Welcome!

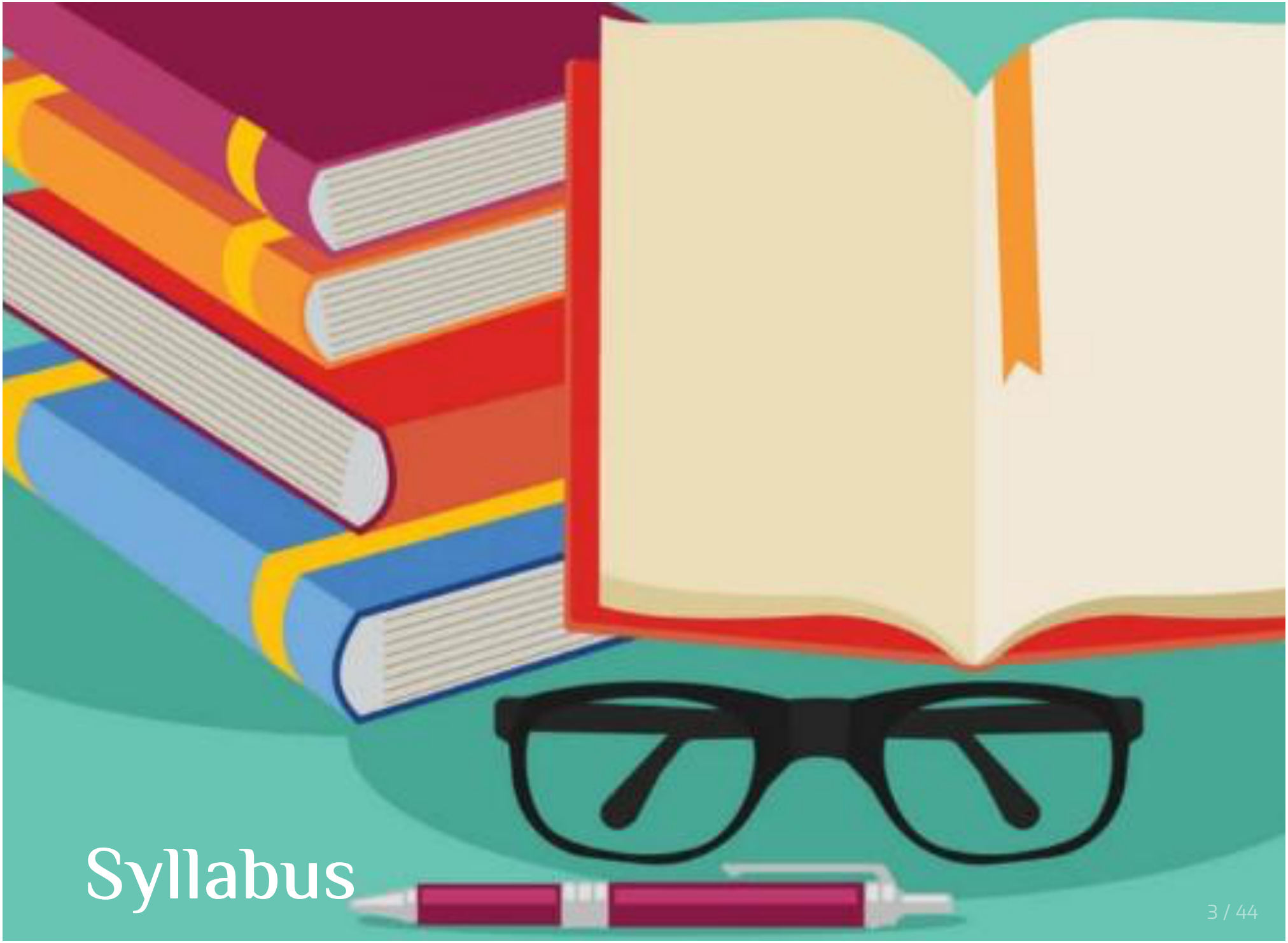
An overview of the course

Daniel Anderson
Week 1, Class 1

UNIVERSITY OF
OREGON

Agenda

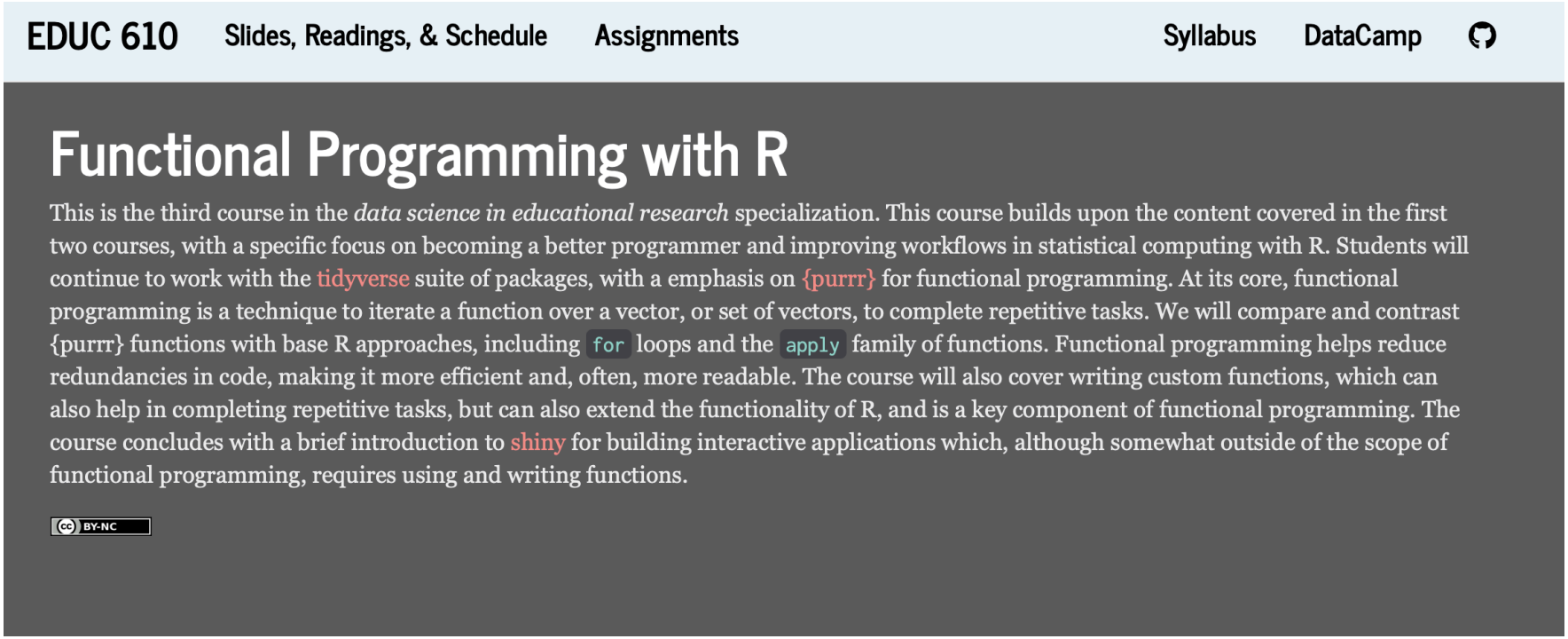
- Getting on the same page
- Syllabus
- Intro to data types



Syllabus

Course Website(s)

website repo



Learning objects for this class

Learning objects for this class

- Understand and be able to describe the differences in R's data structures and when each is most appropriate for a given task

Learning objects for this class

- Understand and be able to describe the differences in R's data structures and when each is most appropriate for a given task
- Explore `purrr::map` and its variants, how they relate to base R functions, and why the {purrr} variants are often preferable.

Learning objects for this class

- Understand and be able to describe the differences in R's data structures and when each is most appropriate for a given task
- Explore `purrr::map` and its variants, how they relate to base R functions, and why the {purrr} variants are often preferable.
- Work with lists and list columns using `purrr::nest` and `purrr:unnest`

Learning objects for this class

- Understand and be able to describe the differences in R's data structures and when each is most appropriate for a given task
- Explore `purrr::map` and its variants, how they relate to base R functions, and why the {purrr} variants are often preferable.
- Work with lists and list columns using `purrr::nest` and `purrr:unnest`
- Fit multiple (basic) models using the {broom} package with {purrr}

Learning objects for this class

- Understand and be able to describe the differences in R's data structures and when each is most appropriate for a given task
- Explore `purrr::map` and its variants, how they relate to base R functions, and why the {purrr} variants are often preferable.
- Work with lists and list columns using `purrr::nest` and `purrr:unnest`
- Fit multiple (basic) models using the {broom} package with {purrr}
- Convert repetitive tasks into functions

Learning objects for this class

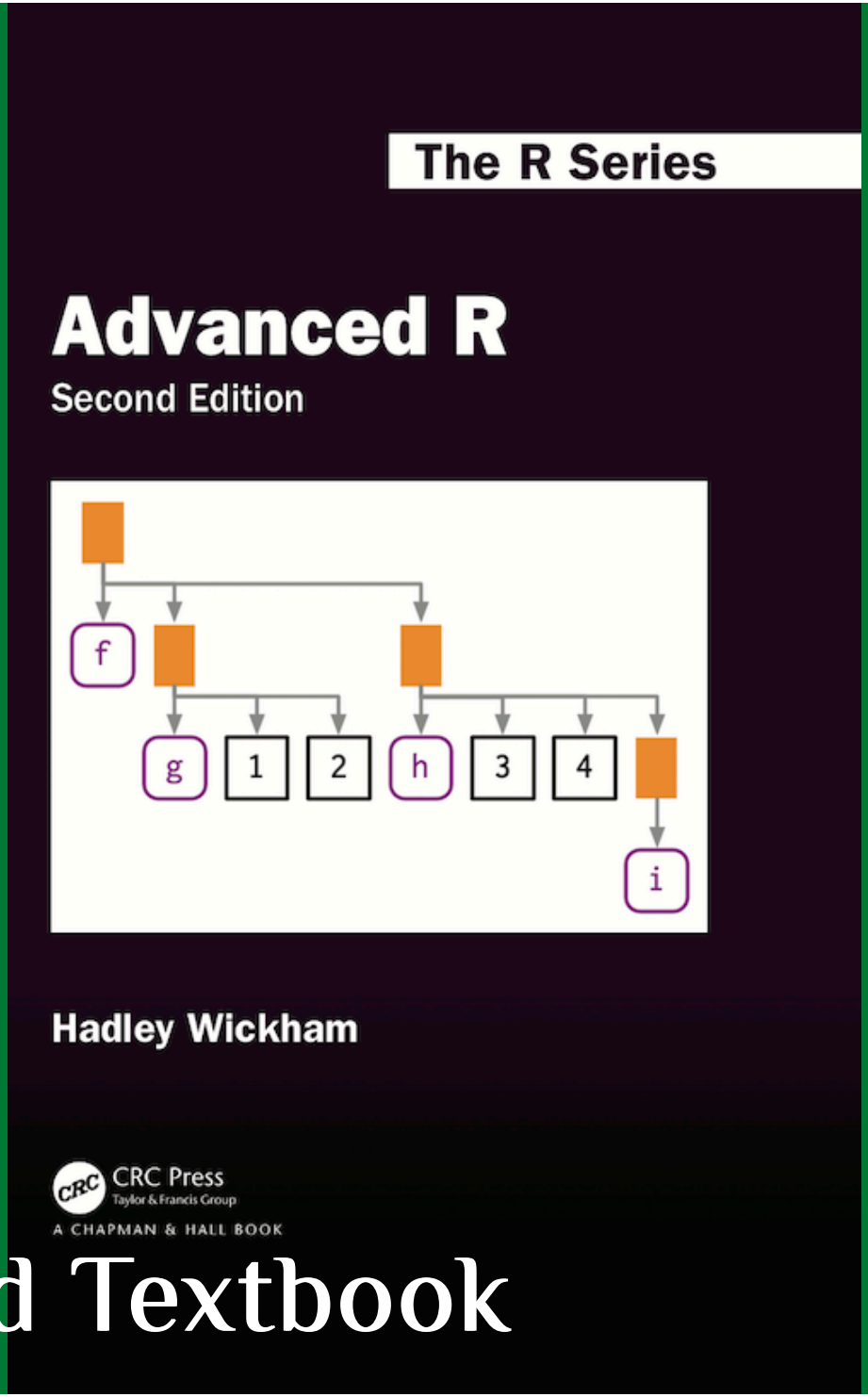
- Understand and be able to describe the differences in R's data structures and when each is most appropriate for a given task
- Explore `purrr::map` and its variants, how they relate to base R functions, and why the {purrr} variants are often preferable.
- Work with lists and list columns using `purrr::nest` and `purrr:unnest`
- Fit multiple (basic) models using the {broom} package with {purrr}
- Convert repetitive tasks into functions
- Understand elements of good functions, and things to avoid

Learning objects for this class

- Understand and be able to describe the differences in R's data structures and when each is most appropriate for a given task
- Explore `purrr::map` and its variants, how they relate to base R functions, and why the {purrr} variants are often preferable.
- Work with lists and list columns using `purrr::nest` and `purrr:unnest`
- Fit multiple (basic) models using the {broom} package with {purrr}
- Convert repetitive tasks into functions
- Understand elements of good functions, and things to avoid
- Write effective and clear functions to continue with the mantra of "Don't Repeat Yourself"

This Week's learning objectives

- Understand the requirements of the course
- Understand the requirements of the final project

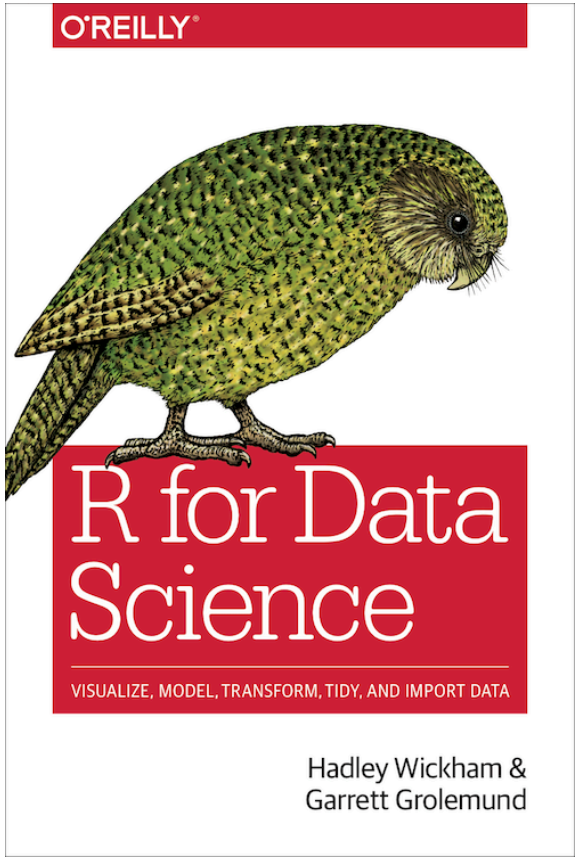


Required Textbook

Other books (also free)

Bryan (http://happygitwithr.com)

Wickham & Grolemund (https://r4ds.had.co.nz)



Structure of the course

- First 5 weeks - mostly iteration
    - Data types
    - Base R iterations
    - {purrr}
    - Batching processes and working with list columns
    - Parallel iterations (and a few extras)

Structure of the course

- First 5 weeks - mostly iteration
    - Data types
    - Base R iterations
    - {purrr}
    - Batching processes and working with list columns
    - Parallel iterations (and a few extras)
- Second 5 weeks - mostly writing functions
    - Writing functions 1
    - Writing functions 2
    - Shiny 1
    - Shiny 2
    - Code performance & packages (briefly)

Labs

16%

4 @ 10 points each

Two labs during each 5 week segment

- Iteration w/Base R
  - Mostly `for` loops
- Iteration w/{purrr}
  - Mostly `map_*`
- Writing functions 1
  - Basic functions
- Writing functions 2
  - Making functions "pure"
  - Building up functionality (one function calling another)

DataCamp

DataCamp

One required module (10 points)

DataCamp

One required module (10 points)

One module available for extra credit (5 points)

DataCamp

One required module (10 points)

One module available for extra credit (5 points)

Split by 5 week sections

Must complete one (due dates differ by section)

If you want the extra credit, one must come from each section

First 5 week DataCamp courses

Due by 11:59 PM, 5/6/19

- Intermediate R
- Foundations of functional programming with purrr
- Intermediate functional programming with purrr
- Machine learning in the tidyverse

Second 5 week DataCamp courses

Due by 11:59 PM, 6/12/19

- Writing functions in R
- Building web applications in R with Shiny
- Building web applications in R with Shiny: Case studies
- Developing R packages
- Writing efficient R code

Midterm

50 points (20%)

- Final format still yet to be determined - happy to take input on what it should look like

Midterm

50 points (20%)

- Final format still yet to be determined - happy to take input on what it should look like
- **Will** include you writing new code

Midterm

50 points (20%)

- Final format still yet to be determined - happy to take input on what it should look like
- **Will** include you writing new code
- **May** include you
  - Identifying bugs in code
  - Automating an existing approach
  - Multiple choice/fill-in the blank questions

Final Project

150 points total (60%)

5 parts

Component	Points	Due
Groups Finalized	0	4/10/19
Outline	10	5/1/19
Draft	20	5/27/19
Peer review	20	6/3/19
Product	100	6/12/19 (11:59 pm)

What is it?

Two basic options

What is it?

Two basic options

Data Product

Similar to first class

- Brief research manuscript (can be APA or not, I don't really care 🤷‍♀️)
- Shiny app
- Dashboard
    - Probably unlikely to work well though, unless you make it a shiny dashboard
- Blog post
- For the ambitious - a documented R package

Tutorial

- Probably best done through a blog post or series of blog posts
- Approach as if you're teaching others about the content I'll ask you to cover
- BONUS: You can actually release the blog post(s) and may get some traffic 🎉👏🥳


Make it your own

What you have to have

- Everything on GitHub
- Publicly available dataset
- Team of 2-4

What you have to cover

Unfortunately, this still is a class assignment. I have to be able to evaluate that you can actually apply the content within a messy, real-world setting.

What you have to cover

Unfortunately, this still is a class assignment. I have to be able to evaluate that you can actually apply the content within a messy, real-world setting.

The grading criteria (which follow) may force you into some use cases that are a bit artificial. This is okay.

Grading criteria

- No code is used repetitively (no more than twice) 10 points

Grading criteria

- No code is used repetitively (no more than twice) 10 points
- More than one variant of `purrr::map` is used 10 points

Grading criteria

- No code is used repetitively (no more than twice) 10 points
- More than one variant of `purrr::map` is used 10 points
- At least one {purrr} function outside the basic `map` family (`walk_*`, `reduce`, `modify_*`, etc.) 10 points

Grading criteria

- No code is used repetitively (no more than twice) 10 points
- More than one variant of `purrr::map` is used 10 points
- At least one {purrr} function outside the basic `map` family (`walk_*`, `reduce`, `modify_*`, etc.) 10 points
- At least one instance of parallel iteration (e.g., `map2_*`, `pmap_*`)

  10 points

Grading criteria

- No code is used repetitively (no more than twice) 10 points
- More than one variant of `purrr::map` is used 10 points
- At least one {purrr} function outside the basic `map` family (`walk_*`, `reduce`, `modify_*`, etc.) 10 points
- At least one instance of parallel iteration (e.g., `map2_*`, `pmap_*`)

  10 points
- At least one use case of `purrr::nest %>% mutate()` 10 points

- At least two custom functions 20 points; 10 points each
  - Each function must be "pure" unless it is clear that it should have side effects (e.g., a plotting function)
  - Each function must do exactly one thing
  - The functions **may** replicate the behavior of a base function - as noted above this is about practicing the skills you learn in class

- At least two custom functions 20 points; 10 points each
  - Each function must be "pure" unless it is clear that it should have side effects (e.g., a plotting function)
  - Each function must do exactly one thing
  - The functions **may** replicate the behavior of a base function - as noted above this is about practicing the skills you learn in class

- Code is fully reproducible and housed on GitHub 10 points

- At least two custom functions 20 points; 10 points each
  - Each function must be "pure" unless it is clear that it should have side effects (e.g., a plotting function)
  - Each function must do exactly one thing
  - The functions **may** replicate the behavior of a base function - as noted above this is about practicing the skills you learn in class
- Code is fully reproducible and housed on GitHub 10 points
- No obvious errors in chosen output format 10 points

- At least two custom functions 20 points; 10 points each
  - Each function must be "pure" unless it is clear that it should have side effects (e.g., a plotting function)
  - Each function must do exactly one thing
  - The functions **may** replicate the behavior of a base function - as noted above this is about practicing the skills you learn in class
- Code is fully reproducible and housed on GitHub 10 points
- No obvious errors in chosen output format 10 points
- Deployed on the web and shareable through a link 10 points

Outline

Due 5/1/19

Four components:

- Description of data source (must be publicly available)
- Purpose (tutorial or substantive)
- Chosen format
- Lingering questions
    - How can I help?

Outline

Due 5/1/19

Four components:

- Description of data source (must be publicly available)
- Purpose (tutorial or substantive)
- Chosen format
- Lingering questions
    - How can I help?

Please include all components - including the question(s) section!

Draft

Due 5/27/19, before class

- Expected to still be a work in progress
    - This means some of your code may be rough and/or incomplete. However:
- Direction should be obvious
- Most, if not all, grading elements should be present
- Provided to your peers so they can learn from you as much as you can learn from their feedback

Peer Review

- Exact same process we've used before
- It has always gone very well - let's keep it going
- If, during your peer review, you find grading elements not present, definitely note them

Utilizing GitHub (required)

- You'll be assigned two groups to review (10 points each)
- Fork their repo
- Embed comments, suggest changes to their code
- Submit a PR
    - Summarize your overall review in the PR
- I WILL cover this this term, barring a tornado or similar

Grading

250 points total

- 4 labs at 10 points each (40 points; 16%)
- 1 DataCamp modules (10 points; 4%)
- Midterm (50 points; 20%)
- Final Project (150 points; 60%)
  - Proposal (10 points; 4%)
  - Draft (20 points; 8%)
  - Peer review (20 points; 8%)
  - Product (100 points; 40%)

Grading

Lower percent	Lower point range	Grade	Upper point range	Upper percent
0.97	(242 pts)	A+	(250 pts)	
0.93	(232 pts)	A	(242 pts)	0.97
0.90	(225 pts)	A-	(232 pts)	0.93
0.87	(218 pts)	B+	(225 pts)	0.90
0.83	(208 pts)	B	(218 pts)	0.87
0.80	(200 pts)	B-	(208 pts)	0.83
0.77	(192 pts)	C+	(200 pts)	0.80
0.73	(182 pts)	C	(192 pts)	0.77
0.70	(175 pts)	C-	(182 pts)	0.73
		F	(175 pts)	0.70


COMMUNICATION

Missing class

I will be out Monday next week

- Trying to get a guest speaker
- May end up having to adjust the schedule a bit

Any time left?

Any time left?

Basic data types

Vectors

Pop quiz

Discuss with your neighbor

- What are the four basic types of atomic vectors?
- What function creates a vector?
- T/F: A list (an R list) is not a vector.
- What is the fundamental difference between a matrix and a data frame?
- What does *coercion* mean, and when does it come into play?

Vector types

4 basic types

Note there are two others (complex and raw), but we don't care about them (I've never even seen them used).

- Integer
- Double
- Logical
- Character

Vector types

4 basic types

Note there are two others (complex and raw), but we don't care about them (I've never even seen them used).

- Integer
- Double
- Logical
- Character

Integer and double vectors are both numeric.

Creating vectors

Vectors are created with `c`. Below are examples of each of the four main types of vectors.

```
integer <- c(5L, 7L, 3L, 94L) # L explicitly an integer, not double
double <- c(3.27, 8.41, Inf, -Inf)
logical <- c(TRUE, TRUE, TRUE, FALSE, TRUE, FALSE, FALSE)
character <- c("red", "orange", "yellow", "green", "blue", "violet", "rainbo
```

Coercion

- Vectors **must** be of the same type.
- If you try to mix types, implicit coercion will occur
- Implicit coercion defaults to the most flexible type
    - which is... ?

Coercion

- Vectors **must** be of the same type.
- If you try to mix types, implicit coercion will occur
- Implicit coercion defaults to the most flexible type
    - which is... ?

```
c(7L, 3.25)
```

```
## [1] 7.00 3.25
```

```
c(3.24, TRUE, "April")
```

```
## [1] "3.24"  "TRUE"  "April"
```

```
c(TRUE, 5)
```

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

Explicit coercion

- You can alternatively define the coercion to occur

```
as.integer(c(7L, 3.25))
```

```
## [1] 7 3
```

```
as.logical(c(3.24, TRUE, "April"))
```

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

```
as.character(c(TRUE, 5))
```

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

Checking types

- Use `typeof` to verify the type of vector

```
typeof(c(7L, 3.25))
```

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

```
typeof(as.integer(c(7L, 3.25)))
```

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

Piping

- Although traditionally used within the tidyverse (not what we're doing here), it can still be useful. The following are equivalent

```
library(magrittr)

typeof(as.integer(c(7L, 3.25)))
```

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

```
c(7L, 3.25) %>%
  as.integer() %>%
  typeof()
```

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

Pop quiz

Without actually running the code, talk with your neighbor: Which will each of the following coerce to?

```
c(1.25, TRUE, 4L)
c(1L, FALSE)
c(7L, 6.23, "eight")
c(TRUE, 1L, 0L, "False")
```

Answers

```
typeof(c(1.25, TRUE, 4L))
```

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

```
typeof(c(1L, FALSE))
```

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

```
typeof(c(7L, 6.23, "eight"))
```

```
## [1] "character"
```

```
typeof(c(TRUE, 1L, 0L, "False"))
```

```
## [1] "character"
```

Missing values

- Missing values breed missing values

```
NA > 5
```

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

```
NA * 7
```

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

Missing values

- Missing values breed missing values

```
NA > 5
```

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

```
NA * 7
```

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

- But this one may surprise you

```
NA == NA
```

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

Missing values

- Missing values breed missing values

```
NA > 5
```

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

```
NA * 7
```

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

- But this one may surprise you

```
NA == NA
```

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

It is correct, however, because there's no reason to presume that one missing value is equal to another missing value.

When missing values don't propagate

```
NA & FALSE
```

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

```
NA | TRUE
```

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

```
x <- c(NA, 3, NA, 5)
any(x > 4)
```

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

How to test missingness?

- We've already seen the following doesn't work

```
x == NA
```

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

How to test missingness?

- We've already seen the following doesn't work

```
x == NA
```

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

- Instead, use `is.na`

```
is.na(x)
```

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

Next time

- More on data types
  - Subsetting
  - Attributes
  - More on coercion (good to be fluent)
- Lists

Any lingering questions?