

# ETC1010: Introduction to Data Analysis

## Week 5, part B

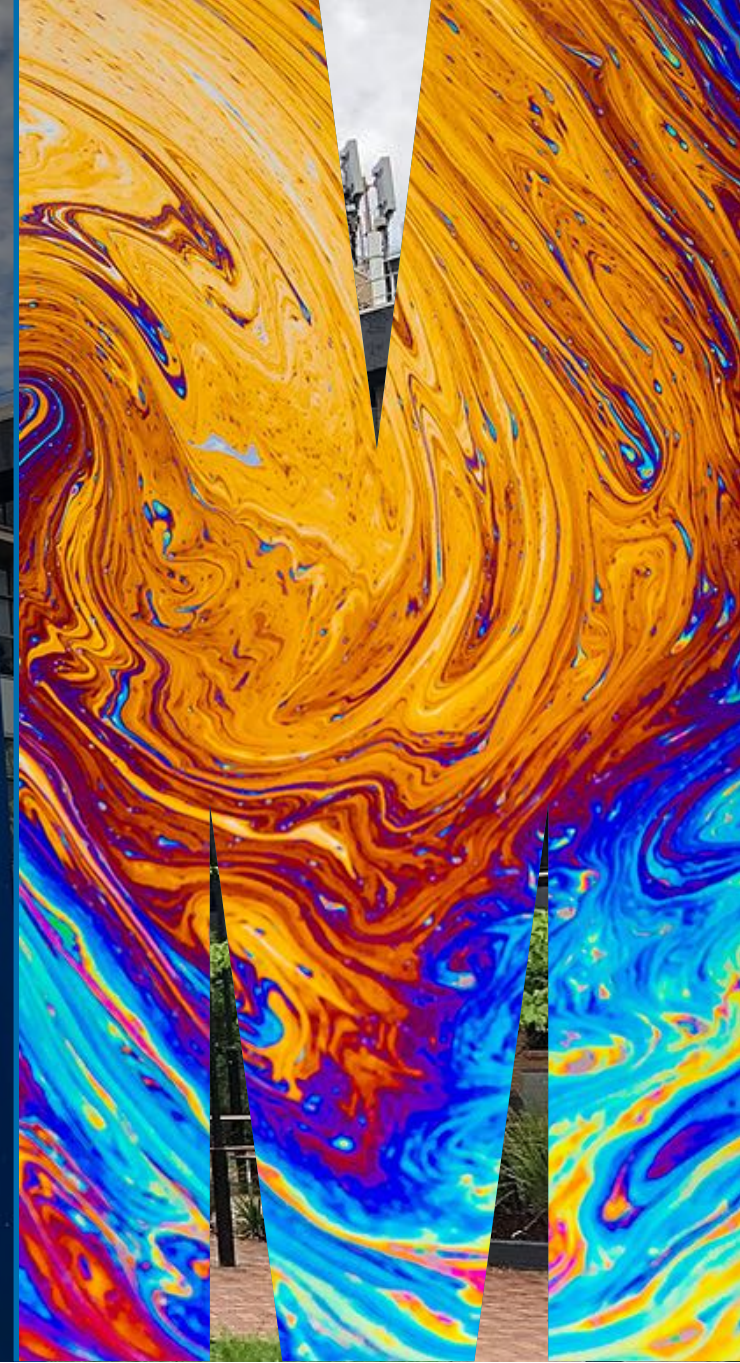
Week of introduction

Lecturer: *Nicholas Tierney*

Department of Econometrics and Business Statistics

✉ [ETC1010.Clayton-x@monash.edu](mailto:ETC1010.Clayton-x@monash.edu)

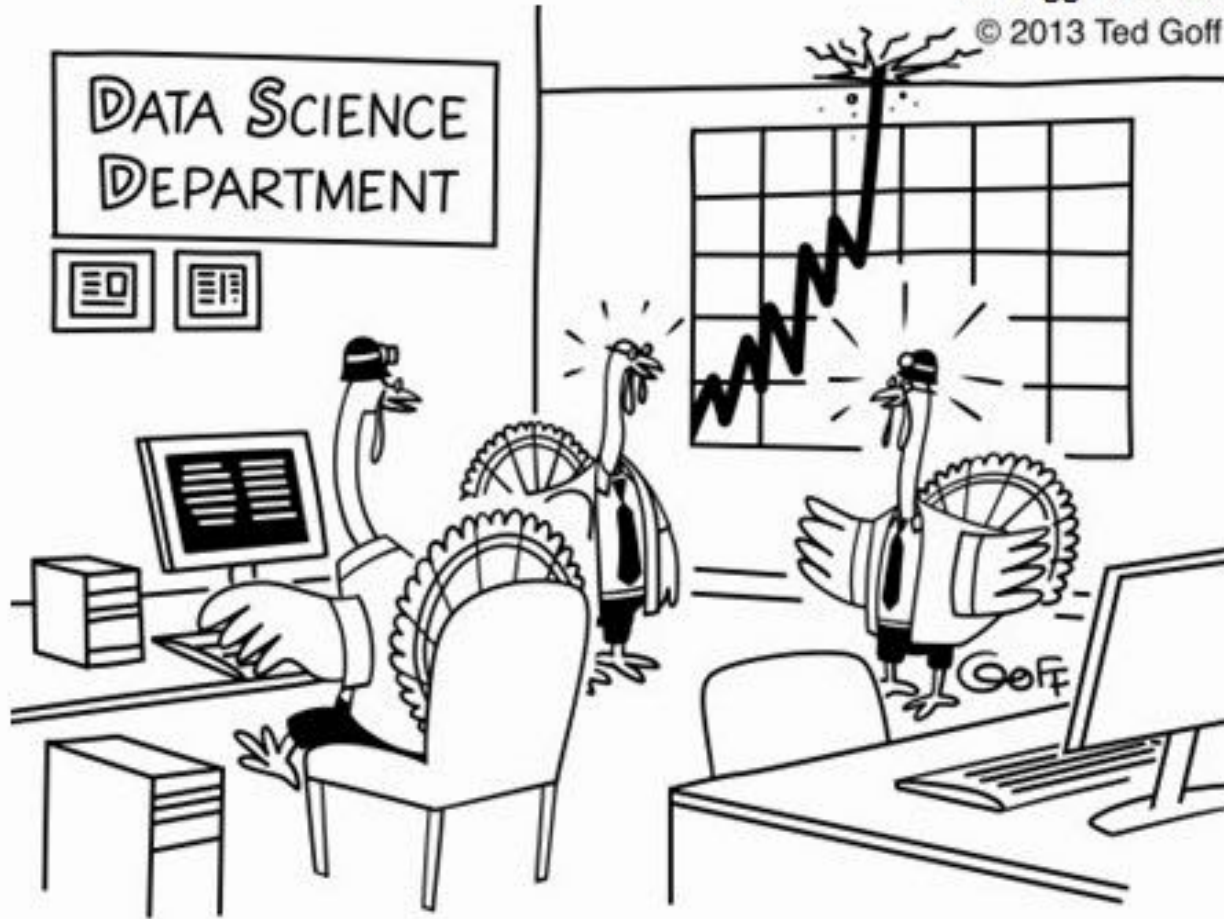
April 2020





# While the song is playing...

Draw a mental model / concept map of last lectures content on Missing Data.



**“I don’t like the look of this.  
Searches for gravy and turkey stuffing  
are going through the roof !”**

# Overview

- Different file formats
  - Audio / binary
- Web data
  - responsible scraping
  - scraping
  - JSON

# Recap on some tricky topics

- pipes %>% ("then")
- assignment <- ("gets")

# The pipe operator: %>%

- Code to tell a story about a little bunny foo foo (borrowed from <https://r4ds.had.co.nz/pipes.html>):
- Using functions for each verb: hop( ), scoop( ), bop( ).

*Little bunny Foo Foo Went hopping through the  
forest Scooping up the field mice And bopping them  
on the head*

# Approach: Intermediate steps

```
foo_foo_1 <- hop(foo_foo, through = forest)
foo_foo_2 <- scoop(foo_foo_1, up = field_mice)
foo_foo_3 <- bop(foo_foo_2, on = head)
```

- Main downside: forces you to name each intermediate element.
- Sometimes these steps form natural names. If this is the case - go ahead.
- **But many times there are not natural names**
- Adding number suffixes to make the names unique leads to problems.

# Approach: Intermediate steps

```
foo_foo_1 <- hop(foo_foo, through = forest)
foo_foo_2 <- scoop(foo_foo_1, up = field_mice)
foo_foo_3 <- bop(foo_foo_2, on = head)
```

- Code is cluttered with unimportant names
- Suffix has to be carefully incremented on each line.
- I've done this!
- 99% of the time I miss a number somewhere, and there goes my evening ... debugging my code.



# Another Approach: Overwrite the original

```
foo_foo <- hop(foo_foo, through = forest)
foo_foo <- scoop(foo_foo, up = field_mice)
foo_foo <- bop(foo_foo, on = head)
```

- Overwrite originals instead of creating intermediate objects
- Less typing (and less thinking). Less likely to make mistakes?
- **Painful debugging:** need to re-run the code from the top.
- Repitition of object - (foo\_foo written 6 times!) Obscures what changes.

# (Yet) Another approach: function composition

```
bop(  
  scoop(  
    hop(foo_foo, through = forest),  
    up = field_mice  
  ),  
  on = head  
)
```

- You need to read inside-out, and right-to-left.
- Arguments are spread far apart
- Harder to read

# Pipe %>% can help!

$f(x)$

$g(f(x))$

$h(g(f(x)))$

$x \%>\% f()$

$x \%>\% f() \%>\% g()$

$x \%>\% f() \%>\% g() \%>\%$   
 $h()$

# Solution: Use the pipe - %>%

```
foo_foo %>%  
  hop(through = forest) %>%  
  scoop(up = field_mice) %>%  
  bop(on = head)
```

- focusses on verbs, not nouns.
- Can be read as a series of function compositions like actions.

■ *Foo Foo hops, then scoops, then bops.*

- read more at: <https://r4ds.had.co.nz/pipes.html>

# Assignment < -

"gets"



# Assignment

We can perform calculations in R:

```
1 + 1
```

```
read_csv("data.csv")
```

# Assignment

But what if we want to use that information later?

```
1 + 1
```

```
read_csv("data.csv")
```

# Assignment

We can assign these things to an object using `<-`

This reads as "gets".

```
x <- 1 + 1  
my_data <- read_csv("data.csv")
```

- `x` 'gets' `1+1`
- `my_data` 'gets' the output of `read_csv...`

# Assignment

Then we can use those things in other calculations

```
x <- 1 + 1
my_data <- read_csv("data.csv")

x * x

my_data %>%
  select(age, height, weight) %>%
  mutate(bmi = weight / height^2)
```

# Take 3 minutes to discuss these two concepts with your breakout room

- What are pipes `%>%`
- What is assignment? `<-`

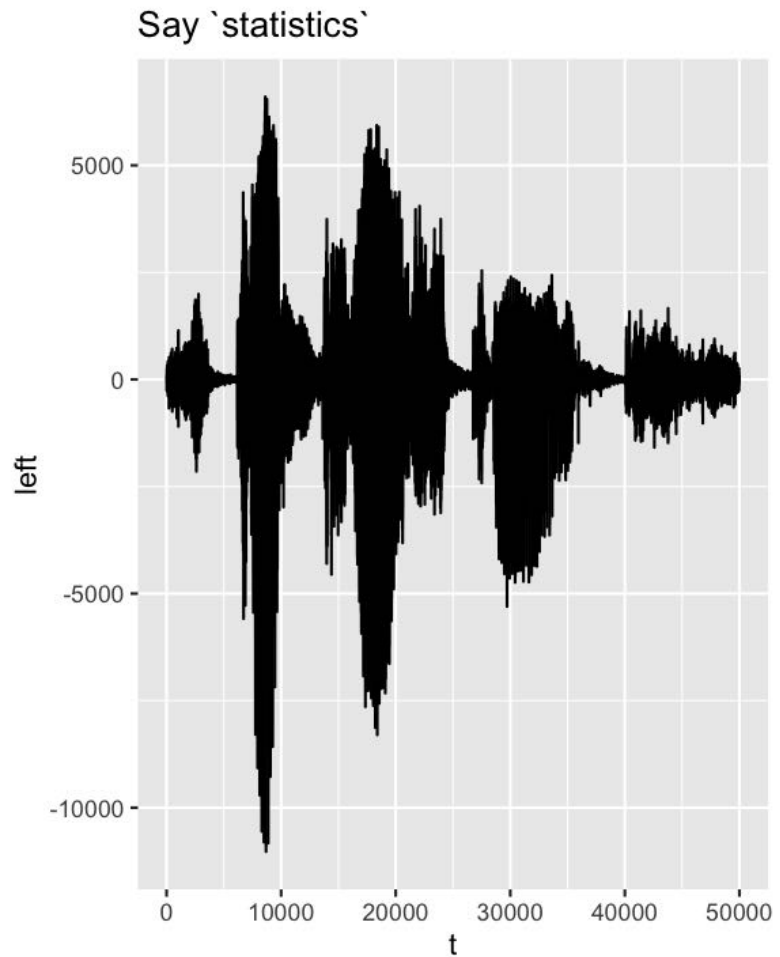
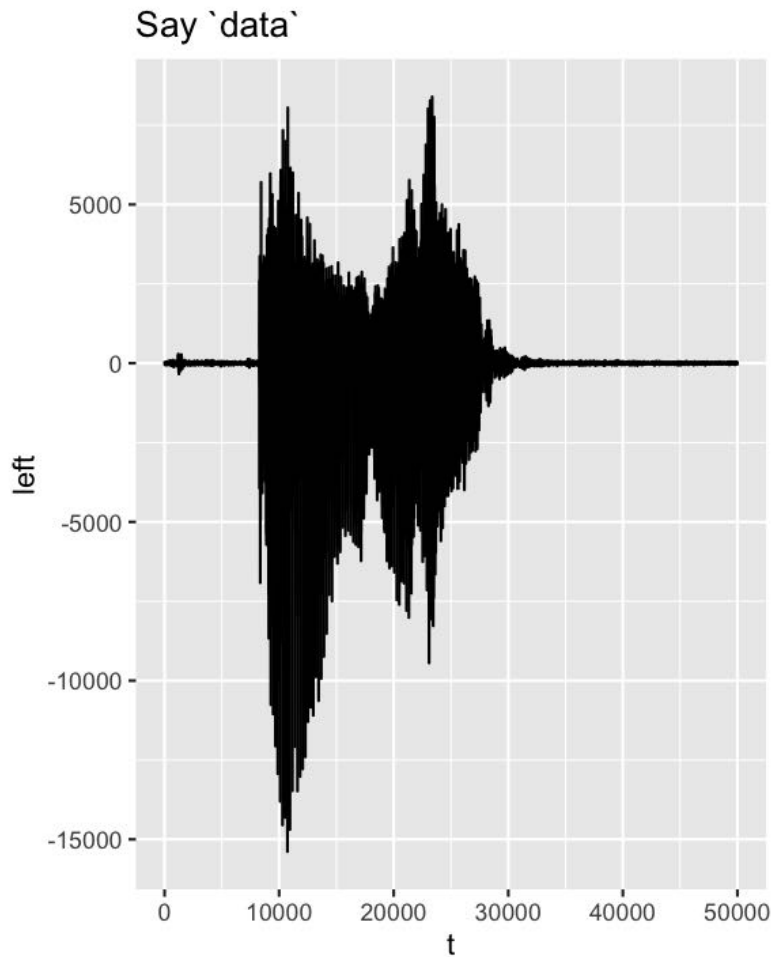


# The many shapes and sizes of data

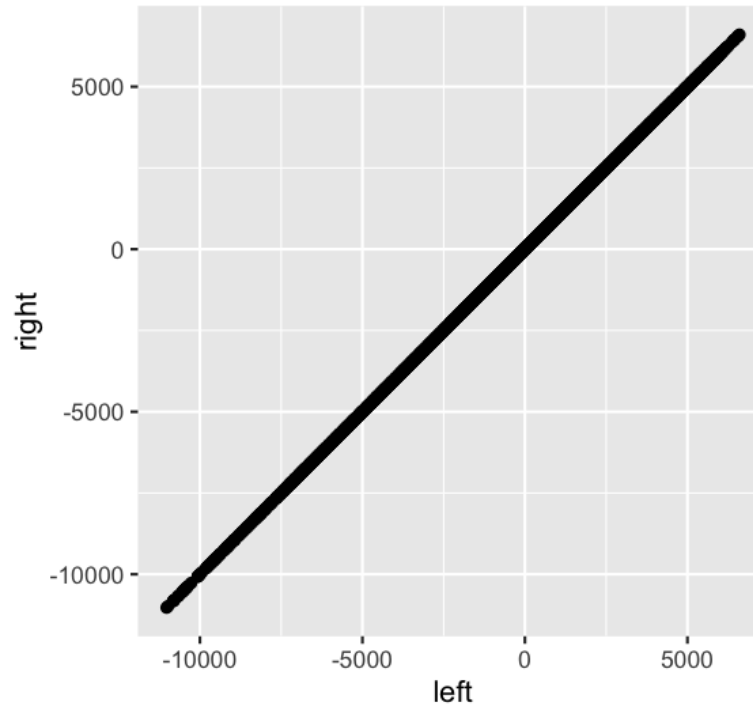
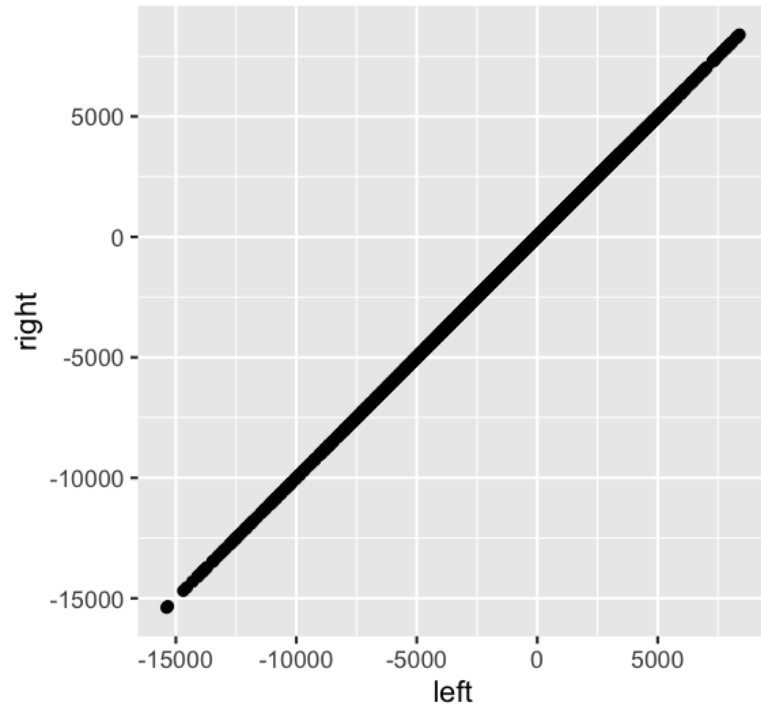
# Data as an audio file

```
## # A tibble: 50,001 x 3
##       t left right
##   <int> <int> <int>
## 1     1     28     29
## 2     2     27     28
## 3     3     26     24
## 4     4     24     27
## 5     5     22     18
## 6     6     15     19
## 7     7     15     13
## 8     8     12     13
## 9     9     15     16
## 10    10     18     16
## # ... with 49,991 more rows
```

# Plotting audio data?



# Compare left and right channels



# Compute statistics

```
df_wavs %>%  
  filter(channel == "left") %>%  
  group_by(word) %>%  
  summarise(  
    m = mean(value),  
    s = sd(value),  
    mx = max(value),  
    mn = min(value)  
  ) %>%  
  as_table()
```

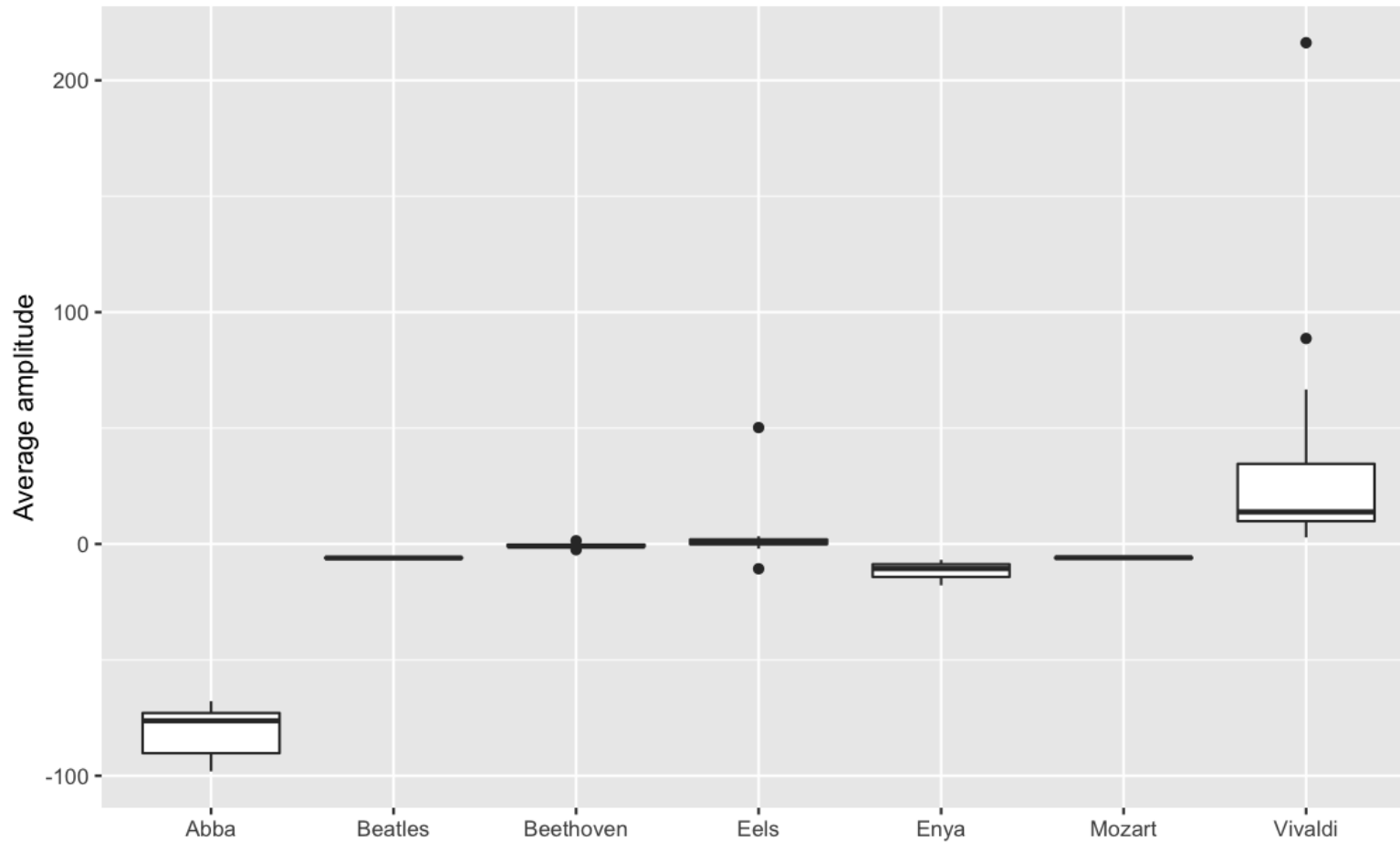
<b>word</b>	<b>m</b>	<b>s</b>	<b>mx</b>	<b>mn</b>
data	0.004	1602.577	8393	-15386
word	0.009	1506.626	6601	-11026



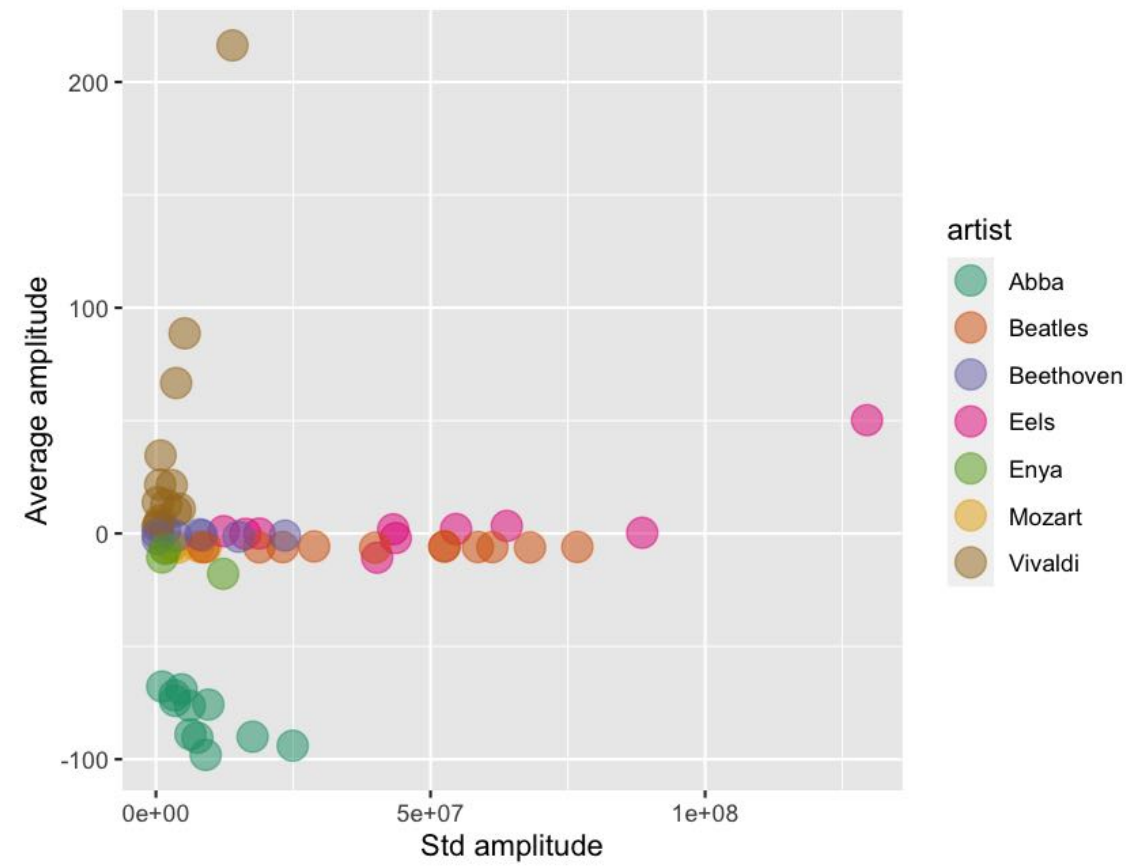
# Di's music

##	artist	type	lvar	lave	lmax	lfene
## Dancing Queen	Abba	Rock	17600755.6	-90.0068667	29921	105.9209
## Knowing Me	Abba	Rock	9543020.9	-75.7667187	27626	102.8361
## Take a Chance	Abba	Rock	9049481.5	-98.0629244	26372	102.3248
## Mamma Mia	Abba	Rock	7557437.3	-90.4710616	28898	101.6164
## Lay All You	Abba	Rock	6282285.6	-88.9526309	27940	100.3007
## Super Trouper	Abba	Rock	4665866.7	-69.0208450	25531	100.2484
## I Have A Dream	Abba	Rock	3369670.3	-71.6828788	14699	104.5968
## The Winner	Abba	Rock	1135862.0	-67.8190467	8928	104.3492
## Money	Abba	Rock	6146942.6	-76.2807500	22962	102.2406
## SOS	Abba	Rock	3482881.8	-74.1300038	15517	104.3624
## V1	Vivaldi	Classical	3677296.2	66.6530970	24229	99.2524
## V2	Vivaldi	Classical	771491.8	21.6560941	6936	104.3673
## V3	Vivaldi	Classical	5227573.2	88.6465556	17721	104.6125
## V4	Vivaldi	Classical	334719.1	13.8318847	4123	104.3500
## V5	Vivaldi	Classical	836904.9	34.5677377	17306	88.8282
## V6	Vivaldi	Classical	13936436.7	216.2317586	30355	104.8235
## V7	Vivaldi	Classical	3636324.3	9.8366363	21450	100.5272
## V8	Vivaldi	Classical	295397.4	2.8143227	2985	108.2871
## V9	Vivaldi	Classical	4335879.0	10.9015767	22271	101.3288

# Plot Di's music



# Plot Di's Music



Abba is just different from everyone else!

# Question time:

- "How does data appear different than statistics in the time series?"
- "What format is the data in an audio file?"
- "How is Abba different from the other music clips?",

01 : 30

# Why look at audio data?

- Data comes in many shapes and sizes
- Audio data can be transformed ("rectangled") into a data.frame
- Another type of data is data on the web.
- Extracting data from websites is called "web scraping".

# 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.

# Scraping the web: what? why?

1. Screen scraping: extract data from source code of website, with html parser (easy) or regular expression matching (less easy).
  2. Web APIs (application programming interface): website offers a set of structured http requests that return JSON or XML files.
- Why R? It includes all tools necessary to do web scraping, familiarity, direct analysis of data... But python, perl, java are also efficient tools.

# Web Scraping with `rvest` and `polite`



# Hypertext Markup Language

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>
```

# What if we want to extract parts of this text out?

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

- `read_html()`: read HTML in (like `read_csv` and co!)
- `html_nodes()`: select specified nodes from the HTML document using CSS selectors.

# Let's read it in with read\_html

```
example <- read_html(here::here("slides/data/example.html"))
example
## {html_document}
## <html>
## [1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">\n<t
## [2] <body>\n      <p align="center">Hello world!</p>\n    </body>
```

- We have two parts - head and body - which makes sense:

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

# Now let's get the title

```
example %>%  
  html_nodes("title")  
## {xml_nodeset (1)}  
## [1] <title>This is a title</title>  
  
<html>  
  <head>  
    <title>This is a title</title>  
  </head>  
  <body>  
    <p align="center">Hello world!</p>  
  </body>  
</html>
```

# Now let's get the paragraph text

```
example %>%  
  html_nodes("p")  
## {xml_nodeset (1)}  
## [1] <p align="center">Hello world!</p>  
  
<html>  
  <head>  
    <title>This is a title</title>  
  </head>  
  <body>  
    <p align="center">Hello world!</p>  
  </body>  
</html>
```

# Rough summary

- `read_html` - read in a html file
- `html_nodes` - select the parts of the html file we want to look at
  - This requires knowing about the website structure
  - But it turns out website are much...much more complicated than out little example file

# rvest + polite:

Simplify processing and manipulating HTML data

- `bow()` - check if the data can be scraped appropriately
- `scrape()` - scrape website data (with nice defaults)
- `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.

# SelectorGadget: css selectors

- Using a tool called selector gadget to **help** identify the html elements of interest
- Does this by constructing a css selector which can be used to subset the html document.



# SelectorGadget: css selectors

Selector	Example	Description
element	p	Select all <p> elements
element element	div p	Select all <p> elements inside a <div> element
element>element	div > p	Select all <p> elements with <div> as a parent
.class	.title	Select all elements with class="title"
#id	.name	Select all elements with id="name"
[attribute]	[class]	Select all elements with a class attribute
[attribute=value]	[class=title]	Select all elements with class="title"

# SelectorGadget

- SelectorGadget: Open source tool that eases CSS selector generation and discovery
- Install the [Chrome Extension](#)
- 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

Take a look at the source code, look for the tag `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. <a href="#">The Shawshank Redemption (1994)</a>	★ 9.2	☆ +
 2. <a href="#">The Godfather (1972)</a>	★ 9.2	☆ +
 3. <a href="#">The Godfather: Part II (1974)</a>	★ 9.0	☆ +

# First check to make sure you're allowed!

```
# install.packages("remotes")
# remotes::install_github("dmi3kno/polite")
library(polite)
bow("http://www.imdb.com")
## <polite session> http://www.imdb.com
##   User-agent: polite R package - https://github.com/dmi3kno/polite
##   robots.txt: 26 rules are defined for 1 bots
##   Crawl delay: 5 sec
##   The path is scrapable for this user-agent

bow("http://www.facebook.com")
## <polite session> http://www.facebook.com
##   User-agent: polite R package - https://github.com/dmi3kno/polite
##   robots.txt: 438 rules are defined for 20 bots
##   Crawl delay: 5 sec
##   The path is not scrapable for this user-agent
```

# Join in

- Go to [rstudio.cloud](https://rstudio.cloud) → ida-exercise-5a
- If you want to use R / Rstudio on your laptop:
  - Install R + Rstudio (see [Stuart Lee's Guide](#))
  - open RStudio
  - type the following:

```
# install.packages("usethis")  
library(usethis)  
use_course("https://ida.numbat.space/exercises/5b/ida-exercise-5b.zip")
```

# Demo

Let's go to <http://www.imdb.com/chart/top>

# Bow and scrape

```
imdb_session <- bow("http://www.imdb.com/chart/top")
```

```
imdb_session
```

```
## <polite session> http://www.imdb.com/chart/top  
##   User-agent: polite R package - https://github.com/dmi3kno/polite  
##   robots.txt: 26 rules are defined for 1 bots  
##   Crawl delay: 5 sec  
##   The path is scrapable for this user-agent
```

```
imdb_data <- scrape(imdb_session)
```

```
imdb_data
```

```
## {html_document}  
## <html xmlns:og="http://ogp.me/ns#" xmlns:fb="http://www.facebook.com/2008/fbml">  
## [1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">\n<s  
## [2] <body id="styleguide-v2" class="fixed">\n           <img height="1" width="1"
```

# Select and format pieces: titles - `html_nodes()`

```
library(rvest)
imdb_data %>%
  html_nodes(".titleColumn a")
## {xml_nodeset (250)}
## [1] <a href="/title/tt0111161/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=e31d89dd-322d-46
## [2] <a href="/title/tt0068646/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=e31d89dd-322d-46
## [3] <a href="/title/tt0071562/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=e31d89dd-322d-46
## [4] <a href="/title/tt0468569/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=e31d89dd-322d-46
## [5] <a href="/title/tt0050083/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=e31d89dd-322d-46
## [6] <a href="/title/tt0108052/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=e31d89dd-322d-46
## [7] <a href="/title/tt0167260/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=e31d89dd-322d-46
## [8] <a href="/title/tt0110912/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=e31d89dd-322d-46
## [9] <a href="/title/tt0060196/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=e31d89dd-322d-46
## [10] <a href="/title/tt0120737/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=e31d89dd-322d-46
## [11] <a href="/title/tt0137523/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=e31d89dd-322d-46
## [12] <a href="/title/tt0109830/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=e31d89dd-322d-46
## [13] <a href="/title/tt1375666/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=e31d89dd-322d-46
## [14] <a href="/title/tt0080684/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=e31d89dd-322d-46
## [15] <a href="/title/tt0167261/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=e31d89dd-322d-46
```



# Select and format pieces: titles - `html_text()`

```
imdb_data %>%  
  html_nodes(".titleColumn a") %>%  
  html_text()  
## [1] "The Shawshank Redemption"  
## [2] "The Godfather"  
## [3] "The Godfather: Part II"  
## [4] "The Dark Knight"  
## [5] "12 Angry Men"  
## [6] "Schindler's List"  
## [7] "The Lord of the Rings: The Return of the King"  
## [8] "Pulp Fiction"  
## [9] "The Good, the Bad and the Ugly"  
## [10] "The Lord of the Rings: The Fellowship of the Ring"  
## [11] "Fight Club"  
## [12] "Forrest Gump"  
## [13] "Inception"  
## [14] "Star Wars: Episode V - The Empire Strikes Back"  
## [15] "The Lord of the Rings: The Two Towers"  
## [16] "The Matrix"
```

# Select and format pieces: save it

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

# Select and format pieces: years - nodes

```
imdb_data %>%  
  html_nodes(".secondaryInfo")  
## {xml_nodeset (250)}  
## [1] <span class="secondaryInfo">(1994)</span>  
## [2] <span class="secondaryInfo">(1972)</span>  
## [3] <span class="secondaryInfo">(1974)</span>  
## [4] <span class="secondaryInfo">(2008)</span>  
## [5] <span class="secondaryInfo">(1957)</span>  
## [6] <span class="secondaryInfo">(1993)</span>  
## [7] <span class="secondaryInfo">(2003)</span>  
## [8] <span class="secondaryInfo">(1994)</span>  
## [9] <span class="secondaryInfo">(1966)</span>  
## [10] <span class="secondaryInfo">(2001)</span>  
## [11] <span class="secondaryInfo">(1999)</span>  
## [12] <span class="secondaryInfo">(1994)</span>  
## [13] <span class="secondaryInfo">(2010)</span>  
## [14] <span class="secondaryInfo">(1980)</span>  
## [15] <span class="secondaryInfo">(2002)</span>  
## [16] <span class="secondaryInfo">(1999)</span>  
## [17] <span class="secondaryInfo">(1990)</span>
```

# Select and format pieces: years - text

```
imdb_data %>%
  html_nodes(".secondaryInfo") %>%
  html_text()

## [1] "(1994)" "(1972)" "(1974)" "(2008)" "(1957)" "(1993)" "(2003)" "(1994)" "(196
## [11] "(1999)" "(1994)" "(2010)" "(1980)" "(2002)" "(1999)" "(1990)" "(1975)" "(195
## [21] "(2002)" "(1997)" "(1991)" "(1946)" "(1977)" "(2019)" "(1998)" "(2001)" "(199
## [31] "(1994)" "(1995)" "(1962)" "(1994)" "(1998)" "(2002)" "(1991)" "(1985)" "(193
## [41] "(2000)" "(1931)" "(2011)" "(2006)" "(2014)" "(2006)" "(1968)" "(1988)" "(194
## [51] "(1988)" "(1954)" "(1979)" "(1979)" "(2000)" "(1981)" "(1940)" "(2006)" "(201
## [61] "(1980)" "(2018)" "(2008)" "(1950)" "(2018)" "(1997)" "(1964)" "(2003)" "(195
## [71] "(2012)" "(1984)" "(1986)" "(2016)" "(2017)" "(1999)" "(1995)" "(2019)" "(198
## [81] "(1995)" "(1963)" "(2007)" "(1983)" "(1984)" "(1992)" "(2009)" "(1997)" "(196
## [91] "(1958)" "(1931)" "(2016)" "(2004)" "(1941)" "(2012)" "(2018)" "(1987)" "(195
## [101] "(2000)" "(1921)" "(1948)" "(1952)" "(1983)" "(1976)" "(2001)" "(1962)" "(197
## [111] "(1927)" "(1965)" "(1952)" "(2011)" "(1944)" "(1960)" "(1962)" "(2010)" "(198
## [121] "(1997)" "(1975)" "(1995)" "(1950)" "(1988)" "(1961)" "(2005)" "(2018)" "(200
## [131] "(1985)" "(1997)" "(1959)" "(2004)" "(1985)" "(1963)" "(1950)" "(2001)" "(199
## [141] "(1988)" "(2009)" "(1980)" "(1998)" "(2013)" "(1948)" "(1961)" "(2007)" "(200
## [151] "(1925)" "(1974)" "(1954)" "(2005)" "(1957)" "(2015)" "(2007)" "(2011)" "(201
## [161] "(1980)" "(1999)" "(1982)" "(2015)" "(1939)" "(1993)" "(1982)" "(1957)" "(195
```

# Select and format pieces: years - remove-brackets

```
imdb_data %>%
  html_nodes(".secondaryInfo") %>%
  html_text() %>%
  str_remove("\\(") %>% # remove (
  str_remove("\\)") %>% # remove )
  as.numeric()
## [1] 1994 1972 1974 2008 1957 1993 2003 1994 1966 2001 1999 1994 2010 1980 2002 19
## [19] 1954 1995 2002 1997 1991 1946 1977 2019 1998 2001 1999 2014 1994 1995 1962 19
## [37] 1991 1985 1936 1960 2000 1931 2011 2006 2014 2006 1968 1988 1942 2019 1988 19
## [55] 2000 1981 1940 2006 2012 1957 1980 2018 2008 1950 2018 1997 1964 2003 1957 20
## [73] 1986 2016 2017 1999 1995 2019 1981 2009 1995 1963 2007 1983 1984 1992 2009 19
## [91] 1958 1931 2016 2004 1941 2012 2018 1987 1959 1971 2000 1921 1948 1952 1983 19
## [109] 1973 2010 1927 1965 1952 2011 1944 1960 1962 2010 1989 2009 1997 1975 1995 19
## [127] 2005 2018 2004 1992 1985 1997 1959 2004 1985 1963 1950 2001 1995 2006 1988 20
## [145] 2013 1948 1961 2007 2005 2017 1925 1974 1954 2005 1957 2015 2007 2011 2010 19
## [163] 1982 2015 1939 1993 1982 1957 1957 2003 1949 1954 1979 1996 2003 1998 2008 19
## [181] 2014 1996 2018 2019 1998 2009 1993 2014 2016 2014 2019 1966 1939 2010 1995 19
## [199] 2013 2013 1967 2015 1986 1976 1953 1975 2007 2004 1979 1986 2009 1959 2013 19
## [217] 2011 1989 1959 2006 2004 2000 1984 2015 2016 1940 1976 2019 2001 1995 1934 20
## [235] 1987 2011 1951 1984 1948 2016 2015 1984 2001 1969 1988 1992 2014 2014 1982 20
```

# Select and format pieces: years - parse\_number()

```
imdb_data %>%  
  html_nodes(".secondaryInfo") %>%  
  html_text() %>%  
  parse_number()  
## [1] 1994 1972 1974 2008 1957 1993 2003 1994 1966 2001 1999 1994 2010 1980 2002 19  
## [19] 1954 1995 2002 1997 1991 1946 1977 2019 1998 2001 1999 2014 1994 1995 1962 19  
## [37] 1991 1985 1936 1960 2000 1931 2011 2006 2014 2006 1968 1988 1942 2019 1988 19  
## [55] 2000 1981 1940 2006 2012 1957 1980 2018 2008 1950 2018 1997 1964 2003 1957 20  
## [73] 1986 2016 2017 1999 1995 2019 1981 2009 1995 1963 2007 1983 1984 1992 2009 19  
## [91] 1958 1931 2016 2004 1941 2012 2018 1987 1959 1971 2000 1921 1948 1952 1983 19  
## [109] 1973 2010 1927 1965 1952 2011 1944 1960 1962 2010 1989 2009 1997 1975 1995 19  
## [127] 2005 2018 2004 1992 1985 1997 1959 2004 1985 1963 1950 2001 1995 2006 1988 20  
## [145] 2013 1948 1961 2007 2005 2017 1925 1974 1954 2005 1957 2015 2007 2011 2010 19  
## [163] 1982 2015 1939 1993 1982 1957 1957 2003 1949 1954 1979 1996 2003 1998 2008 19  
## [181] 2014 1996 2018 2019 1998 2009 1993 2014 2016 2014 2019 1966 1939 2010 1995 19  
## [199] 2013 2013 1967 2015 1986 1976 1953 1975 2007 2004 1979 1986 2009 1959 2013 19  
## [217] 2011 1989 1959 2006 2004 2000 1984 2015 2016 1940 1976 2019 2001 1995 1934 20  
## [235] 1987 2011 1951 1984 1948 2016 2015 1984 2001 1969 1988 1992 2014 2014 1982 20
```

# Select and format pieces: years - remove-brackets

```
years <- imdb_data %>%  
  html_nodes(".secondaryInfo") %>%  
  html_text() %>%  
  str_remove("\\(") %>% # remove (  
  str_remove("\\)") %>% # remove )  
  as.numeric()
```

# Select and format pieces: scores - nodes

```
imdb_data %>%
  html_nodes(".imdbRating strong")
## {xml_nodeset (250)}
## [1] <strong title="9.2 based on 2,223,299 user ratings">9.2</strong>
## [2] <strong title="9.1 based on 1,532,071 user ratings">9.1</strong>
## [3] <strong title="9.0 based on 1,072,485 user ratings">9.0</strong>
## [4] <strong title="9.0 based on 2,197,842 user ratings">9.0</strong>
## [5] <strong title="8.9 based on 649,752 user ratings">8.9</strong>
## [6] <strong title="8.9 based on 1,157,929 user ratings">8.9</strong>
## [7] <strong title="8.9 based on 1,575,407 user ratings">8.9</strong>
## [8] <strong title="8.8 based on 1,744,136 user ratings">8.8</strong>
## [9] <strong title="8.8 based on 657,802 user ratings">8.8</strong>
## [10] <strong title="8.8 based on 1,588,133 user ratings">8.8</strong>
## [11] <strong title="8.8 based on 1,772,059 user ratings">8.8</strong>
## [12] <strong title="8.8 based on 1,715,120 user ratings">8.8</strong>
## [13] <strong title="8.7 based on 1,949,598 user ratings">8.7</strong>
## [14] <strong title="8.7 based on 1,111,637 user ratings">8.7</strong>
## [15] <strong title="8.7 based on 1,422,865 user ratings">8.7</strong>
## [16] <strong title="8.6 based on 1,597,589 user ratings">8.6</strong>
## [17] <strong title="8.6 based on 967,644 user ratings">8.6</strong>
```



# Select and format pieces: scores - text

```
imdb_data %>%
  html_nodes(".imdbRating strong") %>%
  html_text()
## [1] "9.2" "9.1" "9.0" "9.0" "8.9" "8.9" "8.9" "8.8" "8.8" "8.8" "8.8" "8.8" "8.8" "8.7"
## [16] "8.6" "8.6" "8.6" "8.6" "8.6" "8.6" "8.6" "8.6" "8.6" "8.6" "8.6" "8.6" "8.6" "8.5" "8.5"
## [31] "8.5" "8.5" "8.5" "8.5" "8.5" "8.5" "8.5" "8.5" "8.5" "8.5" "8.5" "8.5" "8.5" "8.5" "8.5"
## [46] "8.5" "8.5" "8.4" "8.4" "8.4" "8.4" "8.4" "8.4" "8.4" "8.4" "8.4" "8.4" "8.4" "8.4" "8.4"
## [61] "8.4" "8.4" "8.4" "8.4" "8.4" "8.4" "8.4" "8.4" "8.4" "8.4" "8.4" "8.3" "8.3" "8.3" "8.3"
## [76] "8.3" "8.3" "8.3" "8.3" "8.3" "8.3" "8.3" "8.3" "8.3" "8.3" "8.3" "8.3" "8.3" "8.3" "8.3"
## [91] "8.3" "8.3" "8.3" "8.3" "8.3" "8.3" "8.3" "8.3" "8.3" "8.3" "8.3" "8.2" "8.2" "8.2" "8.2"
## [106] "8.2" "8.2" "8.2" "8.2" "8.2" "8.2" "8.2" "8.2" "8.2" "8.2" "8.2" "8.2" "8.2" "8.2" "8.2"
## [121] "8.2" "8.2" "8.2" "8.2" "8.2" "8.2" "8.2" "8.2" "8.2" "8.2" "8.2" "8.2" "8.2" "8.2" "8.2"
## [136] "8.2" "8.2" "8.2" "8.2" "8.2" "8.2" "8.2" "8.2" "8.2" "8.2" "8.2" "8.2" "8.1" "8.1" "8.1"
## [151] "8.1" "8.1" "8.1" "8.1" "8.1" "8.1" "8.1" "8.1" "8.1" "8.1" "8.1" "8.1" "8.1" "8.1" "8.1"
## [166] "8.1" "8.1" "8.1" "8.1" "8.1" "8.1" "8.1" "8.1" "8.1" "8.1" "8.1" "8.1" "8.1" "8.1" "8.1"
## [181] "8.1" "8.1" "8.1" "8.1" "8.1" "8.1" "8.1" "8.1" "8.1" "8.1" "8.1" "8.1" "8.1" "8.1" "8.1"
## [196] "8.1" "8.1" "8.1" "8.1" "8.1" "8.1" "8.1" "8.1" "8.1" "8.1" "8.1" "8.1" "8.1" "8.1" "8.1"
## [211] "8.1" "8.1" "8.1" "8.1" "8.0" "8.0" "8.0" "8.0" "8.0" "8.0" "8.0" "8.0" "8.0" "8.0" "8.0"
## [226] "8.0" "8.0" "8.0" "8.0" "8.0" "8.0" "8.0" "8.0" "8.0" "8.0" "8.0" "8.0" "8.0" "8.0" "8.0"
## [241] "8.0" "8.0" "8.0" "8.0" "8.0" "8.0" "8.0" "8.0" "8.0" "8.0" "8.0" "8.0" "8.0" "8.0" "8.0"
```

# Select and format pieces: scores - as-numeric

```
imdb_data %>%
  html_nodes(".imdbRating strong") %>%
  html_text() %>%
  as.numeric()
## [1] 9.2 9.1 9.0 9.0 8.9 8.9 8.9 8.8 8.8 8.8 8.8 8.8 8.8 8.7 8.7 8.7 8.6 8.6 8.6 8.6 8
## [23] 8.6 8.6 8.6 8.6 8.5 8.5 8.5 8.5 8.5 8.5 8.5 8.5 8.5 8.5 8.5 8.5 8.5 8.5 8.5 8.5 8
## [45] 8.5 8.5 8.5 8.4 8.4 8.4 8.4 8.4 8.4 8.4 8.4 8.4 8.4 8.4 8.4 8.4 8.4 8.4 8.4 8.4 8
## [67] 8.4 8.4 8.4 8.3 8.3 8.3 8.3 8.3 8.3 8.3 8.3 8.3 8.3 8.3 8.3 8.3 8.3 8.3 8.3 8.3 8
## [89] 8.3 8.3 8.3 8.3 8.3 8.3 8.3 8.3 8.3 8.3 8.3 8.3 8.2 8.2 8.2 8.2 8.2 8.2 8.2 8.2 8
## [111] 8.2 8.2 8.2 8.2 8.2 8.2 8.2 8.2 8.2 8.2 8.2 8.2 8.2 8.2 8.2 8.2 8.2 8.2 8.2 8.2 8
## [133] 8.2 8.2 8.2 8.2 8.2 8.2 8.2 8.2 8.2 8.2 8.2 8.2 8.2 8.2 8.1 8.1 8.1 8.1 8.1 8.1 8
## [155] 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8
## [177] 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8
## [199] 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.0 8.0 8.0 8
## [221] 8.0 8.0 8.0 8.0 8.0 8.0 8.0 8.0 8.0 8.0 8.0 8.0 8.0 8.0 8.0 8.0 8.0 8.0 8.0 8
## [243] 8.0 8.0 8.0 8.0 8.0 8.0 8.0 8.0
```

# Select and format pieces: scores - save

```
scores <- imdb_data %>%  
  html_nodes(".imdbRating strong") %>%  
  html_text() %>%  
  as.numeric()
```

# Select and format pieces: put it all together

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

```
imdb_top_250
```

```
## # A tibble: 250 x 3
```

```
##   title                                year score  
##   <chr>                                <dbl> <dbl>  
## 1 The Shawshank Redemption             1994  9.2  
## 2 The Godfather                        1972  9.1  
## 3 The Godfather: Part II               1974   9  
## 4 The Dark Knight                      2008   9  
## 5 12 Angry Men                         1957  8.9  
## 6 Schindler's List                     1993  8.9  
## 7 The Lord of the Rings: The Return of the King 2003  8.9  
## 8 Pulp Fiction                         1994  8.8  
## 9 The Good, the Bad and the Ugly       1966  8.8  
## 10 The Lord of the Rings: The Fellowship of the Ring 2001  8.8  
## # ... with 240 more rows
```

<b>title</b>	<b>year</b>	<b>score</b>
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	8.8
The Good, the Bad and the Ugly	1966	8.8
...	...	...

# Aside: Yet another approach - pull the table with `html_table()`

- requires notation we haven't used yet (e.g., what is `[[ ]]`)
- requires substantial text cleaning
- If there is time we can cover this at the end of class

```
imdb_table <- html_table(imdb_data)

glimpse(imdb_table[[1]])
## Rows: 250
## Columns: 5
## $ ` ` <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,
## $ `Rank & Title` <chr> "1.\n      The Shawshank Redemption\n      (1994)", "2.\n
## $ `IMDb Rating` <dbl> 9.2, 9.1, 9.0, 9.0, 8.9, 8.9, 8.9, 8.8, 8.8, 8.8, 8.8, 8.8,
## $ `Your Rating` <chr> "12345678910\n      \n      \n      \n
## $ ` ` <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,
```

# 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)
## Rows: 250
## Columns: 3
## $ title <chr> "The Shawshank Redemption", "The Godfather", "The Godfather: Part II"
## $ year <dbl> 1994, 1972, 1974, 2008, 1957, 1993, 2003, 1994, 1966, 2001, 1999, 199
## $ score <dbl> 9.2, 9.1, 9.0, 9.0, 8.9, 8.9, 8.9, 8.8, 8.8, 8.8, 8.8, 8.7, 8.7,
```

# Clean up / enhance

- Add a variable for rank

```
imdb_top_250 %>%
  mutate(
    rank = 1:nrow(imdb_top_250)
  )
## # A tibble: 250 x 4
##   title                                year score  rank
##   <chr>                                <dbl> <dbl> <int>
## 1 The Shawshank Redemption             1994   9.2    1
## 2 The Godfather                        1972   9.1    2
## 3 The Godfather: Part II               1974    9     3
## 4 The Dark Knight                     2008    9     4
## 5 12 Angry Men                        1957   8.9    5
## 6 Schindler's List                    1993   8.9    6
## 7 The Lord of the Rings: The Return of the King 2003   8.9    7
## 8 Pulp Fiction                        1994   8.8    8
## 9 The Good, the Bad and the Ugly      1966   8.8    9
## 10 The Lord of the Rings: The Fellowship of the Ring 2001   8.8   10
## # ... with 240 more rows
```



<b>title</b>	<b>year</b>	<b>score</b>	<b>rank</b>
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	8.8	8
The Good, the Bad and the Ugly	1966	8.8	9
...	...	...	...

# Your Turn: Which 1995 movies made the list?

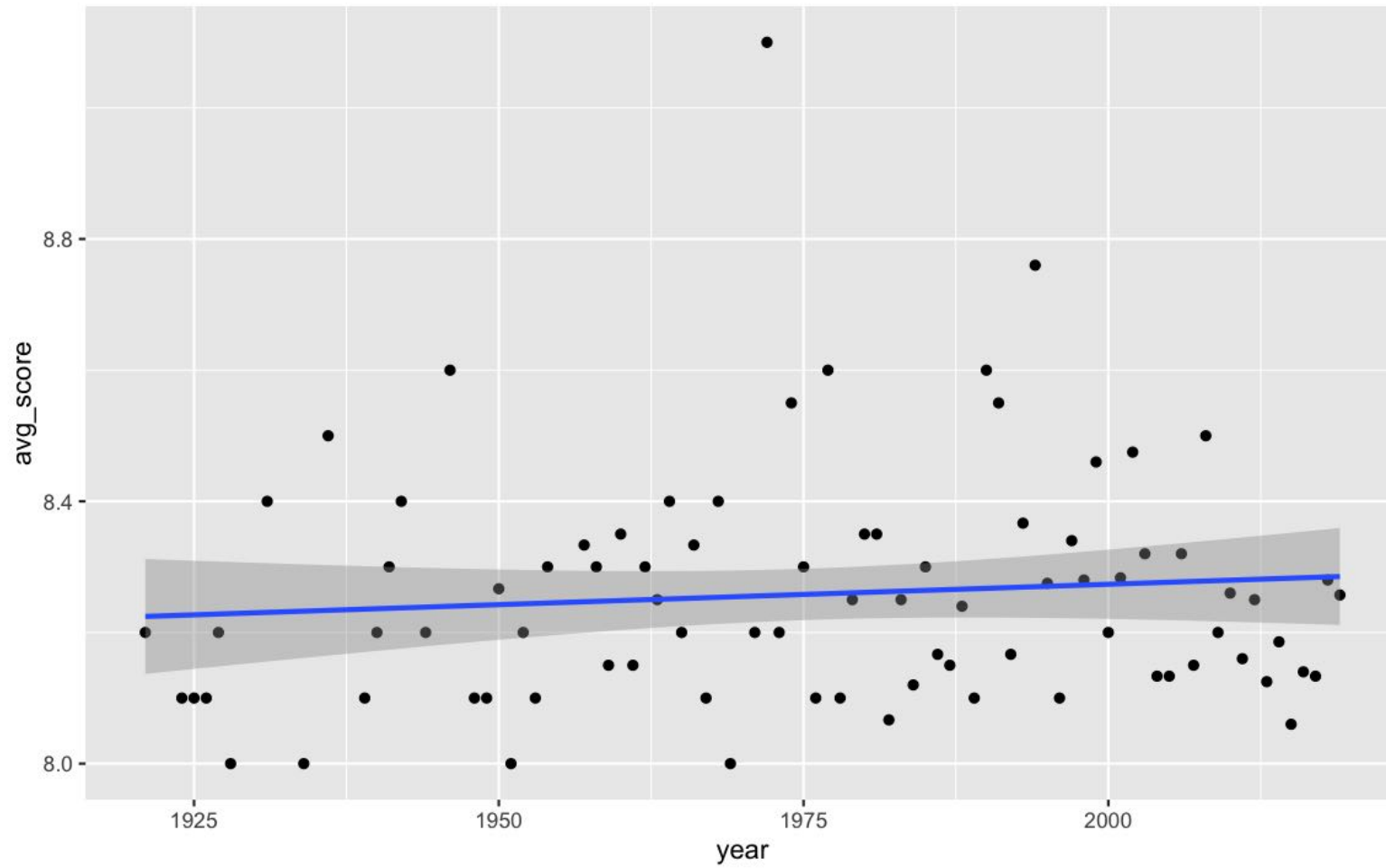
```
imdb_top_250 %>%  
  filter(year == 1995)  
## # A tibble: 8 x 3  
##   title          year score  
##   <chr>         <dbl> <dbl>  
## 1 Se7en          1995  8.6  
## 2 The Usual Suspects 1995  8.5  
## 3 Braveheart      1995  8.3  
## 4 Toy Story       1995  8.3  
## 5 Heat            1995  8.2  
## 6 Casino          1995  8.2  
## 7 Before Sunrise   1995  8.1  
## 8 La haine        1995  8
```

# Your turn: 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  2000     7  
## 3  2014     7  
## 4  2019     7  
## 5  1957     6
```

# Your Turn: Visualize top 250 yearly mean score over time

```
imdb_top_250 %>%  
  group_by(year) %>%  
  summarise(avg_score = mean(score)) %>%  
  ggplot(aes(y = avg_score, x = year)) +  
    geom_point() +  
    geom_smooth(method = "lm") +  
    xlab("year")
```



# Other common formats: JSON

- JavaScript Object Notation (JSON).
- A language-independent data format, and supplants extensible markup language (XML).
- Data are sometimes stored as JSON, which requires special unpacking

# Unpacking JSON: Example JSON from [jsonlite](#)

```
library(jsonlite)
json_mario <- '[
  {
    "Name": "Mario",
    "Age": 32,
    "Occupation": "Plumber"
  },
  {
    "Name": "Peach",
    "Age": 21,
    "Occupation": "Princess"
  },
  {}
  {
    "Name": "Bowser",
    "Occupation": "Koopa"
  }
]'
```

```
mydf <- fromJSON(json_mario)
mydf
##      Name Age Occupation
## 1  Mario  32   Plumber
## 2  Peach  21   Princess
## 3   <NA>  NA     <NA>
## 4  Bowser  NA     Koopa
```

# Potential challenges with web scraping

- Unreliable formatting at the source
- Data broken into many pages
- Data arriving in multiple excel file formats
- ... We will come back to this when we learn about functions next week.

*Compare the display of information at [gumtree melbourne](#) to the list on the IMDB top 250 list. What challenges can you foresee in scraping a list of the available apartments? ]*



# Further exploring

People write R packages to access online data! Check out:

- [crinfo by Sayani Gupta and Rob Hyndman](#)
- [rwalkr by Earo Wang](#)
- [fitzRoy for AFL data](#)
- [Top 40 lists of R packages by Joe Rickert](#) - they usually include a "data" section.