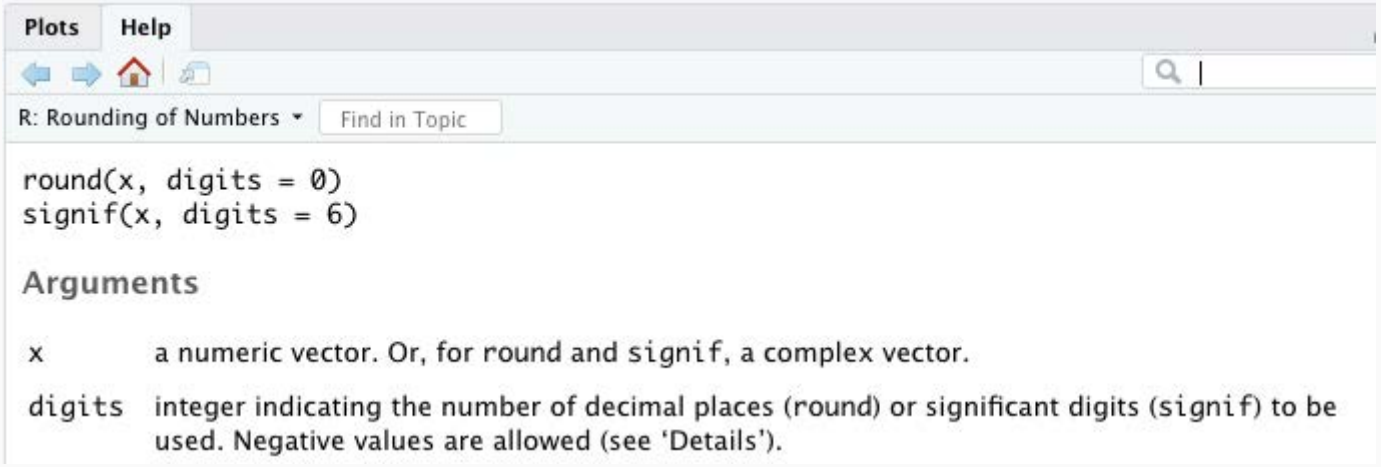
```
mean(goals_against[42:82])
```

```
[1] 3.365854
```

The Leafs' defense allowed roughly 0.6 more goals per game in the second half.

Arguments of functions

Access help for the function `round()` using `?round`



`round()` has two arguments:

- `x` contains the number(s) to be rounded (no default value)
- `digits` specifies the decimal places (default value is `0`)

Arguments of functions

Argument assignment examples for `round(x, digits = 0)`

```
mynum = 7.473
```

```
round(mynum)    # Omitting 'digits' argument uses default value of 0 decimals
```

```
[1] 7
```

```
round(mynum, 1) # Assigning 'digits' manually, using default argument order
```

```
[1] 7.5
```

```
round(x = mynum, digits = 1)   # Naming our arguments
```

```
[1] 7.5
```

```
round(digits = 1, x = mynum)   # Named arguments can appear in any order
```

```
[1] 7.5
```

Careful. Only use `=` inside functions. Do not use `<-`.

Summary

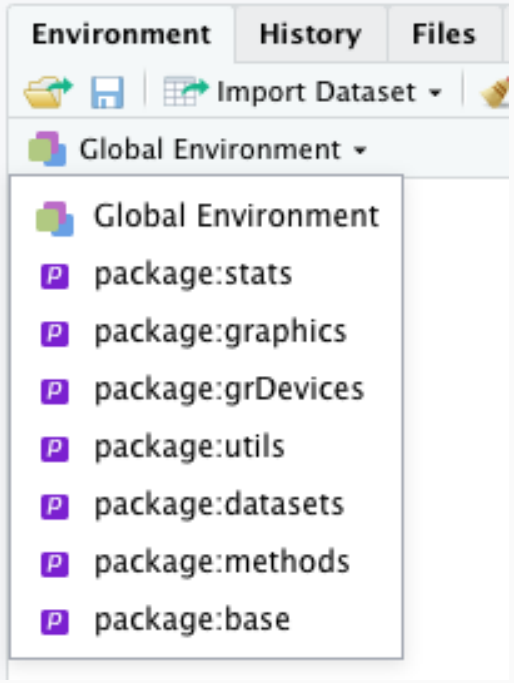
- Basic math operators (`+` `-` `*` `/` `^`) and functions like `sqrt()` or `log10()`
- Storing values to memory using `=` or `<-`
- Creating vectors using `c()`
- Math on vectors, including functions `sum()`, `length()`, and `mean()`
- Relational operators like `<`, `!=` etc. to create logicals (`TRUE` or `FALSE`)
- Summarizing unique values of vectors using `table()`

Extending R's Functions with Packages

Extending functionality with packages

- R's functions are contained in **packages**
- For example, `c()` and `sum()` are in the `base` package
- These expressions are equivalent: `c(4, 1, 2)` and `base::c(4, 1, 2)`

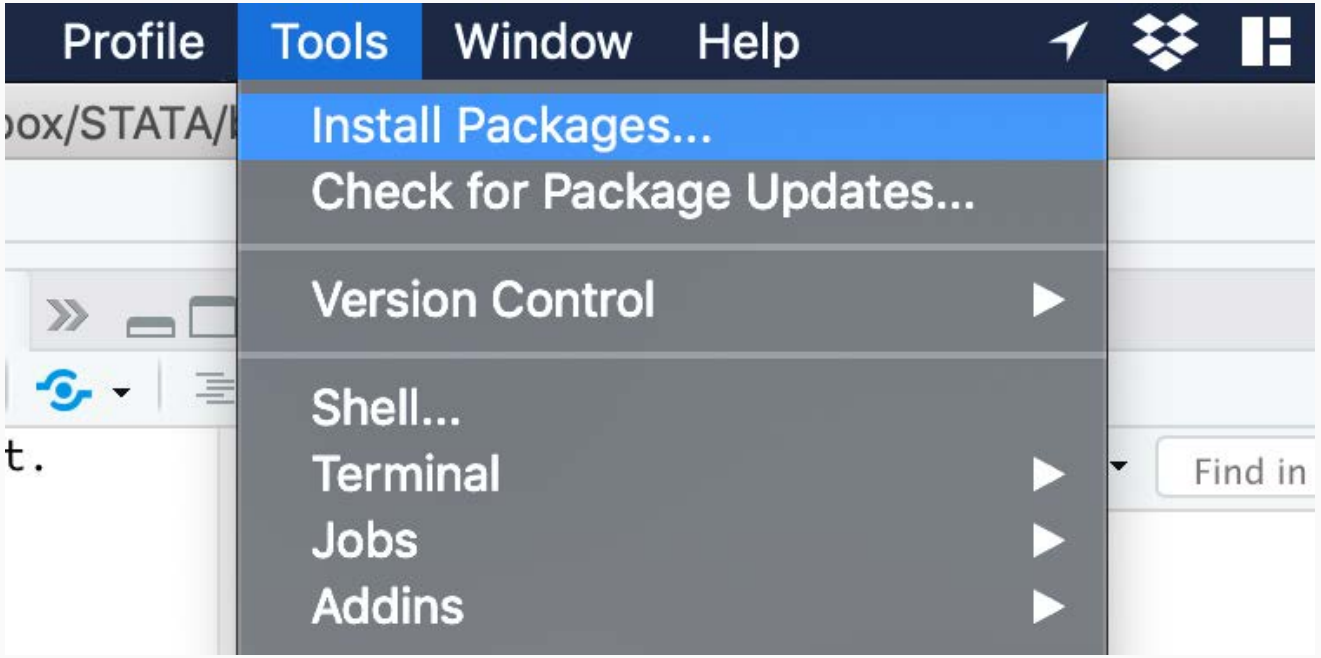
- These packages are automatically loaded in new R sessions $\longrightarrow$
- To add functions to R:
  - Install new package from internet
  - Then load into R's memory using `library()`



Extending functionality with packages

Let's install a new package: `cowsay`

- Option 1: `install.packages('cowsay')`
- Option 2:

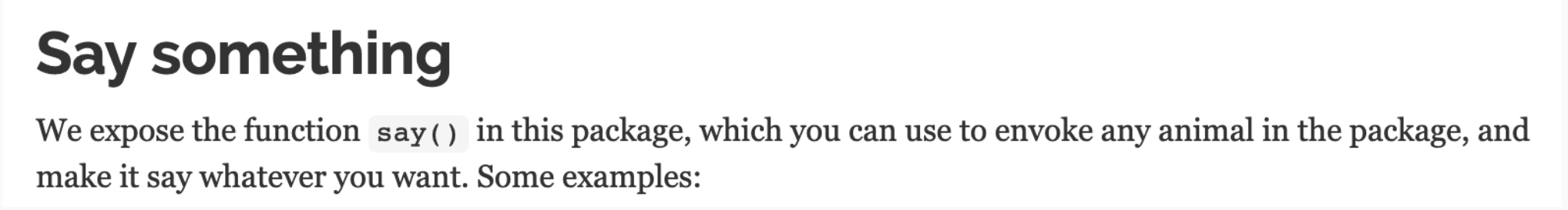


Extending functionality with packages

After installation, load into memory: `library(cowsay)`

What does `cowsay` do?

- CRAN has info about its packages:
  - https://cran.r-project.org/web/packages/cowsay/index.html



Say something

We expose the function `say()` in this package, which you can use to envoke any animal in the package, and make it say whatever you want. Some examples:

```
say("why did the chicken cross the road", "chicken")
```

Try this in R.

Extending functionality with packages

```
say("why did the chicken cross the road", "chicken")
```

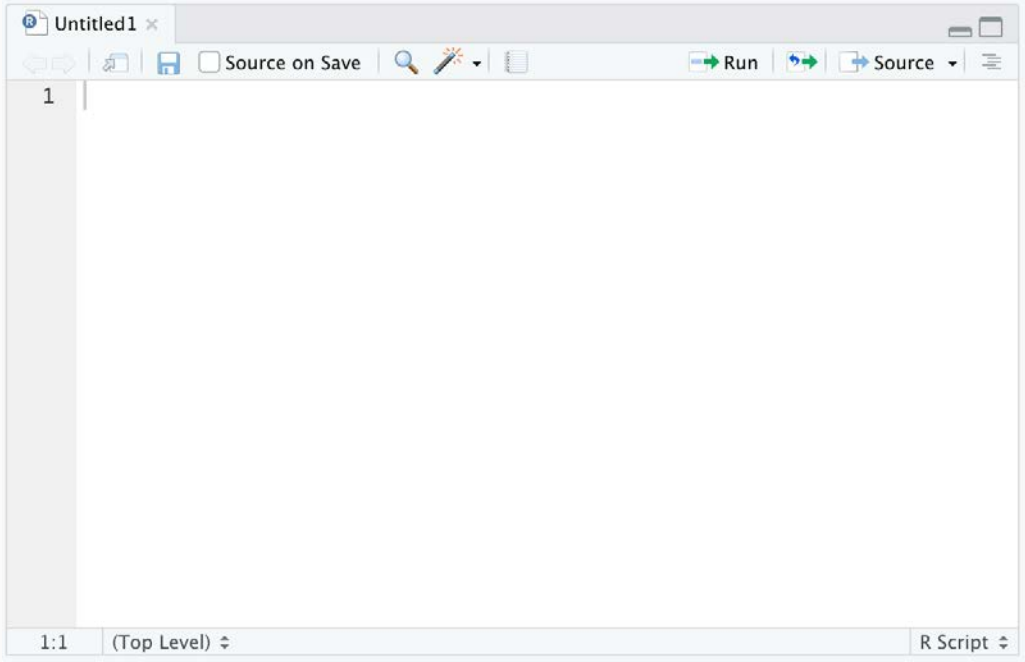
```
Colors cannot be applied in this environment :( Try using a terminal or RStudio.


 -----
why did the chicken cross the road
 ------
    \
     \
         _
       _/ }
      `>'  \
      `|    \
       |    /'-.      .-.
       \'       ';`--'  .'
        \'.      `'-./
         '.`-..-;`
           `;-..'
           _|  _|
          /` /` [nosig]
```

Scripting and Managing Your File System

Scripting

Open a new script (.R file)

File → New File → R Script



- Scripts = text documents containing R commands
- We should mostly work in scripts, not interactively in the console.
- Send the current line of the script to the console by hitting:
  - Cmd-Enter (Mac)
  - Control-Enter (PC)
- Try it with `say("Hello")`

Scripting

Here's a short script:

```
# congratulations.R  -  Congratulate the Raptors
library(cowsay) # load the library into memory

message = "Raptors 2019 NBA Champs"

say(message, by = "owl")
```

- `#` is the commenting character. R ignores the rest of the line after `#`
  - **Comment your code.** It'll remind you what you were trying to do.
- You can walk through a script line-by-line using Cmd-Enter, or...
- ...send the whole script to the console using Cmd-Shift-Enter
- Make the script above and try it out

Scripting

Here's a short script:

```
# congratulations.R  -  Congratulate the Raptors
library(cowsay) # load the library into memory

message = "Raptors 2019 NBA Champs"

say(message, by = "owl")
```

```
Colors cannot be applied in this environment :( Try using a terminal or RStudio.

 -----
Raptors 2019 NBA Champs
 ------
    \
     \
      \
       /\___/\
       {o}{o}|
       \ v  /|
       |    \ \
        \___/_/       [ab]
          | |
```

Project file system

- Scripts are like other documents. They need to be saved and backed up.
- Make a `projects/` folder in your Dropbox, Google Drive, iCloud, etc. folder
- Suggestion: store all your future statistical work in `projects/` subfolders
  - Statistical coursework like `pol2504/`
  - Projects like `voting_experiment/` (avoid spaces in folder names)

- Make a `projects/` subfolder `bootcamp/`
- Save a new R script into `bootcamp/` as `bootcamp_notes.R`
- Set R's working directory to the location of the current R script:
  - Session → Set Working Directory → Source File Location

- Save all your code and notes from this course in `bootcamp_notes.R`

Working with Data in R (Tidyverse edition)

Preliminaries

Install 'The Tidyverse': `install.packages('tidyverse')`

At top of `bootcamp_notes.R`, load into memory: `library(tidyverse)`

Next session - **Working with Data in R**