

Introduction to R

Week 1: The basics

Louisa Smith

July 13 - July 17

LET'S START WITH...

The basics

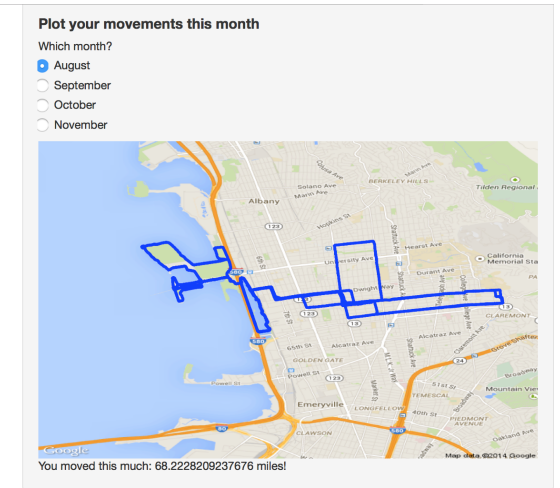
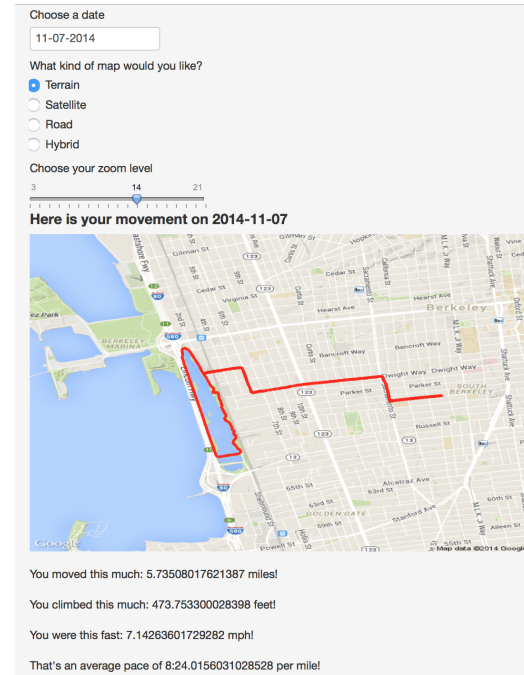
About this class

- Non-credit
- 6 weeks
- Watch the videos and do the exercises on your own (or with friends/classmates), come together for lab
- Practice by yourself in between classes
- Everything you need is at <http://intro-to-r-2020.louisahsmith.com>

You are not going to break anything!

About me

- Rising 5th-year PhD candidate in Epidemiology
- Started using R during my master's (so 6 years of experience); learned mostly by doing
- Problem sets, manuscripts, slides, website all in R
- Almost 100 R projects on my computer, over 1000 R scripts



I have to Google things literally every time I use R!

Plan

Week 1: The basics

Week 2: Figures

Week 3: Selecting, filtering, and mutating

Week 4: Grouping and tables

Week 5: Functions

Week 6: Analyze your data



An IDE for R

An integrated development environment is software that makes coding easier

- see objects you've imported and created
- autocomplete
- syntax highlighting
- run part or all of your code

SETUP...

YOUR TURN...

1

- Install R
- Install R Studio

The image shows the RStudio interface with several components and annotations:

- Environment:** This is where you see what objects you've created and data you've loaded.
- Files:** This is where you write code you want to save.
- Plots:** This is where results print out and where you write code you don't want to save.
- Help:** This is where you write code you don't want to save.

The RStudio interface includes a menu bar (File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, Help), a toolbar, a source editor (day1-script1.R), a console, and a file explorer. The source editor contains R code for creating a vector and running a script. The console shows the R version (3.6.0) and the RStudio logo.

```
1 #-----
2 # Title: Day 1, script 1
3 # Name: Your name here
4 #-----
5
6 #### Code -----
7 # Run through this code on your own to practice what we covered in
8 # the slides.
9
10 # create a vector of numeric values
11 vals = c(1, 645, 329)
12
13 # run
14 # what does each does
15 # what does each does
16 new_v
17 c(13,
18 new_v
19 new_v
20
21 # create and view different types of vectors
22
```

R version 3.6.0 (2019-04-26) -- "Planting of a Tree"
Copyright (C) 2019 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'q()' to quit R.

Files

Name	Size	Modified
..		
.Rhistory	0 B	Sep 5, 2019, 4:38 PM
day1-script1.R	1.5 KB	Sep 5, 2019, 4:05 PM
day1-script2.R	2.8 KB	Sep 5, 2019, 4:05 PM
day1-script3.R	2.8 KB	Sep 5, 2019, 4:06 PM
day1-script4.R	1.9 KB	Sep 5, 2019, 4:05 PM
day1-script5.R	1.2 KB	Sep 5, 2019, 4:05 PM
day1.Rproj	205 B	Sep 5, 2019, 4:38 PM
nlsy_cc.csv	44.5 KB	Sep 5, 2019, 4:05 PM
nlsy_cc.xlsx	79.8 KB	Sep 5, 2019, 4:05 PM

Packages

- Some functions are built into R
 - `mean()`, `lm()`, `table()`, etc.
- They actually come from built-in packages
 - `base`, `stats`, `graphics`, etc.
- Anyone (yes, *anyone*) build their own package to add to the functionality of R
 - `ggplot2`, `dplyr`, `data.table`, `survival`, etc.



Image from Zhi Yang

Packages

- You have to **install** a package once*

```
install.packages("survival")
```

- You then have to **load** the package every time you want to use it

```
library(survival)
```

*Actually, with every new major R release, but we won't worry about that.

Packages

"You only have to buy the book once, but you have to go get it out of the bookshelf every time you want to read it."

```
install.packages("survival")  
library(survival)  
survfit(...)
```

SEVERAL DAYS LATER...

```
library(survival)  
coxph(...)
```

DEMONSTRATION...

Package details

- When you use `install.packages`, packages are downloaded from **CRAN** (The Comprehensive R Archive Network)
 - This is also where you downloaded R
- Packages can be hosted lots of other places, such as **Bioconductor** (for bioinformatics), and **Github** (for personal projects or while still developing)
- The folks at CRAN check to make things "work" in some sense, but don't check on the statistical methods...
 - But because R is open-source, you can always read the code yourself
- Two functions from different packages can have the same name... if you load them both, you may have some trouble

tidyverse

- The same people who make RStudio also are responsible for a set of packages called the tidyverse



Journal of Statistical Software

August 2014, Volume 59, Issue 10.

<http://www.jstatsoft.org/>

Tidy Data

tidyverse

- Running `install.packages(tidyverse)` actually downloads more than a dozen packages*
- Running `library(tidyverse)` loads:
`ggplot2`, `dplyr`, `tidyr`, `readr`, `purrr`, `tibble`,
`stringr`, `forcats`
- This is by no means the only way to manage your data, but I find that a lot of the time, it's the easiest and simplest way to get things done.

*See which ones at <https://tidyverse.tidyverse.org>



2

YOUR TURN...

- Install the tidyverse "package"
- Load *one* of the tidyverse packages

R projects

```
my-project/  
├── my-project.Rproj  
├── README  
├── data/  
│   ├── raw/  
│   └── processed/  
├── code/  
├── results/  
│   ├── tables/  
│   ├── figures/  
│   └── output/  
└── docs/
```

- An `.Rproj` file is mostly just a placeholder. It remembers various options, and makes it easy to open a new RStudio session that starts up in the correct working directory. You never need to edit it directly.
- A README file can just be a text file that includes notes for yourself or future users.
- I like to have a folder for raw data -- which I never touch -- and a folder(s) for datasets that I create along the way.

This course

```
R-course/  
├── 01-week/  
│   ├── 01-week.Rproj  
│   ├── 01-exercises.R  
│   ├── 01-lab.Rmd  
│   ├── 01-slides.pdf  
│   └── data/  
│       └── nlsy.csv  
├── 02-week/  
│   ├── 02-week.Rproj  
│   ├── 02-exercises.R  
│   ├── 02-lab.Rmd  
│   ├── 02-slides.pdf  
│   └── data/  
│       └── nhanes.xlsx  
└── 03-week/
```

- Each week you'll download a zip file of some or all of the things you need for the week
 - You may be adding more later!
- Open the week's work by opening the `.Rproj` file
 - This will ensure you're in the right working directory to easily access the data, etc.

DEMONSTRATION...

3

YOUR TURN...

- Download the `01-week.zip` file here
- Open up the `01-week.Rproj` file

R uses `<-` for assignment

Create an object `vals` that contains a sequence of numbers:

```
# create values  
vals <- c(1, 645, 329)
```

Put your cursor at the end of the line and hit ctrl/cmd + enter.

Now `vals` holds those values.

We can see them again by running just the name (put your cursor after the name and press ctrl/cmd + enter again).

```
vals
```

```
## [1] 1 645 329
```

No assignment arrow means that the object will be printed to the console.

Types of data (*classes*)

We could also create a character *vector*:

```
chars <- c("dog", "cat", "rhino")  
chars
```

```
## [1] "dog" "cat" "rhino"
```

Or a *logical* vector:

```
logs <- c(TRUE, FALSE, FALSE)  
logs
```

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

We'll see more options as we go along!

Types of objects

We created *vectors* with the `c()` function (`c` stands for concatenate)

We could also create a *matrix* of values with the `matrix()` function:

```
# turn the vector of numbers into a 2-row matrix
mat <- matrix(c(234, 7456, 12, 654, 183, 753), nrow = 2)
mat
```

```
##      [,1] [,2] [,3]
## [1,]  234   12  183
## [2,] 7456  654  753
```

The numbers in square brackets are *indices*, which we can use to pull out values:

```
# extract second row
mat[2, ]
```

```
## [1] 7456  654  753
```

Dataframes

We usually do analysis in R with dataframes (or some variant).

Dataframes are basically like spreadsheets: columns are variables, and rows are observations.

```
gss_cat
```

```
## # A tibble: 21,483 x 9
```

```
##   year marital      age race  rincome      partyid      relig      denom
##   <int> <fct>      <int> <fct> <fct>      <fct>      <fct>      <fct>
## 1  2000 Never marr...    26 White $8000 to 99... Ind,near rep Protestant Southern
## 2  2000 Divorced      48 White $8000 to 99... Not str repu... Protestant Baptist-
## 3  2000 Widowed      67 White Not applica... Independent Protestant No denom
## 4  2000 Never marr...    39 White Not applica... Ind,near rep Orthodox-ch... Not app
## 5  2000 Divorced      25 White Not applica... Not str demo... None Not app
## 6  2000 Married      25 White $20000 - 24... Strong democ... Protestant Southern
## 7  2000 Never marr...    36 White $25000 or m... Not str repu... Christian Not app
## 8  2000 Divorced      44 White $7000 to 79... Ind,near dem Protestant Lutheran
```

tibble???



tibbles are basically just pretty dataframes

```
as_tibble(gss_cat)[, 1:4]
```

```
# A tibble: 21,483 x 4
   year marital    age race
  <int> <fct>    <int> <fct>
1  2000 Never married    26 White
2  2000 Divorced        48 White
3  2000 Widowed         67 White
4  2000 Never married    39 White
5  2000 Divorced         25 White
6  2000 Married         25 White
7  2000 Never married    36 White
8  2000 Divorced         44 White
9  2000 Married         44 White
10 2000 Married         47 White
# ... with 21,473 more rows
```

```
as.data.frame(gss_cat)[, 1:4]
```

```
   year marital age race
1  2000 Never married 26 White
2  2000      Divorced 48 White
3  2000      Widowed 67 White
4  2000 Never married 39 White
5  2000      Divorced 25 White
6  2000      Married 25 White
7  2000 Never married 36 White
8  2000      Divorced 44 White
9  2000      Married 44 White
10 2000      Married 47 White
11 2000      Married 53 White
12 2000      Married 52 White
13 2000      Married 52 White
14 2000      Married 51 White
```


and tibbles are the quickest and most intuitive way to make and read a dataset

```
dat1 <- tibble(  
  age = c(24, 76, 38),  
  height_in = c(70, 64, 68),  
  height_cm = height_in * 2.54  
)  
dat1
```

```
## # A tibble: 3 x 3  
##       age height_in height_cm  
##   <dbl>   <dbl>   <dbl>  
## 1     24       70     178.  
## 2     76       64     163.  
## 3     38       68     173.
```

```
dat2 <- tribble(  
  ~n, ~food, ~animal,  
  39, "banana", "monkey",  
  21, "milk", "cat",  
  18, "bone", "dog"  
)  
dat2
```

```
## # A tibble: 3 x 3  
##       n food  animal  
##   <dbl> <chr> <chr>  
## 1     39 banana monkey  
## 2     21 milk   cat  
## 3     18 bone   dog
```

4

YOUR TURN...

- Work through the code in `01-week/01-todo.R`