

Web scraping

Dr. Maria Tackett

09.24.19

[Click for PDF of slides](#)



Announcements

- Lab 04 - **due Wednesday, 9/25 at 11:59p**
- Writing Exercise #2 initial draft - **due Thursday, 9/26 at 11:59p**
 - Access Eli at <https://app.elireview.com/unit> or through the link on Sakai
- HW 02 - **due Thursday, 9/26 at 11:59p**
- Team Feedback #1 **due Thursday, 9/26 at 11:59p**
 - Please provide honest and constructive feedback. This team feedback will be graded for completion.

Today's Agenda

- Data frames and tibbles
- Factors
- Web scraping

Data "set"

Data "sets" in R

- "set" is in quotation marks because it is not a formal data class
- A tidy data "set" can be one of the following types:
 - **tibble**
 - **data.frame**

Data frames & tibbles

- A **data.frame** is the most commonly used data structure in R, they are just a list of equal length vectors. Each vector is treated as a column and elements of the vectors as rows.
- A tibble is a type of data frame that ... makes the data analysis easier.
- Most often a data frame will be constructed by reading in from a file, but we can also create them from scratch.
 - **readr** package (e.g. **read_csv** function) loads data as a **tibble** by default
 - **tibbles** are part of the tidyverse, so they work well with other packages we are using
 - they make minimal assumptions about your data, so are less likely to cause hard to track bugs in your code

Creating data frames

```
df <- tibble(x = 1:3, y = c("a", "b", "c"))  
class(df)
```

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

```
glimpse(df)
```

```
## Observations: 3  
## Variables: 2  
## $ x <int> 1, 2, 3  
## $ y <chr> "a", "b", "c"
```


Features of data frames

```
attributes(df)
```

```
## $names  
## [1] "x" "y"  
##  
## $row.names  
## [1] 1 2 3  
##  
## $class  
## [1] "tbl_df"      "tbl"        "data.frame"
```

```
class(df$x)
```

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

```
class(df$y)
```

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

Working with tibbles in pipelines

- Recall the [cat lovers](#) data set

How many respondents have below average number of cats?

```
mean_cats <- cat_lovers %>%  
  summarise(mean_cats = mean(number_of_cats))  
  
cat_lovers %>%  
  filter(number_of_cats < mean_cats) %>%  
  nrow()
```

```
## [1] 60
```

Do you believe this number? Why, why not?

A result of a pipeline is always a tibble

```
mean_cats
```

```
## # A tibble: 1 x 1  
##   mean_cats  
##   <dbl>  
## 1      0.817
```

```
class(mean_cats)
```

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

pull() can be great

But use it sparingly!

```
mean_cats <- cat_lovers %>%  
  summarise(mean_cats = mean(number_of_cats)) %>%  
  pull()  
  
cat_lovers %>%  
  filter(number_of_cats < mean_cats) %>%  
  nrow()
```

```
## [1] 33
```

```
mean_cats
```

```
## [1] 0.8166667
```

```
class(mean_cats)
```

```
## [1] "numeric"
```

Factors

Factors

Factor objects are what R uses to store data for categorical variables (fixed numbers of discrete values).

```
(x = factor(c("BS", "MS", "PhD", "MS")))
```

```
## [1] BS  MS  PhD MS  
## Levels: BS MS PhD
```

```
glimpse(x)
```

```
## Factor w/ 3 levels "BS","MS","PhD": 1 2 3 2
```

```
typeof(x)
```

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

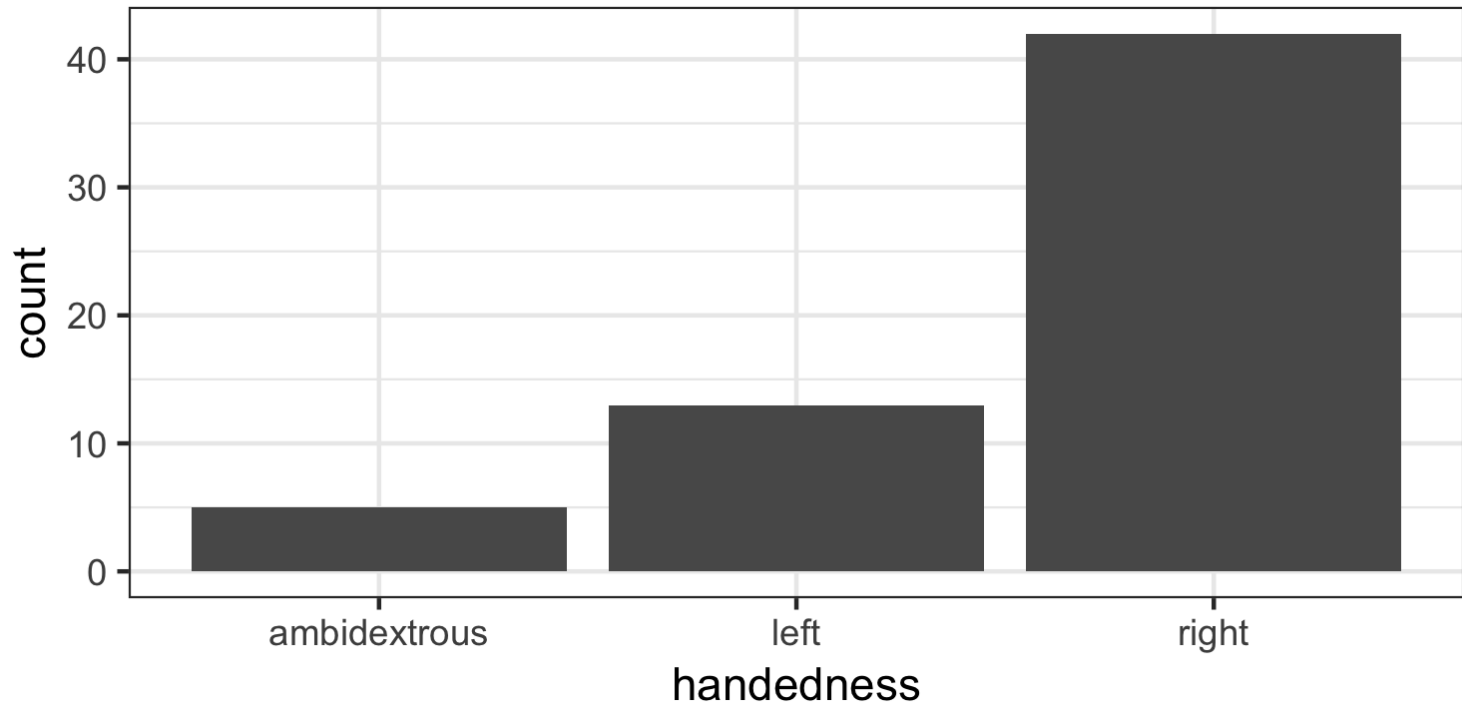
Read data in as character strings

```
glimpse(cat_lovers)
```

```
## Observations: 60
## Variables: 3
## $ name          <chr> "Bernice Warren", "Woodrow Stone", "Willie Bass",...
## $ number_of_cats <dbl> 0, 0, 1, 3, 3, 2, 1, 1, 0, 0, 0, 0, 1, 3, 3, 2, 1...
## $ handedness     <chr> "left", "left", "left", "left", "left", "left", "...
```

But coerce when plotting

```
ggplot(cat_lovers, mapping = aes(x = handedness)) +  
  geom_bar()
```



Use forcats to manipulate factors

```
cat_lovers <- cat_lovers %>%  
  mutate(handedness = fct_relevel(handedness,  
                                   "right", "left", "ambidextrous"))
```

```
ggplot(cat_lovers, mapping = aes(x = handedness)) +  
  geom_bar()
```

forcats



- R uses factors to handle categorical variables, variables that have a fixed and known set of possible values. Historically, factors were much easier to work with than character vectors, so many base R functions automatically convert character vectors to factors.
- However, factors are still useful when you have true categorical data, and when you want to override the ordering of character vectors to improve display. The goal of the forcats package is to provide a suite of useful tools that solve common problems with factors.

Recap

- Always best to think of data as part of a tibble
 - This works nicely with the **tidyverse** as well
 - Rows are observations, columns are variables
- Be careful about data types / classes
 - Sometimes **R** makes silly assumptions about your data class
 - Using **tibbles** help, but it might not solve all issues
 - Think about your data in context, e.g. 0/1 variable is most likely a **factor**
 - If a plot/output is not behaving the way you expect, first investigate the data class
 - If you are absolutely sure of a data class, over-write it in your tibble so that you don't need to keep having to keep track of it
 - **mutate** the variable with the correct class

Scraping the web

Scraping the web: what? why?

- Increasing amount of data is available on the web.
- These data are provided in an unstructured format: you can always copy&paste, but it's time-consuming and prone to errors.
- Web scraping is the process of extracting this information automatically and transform it into a structured dataset.
- Two different scenarios:
 - **Screen scraping**: extract data from source code of website, with html parser (easy) or regular expression matching (less easy).
 - **Web APIs (application programming interface)**: website offers a set of structured http requests that return JSON or XML files.

Web Scraping with rvest

Hypertext Markup Language (HTML)

Most of the data on the web is still largely available as HTML - while it is structured (hierarchical / tree based) it often is not available in a form useful for analysis (flat / tidy).

```
<html>
  <head>
    <title>This is a title</title>
  </head>
  <body>
    <p align="center">Hello world!</p>
  </body>
</html>
```

rvest

- **rvest** is a package that makes basic processing and manipulation of HTML data straight forward
- It's designed to work with pipelines built with `%>%`



Core functions in rvest:

- **read_html** - read HTML data from a url or character string.
- **html_nodes** - select specified nodes from the HTML document using CSS selectors.
- **html_table** - parse an HTML table into a data frame.
- **html_text** - extract tag pairs' content.
- **html_name** - extract tags' names.
- **html_attrs** - extract all of each tag's attributes.
- **html_attr** - extract tags' attribute value by name.

SelectorGadget

- **SelectorGadget**: Open source tool that eases CSS selector generation and discovery
- Easiest to use with the [Chrome Extension](#)
- Learn more on the [Selector Gadget Vignette](#)

Using SelectorGadget

- Click on the app logo next to the search bar
- A box will open in the bottom right of the website. Click on a page element that you would like your selector to match (it will turn green). SelectorGadget will then generate a minimal CSS selector for that element, and will highlight (yellow) everything that is matched by the selector.
- Now click on a highlighted element to remove it from the selector (red), or click on an unhighlighted element to add it to the selector. Through this process of selection and rejection, SelectorGadget helps you come up with the appropriate CSS selector for your needs.

Top 250 movies on IMDB

Top 250 movies on IMDB

Take a look at the source code, look for **table** tag:

<http://www.imdb.com/chart/top>




IMDb Charts

Top Rated Movies

Top 250 as rated by IMDb Users

Showing 250 Titles

Sort by: Ranking

Rank & Title	IMDb Rating	Your Rating
 1. The Shawshank Redemption (1994)	★ 9.2	☆ +
 2. The Godfather (1972)	★ 9.2	☆ +
 3. The Godfather: Part II (1974)	★ 9.0	☆ +

First check to make sure you're allowed!

```
# install.packages("robotstxt")  
library(robotstxt)  
paths_allowed("http://www.imdb.com")
```

```
##  
www.imdb.com                      No encoding supplied: defaulting to UTF-8.  
  
## [1] TRUE
```

versus

```
paths_allowed("http://www.facebook.com")
```

```
##  
www.facebook.com  
  
## [1] FALSE
```

Demo



Go to rstudio.cloud → Web scraping → Make a copy → **scrape-250.R**

Select and format pieces

```
library(rvest)

page <- read_html("http://www.imdb.com/chart/top")

titles <- page %>%
  html_nodes(".titleColumn a") %>%
  html_text()

years <- page %>%
  html_nodes(".secondaryInfo") %>%
  html_text() %>%
  str_replace("\\(", "") %>% # remove (
  str_replace("\\)", "") %>% # remove )
  as.numeric()

scores <- page %>%
  html_nodes(".imdbRating") %>%
  html_text() %>%
  as.numeric()

imdb_top_250 <- tibble(
  title = titles,
  year = years,
  score = scores
)
```


title	year	score
The Shawshank Redemption	1994	9.2
The Godfather	1972	9.1
The Godfather: Part II	1974	9
The Dark Knight	2008	9
12 Angry Men	1957	8.9
Schindler's List	1993	8.9
The Lord of the Rings: The Return of the King	2003	8.9
Pulp Fiction	1994	8.9
The Good, the Bad and the Ugly	1966	8.8
Fight Club	1999	8.8
...

Clean up / enhance

May or may not be a lot of work depending on how messy the data are

- See if you like what you got:

```
glimpse(imdb_top_250)
```

```
## Observations: 250
## Variables: 3
## $ title <chr> "The Shawshank Redemption", "The Godfather", "The Godfathe...
## $ year <dbl> 1994, 1972, 1974, 2008, 1957, 1993, 2003, 1994, 1966, 1999...
## $ score <dbl> 9.2, 9.1, 9.0, 9.0, 8.9, 8.9, 8.9, 8.9, 8.8, 8.8, 8.8, 8.8...
```

- Add a variable for rank

```
imdb_top_250 <- imdb_top_250 %>%
  mutate(
    rank = 1:nrow(imdb_top_250)
  )
```

title	year	score	rank
The Shawshank Redemption	1994	9.2	1
The Godfather	1972	9.1	2
The Godfather: Part II	1974	9	3
The Dark Knight	2008	9	4
12 Angry Men	1957	8.9	5
Schindler's List	1993	8.9	6
The Lord of the Rings: The Return of the King	2003	8.9	7
Pulp Fiction	1994	8.9	8
The Good, the Bad and the Ugly	1966	8.8	9
Fight Club	1999	8.8	10
...

Analyze

How would you go about answering this question: Which 1995 movies made the list?

```
imdb_top_250 %>%  
  filter(year == 1995)
```

```
## # A tibble: 8 x 4  
##   title          year score  rank  
##   <chr>        <dbl> <dbl> <int>  
## 1 Se7en        1995   8.6   20  
## 2 The Usual Suspects 1995   8.5   32  
## 3 Braveheart    1995   8.3   75  
## 4 Toy Story     1995   8.3   81  
## 5 Heat         1995   8.2  121  
## 6 Casino       1995   8.2  139  
## 7 Before Sunrise  1995   8.1  194  
## 8 La Haine     1995    8  228
```

Analyze

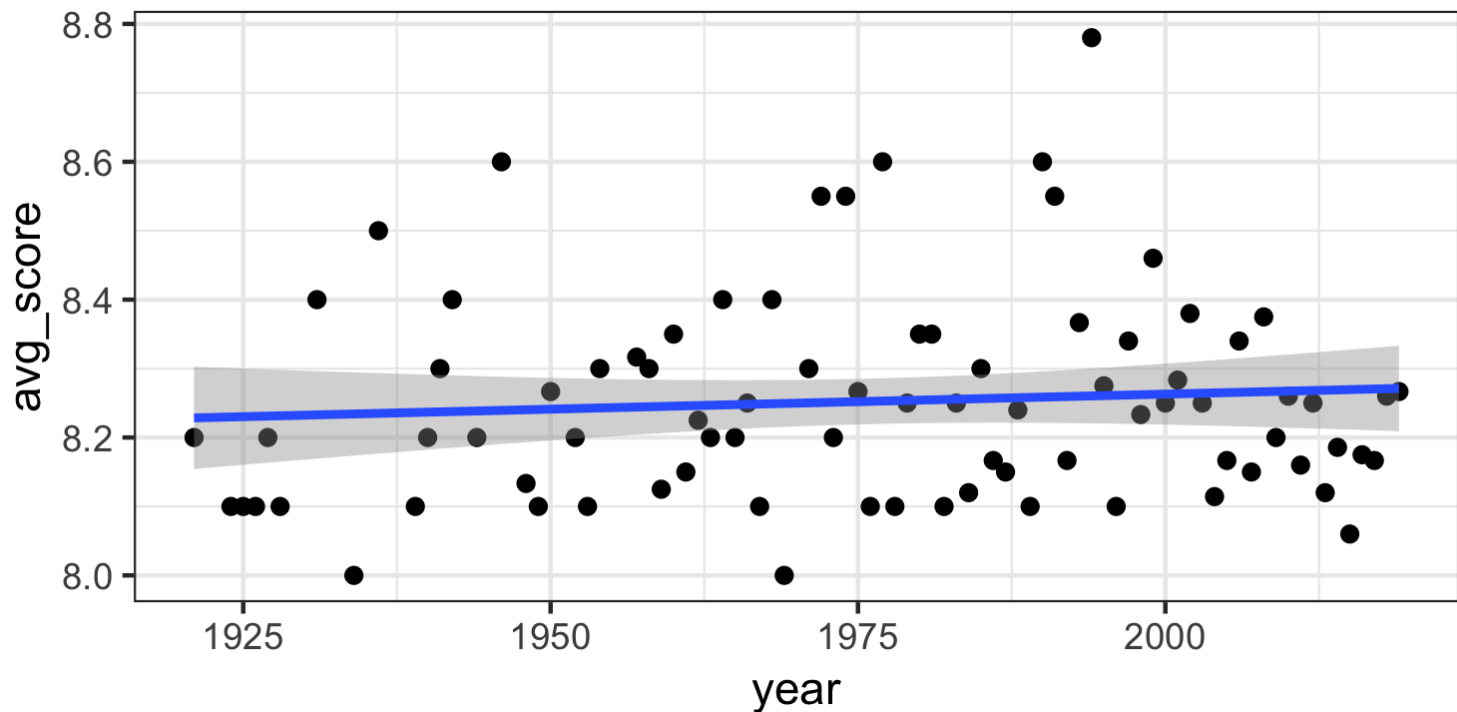
How would you go about answering this question: Which years have the most movies on the list?

```
imdb_top_250 %>%  
  group_by(year) %>%  
  summarise(total = n()) %>%  
  arrange(desc(total)) %>%  
  head(5)
```

```
## # A tibble: 5 x 2  
##   year total  
##   <dbl> <int>  
## 1  1995     8  
## 2  2004     7  
## 3  2014     7  
## 4  1957     6  
## 5  1998     6
```

Visualize

How would you go about creating this visualization: Visualize the average yearly score for movies that made it on the top 250 list over time.



Potential challenges

- Unreliable formatting at the source
- Data broken into many pages
- ...

Compare the display of information at raleigh.craigslist.org/search/apa to the list on the IMDB top 250 list.

What challenges can you foresee in scraping a list of the available apartments?

Application Exercise

Popular TV Shows

RStudio Cloud → Web scraping

1. Scrape the list of most popular TV shows on IMDB:
<http://www.imdb.com/chart/tvmeter>
2. Examine each of the first three (or however many you can get through) tv show subpage to also obtain genre and runtime.
3. Time permitting, also try to get the following:
 - How many episodes so far
 - Certificate
 - First five plot keywords
 - Country
 - Language

Add this information to the data frame you created in step 1.