Making good plots

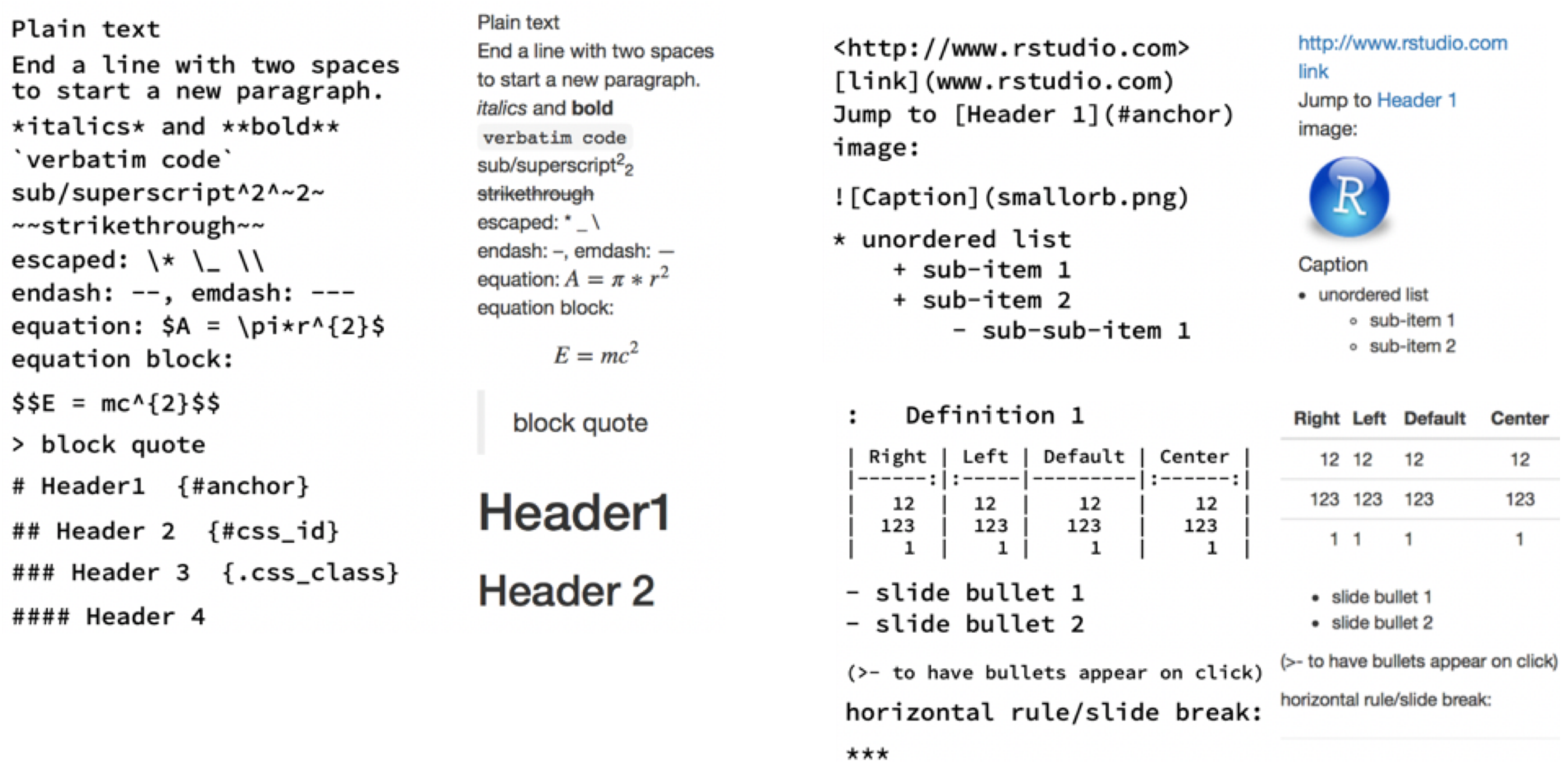
Modern Research Methods

24 September 2021

Working with Rmarkdown

- Can do lots of formatting with markdown/Rmarkdown - check out the resources on the course website

Rmarkdown – formatting text



```
Plain text
End a line with two spaces
to start a new paragraph.
*italics* and **bold**
`verbatim code`
sub/superscript^2^~2~
~~strikethrough~~
escaped: \* \_ \\
endash: --, emdash: ---
equation: $A = \pi*r^{2}$
equation block:

$$E = mc^{2}$$

> block quote

# Header1  {#anchor}

## Header 2  {#css_id}

### Header 3  {.css_class}

#### Header 4
```

Plain text
End a line with two spaces
to start a new paragraph.
italics and **bold**
`verbatim code`
sub/superscript$^{2}{}_{2}$
~~strikethrough~~
escaped: * _ \
endash: –, emdash: —
equation: $A = \pi * r^2$
equation block:

$$E = mc^2$$

> block quote

Header1

Header 2

```
<http://www.rstudio.com>
[link](www.rstudio.com)
Jump to [Header 1](#anchor)
image:

![Caption](smallorb.png)

* unordered list
    + sub-item 1
    + sub-item 2
        - sub-sub-item 1
```

http://www.rstudio.com
link
Jump to Header 1
image:

Caption

- unordered list
  - sub-item 1
  - sub-item 2

```
:   Definition 1

| Right | Left | Default | Center |
|------:|:-----|---------|:------:|
|    12 |   12 |      12 |     12 |
|   123 |  123 |     123 |    123 |
|     1 |    1 |       1 |      1 |

- slide bullet 1
- slide bullet 2

(>- to have bullets appear on click)

horizontal rule/slide break:

***
```

Right	Left	Default	Center
12	12	12	12
123	123	123	123
1	1	1	1

- slide bullet 1
- slide bullet 2

(>- to have bullets appear on click)

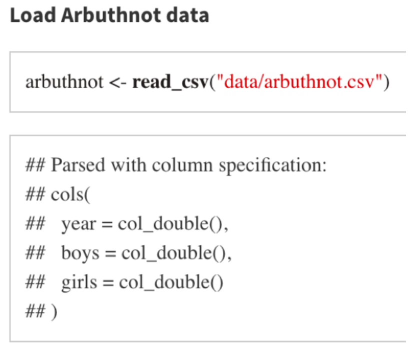
horizontal rule/slide break:

Rmarkdown – setting “options” for R chunks

Useful options:

- **message** – display messages in knitted document?
- **warning** – display warnings?
- **include** – include chunk after running?
- **eval** – run code?

~~~
**Load Arbuthnot data**
```{r, message = TRUE}
arbuthnot <- read_csv("data/arbuthnot.csv")
~~~
```

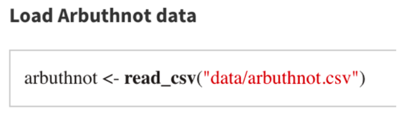


Load Arbuthnot data

```
arbuthnot <- read_csv("data/arbuthnot.csv")
```

```
## Parsed with column specification:
## cols(
##   year = col_double(),
##   boys = col_double(),
##   girls = col_double()
## )
```

```
**Load Arbuthnot data**
```{r, message = FALSE}
arbuthnot <- read_csv("data/arbuthnot.csv")
```
```



Load Arbuthnot data

```
arbuthnot <- read_csv("data/arbuthnot.csv")
```

Can specify ”global” option (all chunks) - **knitr::opts_chunk$set()**

```
knitr::opts_chunk$set(eval = TRUE, message = FALSE)
```

Useful to label chunks

```
```{r calculate_sum}
365 + 112
```
```

- Useful for debugging!
```

- Make sure you look at your .html after you knit. Does it look as you expected? If not, go back to .Rmd.
- No need to use `print()` function in .Rmd - will print output automatically.
- You can change size of plot output in .Rmd by specificy `fig.width` and `fig.height` in the relevant R chunk. In general, aim for the "plot" plot of your plot (i.e. excluding the legend) to be roughly square (or slightly wider than square).

```
{r CHUNKNAME, fig.width = 4.5, fig.height = 4}
```

- Expect you to write answer to questions on assignments in plain text along with your code.
- Also, expect you to give appropriate labels to plots.

Notes on style

Style

- Why does style matter?
- Style doesn't matter to the computer, but it does matter to **humans** who produce, intepret and modify code.
- Having a code specific, consistent code style makes your own code easier to understand and debug, and it helps others do the same.
- In this class, **variable names** in data frames should be all lower case and descriptive. Separate multiple words with an underscore (_).
- BAD: `NEWVARIABLE`, `thing`, `LIFEexpectancy`, `Time`
- GOOD: `num_countries`, `age_years`, `life_expectancy`, `log_reaction_time_seconds`
- In this class, if you can use the pipe, **always use the pipe** (unless there's only a single function)

Line breaks

In the tidyverse, you should think of each **line** as doing **one** thing.

Like instructions in a recipe:

Data frame goes on own line, then each function (verb) on its own line after that (indent after first).

GREAT:

```
gapminder %>%
  group_by(country) %>%
  summarize(num_countries = n()) %>%
  mutate(num_countries_round = round(num_countries))
```

BAD:

```
gapminder %>% group_by(country)  %>% summarize(num_countries = n()) %>%
  mutate(num_countries_round = round(num_countries))
```

Same for ggplot. Imagine your plot is a house and you're building it brick by brick.

Each "brick" of the plot goes on its own line.

Each layer of your plot goes on its own line.

GREAT:

```
ggplot(data = gapminder, mapping = aes(x = gdp_percap,
                                       y = life_exp)) +
  geom_point() +
  geom_smooth(method = "lm") +
  scale_x_log10() +
  ylab("Life Expectancy (Years)") +
  xlab("GDP Per Capita") +
  theme_classic(base_size = 16)
```

BAD:

```
ggplot(data = gapminder, mapping = aes(x = gdp_percap,
                                       y = life_exp)) + geom_point() +
  geom_smooth(method = "lm") + scale_x_log10() + ylab("Life Expectancy (Years)") +
  xlab("GDP Per Capita") +theme_classic(base_size = 16)
```

Points of confusion

%in% vs. %>%

Even though these symbols are made up of three characters, you should think of them as a single symbol.

Despite their apparent similarity, these functions aren't really related to each other.

`%in%` checks whether something is a member of a set.

```
4 %in% c(1,2,3,4)
```

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

```
5 %in% c(1,2,3,4)
```

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

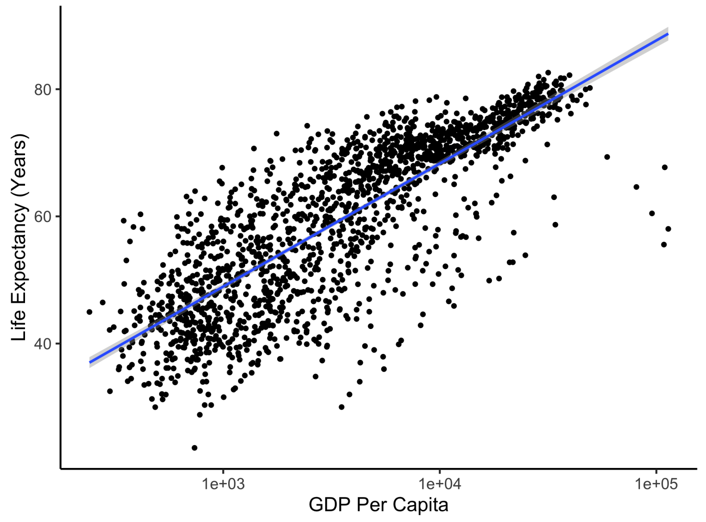
`%>%` ("the pipe") sends the output of one function to another function.

```
gapminder %>%
  group_by(country) %>%
  summarize(num_countries = n())
```

The scope of aes()

Remember this plot?

```
ggplot(data = gapminder, mapping = aes(x = gdp_percap,
                                       y = life_exp)) +
  geom_point() +
  geom_smooth(method = "lm") +
  scale_x_log10() +
  ylab("Life Expectancy (Years)") +
  xlab("GDP Per Capita") +
  theme_classic(base_size = 16)
```

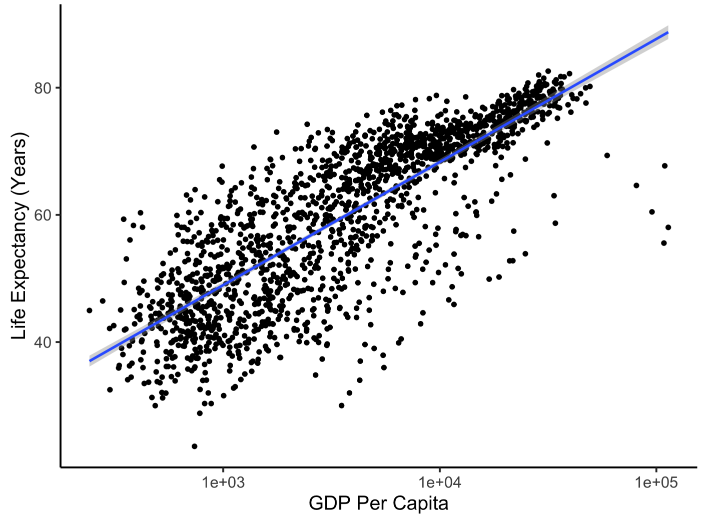


Another way to write this is by putting the aesthetics in the geom functions themselves

```
ggplot(data = gapminder) +
  geom_point(mapping = aes(x = gdp_percap,y = life_exp)) +
  geom_smooth(mapping = aes(x = gdp_percap,  y = life_exp), method = "lm") +
  scale_x_log10() +
  ylab("Life Expectancy (Years)") +
  xlab("GDP Per Capita") +
  theme_classic(base_size = 16)
```

But notice because `geom_point()` and `geom_smooth()` require both `x` and `y` aesthetics we have to include the mappings in both.

Mappings put in the `ggplot()` function apply to all geoms.



A common error: Forgetting a pipe

```
gapminder %>%
  group_by(country)
  summarize(num_countries = n()) %>%
  mutate(num_countries_round = round(num_countries))
```

`Error: n() should only be called in a data context Call`rlang::last_error()`to see a backtrace.`

Error will depend on what exactly you're trying to do. But check this first if you get an error you don't understand!

A common error: Forgetting the +

```r
ggplot(data = gapminder, mapping = aes(x = gdp_percap,
                                       y = life_exp)) +
  geom_point() +
  geom_smooth(method = "lm")
  scale_x_log10() +
  ylab("Life Expectancy (Years)") +
  xlab("GDP Per Capita") +
  theme_classic(base_size = 16)
```

```
Error: Cannot add ggproto objects together. Did you forget to add this object to a
ggplot object?
```

A common error: Forgetting to load packages

```r
cool_data_frame <- read_csv("data/cool_data_frame.csv")
```

```
Error: object 'read_csv' not found
```

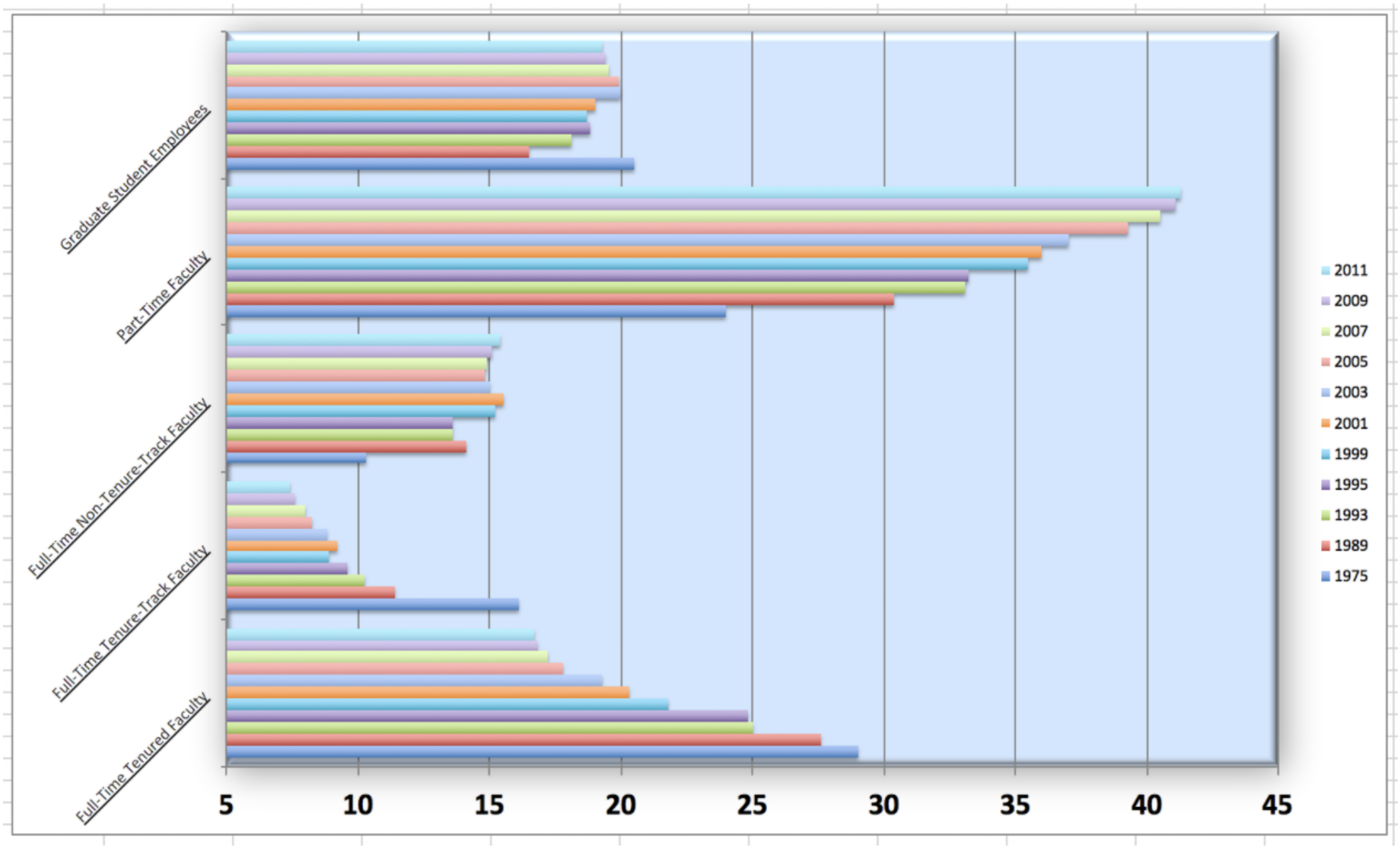
Solves the problem:

```r
library(tidyverse)
cool_data_frame <- read_csv("data/cool_data_frame.csv")
```

You have to load packages before you can use their functions!

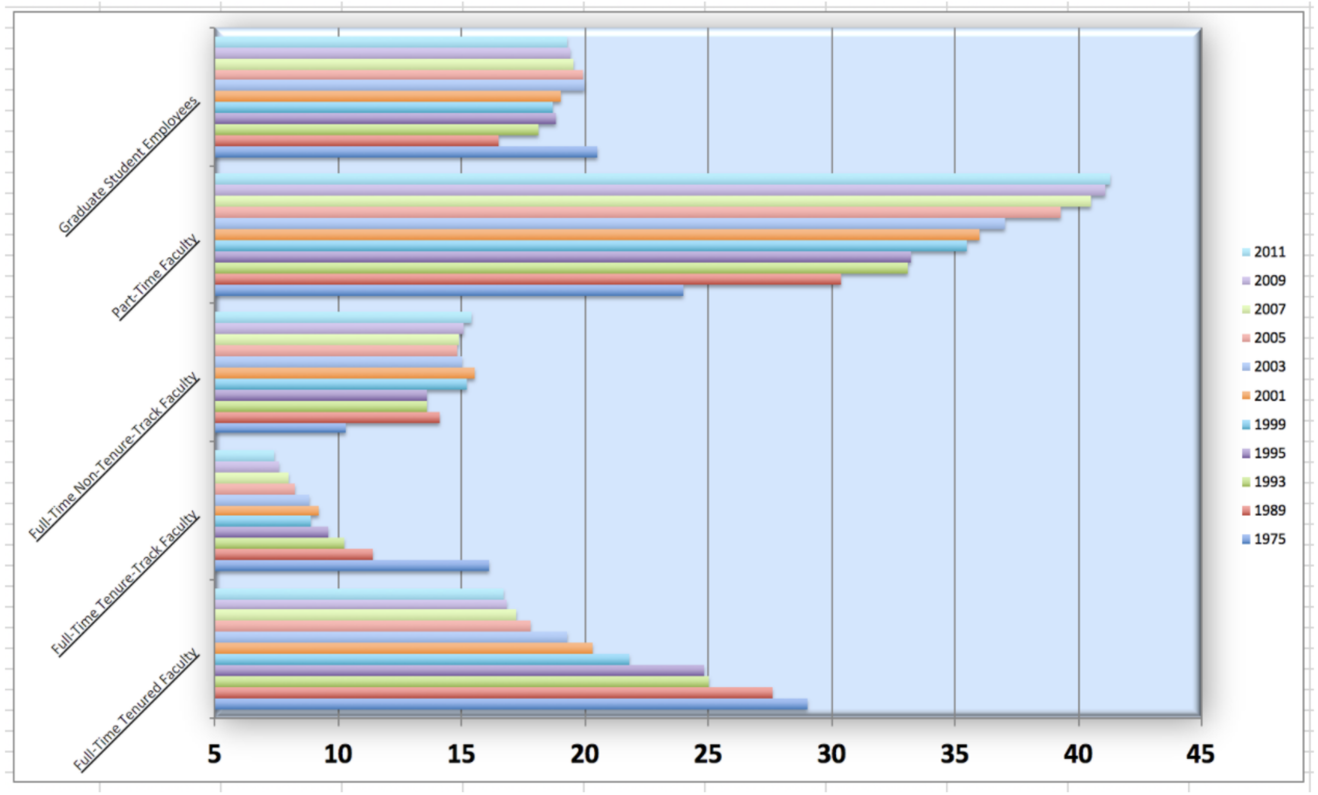
Lab today: Making a sad plot better

The American Association of University Professors (AAUP) is a nonprofit membership association of faculty and other academic professionals. A report compiled by the AAUP shows trends in instructional staff employees between 1975 and 2011, and contains an image very similar to the one given below.



Can you help them improve it? First, decide what **specific** question you want to answer with this data. Second, brainstorm how you would improve it. Then create the improved visualization and write up the changes/decisions you made as bullet points. It's ok if some of your improvements are aspirational, i.e. you don't know how to implement it, but you think it's a good idea.

DATA: "https://raw.githubusercontent.com/mllewis/cumulative-science/master/static/data/instructors.csv"



Activity adapted from Data Science in a Box.