Putting the R in Reed and in Lewis and ClaRk

Chester Ismay

GitHub: `ismayc`
Twitter: `@old_man_chester`

2017-05-25 & 2017-05-26

Bootcamp website at http://bit.ly/rbootcamp17
Slides available at http://bit.ly/rbootcamp17-slides

Table of Contents

Pre-bootcamp HW

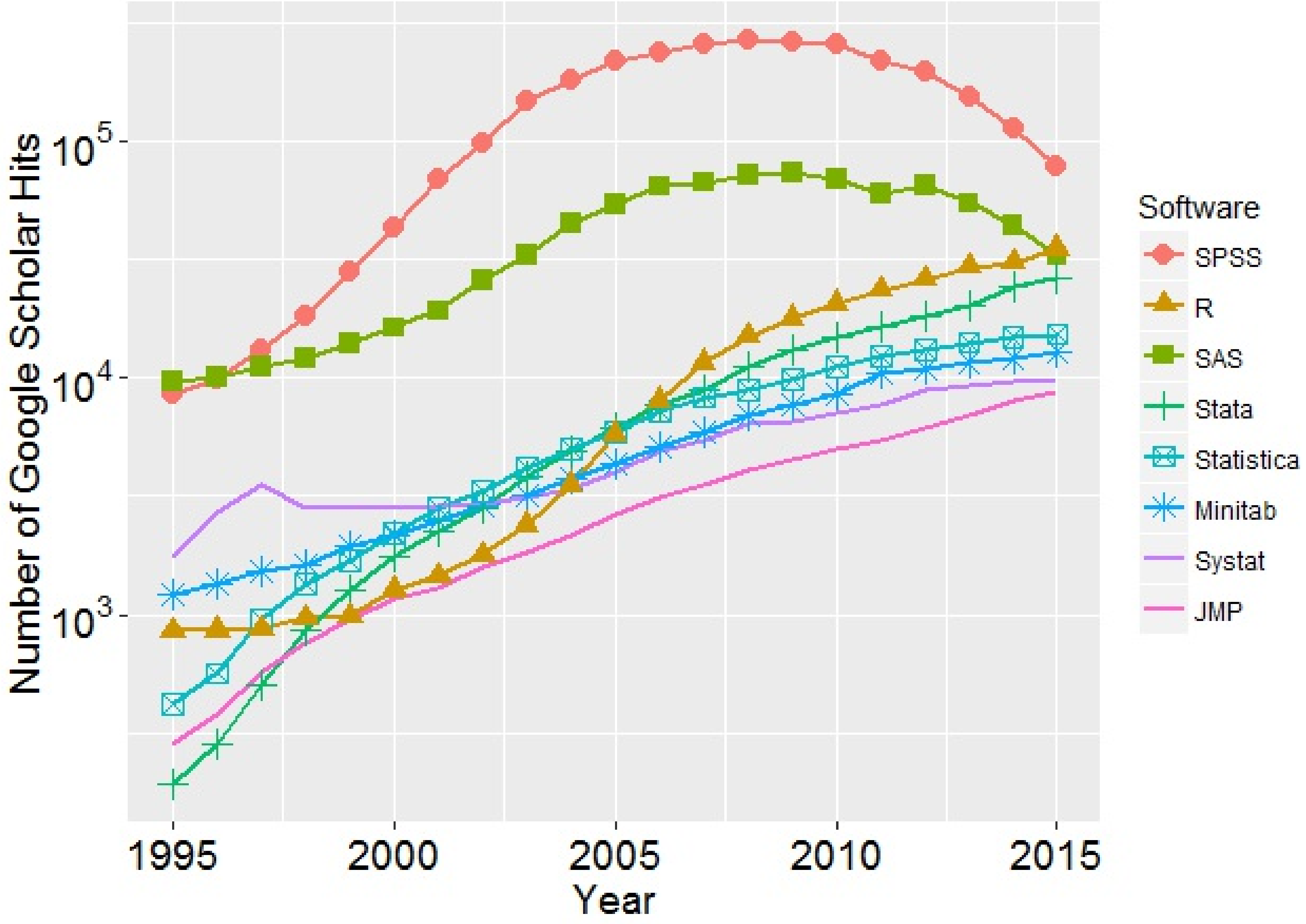
Talk about your analysis on the `weather` data
in groups of 2-3

Let's get it all out there

When you think of R what comes to mind?

Why are you here wanting to learn R?

- ~~Because you enjoy partaking in~~ Schadenfreude



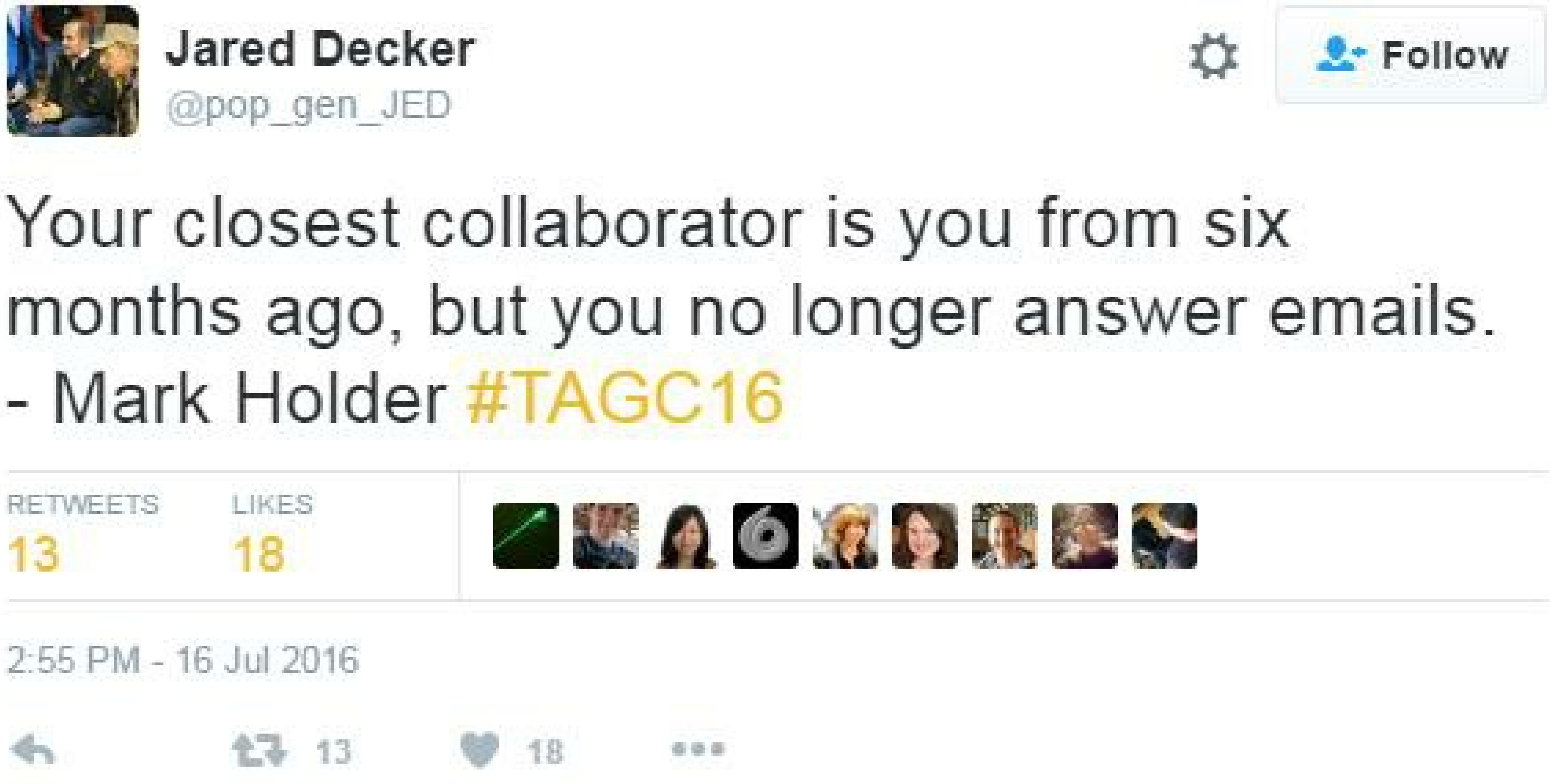
Why should you learn R?

- R is free, R improves, R has existed for 40+ years
- Students can show what they have learned to a potential employer easily
- The support system for R is actually much better than some people say

But as a professor, why should you learn R?

- R and R Markdown will save you TONS of time. Long term thinking is key.
  - You can easily tweak your code if you need to do another analysis
  - Remembering what drop-down menu some thing is in two years from now in a different program will be hard
  - Remembering to copy-and-paste your updated plots/analysis into your word processor is a pain and error prone

But as a professor, why should you learn R?

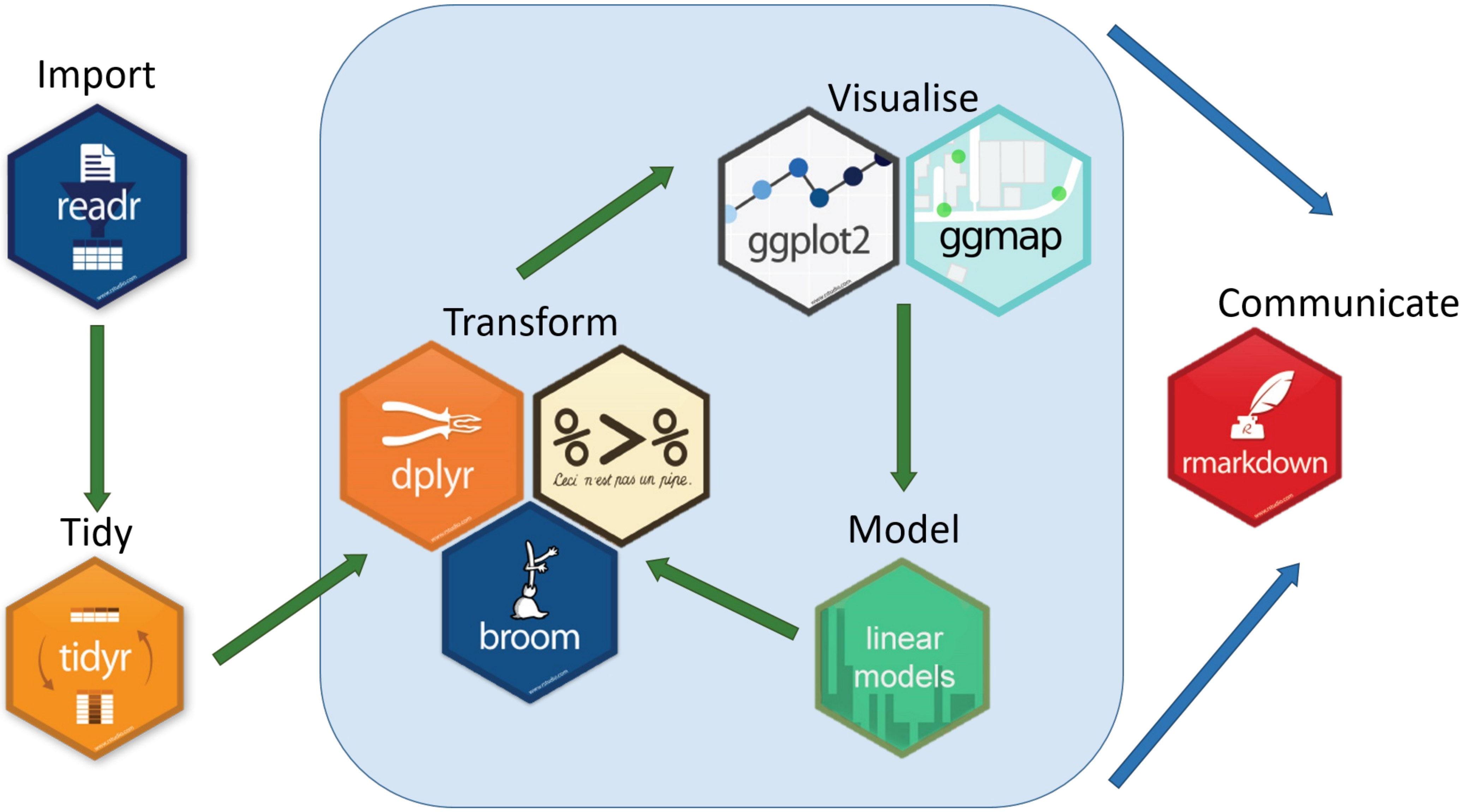


DEMO in RStudio with R Markdown

Analogy

R	RStudio	DataCamp
		

What this bootcamp is



- Designed for all levels of knowledge and background

What this bootcamp is not

- A thorough development of machine learning techniques applied to big data

What this bootcamp is not

- A thorough development of machine learning techniques applied to big data
- A discussion on the role of p-values in science

What this bootcamp is not

- A thorough development of machine learning techniques applied to big data
- A discussion on the role of p-values in science
- A tutorial on how to work with base R subsetting and base R graphics

What I hope you'll learn

- That R is not as scary as it used to be.

What I hope you'll learn

- That R is not as scary as it used to be.
- Learning how to Google is an incredibly valuable skill.

What I hope you'll learn

- That R is not as scary as it used to be.
- Learning how to Google is an incredibly valuable skill.
- That having students turn in assignments using R scripts and R Markdown provides an efficient way to check their work and their analyses easily.

What I hope you'll learn

- That the R packages you will use in this workshop are the same ones that are used by scientists and graduate programs all over the world

What I hope you'll learn

- That the R packages you will use in this workshop are the same ones that are used by scientists and graduate programs all over the world
- That teaching students how to use open-source tools is what is best for them long term

What I hope you'll learn

- That the R packages you will use in this workshop are the same ones that are used by scientists and graduate programs all over the world
- That teaching students how to use open-source tools is what is best for them long term
- That students with no programming background can do great things in only a few weeks
    - Sociology/Criminal Justice majors at Pacific U.
    - A wide range of students at Middlebury College

Pieces of advice

- Scaffold & support as a foreign languages do

Pieces of advice

- Scaffold & support as a foreign languages do
- To be able to use R (and really any other language), students need more than a few assignments and more than a weekly lab

Pieces of advice

- Scaffold & support as a foreign languages do
- To be able to use R (and really any other language), students need more than a few assignments and more than a weekly lab
- Make use of RStudio Projects (students have a really hard time navigating directory structures)

Pieces of advice

My opinion

- Have students work on writing their code in R script files and documenting their errors
    - This is the same workflow that DataCamp uses

Pieces of advice

My opinion

- Have students work on writing their code in R script files and documenting their errors
    - This is the same workflow that DataCamp uses
- Show students R Markdown after a few weeks of working with the script file
    - I've found it is hard for students to learn R and R Markdown from the start
    - Better to have them use R Markdown in groups initially

R Data Types

The bare minimum needed for understanding today

Vector/variable

- Type of vector (`int`, `num`, `chr`, `lgl`, `date`)

The bare minimum needed for understanding today

Vector/variable

- Type of vector (`int`, `num`, `chr`, `lgl`, `date`)

Data frame

- Vectors of (potentially) different types
- Each vector has the same number of rows

The bare minimum needed for understanding today

```r
library(tibble)
library(lubridate)
ex1 <- data_frame(
    vec1 = c(1980, 1990, 2000, 2010),
    vec2 = c(1L, 2L, 3L, 4L),
    vec3 = c("low", "low", "high", "high"),
    vec4 = c(TRUE, FALSE, FALSE, FALSE),
    vec5 = ymd(c("2017-05-23", "1776/07/04", "1983-05/31", "1908/04-01"))
  )
ex1
```

The bare minimum needed for understanding today

```r
library(tibble)
library(lubridate)
ex1 <- data_frame(
    vec1 = c(1980, 1990, 2000, 2010),
    vec2 = c(1L, 2L, 3L, 4L),
    vec3 = c("low", "low", "high", "high"),
    vec4 = c(TRUE, FALSE, FALSE, FALSE),
    vec5 = ymd(c("2017-05-23", "1776/07/04", "1983-05/31", "1908/04-01"))
  )
ex1
```

```
# A tibble: 4 x 5
   vec1  vec2  vec3  vec4       vec5
  <dbl> <int> <chr> <lgl>     <date>
1  1980     1   low  TRUE 2017-05-23
2  1990     2   low FALSE 1776-07-04
3  2000     3  high FALSE 1983-05-31
4  2010     4  high FALSE 1908-04-01
```

Welcome to the tidyverse!

The `tidyverse` is a collection of R packages that share common philosophies and are designed to work together.

Beginning steps

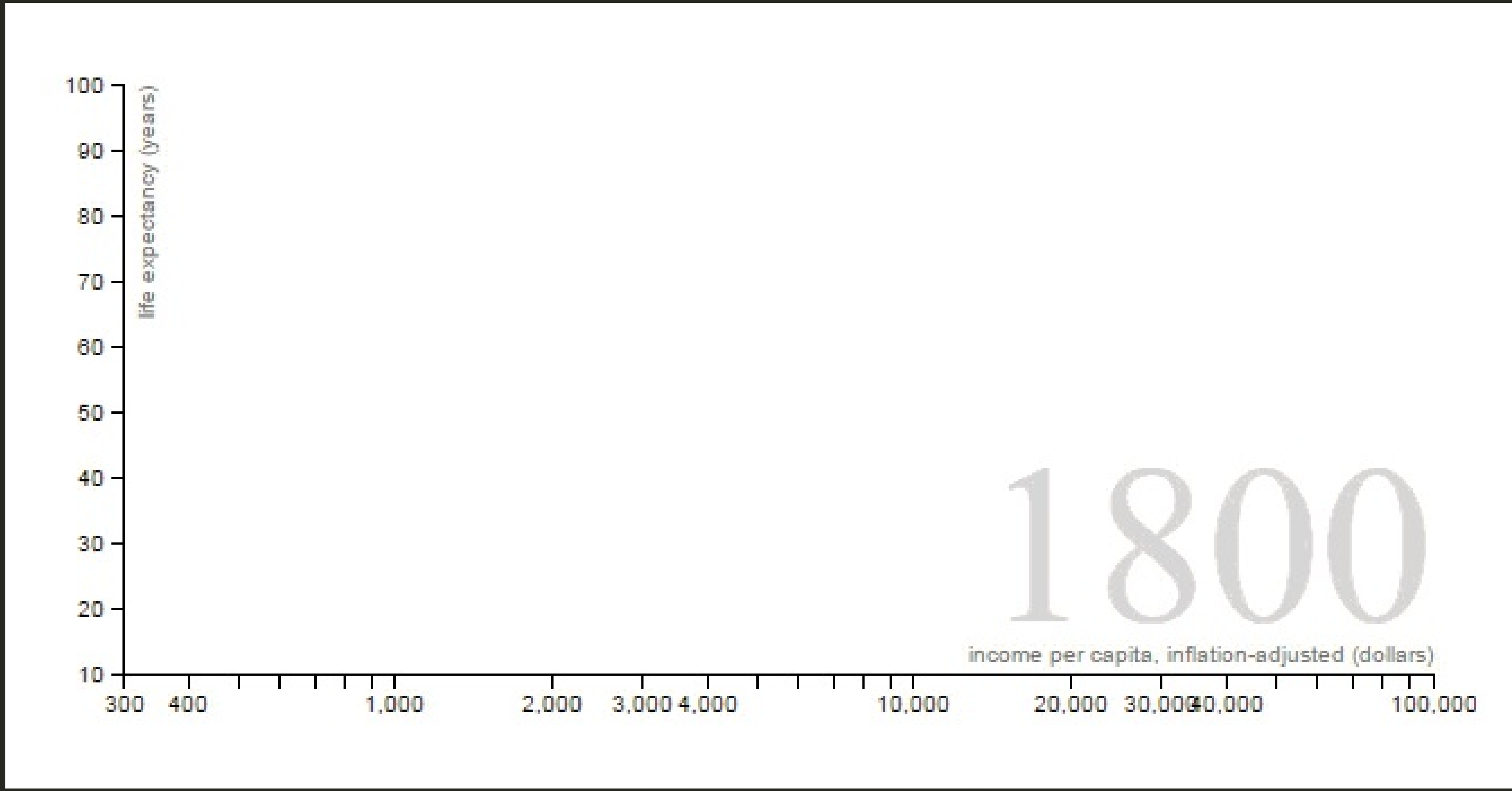
Frequently the first thing to do when given a dataset is to

- identify the observational unit,
- specify the variables,
- give the types of variables you are presented with, and
- check that the data is tidy. (TO COME)

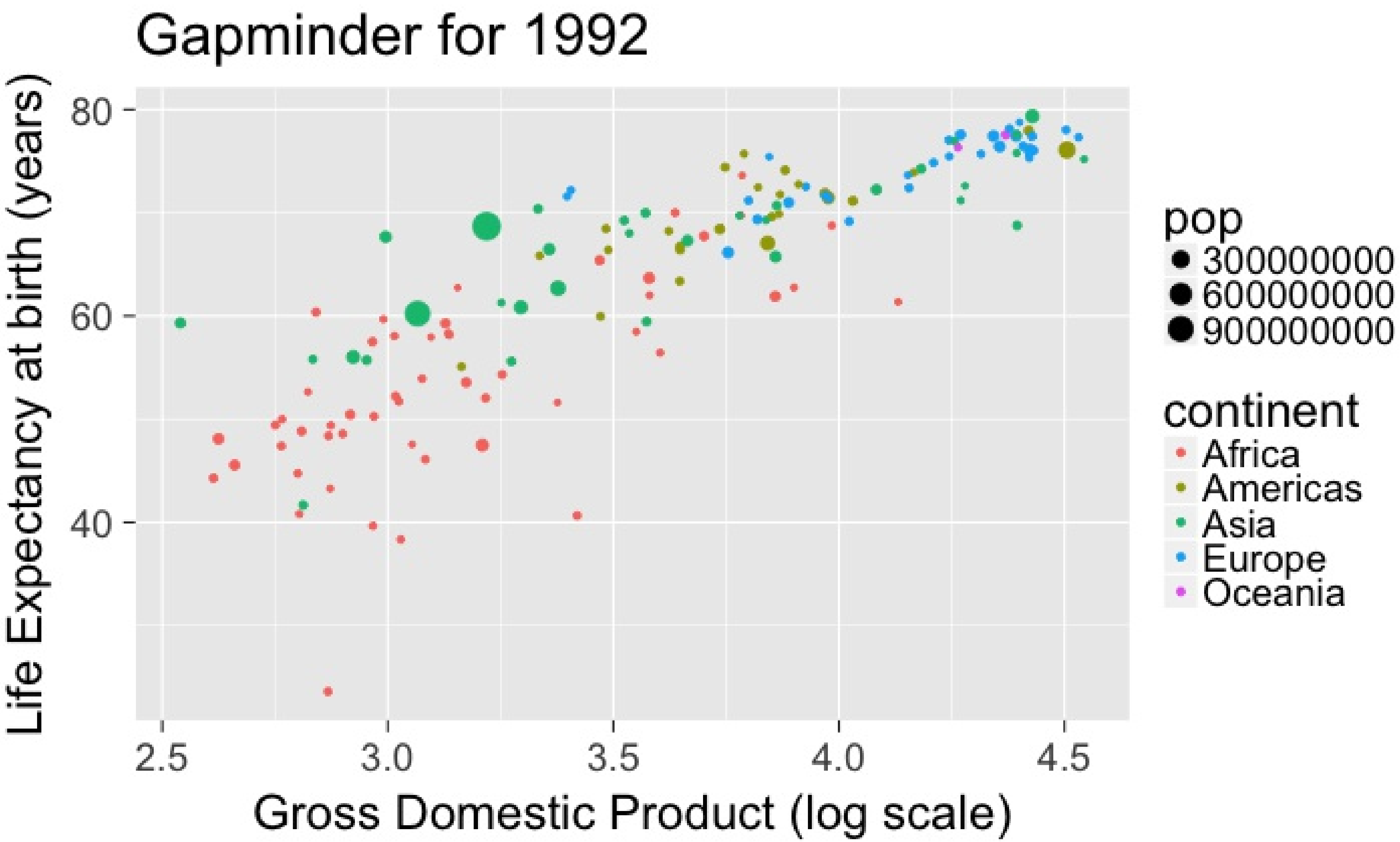
This will help with

- choosing the appropriate plot,
- summarizing the data, and
- understanding which inferences/models can be applied.

Data Visualization



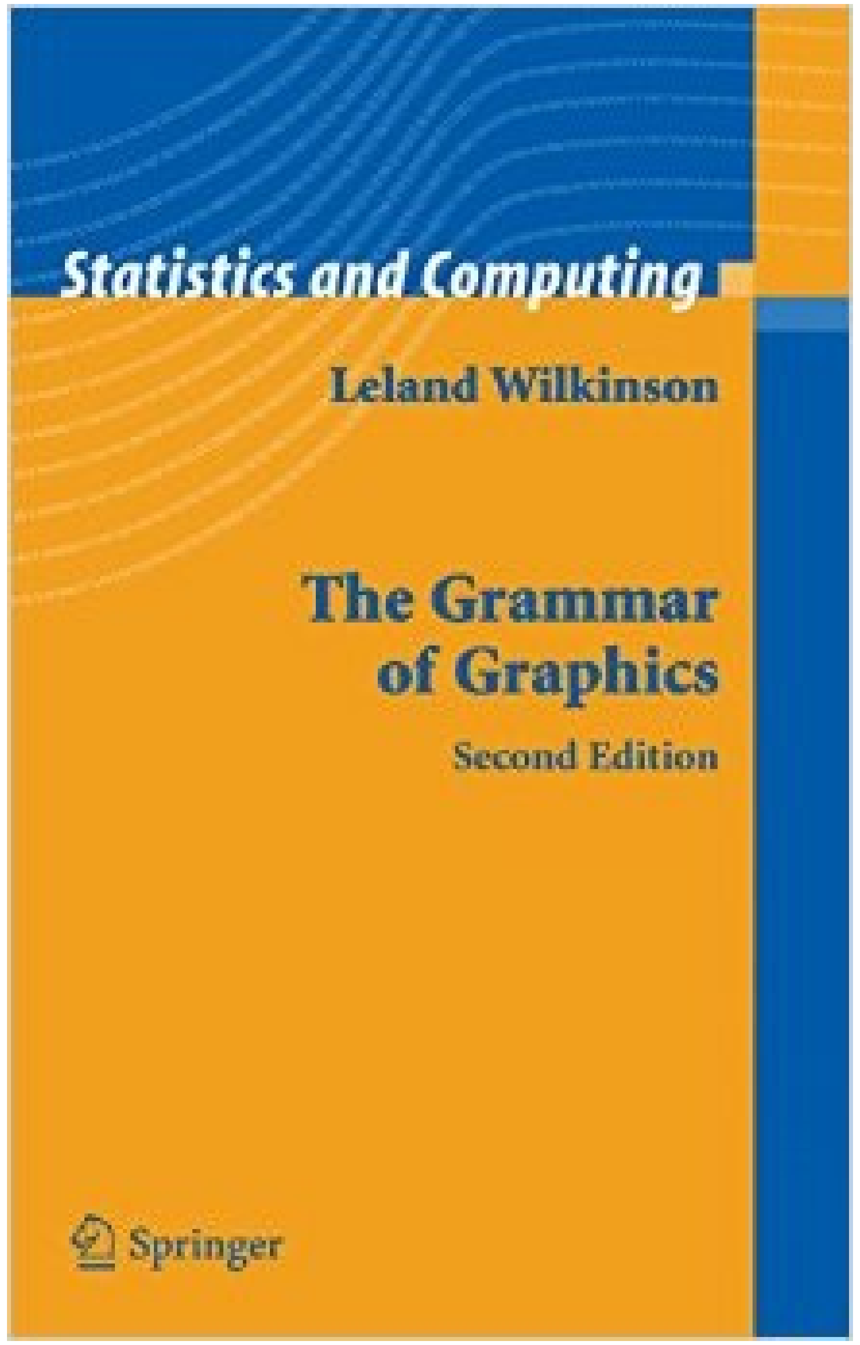
Inspired by Hans Rosling



- What are the variables here?
- What is the observational unit?
- How are the variables mapped to aesthetics?

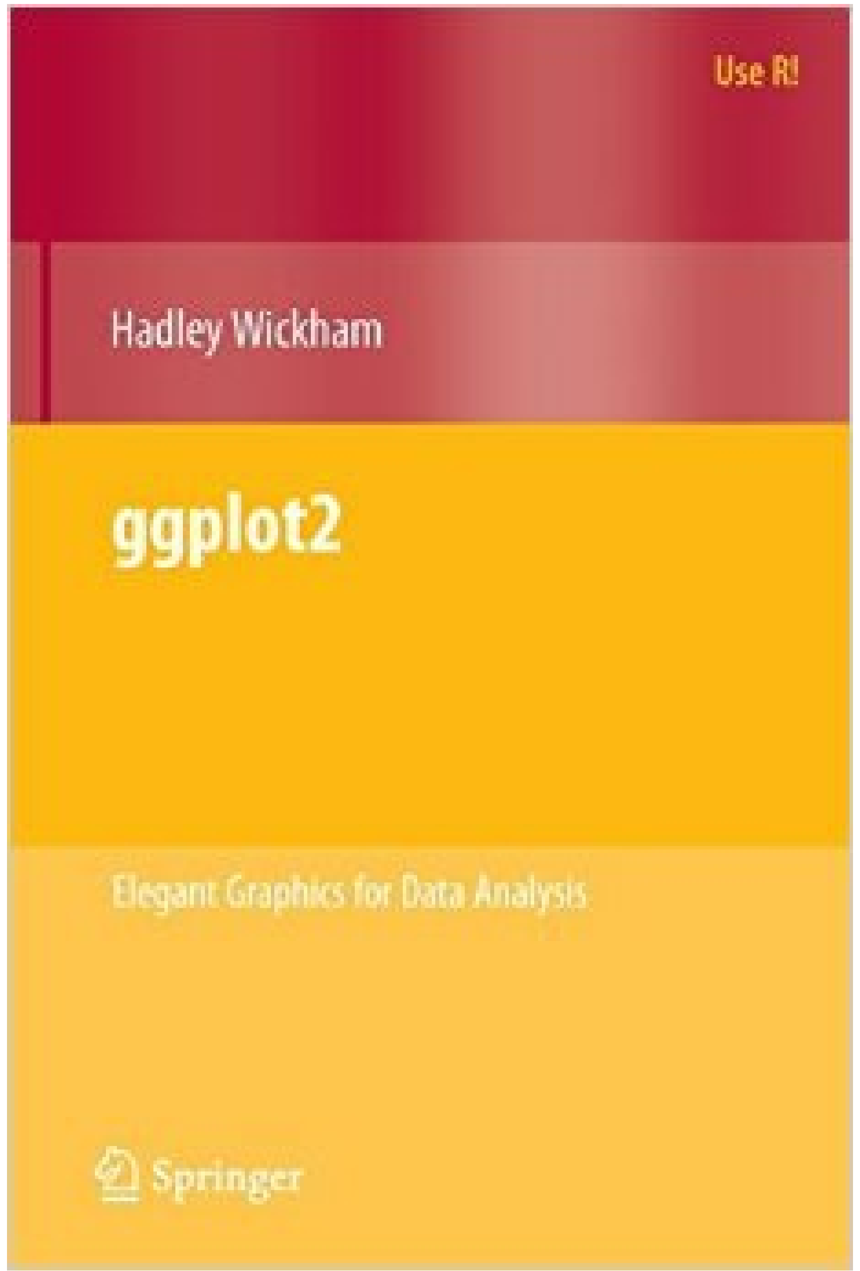
Grammar of Graphics

Wilkinson (2005) laid out the proposed "Grammar of Graphics"



Grammar of Graphics in R

Wickham implemented the grammar in R in the `ggplot2` package



Grammar of Graphics elsewhere

- Make plots online with plotly
- Leland Wilkinson works at Tableau
- The Grammar of Graphics provides a theoretical framework for building and deciphering statistical graphics

What is a statistical graphic?

What is a statistical graphic?

A `mapping` of
`data` variables

What is a statistical graphic?

A `mapping` of
`data` variables

to

`aes()`thetic attributes

What is a statistical graphic?

A `mapping` of
`data` variables

to
`aes()`thetic attributes

of
`geom_`etric objects.

Back to basics

Consider the following data in tidy format:

```
simple_ex <-
  data_frame(
    A = c(1980, 1990, 2000, 2010),
    B = c(1, 2, 3, 4),
    C = c(3, 2, 1, 2),
    D = c("low", "low", "high", "high")
  )
simple_ex
```

```
# A tibble: 4 x 4
      A     B     C     D
  <dbl> <dbl> <dbl> <chr>
1  1980     1     3   low
2  1990     2     2   low
3  2000     3     1  high
4  2010     4     2  high
```

Consider the following data in tidy format:

A	B	C	D
1980	1	3	low
1990	2	2	low
2000	3	1	high
2010	4	2	high

- Sketch the graphics below on paper, where the `x`-axis is variable `A` and the `y`-axis is variable `B`

1. A scatter plot
2. A scatter plot where the `color` of the points corresponds to `D`
3. A scatter plot where the `size` of the points corresponds to `C`
4. A line graph
5. A line graph where the `color` of the line corresponds to `D` with points added that are all forestgreen of size `4`.

Reproducing the plots in `ggplot2`

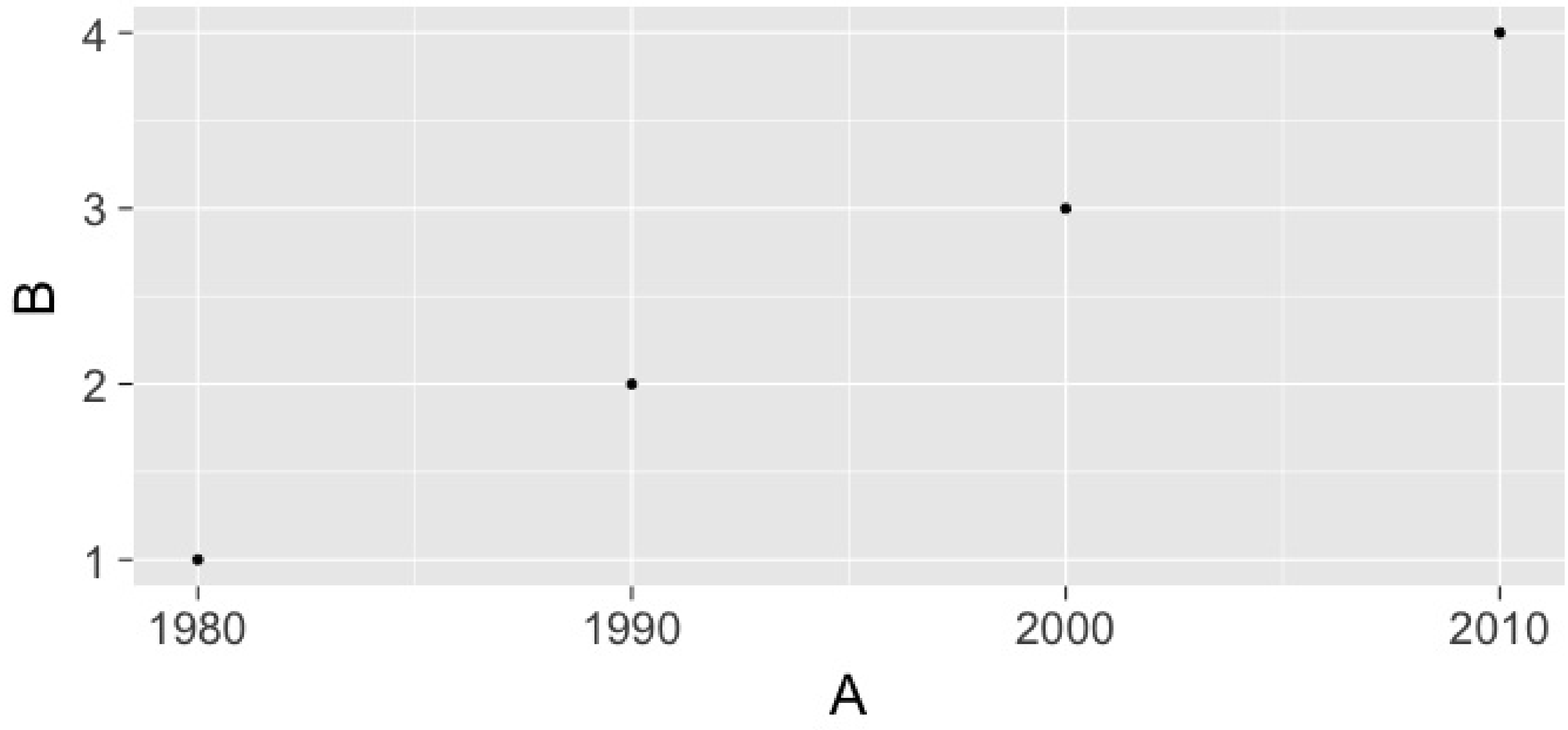
1. A scatterplot

```
library(ggplot2)
ggplot(data = simple_ex, mapping = aes(x = A, y = B)) +
  geom_point()
```

Reproducing the plots in `ggplot2`

1. A scatterplot

```
library(ggplot2)
ggplot(data = simple_ex, mapping = aes(x = A, y = B)) +
  geom_point()
```



Reproducing the plots in `ggplot2`

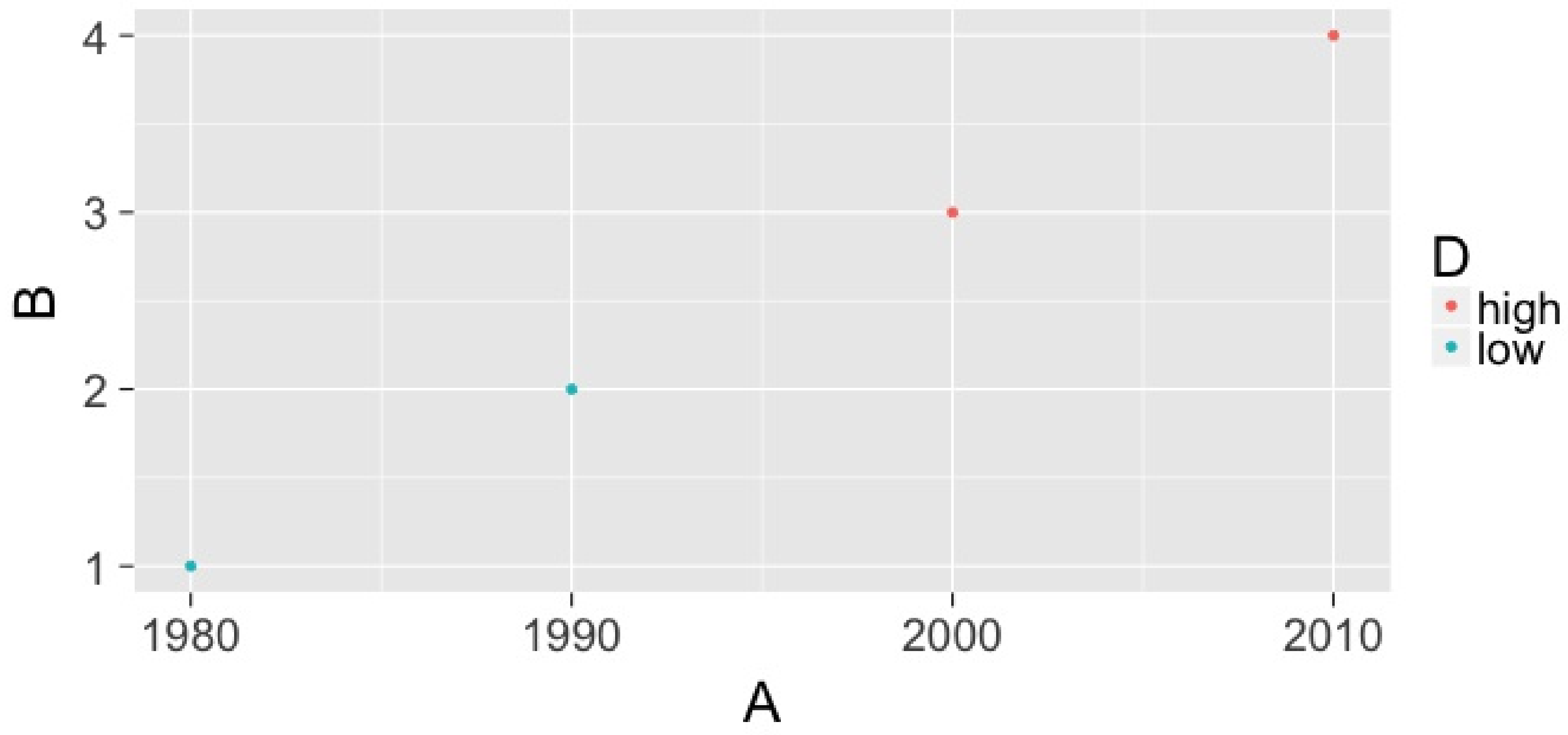
2. A scatter plot where the `color` of the points corresponds to `group`

```
library(ggplot2)
ggplot(data = simple_ex, mapping = aes(x = A, y = B)) +
  geom_point(mapping = aes(color = D))
```

Reproducing the plots in `ggplot2`

2. A scatter plot where the `color` of the points corresponds to `group`

```
library(ggplot2)
ggplot(data = simple_ex, mapping = aes(x = A, y = B)) +
  geom_point(mapping = aes(color = D))
```



Reproducing the plots in `ggplot2`

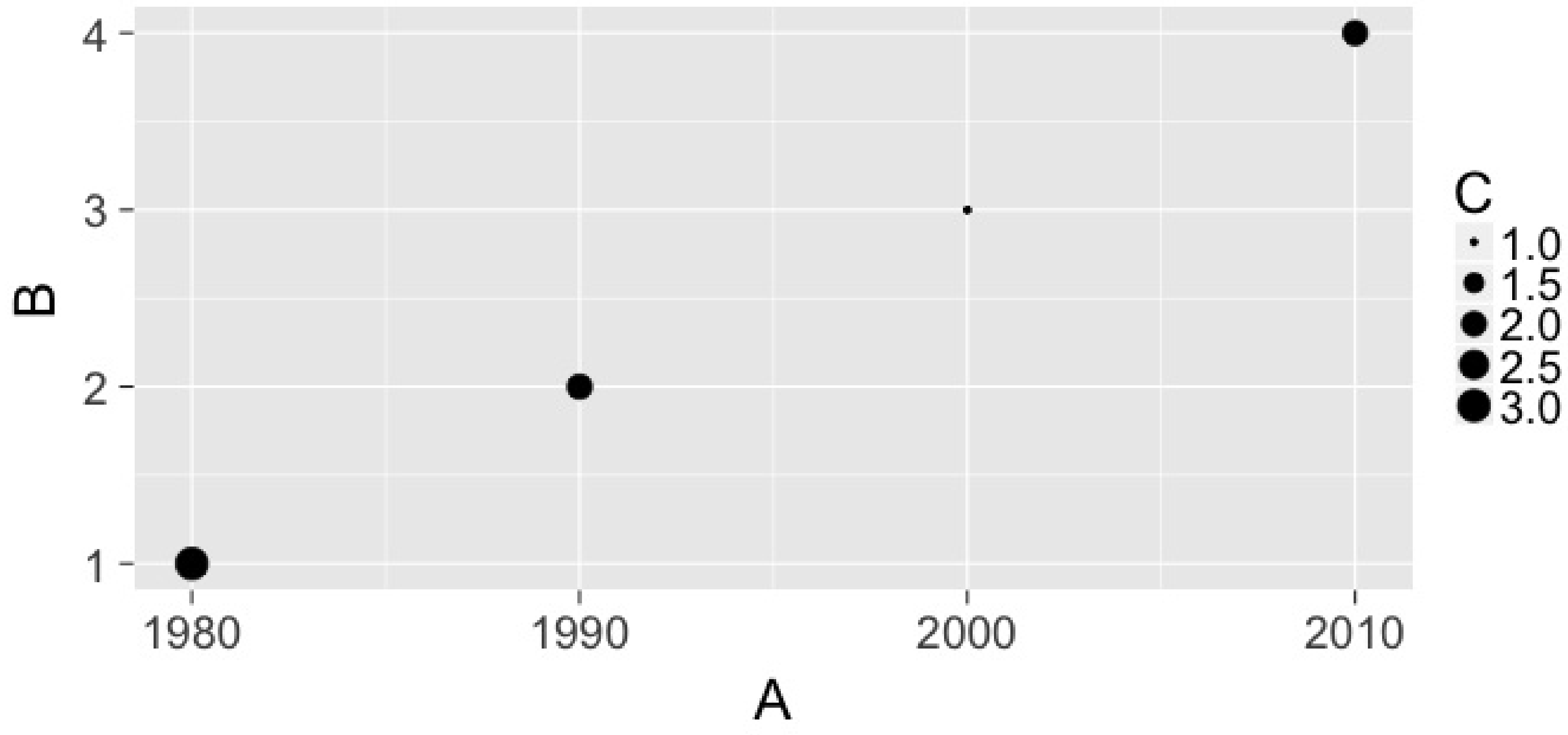
3. A scatter plot where the `size` of the points corresponds to `C`

```
library(ggplot2)
ggplot(data = simple_ex, mapping = aes(x = A, y = B, size = C)) +
  geom_point()
```

Reproducing the plots in `ggplot2`

3. A scatter plot where the `size` of the points corresponds to `C`

```
library(ggplot2)
ggplot(data = simple_ex, mapping = aes(x = A, y = B, size = C)) +
  geom_point()
```



Reproducing the plots in `ggplot2`

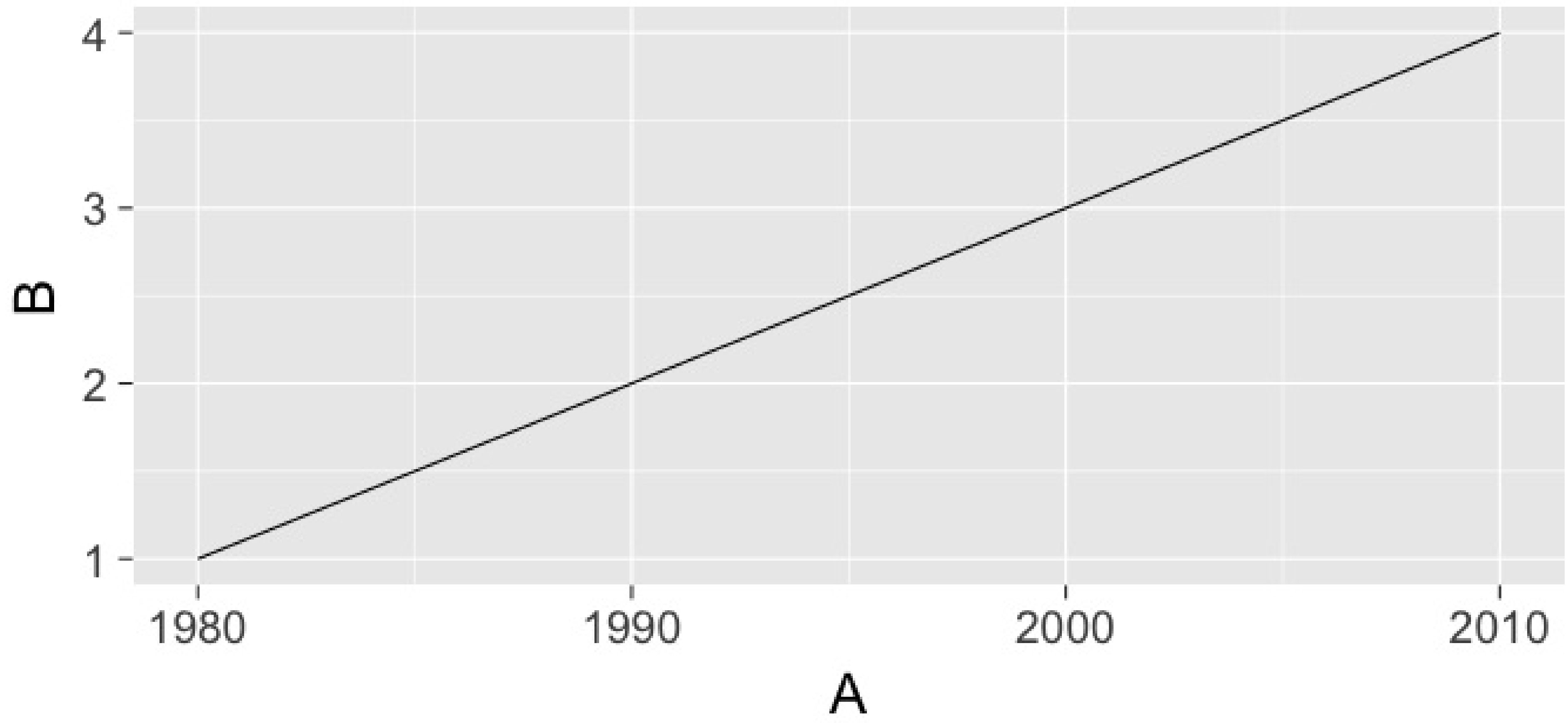
4. A line graph

```
library(ggplot2)
ggplot(data = simple_ex, mapping = aes(x = A, y = B)) +
  geom_line()
```

Reproducing the plots in `ggplot2`

4. A line graph

```
library(ggplot2)
ggplot(data = simple_ex, mapping = aes(x = A, y = B)) +
  geom_line()
```



Reproducing the plots in `ggplot2`

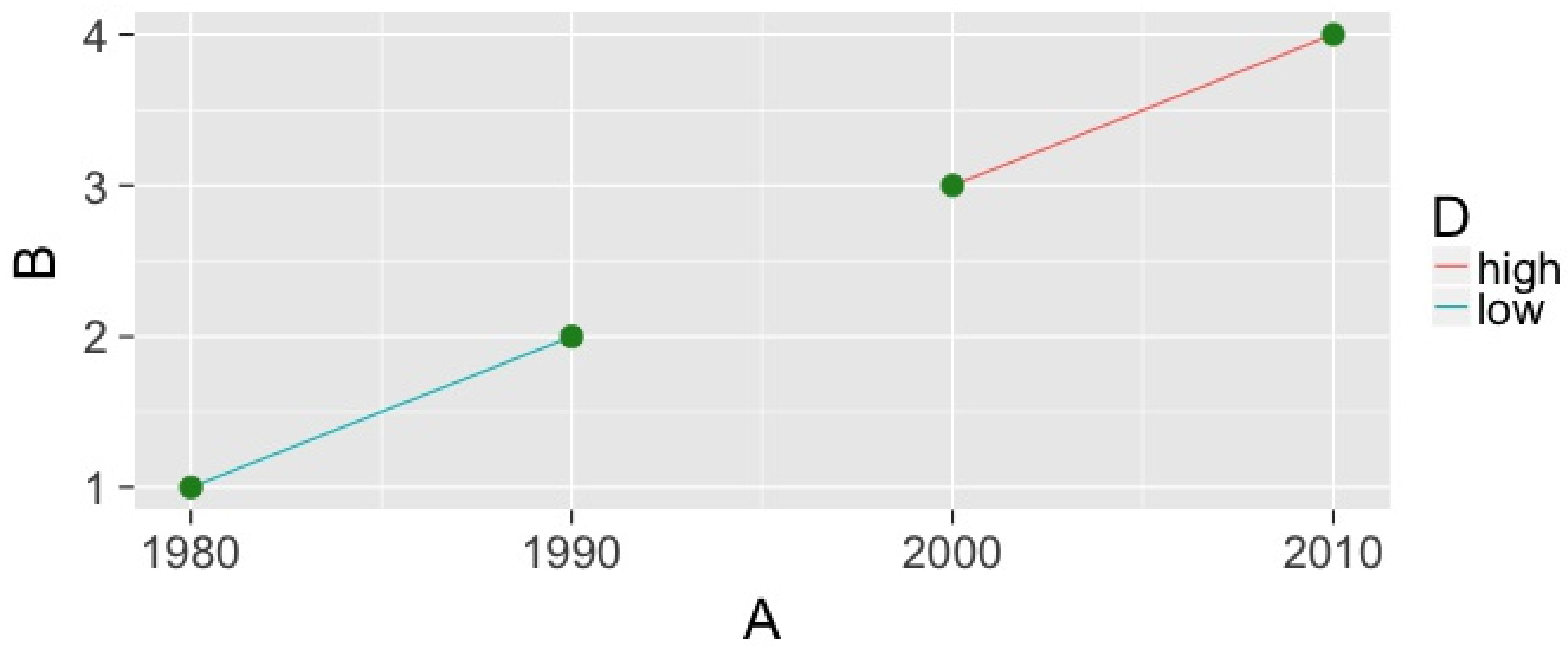
5. A line graph where the `color` of the line corresponds to `D` with points added that are all blue of size 4.

```
library(ggplot2)
ggplot(data = simple_ex, mapping = aes(x = A, y = B)) +
  geom_line(mapping = aes(color = D)) +
  geom_point(color = "forestgreen", size = 4)
```

Reproducing the plots in `ggplot2`

5. A line graph where the `color` of the line corresponds to `D` with points added that are all blue of size 4.

```
library(ggplot2)
ggplot(data = simple_ex, mapping = aes(x = A, y = B)) +
  geom_line(mapping = aes(color = D)) +
  geom_point(color = "forestgreen", size = 4)
```



The Five-Named Graphs

The 5NG of data viz

- Scatterplot: `geom_point()`
- Line graph: `geom_line()`

The Five-Named Graphs

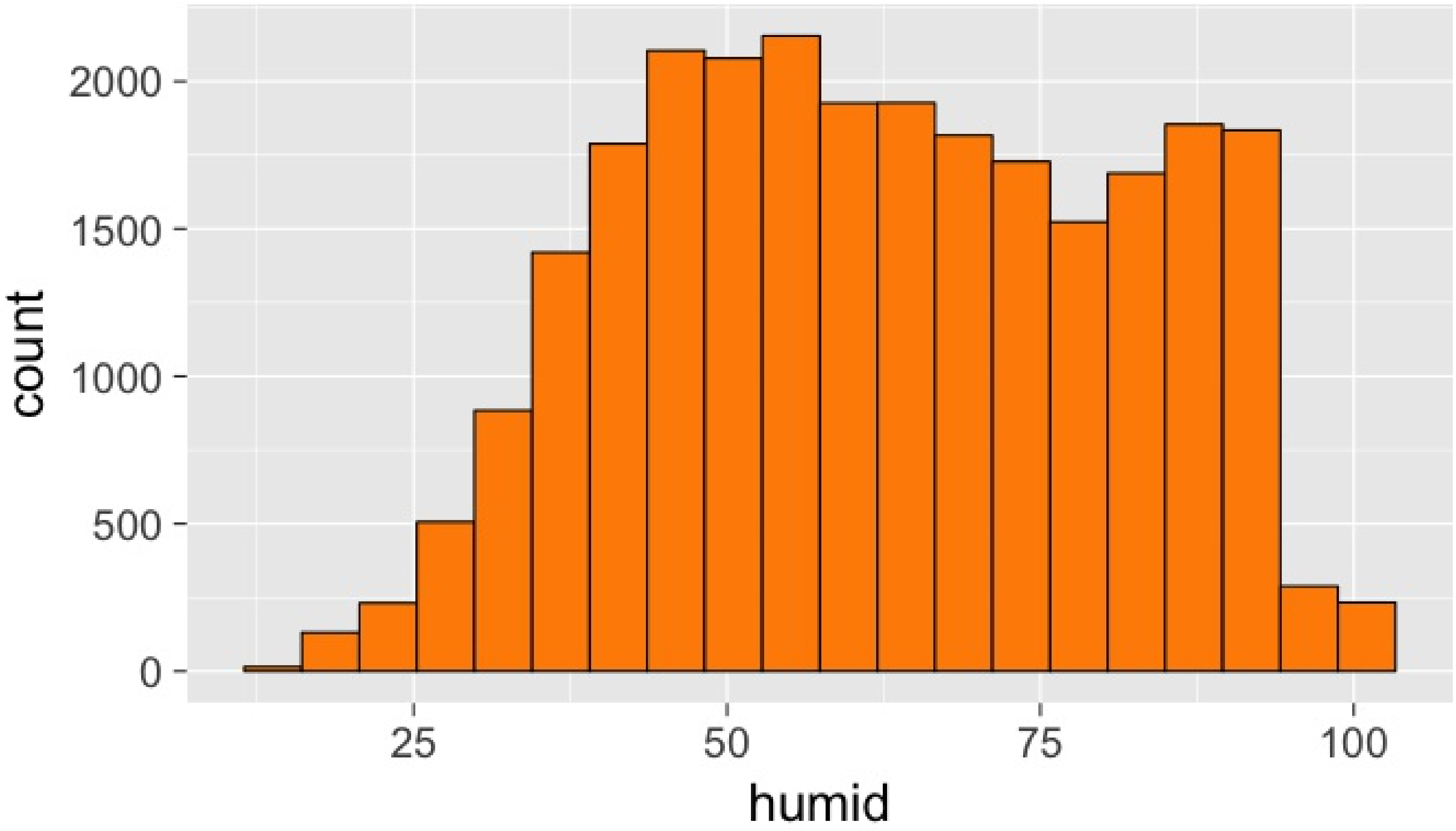
The 5NG of data viz

- Scatterplot: `geom_point()`
- Line graph: `geom_line()`
- Histogram: `geom_histogram()`
- Boxplot: `geom_boxplot()`
- Bar graph: `geom_bar()`

More `ggplot2` examples

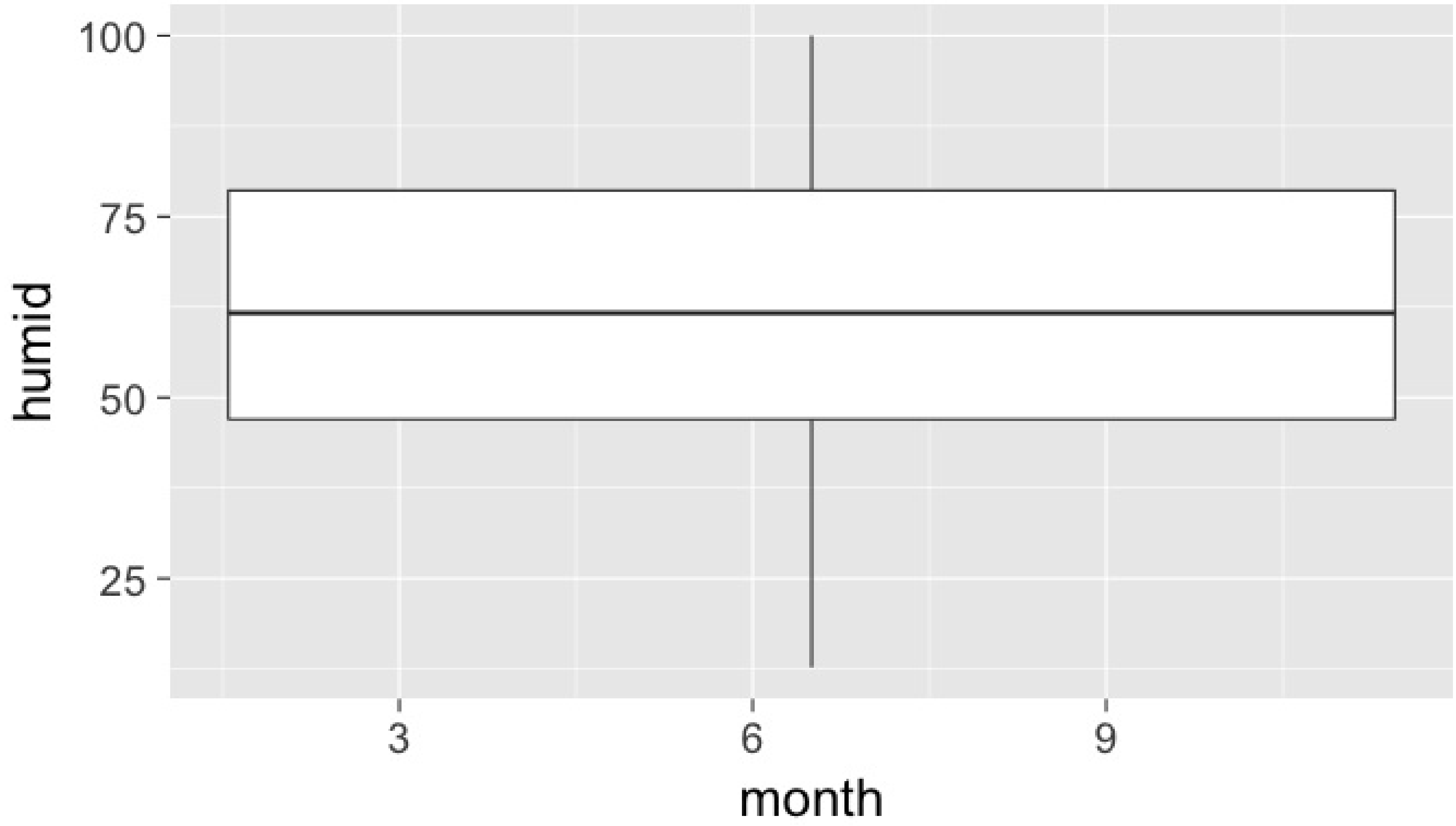
Histogram

```
library(nycflights13)
ggplot(data = weather, mapping = aes(x = humid)) +
  geom_histogram(bins = 20, color = "black", fill = "darkorange")
```



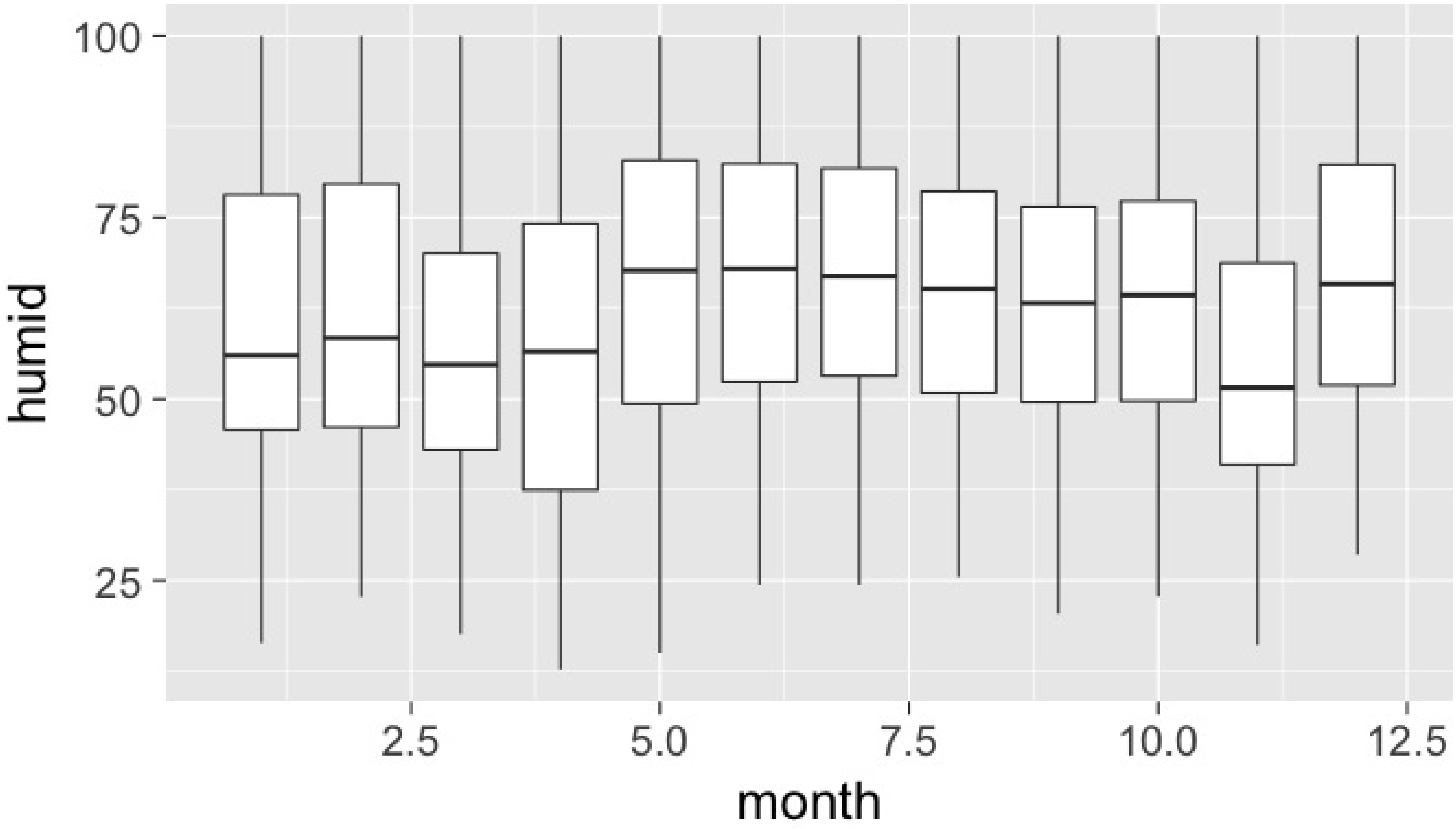
Boxplot (broken)

```
library(nycflights13)
ggplot(data = weather, mapping = aes(x = month, y = humid)) +
  geom_boxplot()
```



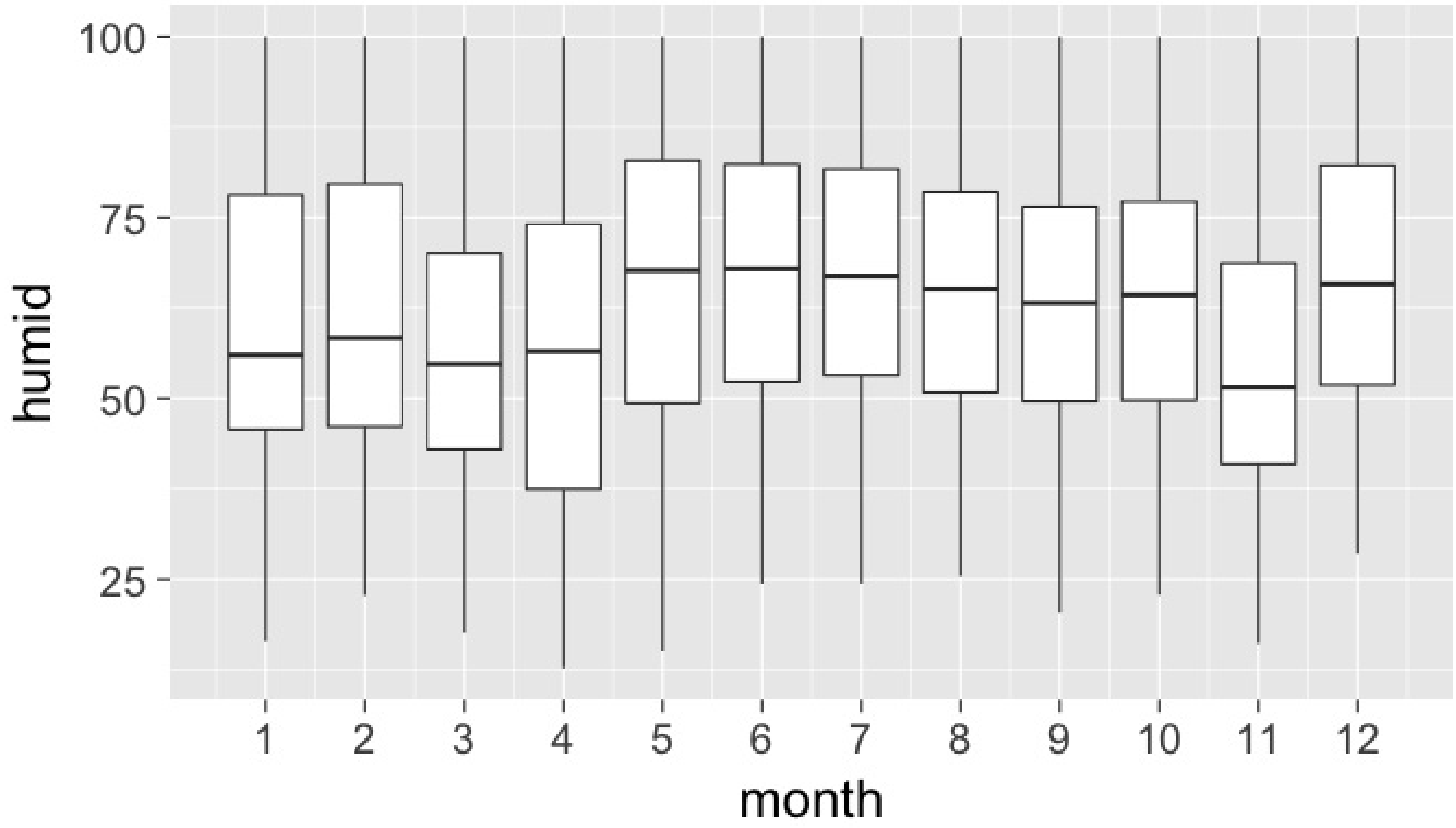
Boxplot (almost fixed)

```
library(nycflights13)
ggplot(data = weather, mapping = aes(x = month, group = month, y = humid)) +
  geom_boxplot()
```



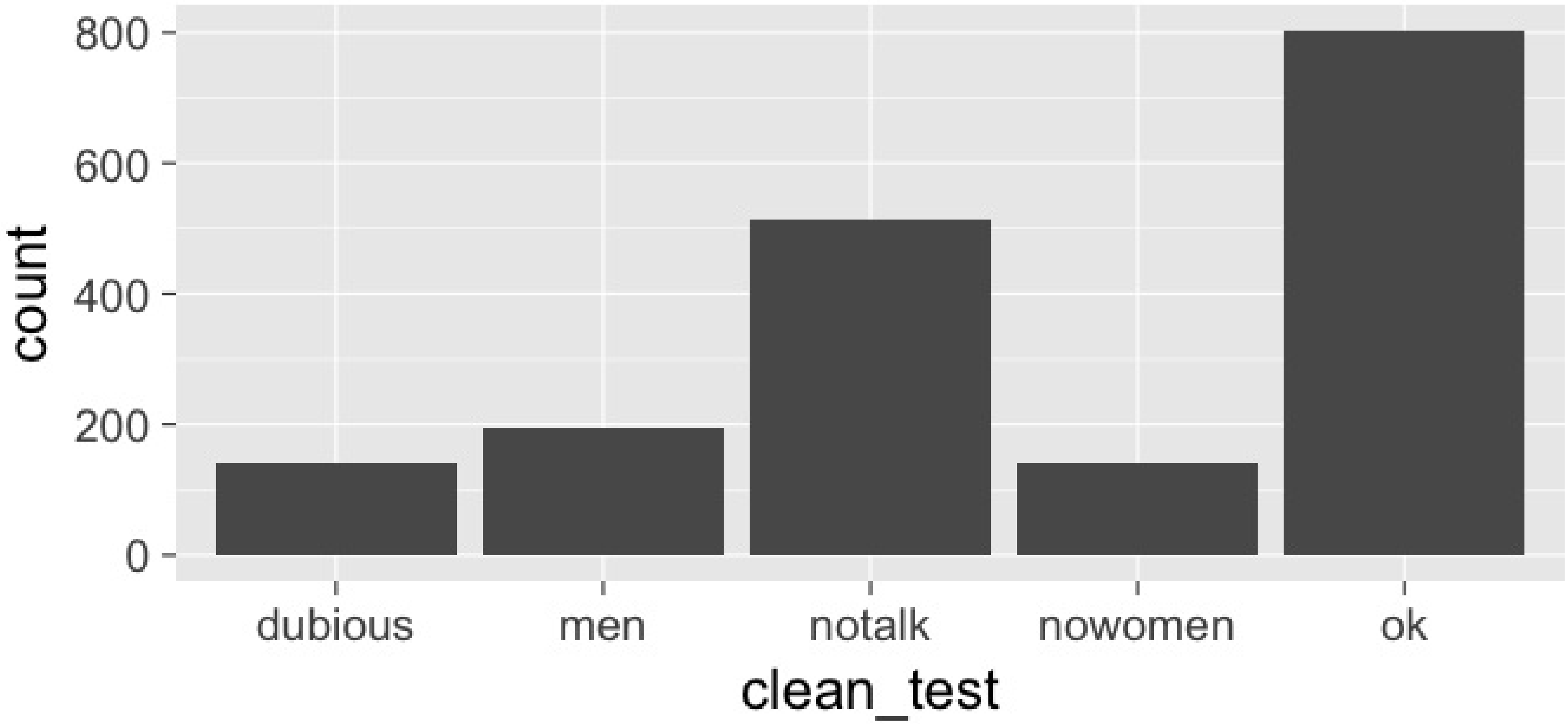
Boxplot (fixed)

```
library(nycflights13)
ggplot(data = weather, mapping = aes(x = month, group = month, y = humid)) +
  geom_boxplot() +
  scale_x_continuous(breaks = 1:12)
```



Bar graph

```
library(fivethirtyeight)
ggplot(data = bechdel, mapping = aes(x = clean_test)) +
  geom_bar()
```



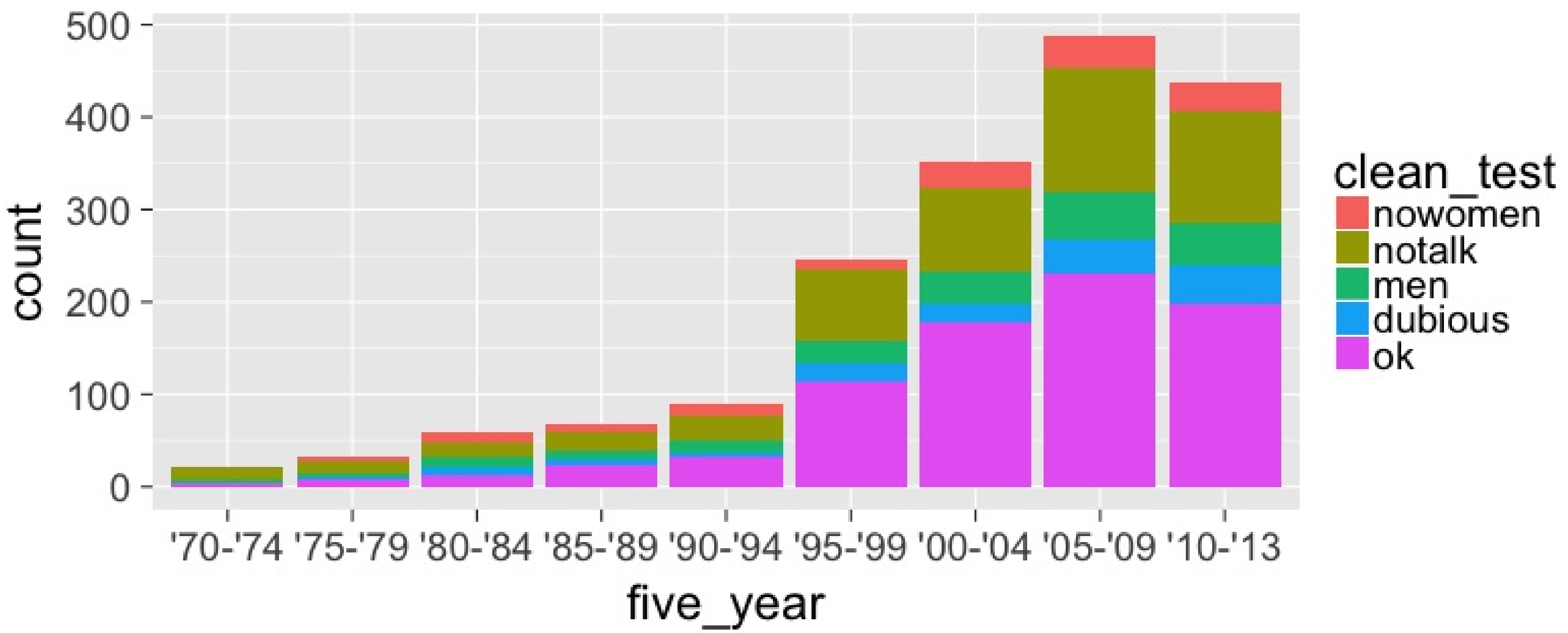
How about over time?

- Hop into `dplyr`

```
library(dplyr)
year_bins <- c("'70-'74", "'75-'79", "'80-'84", "'85-'89",
               "'90-'94", "'95-'99", "'00-'04", "'05-'09",
               "'10-'13")
bechdel <- bechdel %>%
  mutate(five_year = cut(year,
                         breaks = seq(1969, 2014, 5),
                         labels = year_bins)) %>%
  mutate(clean_test = factor(clean_test,
                             levels = c("nowomen", "notalk", "men",
                                        "dubious", "ok")))
```

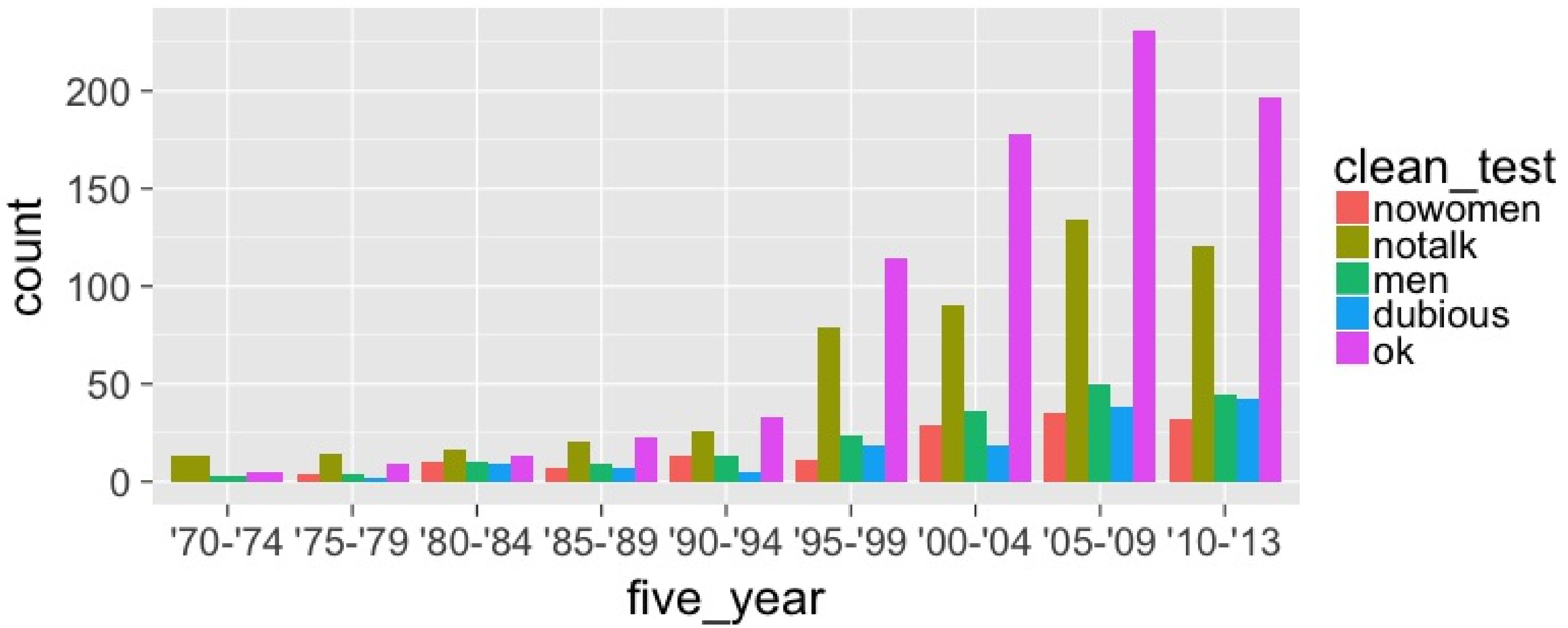
How about over time? (Stacked)

```r
library(fivethirtyeight)
library(ggplot2)
ggplot(data = bechdel,
       mapping = aes(x = five_year, fill = clean_test)) +
  geom_bar()
```



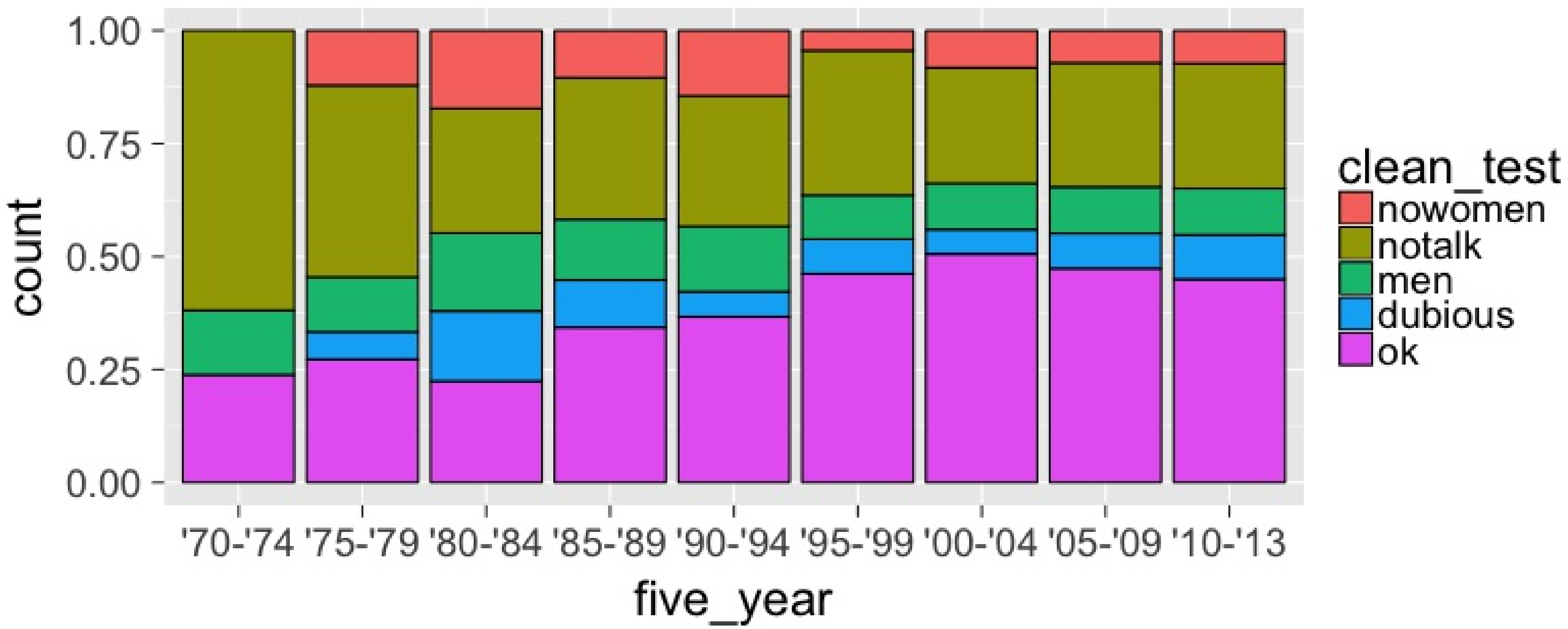
How about over time? (Side-by-side)

```
library(fivethirtyeight)
library(ggplot2)
ggplot(data = bechdel,
       mapping = aes(x = five_year, fill = clean_test)) +
  geom_bar(position = "dodge")
```

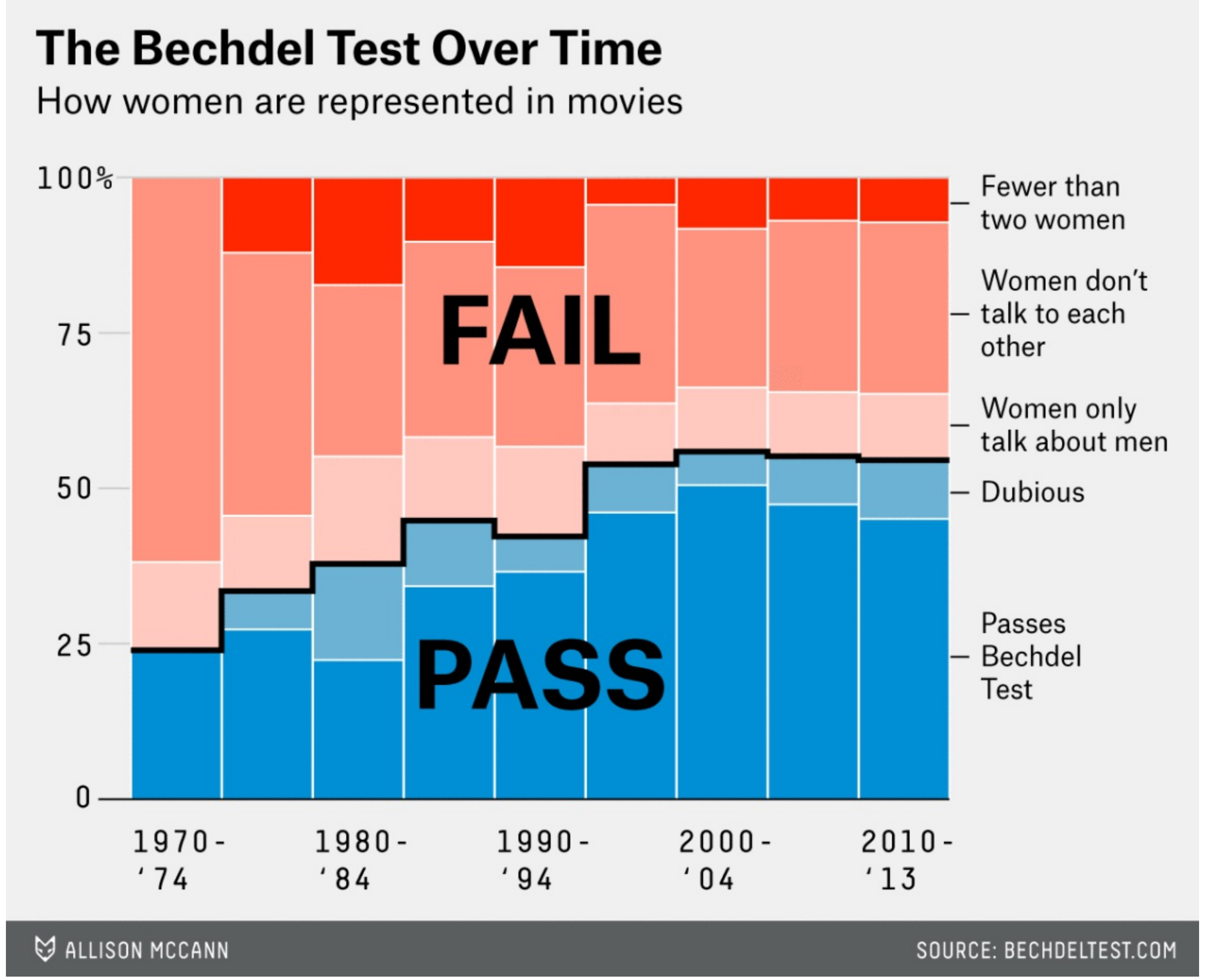


How about over time? (Stacked proportional)

```
library(fivethirtyeight)
library(ggplot2)
ggplot(data = bechdel,
       mapping = aes(x = five_year, fill = clean_test)) +
  geom_bar(position = "fill", color = "black")
```



The `tidyverse`/`ggplot2` is for beginners and for data science professionals!

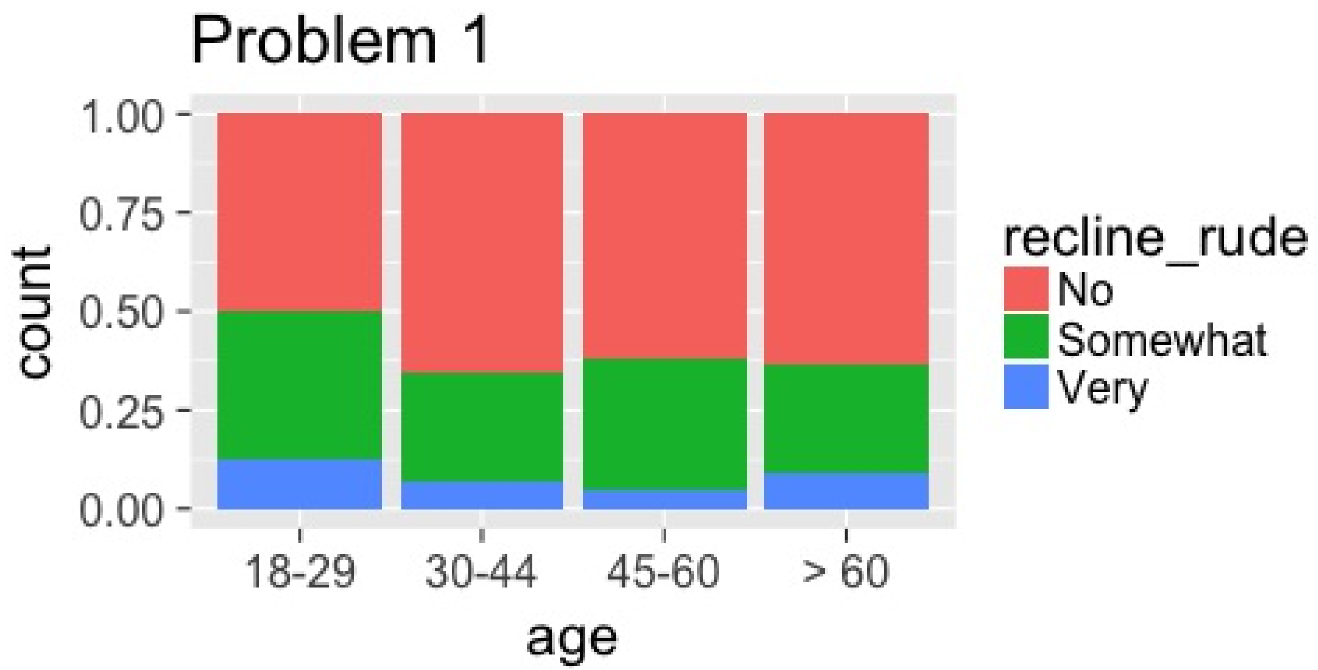


Practice

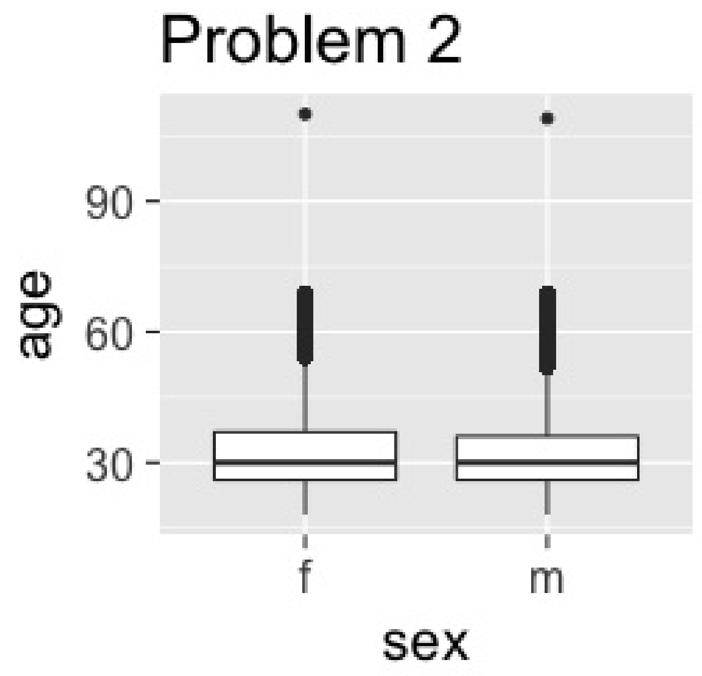
Produce appropriate 5NG with R package & data set in [ ], e.g., [`nycflights13` → `weather`]

1. Does `age` predict `recline_rude`? [`fivethirtyeight` → `na.omit(flying)`]
2. Distribution of `age` by `sex` [`okcupiddata` → `profiles`]
3. Does `budget` predict `rating`? [`ggplot2movies` → `movies`]
4. Distribution of log base 10 scale of `budget_2013` [`fivethirtyeight` → `bechdel`]

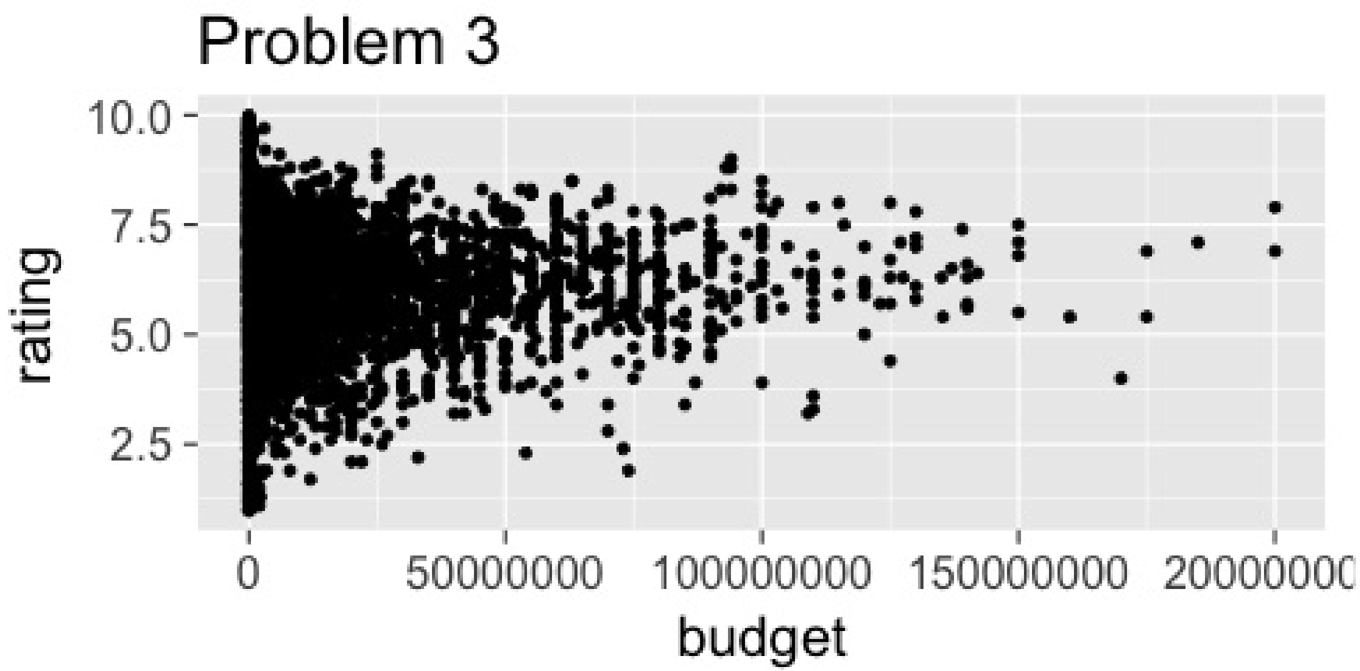
HINTS



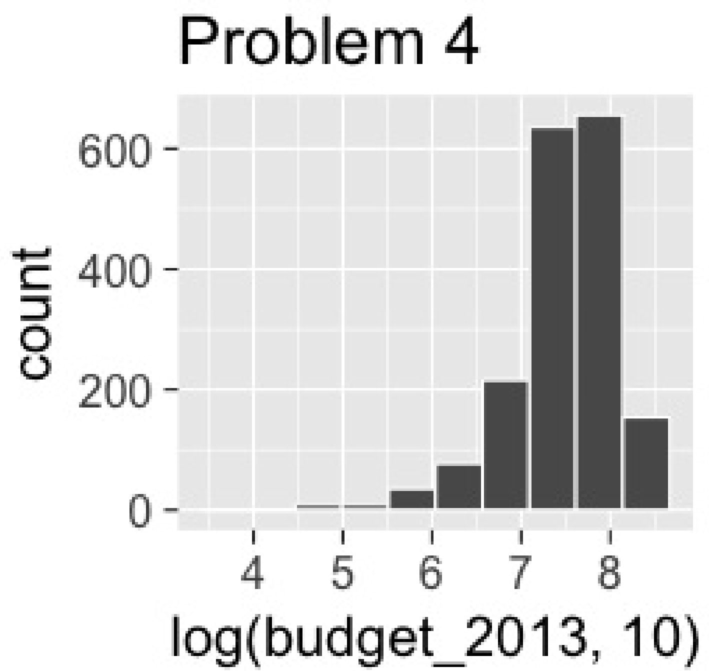
Problem 1
count
1.00
0.75
0.50
0.25
0.00
18-29
30-44
45-60
> 60
age
recline_rude
No
Somewhat
Very



Problem 2
age
90
60
30
f
m
sex



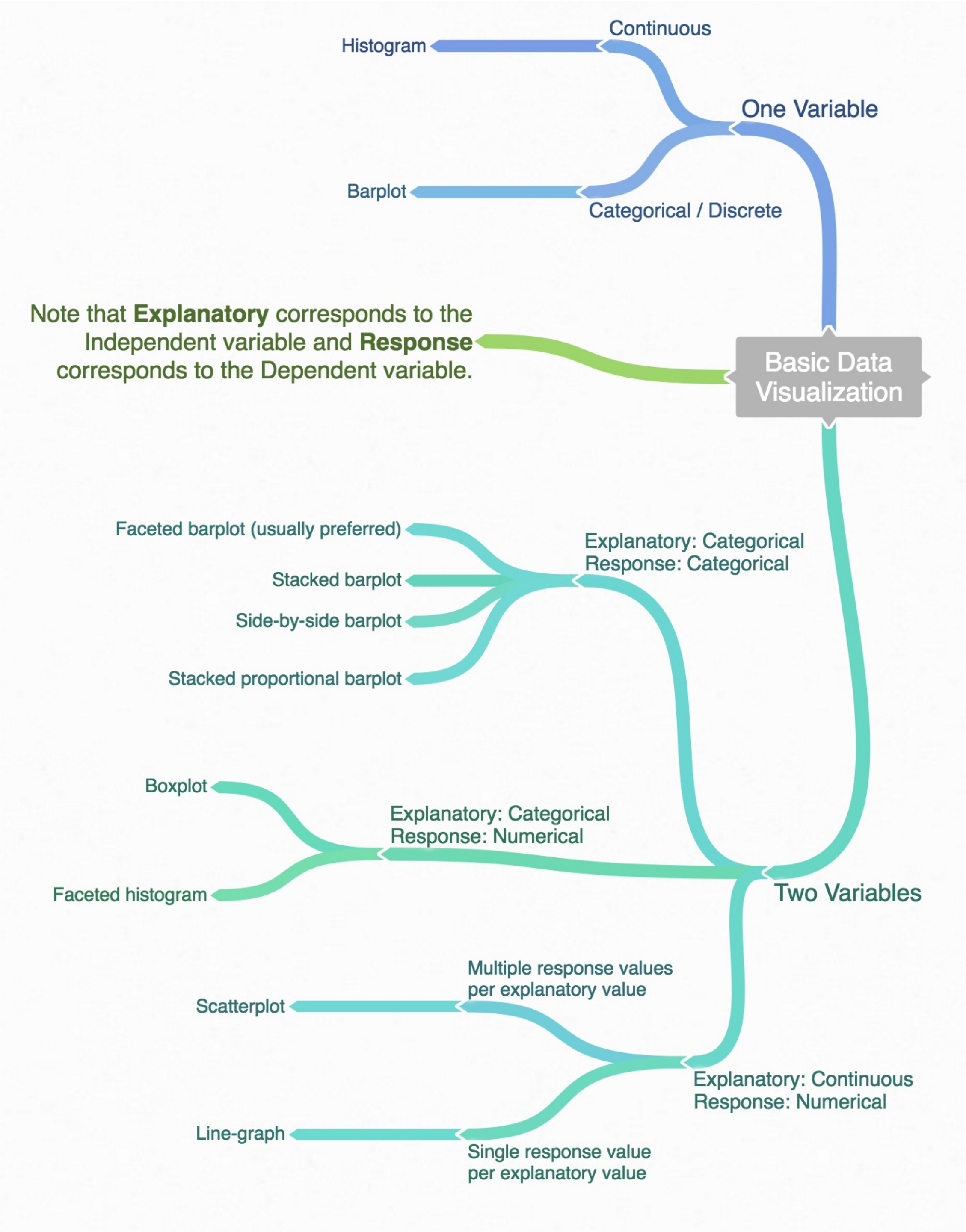
Problem 3
rating
10.0
7.5
5.0
2.5
0
50000000
100000000
150000000
budget



Problem 4
count
600
400
200
0
4
5
6
7
8
log(budget_2013, 10)

DEMO of `ggplot2` in RStudio

Determining the appropriate plot



Day 2

Data Wrangling

`gapminder` data frame in the `gapminder` package

```r
library(gapminder)
gapminder
```

```
# A tibble: 1,704 x 6
       country continent  year lifeExp      pop gdpPercap
        <fctr>    <fctr> <int>   <dbl>    <int>     <dbl>
 1 Afghanistan      Asia  1952  28.801  8425333  779.4453
 2 Afghanistan      Asia  1957  30.332  9240934  820.8530
 3 Afghanistan      Asia  1962  31.997 10267083  853.1007
 4 Afghanistan      Asia  1967  34.020 11537966  836.1971
 5 Afghanistan      Asia  1972  36.088 13079460  739.9811
 6 Afghanistan      Asia  1977  38.438 14880372  786.1134
 7 Afghanistan      Asia  1982  39.854 12881816  978.0114
 8 Afghanistan      Asia  1987  40.822 13867957  852.3959
 9 Afghanistan      Asia  1992  41.674 16317921  649.3414
10 Afghanistan      Asia  1997  41.763 22227415  635.3414
# ... with 1,694 more rows
```

Base R versus the `tidyverse`

Say we wanted mean life expectancy across all years for Asia

Base R versus the `tidyverse`

Say we wanted mean life expectancy across all years for Asia

```
# Base R
asia <- gapminder[gapminder$continent == "Asia", ]
mean(asia$lifeExp)
```

```
[1] 60.0649
```

Base R versus the `tidyverse`

Say we wanted mean life expectancy across all years for Asia

```
# Base R
asia <- gapminder[gapminder$continent == "Asia", ]
mean(asia$lifeExp)
```

```
[1] 60.0649
```

```
library(dplyr)
gapminder %>% filter(continent == "Asia") %>%
  summarize(mean_exp = mean(lifeExp))
```

```
# A tibble: 1 x 1
  mean_exp
     <dbl>
1  60.0649
```

The pipe %>%

The pipe `%>%`

- A way to chain together commands

The pipe `%>%`

- A way to chain together commands
- It is essentially the `dplyr` equivalent to the `+` in `ggplot2`

The 5NG of data viz

The 5NG of data viz

```
geom_point()
geom_line()
geom_histogram()
geom_boxplot()
geom_bar()
```

The Five Main Verbs (5MV) of data wrangling

```
filter()
summarize()
group_by()
mutate()
arrange()
```

`filter()`

- Select a subset of the rows of a data frame.
- The arguments are the "filters" that you'd like to apply.

`filter()`

- Select a subset of the rows of a data frame.
- The arguments are the "filters" that you'd like to apply.

```r
library(gapminder); library(dplyr)
gap_2007 <- gapminder %>% filter(year == 2007)
head(gap_2007)
```

```
# A tibble: 6 x 6
      country continent  year lifeExp      pop  gdpPercap
       <fctr>    <fctr> <int>   <dbl>    <int>      <dbl>
1 Afghanistan      Asia  2007  43.828 31889923   974.5803
2     Albania    Europe  2007  76.423  3600523  5937.0295
3     Algeria    Africa  2007  72.301 33333216  6223.3675
4      Angola    Africa  2007  42.731 12420476  4797.2313
5   Argentina  Americas  2007  75.320 40301927 12779.3796
6   Australia   Oceania  2007  81.235 20434176 34435.3674
```

- Use == to compare a variable to a value

Logical operators

- Use `|` to check for any in multiple filters being true:

Logical operators

- Use `|` to check for any in multiple filters being true:

```
gapminder %>%
  filter(year == 2002 | continent == "Europe")
```

Logical operators

- Use | to check for any in multiple filters being true:

```
gapminder %>%
  filter(year == 2002 | continent == "Europe")
```

```
# A tibble: 472 x 6
       country continent  year lifeExp      pop gdpPercap
        <fctr>    <fctr> <int>   <dbl>    <int>     <dbl>
 1 Afghanistan      Asia  2002  42.129 25268405  726.7341
 2     Albania    Europe  1952  55.230  1282697 1601.0561
 3     Albania    Europe  1957  59.280  1476505 1942.2842
 4     Albania    Europe  1962  64.820  1728137 2312.8890
 5     Albania    Europe  1967  66.220  1984060 2760.1969
 6     Albania    Europe  1972  67.690  2263554 3313.4222
 7     Albania    Europe  1977  68.930  2509048 3533.0039
 8     Albania    Europe  1982  70.420  2780097 3630.8807
 9     Albania    Europe  1987  72.000  3075321 3738.9327
10     Albania    Europe  1992  71.581  3326498 2497.4379
# ... with 462 more rows
```

Logical operators

- Use `&` or `,` to check for all of multiple filters being true:

Logical operators

- Use & or , to check for all of multiple filters being true:

```
gapminder %>%
  filter(year == 2002, continent == "Europe")
```

```
# A tibble: 30 x 6
                  country continent  year lifeExp      pop gdpPercap
                   <fctr>    <fctr> <int>   <dbl>    <int>     <dbl>
 1                Albania    Europe  2002  75.651  3508512  4604.212
 2                Austria    Europe  2002  78.980  8148312 32417.608
 3                Belgium    Europe  2002  78.320 10311970 30485.884
 4 Bosnia and Herzegovina    Europe  2002  74.090  4165416  6018.975
 5               Bulgaria    Europe  2002  72.140  7661799  7696.778
 6                Croatia    Europe  2002  74.876  4481020 11628.389
 7         Czech Republic    Europe  2002  75.510 10256295 17596.210
 8                Denmark    Europe  2002  77.180  5374693 32166.500
 9                Finland    Europe  2002  78.370  5193039 28204.591
10                 France    Europe  2002  79.590 59925035 28926.032
# ... with 20 more rows
```

Logical operators

- Use `%in%` to check for any being true (shortcut to using `|` repeatedly with `==`)

Logical operators

- Use `%in%` to check for any being true (shortcut to using `|` repeatedly with `==`)

```
gapminder %>%
  filter(country %in% c("Argentina", "Belgium", "Mexico"),
         year %in% c(1987, 1992))
```

Logical operators

- Use `%in%` to check for any being true (shortcut to using `|` repeatedly with `==`)

```
gapminder %>%
  filter(country %in% c("Argentina", "Belgium", "Mexico"),
         year %in% c(1987, 1992))
```

```
# A tibble: 6 x 6
    country continent  year lifeExp      pop gdpPercap
     <fctr>    <fctr> <int>   <dbl>    <int>     <dbl>
1 Argentina  Americas  1987  70.774 31620918  9139.671
2 Argentina  Americas  1992  71.868 33958947  9308.419
3   Belgium    Europe  1987  75.350  9870200 22525.563
4   Belgium    Europe  1992  76.460 10045622 25575.571
5    Mexico  Americas  1987  69.498 80122492  8688.156
6    Mexico  Americas  1992  71.455 88111030  9472.384
```

`summarize()`

- Any numerical summary that you want to apply to a column of a data frame is specified within `summarize()`.

```
max_exp_1997 <- gapminder %>%
  filter(year == 1997) %>%
  summarize(max_exp = max(lifeExp))
max_exp_1997
```

`summarize()`

- Any numerical summary that you want to apply to a column of a data frame is specified within `summarize()`.

```
max_exp_1997 <- gapminder %>%
  filter(year == 1997) %>%
  summarize(max_exp = max(lifeExp))
max_exp_1997
```

```
# A tibble: 1 x 1
  max_exp
    <dbl>
1   80.69
```

Combining `summarize()` with `group_by()`

When you'd like to determine a numerical summary for all levels of a different categorical variable

```
max_exp_1997_by_cont <- gapminder %>%
  filter(year == 1997) %>%
  group_by(continent) %>%
  summarize(max_exp = max(lifeExp))
max_exp_1997_by_cont
```

Combining `summarize()` with `group_by()`

When you'd like to determine a numerical summary for all levels of a different categorical variable

```
max_exp_1997_by_cont <- gapminder %>%
  filter(year == 1997) %>%
  group_by(continent) %>%
  summarize(max_exp = max(lifeExp))
max_exp_1997_by_cont
```

```
# A tibble: 5 x 2
  continent max_exp
     <fctr>   <dbl>
1    Africa  74.772
2  Americas  78.610
3      Asia  80.690
4    Europe  79.390
5   Oceania  78.830
```

Without the %>%

It's hard to appreciate the %>% without seeing what the code would look like without it:

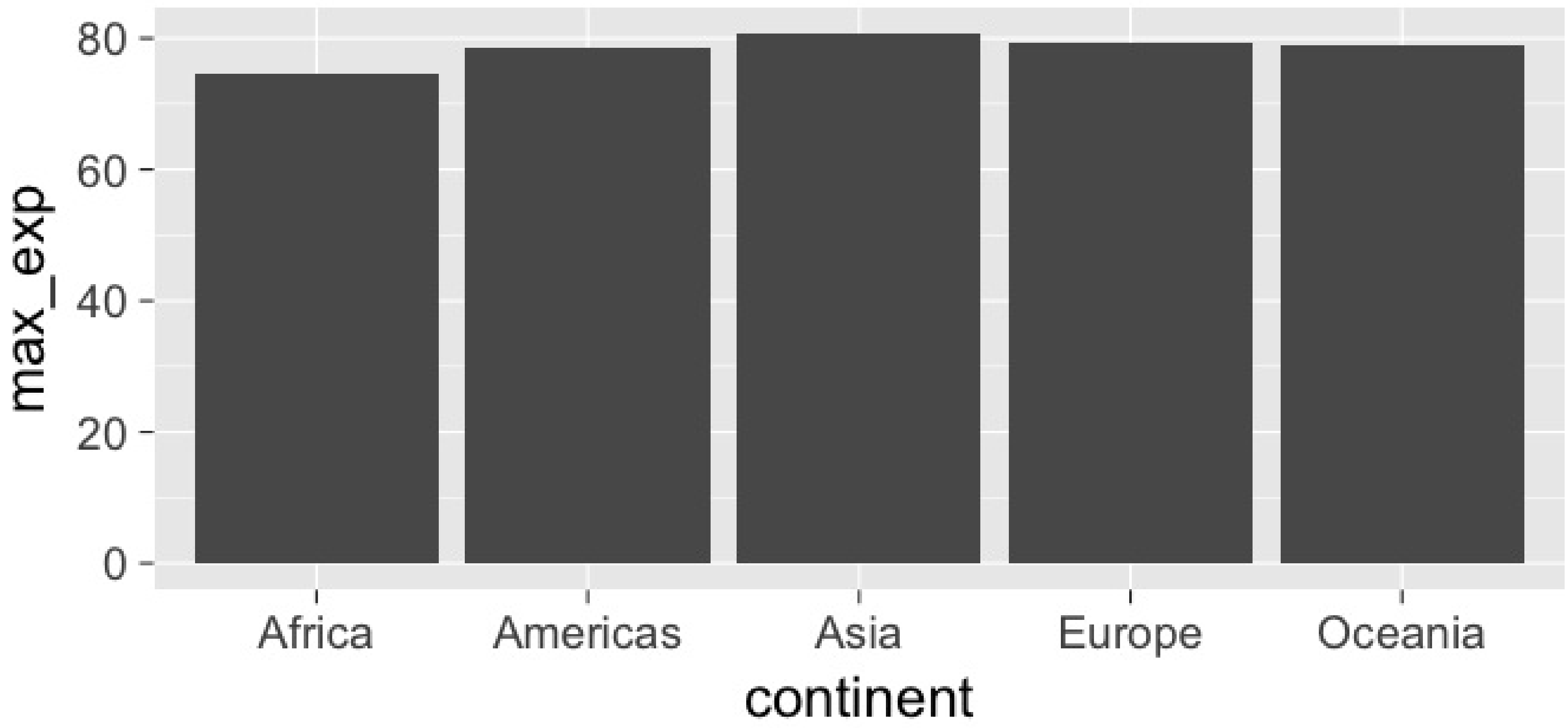
```
max_exp_1997_by_cont <-
  summarize(
    group_by(
      filter(
        gapminder,
          year == 1997),
      continent),
    max_exp = max(lifeExp))
max_exp_1997_by_cont
```

```
# A tibble: 5 x 2
  continent max_exp
     <fctr>   <dbl>
1    Africa  74.772
2  Americas  78.610
3      Asia  80.690
4    Europe  79.390
5   Oceania  78.830
```

`ggplot2` revisited

For aggregated data, use `geom_col`

```
ggplot(data = max_exp_1997_by_cont,
       mapping = aes(x = continent, y = max_exp)) +
  geom_col()
```



The 5MV

- `filter()`
- `summarize()`
- `group_by()`

The 5MV

- `filter()`
- `summarize()`
- `group_by()`
- `mutate()`

The 5MV

- `filter()`
- `summarize()`
- `group_by()`
- `mutate()`
- `arrange()`

`mutate()`

- Allows you to
    1. create a new variable with a specific value OR
    2. create a new variable based on other variables OR
    3. change the contents of an existing variable

mutate()

- Allows you to
  1. create a new variable with a specific value OR
  2. create a new variable based on other variables OR
  3. change the contents of an existing variable

```
gap_plus <- gapminder %>% mutate(just_one = 1)
head(gap_plus)
```

```
# A tibble: 6 x 7
      country continent  year lifeExp      pop gdpPercap just_one
       <fctr>    <fctr> <int>   <dbl>    <int>     <dbl>    <dbl>
1 Afghanistan      Asia  1952  28.801  8425333  779.4453        1
2 Afghanistan      Asia  1957  30.332  9240934  820.8530        1
3 Afghanistan      Asia  1962  31.997 10267083  853.1007        1
4 Afghanistan      Asia  1967  34.020 11537966  836.1971        1
5 Afghanistan      Asia  1972  36.088 13079460  739.9811        1
6 Afghanistan      Asia  1977  38.438 14880372  786.1134        1
```

`mutate()`

- Allows you to
  1. create a new variable with a specific value OR
  2. create a new variable based on other variables OR
  3. change the contents of an existing variable

mutate()

- Allows you to
  1. create a new variable with a specific value OR
  2. create a new variable based on other variables OR
  3. change the contents of an existing variable

```
gap_w_gdp <- gapminder %>% mutate(gdp = pop * gdpPercap)
head(gap_w_gdp)
```

```
# A tibble: 6 x 7
      country continent  year lifeExp      pop gdpPercap         gdp
       <fctr>    <fctr> <int>   <dbl>    <int>     <dbl>       <dbl>
1 Afghanistan      Asia  1952  28.801  8425333  779.4453  6567086330
2 Afghanistan      Asia  1957  30.332  9240934  820.8530  7585448670
3 Afghanistan      Asia  1962  31.997 10267083  853.1007  8758855797
4 Afghanistan      Asia  1967  34.020 11537966  836.1971  9648014150
5 Afghanistan      Asia  1972  36.088 13079460  739.9811  9678553274
6 Afghanistan      Asia  1977  38.438 14880372  786.1134 11697659231
```

`mutate()`

- Allows you to
    1. create a new variable with a specific value OR
    2. create a new variable based on other variables OR
    3. change the contents of an existing variable

mutate()

- Allows you to
  1. create a new variable with a specific value OR
  2. create a new variable based on other variables OR
  3. change the contents of an existing variable

```
gap_weird <- gapminder %>% mutate(pop = pop + 1000)
head(gap_weird)
```

```
# A tibble: 6 x 6
      country continent  year lifeExp      pop gdpPercap
       <fctr>    <fctr> <int>   <dbl>    <dbl>     <dbl>
1 Afghanistan      Asia  1952  28.801  8426333  779.4453
2 Afghanistan      Asia  1957  30.332  9241934  820.8530
3 Afghanistan      Asia  1962  31.997 10268083  853.1007
4 Afghanistan      Asia  1967  34.020 11538966  836.1971
5 Afghanistan      Asia  1972  36.088 13080460  739.9811
6 Afghanistan      Asia  1977  38.438 14881372  786.1134
```

`arrange()`

- Reorders the rows in a data frame based on the values of one or more variables

arrange()

- Reorders the rows in a data frame based on the values of one or more variables

```
gapminder %>%
  arrange(year, country)
```

```
# A tibble: 1,704 x 6
       country continent  year lifeExp      pop  gdpPercap
        <fctr>    <fctr> <int>   <dbl>    <int>      <dbl>
 1 Afghanistan      Asia  1952  28.801  8425333   779.4453
 2     Albania    Europe  1952  55.230  1282697  1601.0561
 3     Algeria    Africa  1952  43.077  9279525  2449.0082
 4      Angola    Africa  1952  30.015  4232095  3520.6103
 5   Argentina  Americas  1952  62.485 17876956  5911.3151
 6   Australia   Oceania  1952  69.120  8691212 10039.5956
 7     Austria    Europe  1952  66.800  6927772  6137.0765
 8     Bahrain      Asia  1952  50.939   120447  9867.0848
 9  Bangladesh      Asia  1952  37.484 46886859   684.2442
10     Belgium    Europe  1952  68.000  8730405  8343.1051
# ... with 1,694 more rows
```

`arrange()`

- Can also put into descending order

arrange()

- Can also put into descending order

```
gapminder %>%
  filter(year > 2000) %>%
  arrange(desc(lifeExp))
```

```
# A tibble: 284 x 6
         country continent  year lifeExp       pop gdpPercap
          <fctr>    <fctr> <int>   <dbl>     <int>     <dbl>
 1         Japan      Asia  2007  82.603 127467972  31656.07
 2 Hong Kong, China   Asia  2007  82.208   6980412  39724.98
 3         Japan      Asia  2002  82.000 127065841  28604.59
 4       Iceland    Europe  2007  81.757    301931  36180.79
 5   Switzerland    Europe  2007  81.701   7554661  37506.42
 6 Hong Kong, China   Asia  2002  81.495   6762476  30209.02
 7     Australia   Oceania  2007  81.235  20434176  34435.37
 8         Spain    Europe  2007  80.941  40448191  28821.06
 9        Sweden    Europe  2007  80.884   9031088  33859.75
10        Israel      Asia  2007  80.745   6426679  25523.28
# ... with 274 more rows
```

Don't mix up `arrange` and `group_by`

- `group_by` is used (mostly) with `summarize` to calculate summaries over groups
- `arrange` is used for sorting

Don't mix up `arrange` and `group_by`

This doesn't really do anything useful

```
gapminder %>% group_by(year)
```

```
Source: local data frame [1,704 x 6]
Groups: year [12]

# A tibble: 1,704 x 6
       country continent  year lifeExp      pop gdpPercap
        <fctr>    <fctr> <int>   <dbl>    <int>     <dbl>
 1 Afghanistan      Asia  1952  28.801  8425333  779.4453
 2 Afghanistan      Asia  1957  30.332  9240934  820.8530
 3 Afghanistan      Asia  1962  31.997 10267083  853.1007
 4 Afghanistan      Asia  1967  34.020 11537966  836.1971
 5 Afghanistan      Asia  1972  36.088 13079460  739.9811
 6 Afghanistan      Asia  1977  38.438 14880372  786.1134
 7 Afghanistan      Asia  1982  39.854 12881816  978.0114
 8 Afghanistan      Asia  1987  40.822 13867957  852.3959
 9 Afghanistan      Asia  1992  41.674 16317921  649.3414
10 Afghanistan      Asia  1997  41.763 22227415  635.3414
# ... with 1,694 more rows
```

Don't mix up `arrange` and `group_by`

But this does

```
gapminder %>% arrange(year)
```

```
# A tibble: 1,704 x 6
       country continent  year lifeExp      pop  gdpPercap
        <fctr>    <fctr> <int>   <dbl>    <int>      <dbl>
 1 Afghanistan      Asia  1952  28.801  8425333   779.4453
 2     Albania    Europe  1952  55.230  1282697  1601.0561
 3     Algeria    Africa  1952  43.077  9279525  2449.0082
 4      Angola    Africa  1952  30.015  4232095  3520.6103
 5   Argentina  Americas  1952  62.485 17876956  5911.3151
 6   Australia   Oceania  1952  69.120  8691212 10039.5956
 7     Austria    Europe  1952  66.800  6927772  6137.0765
 8     Bahrain      Asia  1952  50.939   120447  9867.0848
 9  Bangladesh      Asia  1952  37.484 46886859   684.2442
10     Belgium    Europe  1952  68.000  8730405  8343.1051
# ... with 1,694 more rows
```

Changing of observation unit

True or False

> Each of `filter`, `mutate`, and `arrange` change the observational unit.

Changing of observation unit

True or False

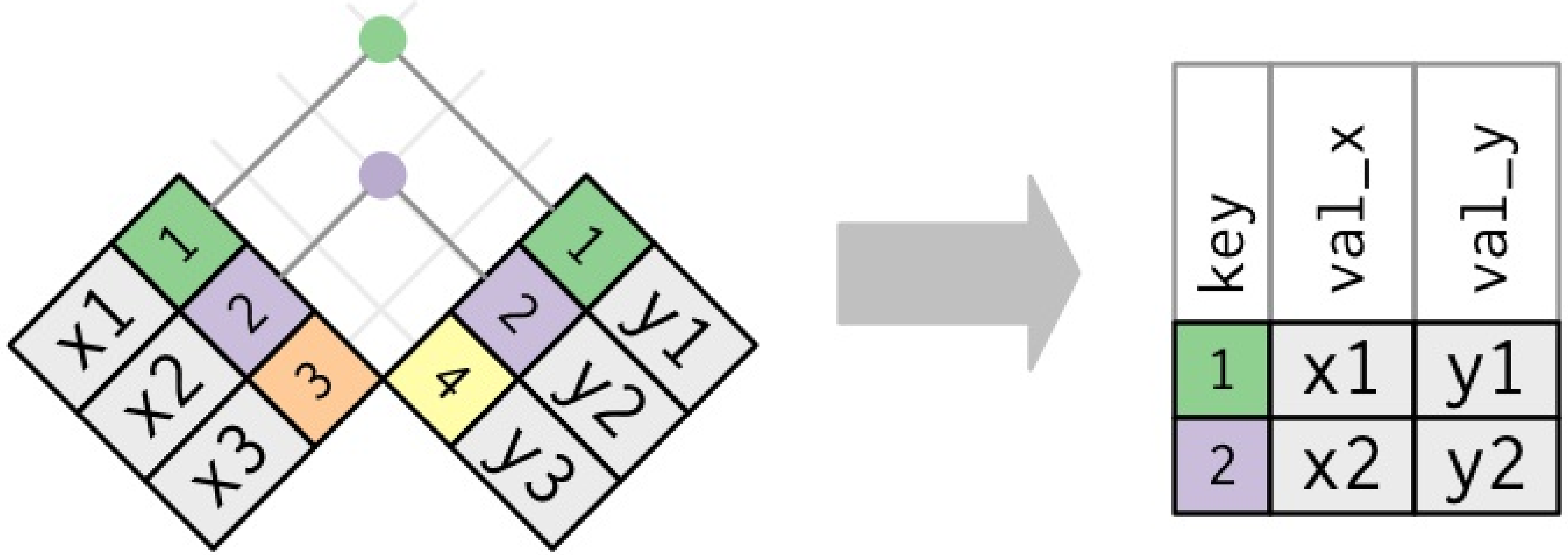
Each of `filter`, `mutate`, and `arrange` change the observational unit.

True or False

`group_by() %>% summarize()` changes the observational unit.

What is meant by "joining data frames" and why is it useful?

What is meant by "joining data frames" and why is it useful?



Does cost of living in a state relate to whether police officers live in the cities they patrol? What about state political ideology?

```
library(fivethirtyeight); library(readr); library(dplyr)
police2 <- police_locals %>% select(city, all)
ideology <- read_csv("https://ismayc.github.io/Effective-Data-Storytelling-us
police_join <- inner_join(x = police2, y = ideology, by = "city")
police_join
```

Does cost of living in a state relate to whether police officers live in the cities they patrol? What about state political ideology?

```
library(fivethirtyeight); library(readr); library(dplyr)
police2 <- police_locals %>% select(city, all)
ideology <- read_csv("https://ismayc.github.io/Effective-Data-Storytelling-us
police_join <- inner_join(x = police2, y = ideology, by = "city")
police_join
```

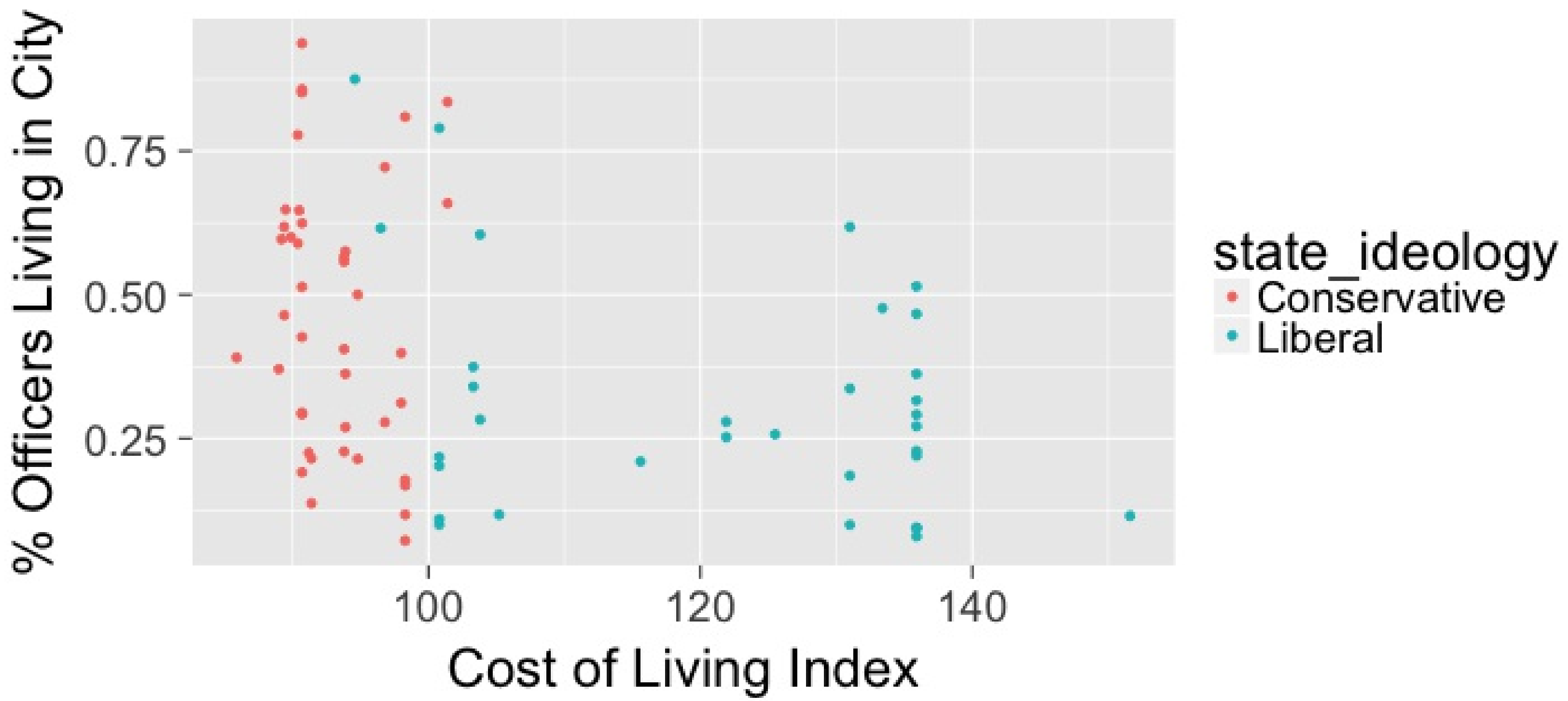
```
# A tibble: 75 x 4
           city       all                state state_ideology
          <chr>     <dbl>                <chr>          <chr>
1      New York 0.6179567             New York        Liberal
2       Chicago 0.8750000             Illinois        Liberal
3   Los Angeles 0.2282178           California        Liberal
4    Washington 0.1156317 District of Columbia        Liberal
5       Houston 0.2922078                Texas   Conservative
6  Philadelphia 0.8354012         Pennsylvania   Conservative
7       Phoenix 0.3117318              Arizona   Conservative
8     San Diego 0.3621076           California        Liberal
9        Dallas 0.1914008                Texas   Conservative
10      Detroit 0.3705972             Michigan   Conservative
# ... with 65 more rows
```

```r
url <- paste0("https://ismayc.github.io/",
              "Effective-Data-Storytelling-using-the-tidyverse/",
              "datasets/cost_of_living.csv")
cost_of_living <- read_csv(url)
police_join_cost <- inner_join(
    x = police_join,
    y = cost_of_living,
    by = "state"
  )
police_join_cost %>% select(-state) %>% arrange(city)
```

```
# A tibble: 75 x 5
                 city       all state_ideology index col_group
                <chr>     <dbl>          <chr> <dbl>     <chr>
 1       Albany, N.Y. 0.1853933        Liberal 131.0      high
 2   Albuquerque, N.M. 0.6156716       Liberal  96.5       mid
 3     Arlington, Va. 0.2022059        Liberal 100.8       mid
 4            Atlanta 0.1372881   Conservative  91.4       low
 5      Austin, Texas 0.2947103   Conservative  90.7       low
 6          Baltimore 0.2571429        Liberal 125.5      high
 7    Baton Rouge, La. 0.2142857  Conservative  94.8       low
 8    Birmingham, Ala. 0.2252252  Conservative  91.2       low
 9             Boston 0.4765625        Liberal 133.4      high
10 Brownsville, Texas 0.5135135   Conservative  90.7       low
# ... with 65 more rows
```

Does cost of living in a state relate to whether police officers live in the cities they patrol? What about state political ideology?

```
ggplot(data = police_join_cost,
       mapping = aes(x = index, y = all)) +
  geom_point(mapping = aes(color = state_ideology)) +
  labs(x = "Cost of Living Index", y = "% Officers Living in City")
```



Practice (Lay out what the resulting table should look like on paper first.)

Use the 5MV to answer problems from R data packages, e.g., [`nycflights13` → `weather`]

1. What is the maximum arrival delay for each carrier departing JFK? [`nycflights13` → `flights`]
2. Determine the top five movies in terms of domestic return on investment for 2013 scaled data [`fivethirtyeight` → `bechdel`]
3. Include the name of the destination airport as a column in the `flights` data frame [`nycflights13` → `flights`, `airports`]

DEMO of `dplyr` in RStudio

Data Importing and Tidying

`tidyverse` packages

IMPORT

- `haven` for SPSS, SAS, and Stata data files
- `jsonlite` for JSON files
- `readxl` for XLS and XLSX files
- `readr` for CSV and TSV files (and R specific RDS files)

TIDYING

- `tidyr` for converting "messy" into "tidy" data frames

Basics of Importing

We will begin by downloading and importing a variety of different "messy" data sets. You can download all of them in a ZIP file at http://bit.ly/reed-messy-data. The links below go to the original sources. I've converted these original sources into different formats.

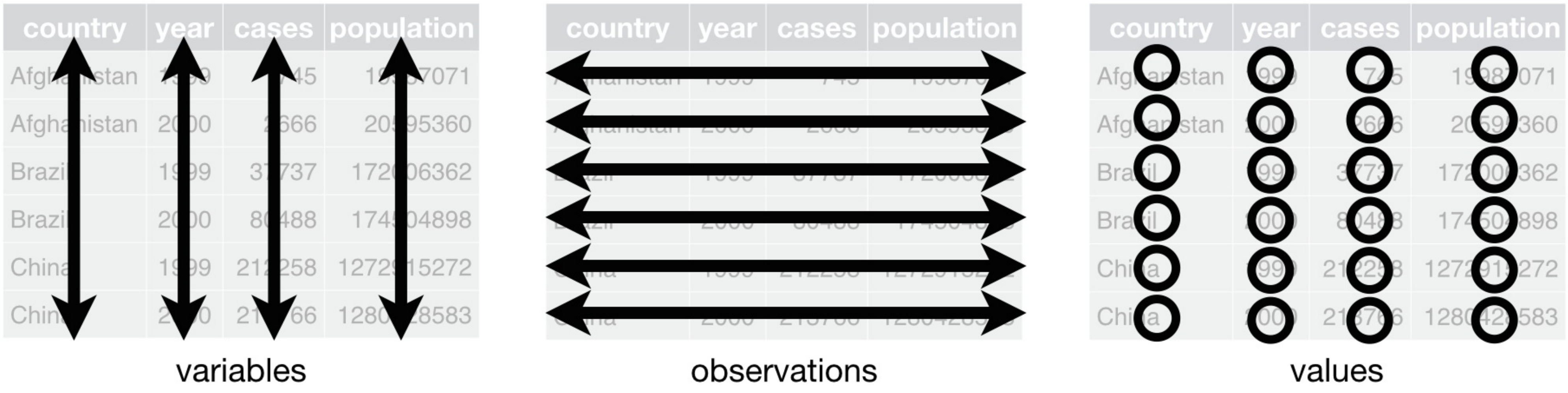
- Life Expectancy by year for countries since 1951 (CSV)
- World Health Organization TB data (Stata DTA)
- Annual Estimates of Population by Age, Sex, Race, and Hispanic Origin for Oregon Counties (XLSX)
- Educational attainment for the U.S., States, and counties, 1970-2014 (JSON)
- County level results for 2012 POTUS election from The Guardian (SPSS SAV)

Demonstration in RStudio

Practice

- Download the four other files and import them into R using the appropriate package
  - You may need to check out the help pages for the different packages
    - `haven` for SPSS, SAS, and Stata data files
    - `jsonlite` for JSON files
    - `readxl` for XLS and XLSX files
    - `readr` for CSV/TSV files (& R specific RDS files)
  - Give them the following names: `who_df`, `county_pop_df`, `edu_county_df`, and `potus12_df`

Tidy Data



1. Each variable forms a column.
2. Each observation forms a row.
3. Each type of observational unit forms a table.

The third point means we don't mix apples and oranges.

What is Tidy Data?

1. Each observation forms a row. In other words, each row corresponds to a single instance of an observational unit
2. Each variable forms a column:
   - Some variables may be used to identify the observational units.
   - For organizational purposes, it's generally better to put these in the left-hand columns
3. Each type of observational unit forms a table with the entries in the table corresponding to values.

Differentiating between <u>neat</u> data and <u>tidy</u> data

- Colloquially, they mean the same thing
- But in our context, one is a subset of the other.

<u>Neat</u> data is

- easy to look at,
- organized nicely, and
- in table form.

Differentiating between neat data and tidy data

- Colloquially, they mean the same thing
- But in our context, one is a subset of the other.

Neat data is

- easy to look at,
- organized nicely, and
- in table form.

Tidy data is neat but also abides by a set of three rules.

variables observations values

Is this tidy?

year	title	clean_test	budget_2013
1995	Apollo 13	ok	99370665
2005	Brokeback Mountain	notalk	16583160
2010	Diary of a Wimpy Kid	ok	16023478
1984	Dune	dubious	100864980
1984	Ghostbusters	notalk	67243320
2003	How to Lose a Guy in 10 Days	men	63304348
2011	Iris	ok	5696299
2004	Sideways	ok	20964279
2000	Songcatcher	ok	2435235
2004	Team America: World Police	men	24663858
2010	Tron Legacy	notalk	213646368
2011	War Horse	notalk	72498355

How about this? Is this tidy?

country	1952	1957	1962	1967	1972	1977	1982	1987	1992	1997	2002	2007
Albania	-9	-9	-9	-9	-9	-9	-9	-9	5	5	7	9
Argentina	-9	-1	-1	-9	-9	-9	-8	8	7	7	8	8
Armenia	-9	-7	-7	-7	-7	-7	-7	-7	7	-6	5	5
Australia	10	10	10	10	10	10	10	10	10	10	10	10
Austria	10	10	10	10	10	10	10	10	10	10	10	10
Azerbaijan	-9	-7	-7	-7	-7	-7	-7	-7	1	-6	-7	-7
Belarus	-9	-7	-7	-7	-7	-7	-7	-7	7	-7	-7	-7
Belgium	10	10	10	10	10	10	10	10	10	10	10	8
Bhutan	-10	-10	-10	-10	-10	-10	-10	-10	-10	-10	-10	-6
Bolivia	-4	-3	-3	-4	-7	-7	8	9	9	9	9	8
Brazil	5	5	5	-9	-9	-4	-3	7	8	8	8	8
Bulgaria	-7	-7	-7	-7	-7	-7	-7	-7	8	8	9	9

Why is tidy data important?

- Think about trying to plot democracy score across years in the simplest way possible.

Why is tidy data important?

- Think about trying to plot democracy score across years in the simplest way possible.
- It would be much easier if the data looked like what follows instead so we could put
  - `year` on the `x`-axis and
  - `dem_score` on the `y`-axis.

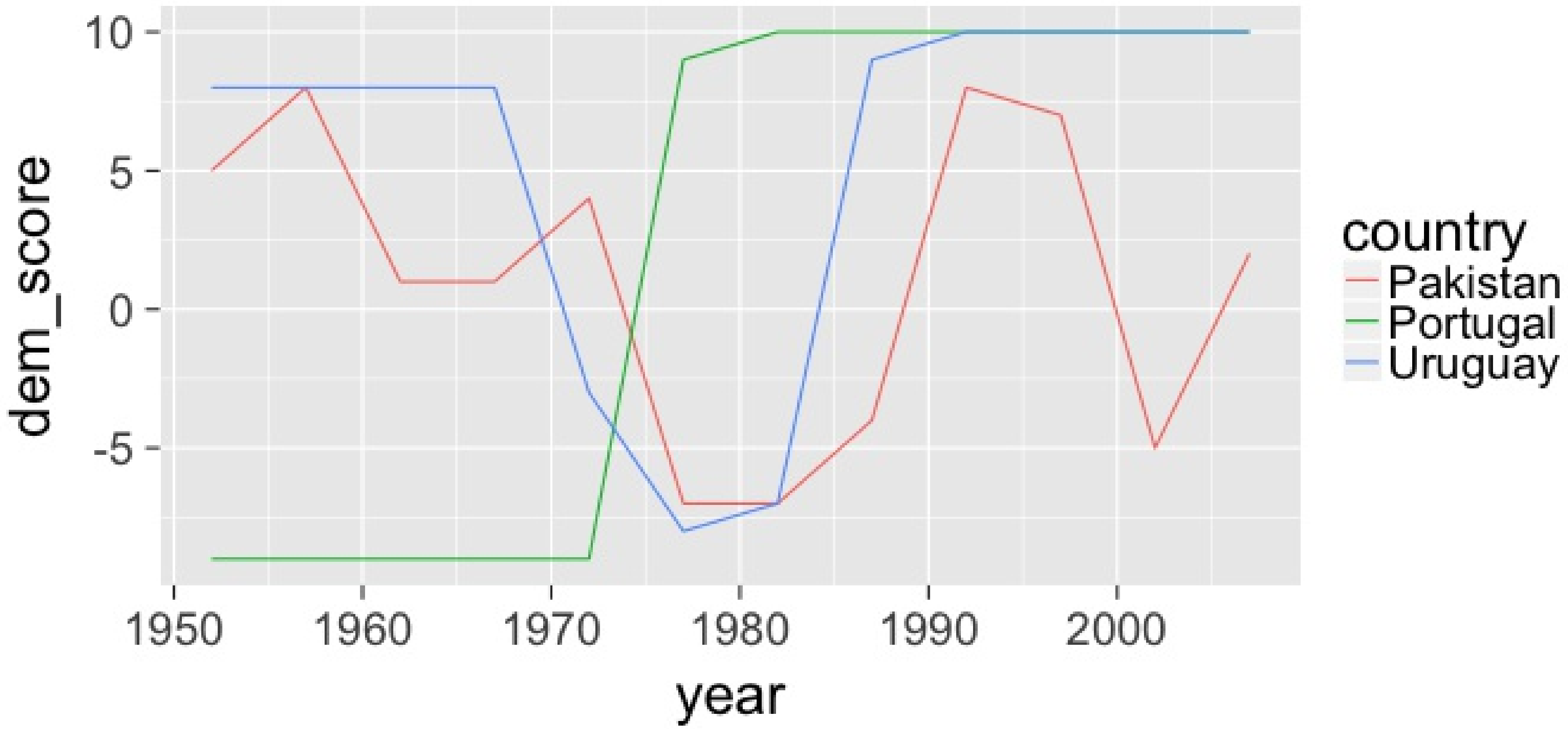
Tidy is good

country	year	dem_score
Argentina	1962	-1
Armenia	1997	-6
Denmark	1962	10
Ethiopia	2007	1
Finland	2007	10
Ireland	1992	10
Libya	1957	-7
Libya	1982	-7
Mexico	1977	-3
Mexico	1982	-3
Spain	1962	-7
Switzerland	1982	10
Ukraine	1997	7

Let's plot it

- Plot the line graph for three countries using `ggplot`

```
dem_score4 <- dem_score_tidy %>%
  filter(country %in% c("Pakistan", "Portugal", "Uruguay"))
ggplot(data = dem_score4, mapping = aes(x = year, y = dem_score)) +
  geom_line(mapping = aes(color = country))
```



Tidying

The Life Expectancy by year data

```
library(readr)
life_exp_df <- read_csv("data/le_mess.csv")
#View(life_exp_df)
```

	country	1951	1952	1953
1	Afghanistan	27.13	27.67	28.19
2	Albania	54.72	55.23	55.85
3	Algeria	43.03	43.50	43.96
4	Angola	31.05	31.59	32.14
5	Antigua and Barbuda	58.26	58.80	59.34

Doing the "tidying"/reshaping

```r
library(tidyr)
library(dplyr)
(life_exp_tidy <- life_exp_df %>%
    gather(key = "year", value = "life_exp", -country))
```

```
# A tibble: 13,332 x 3
               country  year life_exp
                 <chr> <chr>    <dbl>
 1         Afghanistan  1951    27.13
 2             Albania  1951    54.72
 3             Algeria  1951    43.03
 4              Angola  1951    31.05
 5 Antigua and Barbuda  1951    58.26
 6           Argentina  1951    61.93
 7             Armenia  1951    62.67
 8               Aruba  1951    58.96
 9           Australia  1951    68.71
10             Austria  1951    65.24
# ... with 13,322 more rows
```

Tidying

World Health Organization TB data (Stata DTA)

```
library(haven)
who_df <- read_dta("data/who.dta")
#View(who_df)
```

	country	iso2	iso3	year	new_sp_m014	new_sp_m1524	new_sp_m2534	new_sp_m3544
1	Afghanistan	AF	AFG	1980	NaN	NaN	NaN	NaN
2	Afghanistan	AF	AFG	1981	NaN	NaN	NaN	NaN
3	Afghanistan	AF	AFG	1982	NaN	NaN	NaN	NaN
4	Afghanistan	AF	AFG	1983	NaN	NaN	NaN	NaN
5	Afghanistan	AF	AFG	1984	NaN	NaN	NaN	NaN
6	Afghanistan	AF	AFG	1985	NaN	NaN	NaN	NaN
7	Afghanistan	AF	AFG	1986	NaN	NaN	NaN	NaN
8	Afghanistan	AF	AFG	1987	NaN	NaN	NaN	NaN
9	Afghanistan	AF	AFG	1988	NaN	NaN	NaN	NaN
10	Afghanistan	AF	AFG	1989	NaN	NaN	NaN	NaN

WHO ugly...

- This data set contains strange variable names that will require us to look up their meaning in the corresponding data dictionary.

WHO ugly...

- This data set contains strange variable names that will require us to look up their meaning in the corresponding data dictionary.
- What do we know?
    - `country`, `iso2`, and `iso3` are redundant and identify the country
    - `year` is a variable and it looks like it corresponds to each country
- But what in the world is `new_sp_m014`? And the other columns?...

First step

Like before, we can `gather` these non-`country` and non-`year` variables together:

```
who_tidy <- who_df %>%
  gather(new_sp_m014:newrel_f65, key = "key", value = "value")
```

We can now see what this has done to the data frame:

```
View(who_tidy)
```

Data dictionary saves us some...

1. The first three letters of entries in the `key` column correspond to `new` or `old` cases of TB.
2. The next two letters (after the _) correspond to TB type:
    - `rel` for relapse, `ep` for extrapulmonary TB
    - `sn` for smear negative, `sp` for smear positive
3. The next letter after the second _ corresponds to the sex of the TB patient.
4. The remaining numbers correspond to age group:
    - `014` for 0 to 14 years
    - ...
    - `65` for 65 or older

- Looking over the entries of `key` in `who_tidy`, do you see anything else that doesn't match the pattern?
- It may be easier to remember that the entries of `key` correspond to column names in `who_df`:

```
names(who_df)
```

```
 [1] "country"      "iso2"         "iso3"         "year"        
 [5] "new_sp_m014"  "new_sp_m1524" "new_sp_m2534" "new_sp_m3544"
 [9] "new_sp_m4554" "new_sp_m5564" "new_sp_m65"   "new_sp_f014" 
[13] "new_sp_f1524" "new_sp_f2534" "new_sp_f3544" "new_sp_f4554"
[17] "new_sp_f5564" "new_sp_f65"   "new_sn_m014"  "new_sn_m1524"
[21] "new_sn_m2534" "new_sn_m3544" "new_sn_m4554" "new_sn_m5564"
[25] "new_sn_m65"   "new_sn_f014"  "new_sn_f1524" "new_sn_f2534"
[29] "new_sn_f3544" "new_sn_f4554" "new_sn_f5564" "new_sn_f65"  
[33] "new_ep_m014"  "new_ep_m1524" "new_ep_m2534" "new_ep_m3544"
[37] "new_ep_m4554" "new_ep_m5564" "new_ep_m65"   "new_ep_f014" 
[41] "new_ep_f1524" "new_ep_f2534" "new_ep_f3544" "new_ep_f4554"
[45] "new_ep_f5564" "new_ep_f65"   "newrel_m014"  "newrel_m1524"
[49] "newrel_m2534" "newrel_m3544" "newrel_m4554" "newrel_m5564"
[53] "newrel_m65"   "newrel_f014"  "newrel_f1524" "newrel_f2534"
[57] "newrel_f3544" "newrel_f4554" "newrel_f5564" "newrel_f65"  
```

A fix using `stringr`

You can replace all of the entries in `key` that contained `newrel` with `new_rel` via

```
library(stringr)
library(dplyr)
who_tidy <- who_tidy %>%
  mutate(key = str_replace(
      string = key,
      pattern = "newrel",
      replacement = "new_rel")
    )
```

Pulling apart variables

The entry `new_sp_m014` is actually four different variables. Use the `separate` function to pull them apart:

```
who_tidy <- who_tidy %>%
  separate(col = key, into = c("new", "type", "sexage"), sep = "_")
who_tidy
```

```
# A tibble: 405,440 x 8
       country  iso2  iso3  year   new  type sexage value
 *       <chr> <chr> <chr> <dbl> <chr> <chr>  <chr> <dbl>
 1 Afghanistan    AF   AFG  1980   new    sp   m014   NaN
 2 Afghanistan    AF   AFG  1981   new    sp   m014   NaN
 3 Afghanistan    AF   AFG  1982   new    sp   m014   NaN
 4 Afghanistan    AF   AFG  1983   new    sp   m014   NaN
 5 Afghanistan    AF   AFG  1984   new    sp   m014   NaN
 6 Afghanistan    AF   AFG  1985   new    sp   m014   NaN
 7 Afghanistan    AF   AFG  1986   new    sp   m014   NaN
 8 Afghanistan    AF   AFG  1987   new    sp   m014   NaN
 9 Afghanistan    AF   AFG  1988   new    sp   m014   NaN
10 Afghanistan    AF   AFG  1989   new    sp   m014   NaN
# ... with 405,430 more rows
```

One more pull apart

- We need to pull `sexage` into two different variables.
- If you use `?separate`, you'll see that the following is an option:

```
(who_tidy <- who_tidy %>%
  separate(col = sexage, into = c("sex", "age"), sep = 1))
```

```
# A tibble: 405,440 x 9
       country  iso2  iso3  year   new  type   sex   age value
 *       <chr> <chr> <chr> <dbl> <chr> <chr> <chr> <chr> <dbl>
 1 Afghanistan    AF   AFG  1980   new    sp     m   014   NaN
 2 Afghanistan    AF   AFG  1981   new    sp     m   014   NaN
 3 Afghanistan    AF   AFG  1982   new    sp     m   014   NaN
 4 Afghanistan    AF   AFG  1983   new    sp     m   014   NaN
 5 Afghanistan    AF   AFG  1984   new    sp     m   014   NaN
 6 Afghanistan    AF   AFG  1985   new    sp     m   014   NaN
 7 Afghanistan    AF   AFG  1986   new    sp     m   014   NaN
 8 Afghanistan    AF   AFG  1987   new    sp     m   014   NaN
 9 Afghanistan    AF   AFG  1988   new    sp     m   014   NaN
10 Afghanistan    AF   AFG  1989   new    sp     m   014   NaN
# ... with 405,430 more rows
```

Final cleaning

- `iso2` and `iso3` are redundant if we have `country`
- `new` is constant since this only has new cases of TB
- Let's just ignore missing values here
- We also know that `value` is actually `cases` so we can `rename` that column.

```
who_tidy <- who_tidy %>% select(-iso2, -iso3, -new) %>%
  na.omit() %>% rename("cases" = value)
head(who_tidy, 5)
```

```
# A tibble: 5 x 6
      country  year  type   sex   age cases
        <chr> <dbl> <chr> <chr> <chr> <dbl>
1 Afghanistan  1997    sp     m   014     0
2 Afghanistan  1998    sp     m   014    30
3 Afghanistan  1999    sp     m   014     8
4 Afghanistan  2000    sp     m   014    52
5 Afghanistan  2001    sp     m   014   129
```

The power of the %>% (pipe)

Everything we did above can be chained into one sequence:

```
who_tidy <- who_df %>%
  gather(new_sp_m014:newrel_f65, key = "key", value = "value") %>%
  mutate(key = str_replace(key, pattern = "newrel",
                           replacement = "new_rel")) %>%
  separate(col = key, into = c("new", "type", "sexage"), sep = "_") %>%
  separate(col = sexage, into = c("sex", "age"), sep = 1) %>%
  select(-iso2, -iso3, -new) %>%
  na.omit() %>%
  rename("cases" = value)
```

BONUS

A Gapminder tidy dataset read in from a Google Sheet!

- I've updated the `gapminder` data set available in the `gapminder` R data package by Jenny Bryan here. Jenny provides instructions for recreating the data.

BONUS

- You can download the updated data from Google Sheets by running the following in R:

```
# Link is https://docs.google.com/spreadsheets/d/18L5ZiXd1CQ97XWSqb04x1YQ4ncs
library(googlesheets)
updated_gap <- gs_key("18L5ZiXd1CQ97XWSqb04x1YQ4ncsEOdR7eHzgX0ZIuPA") %>%
  gs_read()
```

- A script going through the steps to take the "messy" original Gapminder.org files and turn them into this tidy dataset is available here.

Practice (after the bootcamp)

Try to tidy the other data sets you downloaded and imported or other data sets you have!

Data Modeling

Modeling

1. Experience with `ggplot2` package and knowledge of the Grammar of Graphics primes students for modeling
2. Use of the `broom` package to unpack regression output into a tidy format

1. `ggplot2` Primes Regression

Example:

- All Alaskan Airlines and Frontier flights leaving NYC in 2013
- We want to study the relationship between temperature and departure delay
- For summer (June, July, August) and non-summer months separately

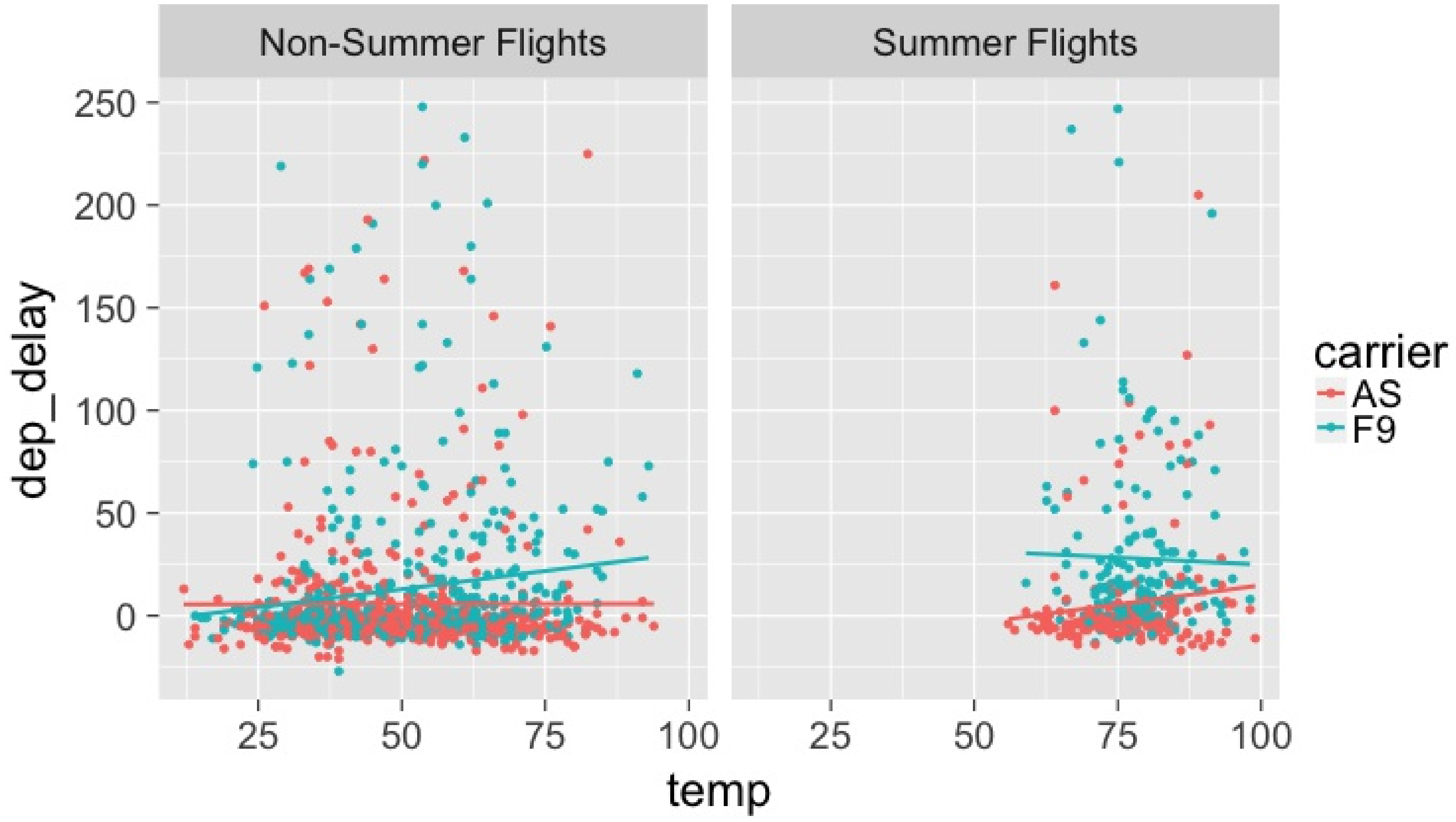
Involves four variables:

- `carrier`, `temp`, `dep_delay`, `summer`

1. ggplot2 Primes Regression

```
flights_subset <- flights %>%
  filter(carrier == "AS" | carrier == "F9") %>%
  left_join(weather, by=c("year", "month", "day", "hour", "origin")) %>%
  filter(dep_delay < 250) %>%
  mutate(summer = ifelse(month == 6 | month == 7 | month == 8, "Summer Flight
```

```r
ggplot(data = flights_subset,
    mapping = aes(x = temp, y = dep_delay, color = carrier)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  facet_wrap(~ summer)
```



1. `ggplot2` Primes Regression

Why? Dig deeper into data. Look at `origin` and `dest` variables as well:

```
flights_subset %>%
  group_by(carrier, origin, dest) %>%
  summarise(`Number of Flights` = n())
```

```
Source: local data frame [2 x 4]
Groups: carrier, origin [?]

# A tibble: 2 x 4
  carrier origin  dest `Number of Flights`
    <chr>  <chr> <chr>               <int>
1      AS    EWR   SEA                 712
2      F9    LGA   DEN                 675
```

2. `broom` Package

- The `broom` package takes the messy output of built-in modeling functions in R, such as `lm`, `nls`, or `t.test`, and turns them into tidy data frames.
- Fits in with `tidyverse` ecosystem
- This works for many R data types!

2. `broom` Package

In our case, `broom` functions take `lm` objects as inputs and return the following in tidy format!

- `tidy()`: regression output table
- `augment()`: point-by-point values (fitted values, residuals, predicted values)
- `glance()`: scalar summaries like R^2 ,

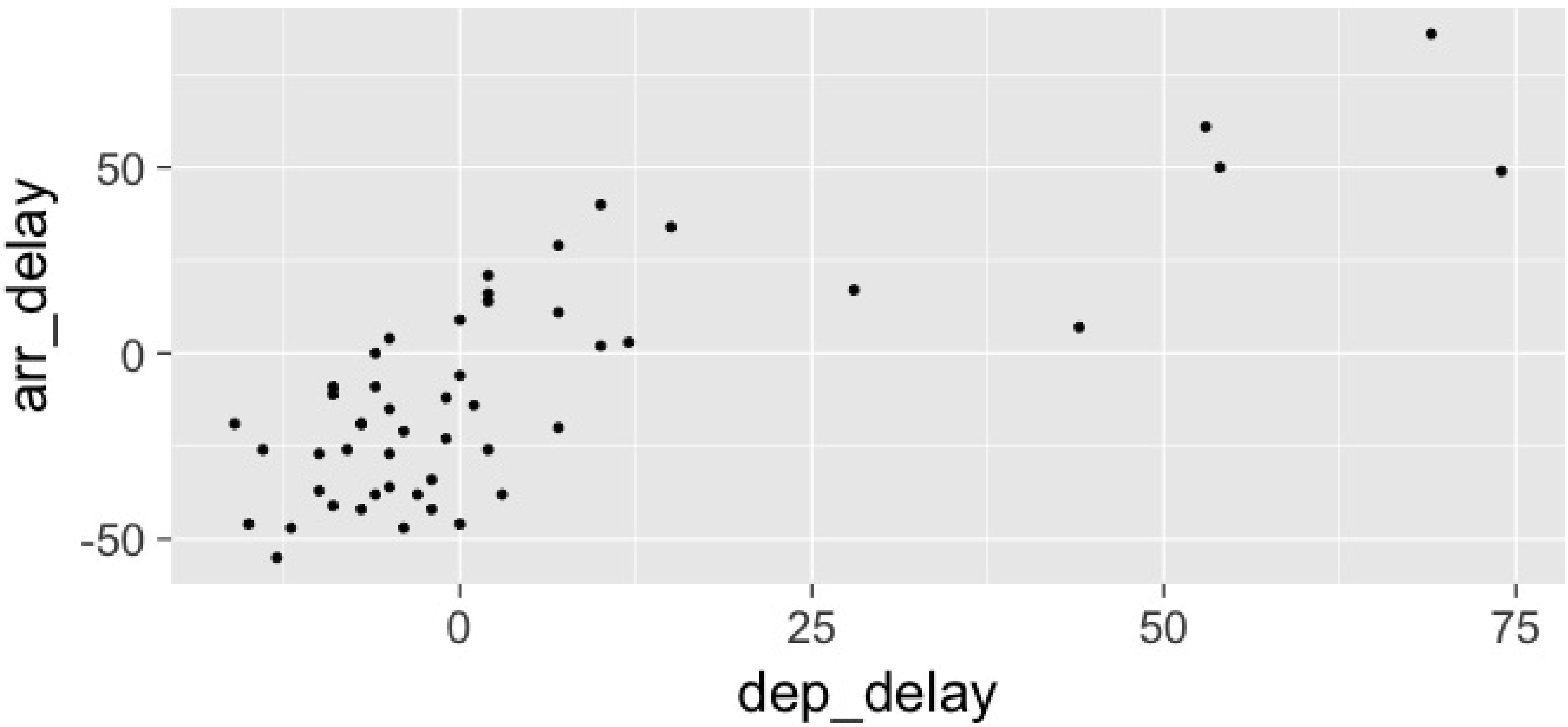
2. broom Package

Example

```
library(tidyverse)
library(nycflights13)
library(knitr)
library(broom)
set.seed(2017)

# Load Alaska data, deleting rows that have missing departure delay
# or arrival delay data
alaska_flights <- flights %>%
  filter(carrier == "AS") %>%
  filter(!is.na(dep_delay) & !is.na(arr_delay)) %>%
  sample_n(50)
```

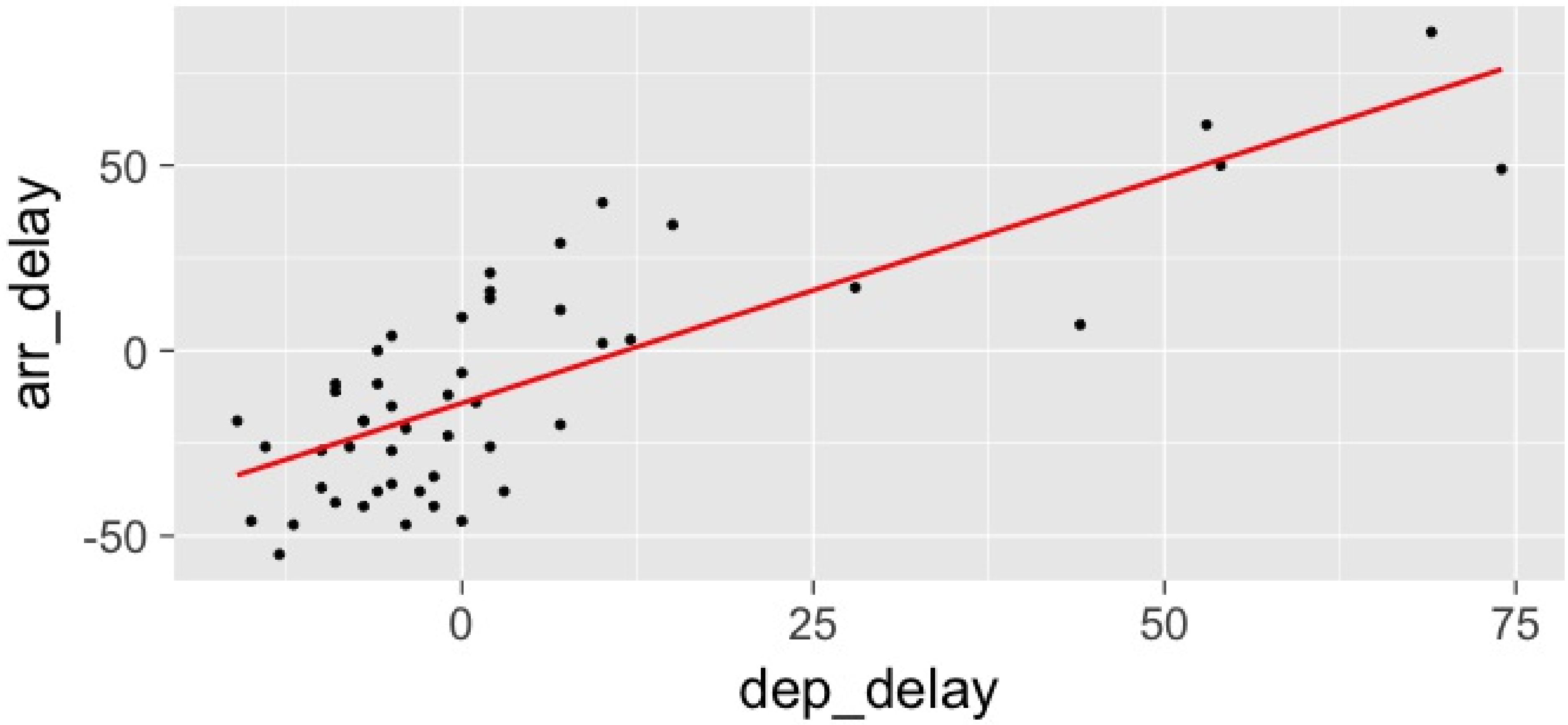
```
# Exploratory Data Analysis---------------------------------------
# Plot of sample of points:
ggplot(data = alaska_flights, mapping = aes(x = dep_delay, y = arr_delay)) +
   geom_point()
```



```
# Correlation coefficient:
alaska_flights %>% summarize(correl = cor(dep_delay, arr_delay))
```

```
# A tibble: 1 x 1
     correl
      <dbl>
1 0.7907993
```

```
# Add regression line
ggplot(data = alaska_flights, mapping = aes(x = dep_delay, y = arr_delay)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE, color = "red")
```



```r
# Fit Regression and Study Output with broom Package-------------------
# Fit regression
options(scipen = 6) # Set scientific notation
delay_fit <- lm(formula = arr_delay ~ dep_delay, data = alaska_flights)

# 1. broom::tidy() regression table with confidence intervals
# and no p-value stars
regression_table <- delay_fit %>%
  tidy(conf.int = TRUE)
regression_table
```

```
         term   estimate std.error statistic      p.value    conf.low
1 (Intercept) -14.155016 2.8094813 -5.038302 7.074778e-06 -19.8038573
2   dep_delay   1.217666 0.1360336  8.951212 8.367030e-12   0.9441519
  conf.high
1 -8.506176
2  1.491180
```

```r
# 2. broom::augment() for point-by-point values
regression_points <- delay_fit %>%
  augment() %>%
  select(arr_delay, dep_delay, .fitted, .resid)
regression_points
```

```
   arr_delay dep_delay    .fitted      .resid
1        -38        -3 -17.808014 -20.1919862
2         86        69  69.863923  16.1360769
3        -38         3 -10.502019 -27.4979809
4         61        53  50.381270  10.6187296
5          3        12   0.456973   2.5430270
6         21         2 -11.719685  32.7196849
7          2        10  -1.978359   3.9783586
8        -41        -9 -25.114009 -15.8859914
9        -19        -7 -22.678677   3.6786770
10         0        -6 -21.461011  21.4610112
11        -6         0 -14.155016   8.1550165
12       -19        -7 -22.678677   3.6786770
13        40        10  -1.978359  41.9783586
14        -9        -6 -21.461011  12.4610112
15        17        28  19.939626  -2.9396257
16       -11        -9 -25.114009  14.1140086
17       -14         1 -12.937351  -1.0626493
18       -37       -10 -26.331674 -10.6683256
19       -21        -4 -19.025680  -1.9743204
20        49        74  75.952252 -26.9522520
21        34        15   4.109970  29.8900296
22        29         7  -5.631356  34.6313559
23       -42        -7 -22.678677 -19.3213230
24        50        54  51.598936  -1.5989362
```

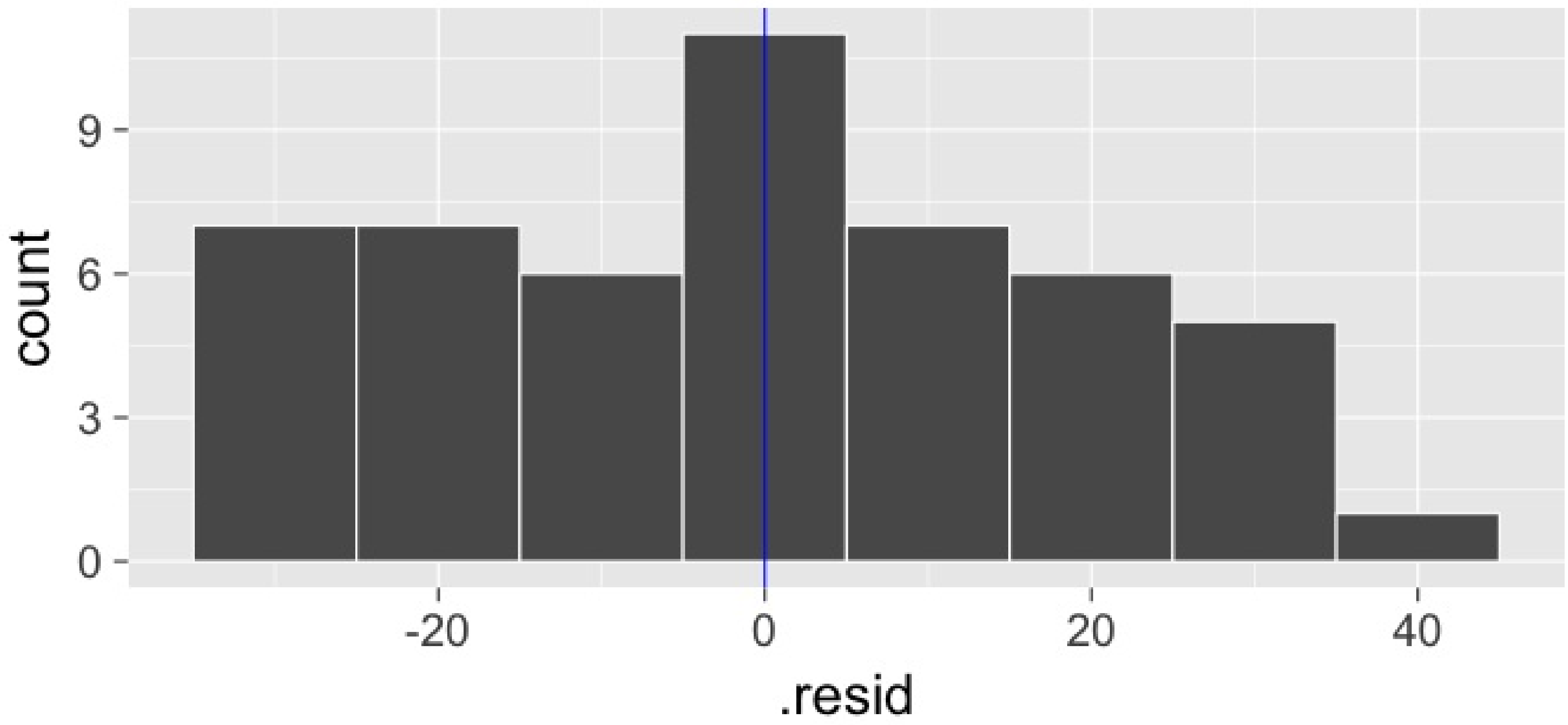
```
# and for prediction
new_flights <- data_frame(dep_delay = c(25, 30, 15))
delay_fit %>%
  augment(newdata = new_flights)
```

```
  dep_delay  .fitted   .se.fit
1        25 16.28663 3.967287
2        30 22.37496 4.481560
3        15  4.10997 3.134505
```

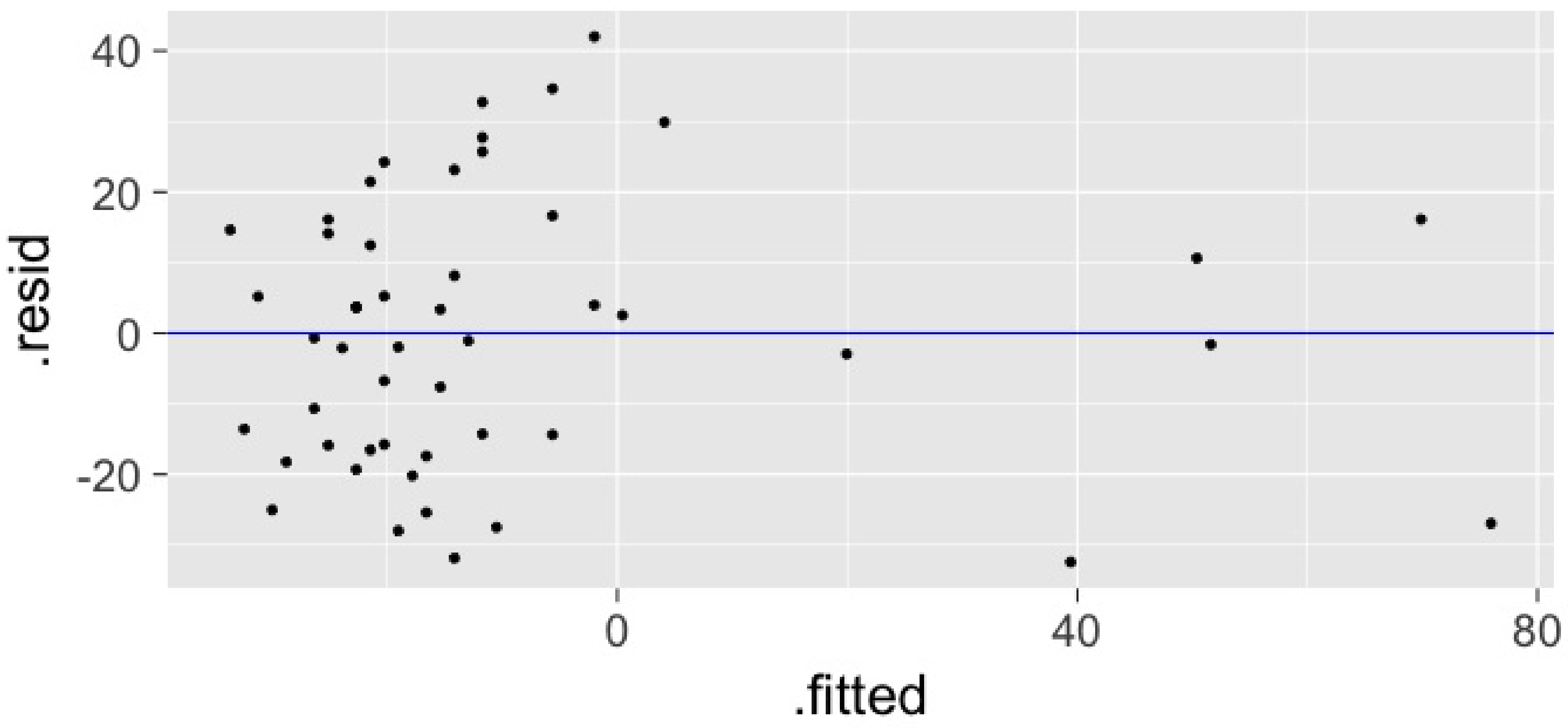
```r
# 3. broom::glance() scalar summaries of regression
regression_summaries <- delay_fit %>%
  glance()
regression_summaries
```

```
  r.squared adj.r.squared    sigma statistic     p.value df    logLik
1 0.6253635     0.6175586 19.48607  80.12419 8.36703e-12  2 -218.4114
       AIC      BIC deviance df.residual
1 442.8227 448.5588 18225.92          48
```

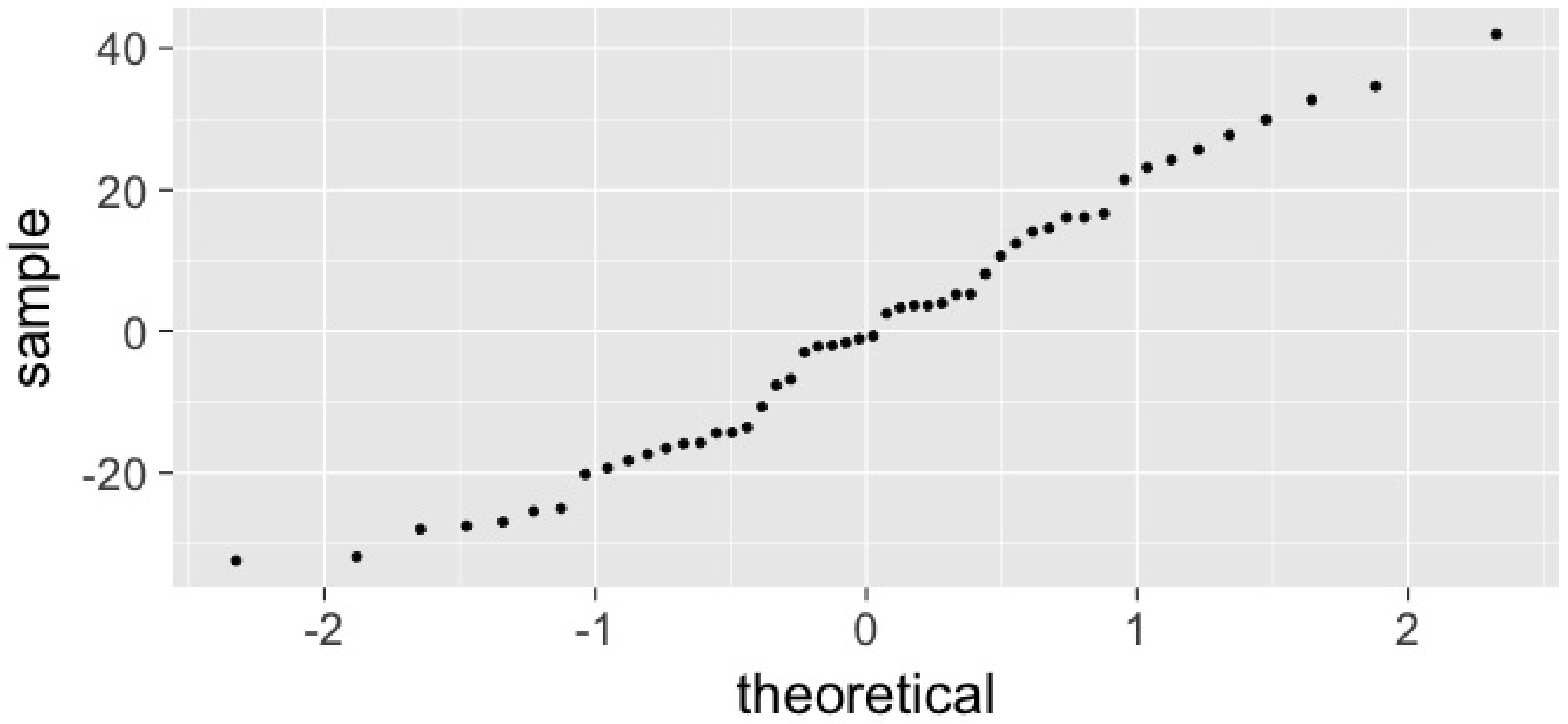
```
# Residual Analysis------------------------------------------------------
ggplot(data = regression_points, mapping = aes(x = .resid)) +
  geom_histogram(binwidth=10, color = "white") +
  geom_vline(xintercept = 0, color = "blue")
```



```
ggplot(data = regression_points, mapping = aes(x = .fitted, y = .resid)) +
  geom_point() +
  geom_abline(intercept = 0, slope = 0, color = "blue")
```



```
ggplot(data = regression_points, mapping = aes(sample = .resid)) +
  stat_qq()
```



Data Communicating

What is Markdown?

- A "plaintext formatting syntax"
- Type in plain text, render to more complex formats
- One step beyond writing a `txt` file
- Render to HTML, PDF, DOCX, etc. using Pandoc

What does it look like?

```
# Header 1

## Header 2

Normal paragraphs of text go here.

**I'm bold**

[links!](http://rstudio.com)

 * Unordered
 * Lists

And  Tables
---- -------
Like This
```

Header 1

Header 2

Normal paragraphs of text go here.

I'm bold

links!

- Unordered
- Lists

And	Tables
Like	This

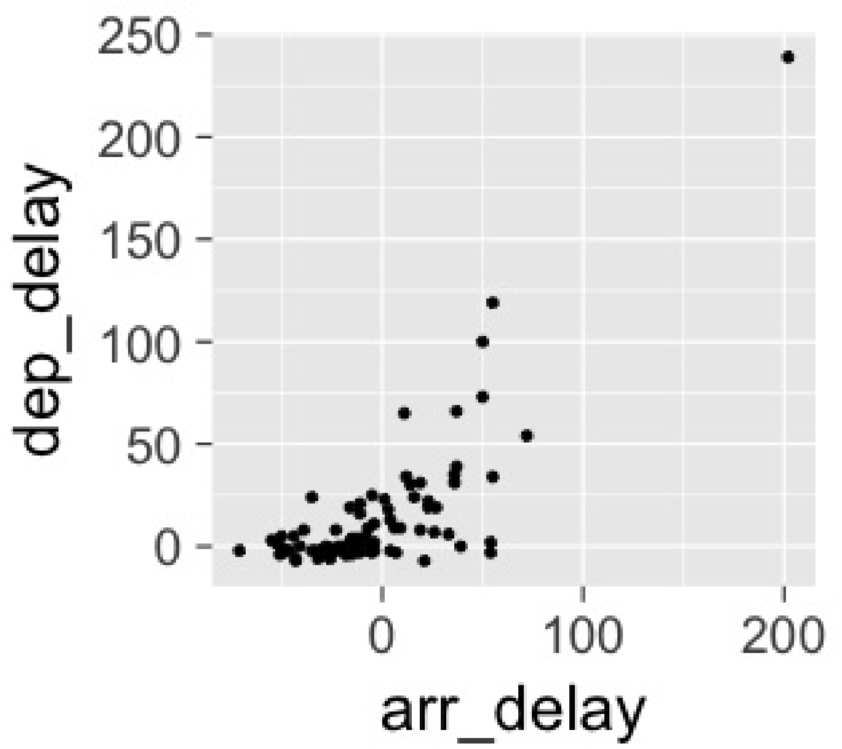
What is R Markdown?

- "Literate programming"
- Embed R code in a Markdown document
- Renders textual output along with graphics

````
```{r chunk1}
library(ggplot2)
library(nycflights13)
pdx_flights <- flights %>%
  filter(dest == "PDX", month == 5)
nrow(pdx_flights)
```

```{r chunk2}
ggplot(data = pdx_flights,
  mapping = aes(x = arr_delay,
                y = dep_delay)) +
  geom_point()
```
````

```
[1] 88
```



Practice

Turn a statistical analysis you have conducted into an R Markdown document.

- You can also publish online easily to http://rpubs.com

Thanks for attending!

- Slides created via the R package xaringan by Yihui Xie
- Source code for these slides and the webpage at https://github.com/ismayc/poRtland-bootcamp17