

# Data Carpentry: Using R to Analyze and Visualize Data

Evan Cortens, PhD and Stephen Childs

October 21, 2018

# Introductions

# Introductions

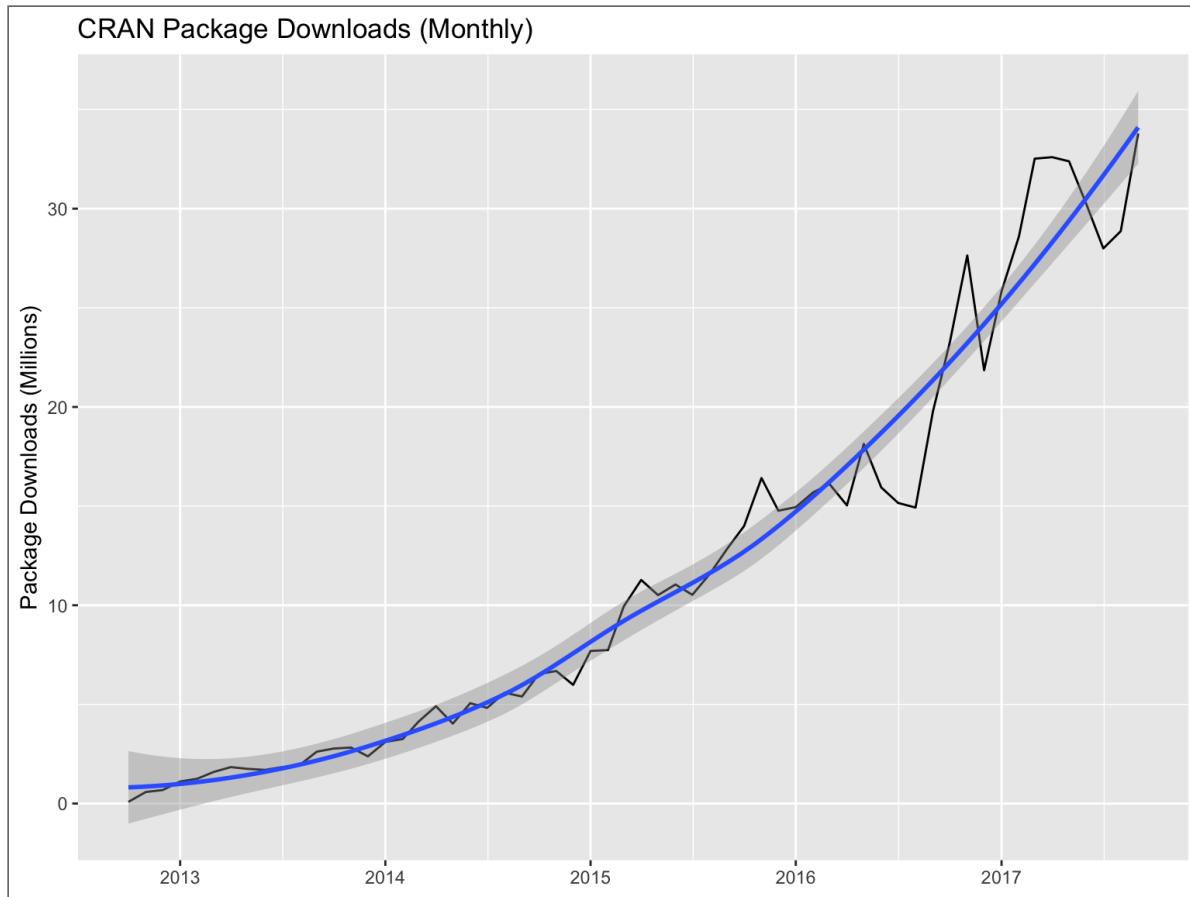
- Who are you/where do you work?
- What software/tools do you spend most time with? (Excel? SPSS? Tableau? etc)
- Do you have any programming experience? (Python? C? Javascript?)
- What do you hope to get out of this workshop? How do you see yourself using R in an IR environment?s

# Why use R?

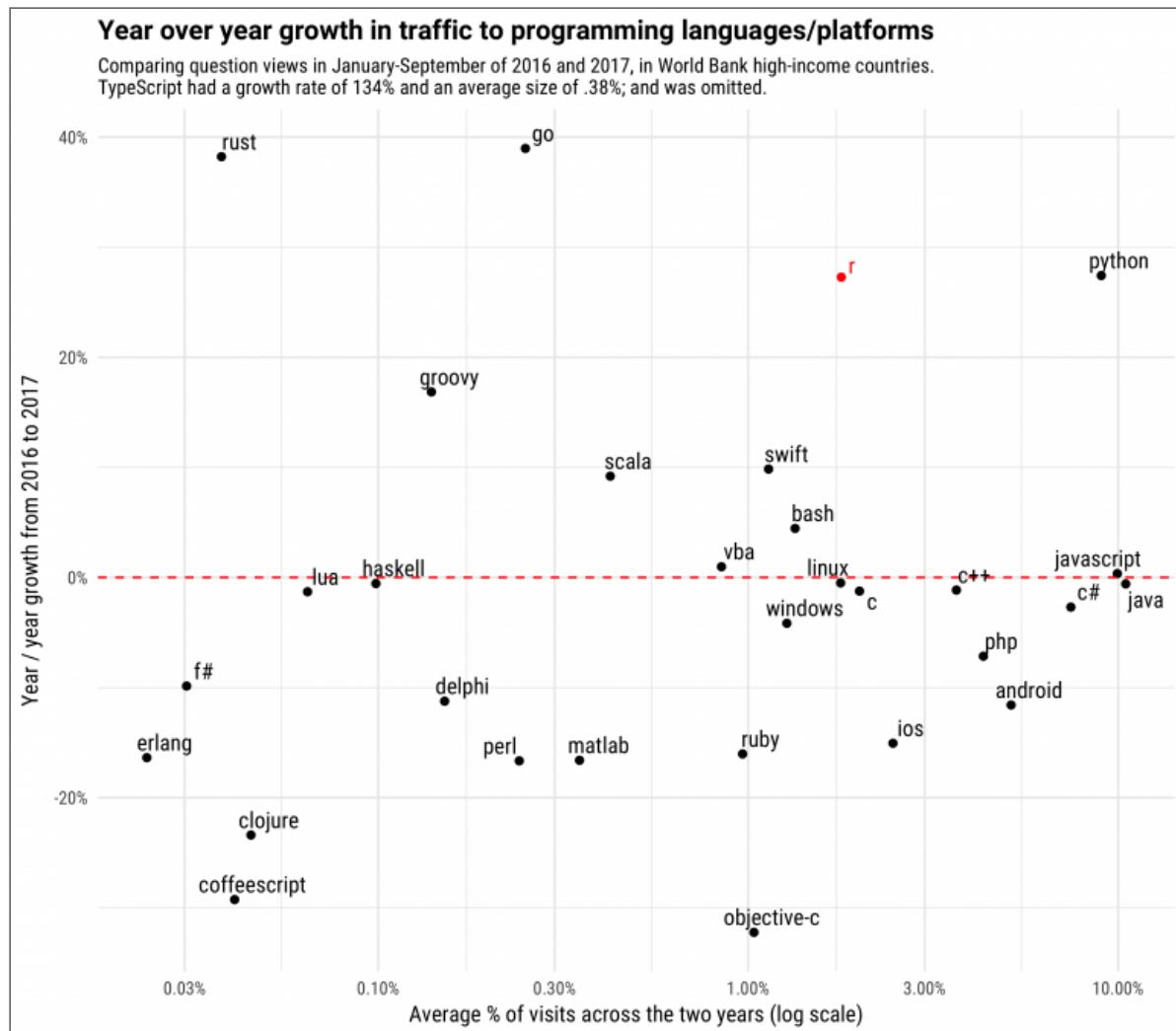
# Why use R?

- Free
- Open-source
- A “statistical” programming language
  - Generally accepted by academic statisticians
  - Thoroughly grounded in statistical theory
- A great community
  - Help and support
  - Innovation and development
- R is a stable, well-supported language
  - Major corporate backing, e.g., Microsoft

# Growing fast!



# Growing fast!



<https://stackoverflow.blog/2017/10/10/impressive-growth-r/>

# Some history

- Based on the S programming language
  - Dates from the late 1970s, substantially revised in the late 1980s
  - (there is still a commercial version, S-PLUS)
- R was developed at the University of Auckland
  - Ross Ihaka and Robert Gentleman
  - First developed in 1993
  - Stable public release in 2000
- RStudio
  - IDE, generally recognized as high quality, even outside R
  - Development began in 2008, first public release in February 2011
    - Version 1.0 in Fall 2016
  - There have been other IDEs for R in the past, but since RStudio came on the scene, most are no longer seriously maintained

# Some history

Why is this relevant?

- You'll likely notice some quirks in R that differ from other programming languages you may have worked in—the unique history often explains why.
- Explains R's position as a “statistical programming language”:
  - Think stats/data first, programming second.

# Today's approach/Learning outcomes

# Today's approach/Learning outcomes

In learning R today, we'll take an approach, used in *R for Data Science* (Wickham & Grolemund, 2016), diagrammed like this:

One of the strengths of R: every step of this workflow can be done using it!

# Import

- Loading the data!
- For IR-type tasks, this will generally be from data files or directly from databases.
- Other possibilities include web APIs, or even web scraping

# Tidy

- “Tidy data” (Wickham, 2014)
  - Each column is a variable, each row is an observation.
- Doing this kind of work up front, right after loading, lets you focus on the modelling/analysis/visualization problem at hand, rather than having to rework your data at each stage of your analysis.

# Transform/Visualize/Model

- Repeat these three steps as necessary:

# Transform

- Together with tidying, sometimes called *data wrangling*
  - Filtering (selecting specific observations)
  - Mutating (creating new variables)
  - Summarizing (means, counts, etc)

# Visualise

- For both exploratory purposes and production/communication

# Model

- Complementary to visualisation
- For the “precise exploration” of “specific questions”

# Communicate

- The final output, whether it's just for yourself, your colleagues, or a wider audience
- Increasingly, there's a trend toward "reproducible research" which integrates even the communication step (final paper, report, etc) into the code/analysis.

# Housekeeping

# Installing R and RStudio

- [\*\*https://cran.r-project.org/\*\*](https://cran.r-project.org/)
- [\*\*https://www.rstudio.com/products/rstudio/\*\*](https://www.rstudio.com/products/rstudio/)
- `install.packages('tidyverse')`

# Basic R

- An interpreted language (like Python)
- Code can be run as:
  - scripts (programmatically/non-interactively)
  - from the prompt (interactively)
- R uses an REPL (Read-Evaluate-Print Loop) just like Python
  - Using R as a calculator (demonstration)

# Outline

# Outline

1. Visualisation
2. Transformation
3. Importing and ‘wrangling’
4. Hands-on portion using what we’ve learned

# Visualisation

# Let's load our package

```
library(tidyverse)
```

A package only needs to be *installed* once (per major version, e.g. 3.3.x to 3.4.x), but must be *loaded* every time.

# The ‘mpg’ data set

Data on the fuel efficiency of 38 models of cars between 1999 and 2008 from the US EPA:

```
mpg
```

```
## # A tibble: 234 x 11
##   manufacturer model displ year cyl trans drv cty hwy
##   <chr>        <chr> <dbl> <int> <int> <chr> <chr> <int> <int>
## 1 audi         a4     1.8  1999    4 auto... f      18   29
## 2 audi         a4     1.8  1999    4 manu... f      21   29
## 3 audi         a4     2    2008    4 manu... f      20   31
## 4 audi         a4     2    2008    4 auto... f      21   30
## 5 audi         a4     2.8  1999    6 auto... f      16   26
## 6 audi         a4     2.8  1999    6 manu... f      18   26
## 7 audi         a4     3.1  2008    6 auto... f      18   27
## 8 audi         a4 q... 1.8  1999    4 manu... 4     18   26
## 9 audi         a4 q... 1.8  1999    4 auto... 4     16   25
## 10 audi        a4 q... 2    2008    4 manu... 4     20   28
## # ... with 224 more rows
```



# The Grammar of Graphics

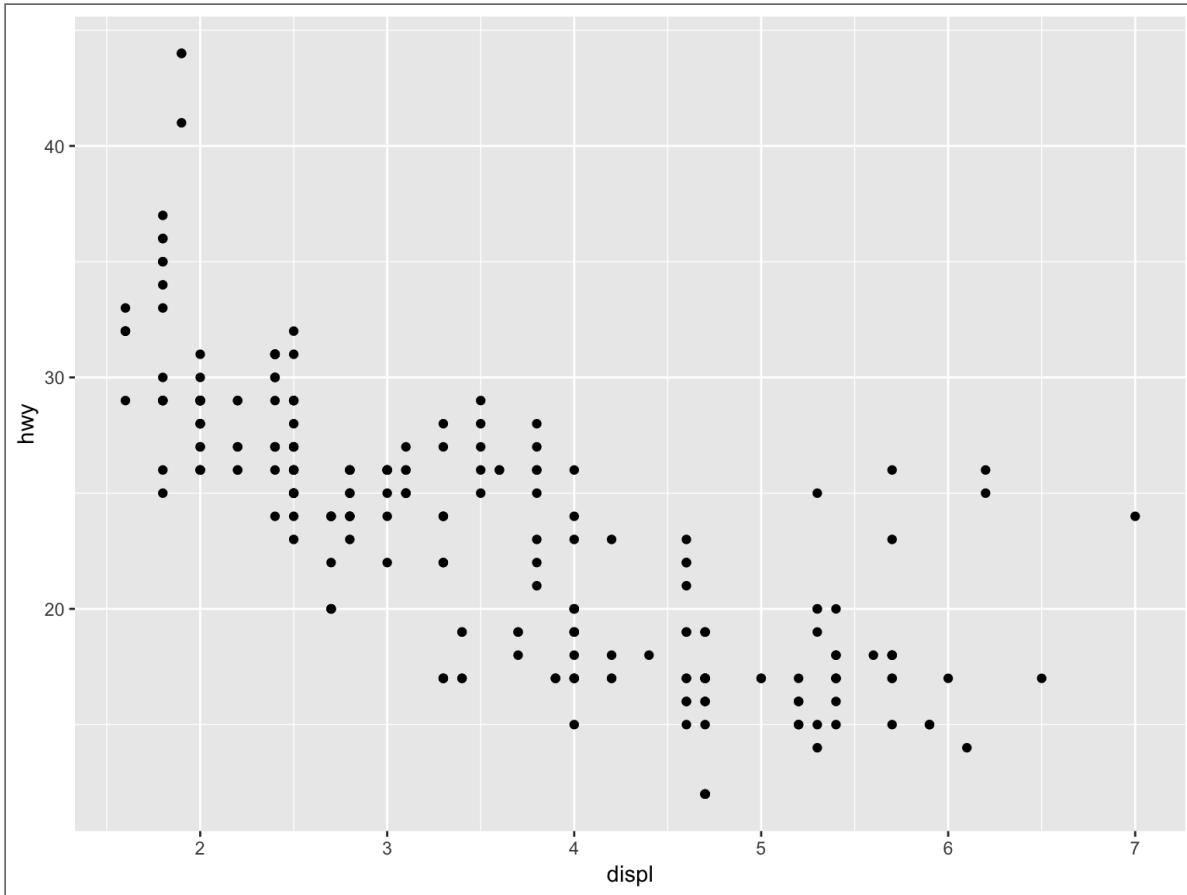
- Layers
- Inheritance
- Mapping (aes)

# Our First Plot

- A car's highway mileage (mpg) vs its engine size (displacement in litres).
- What might the relationship be?

# Our First Plot

```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy))
```



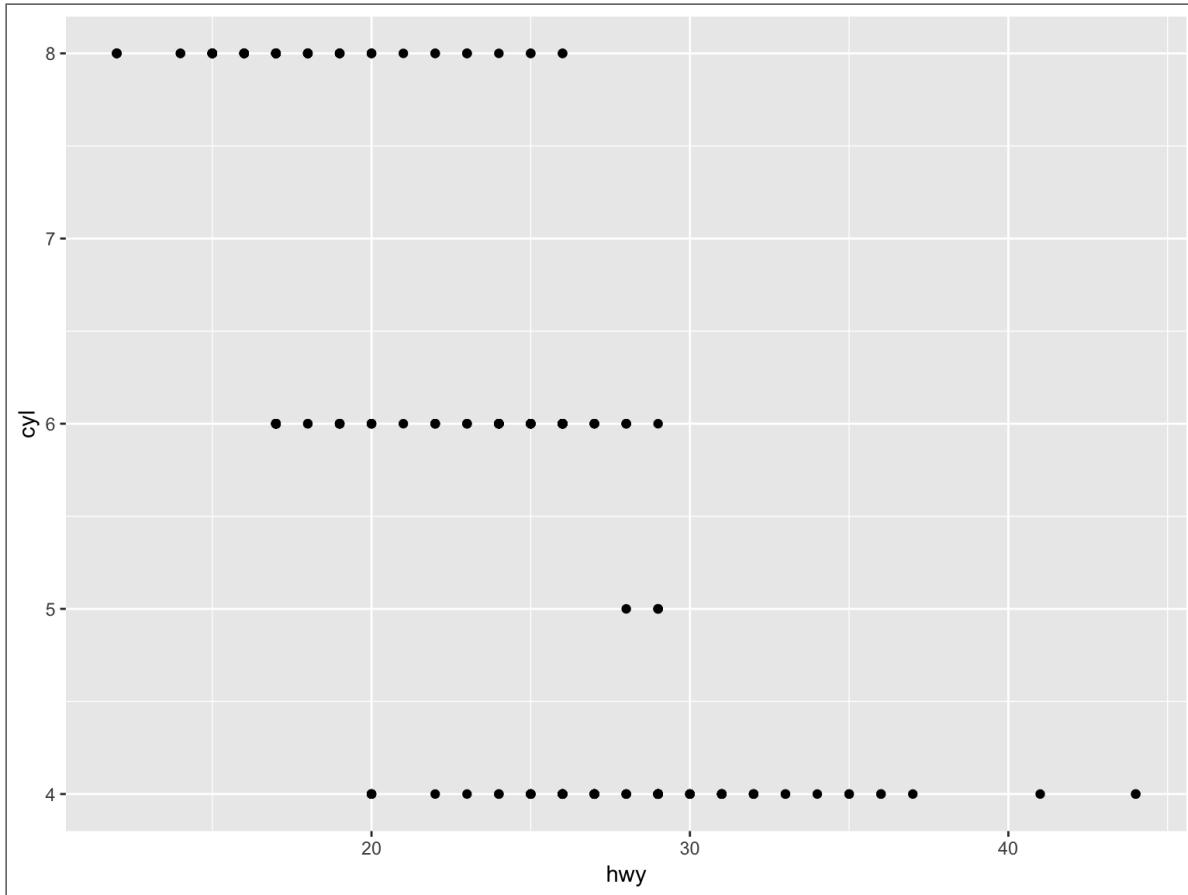
- As engine size increases, fuel efficiency decreases, roughly.

# Exercises

- The first scatterplot was highway mileage vs displacement: how can we make it city mileage vs displacement?
- Make a scatterplot of `hwy` vs `cyl`.
- What about a scatterplot of `class` vs `drv`?

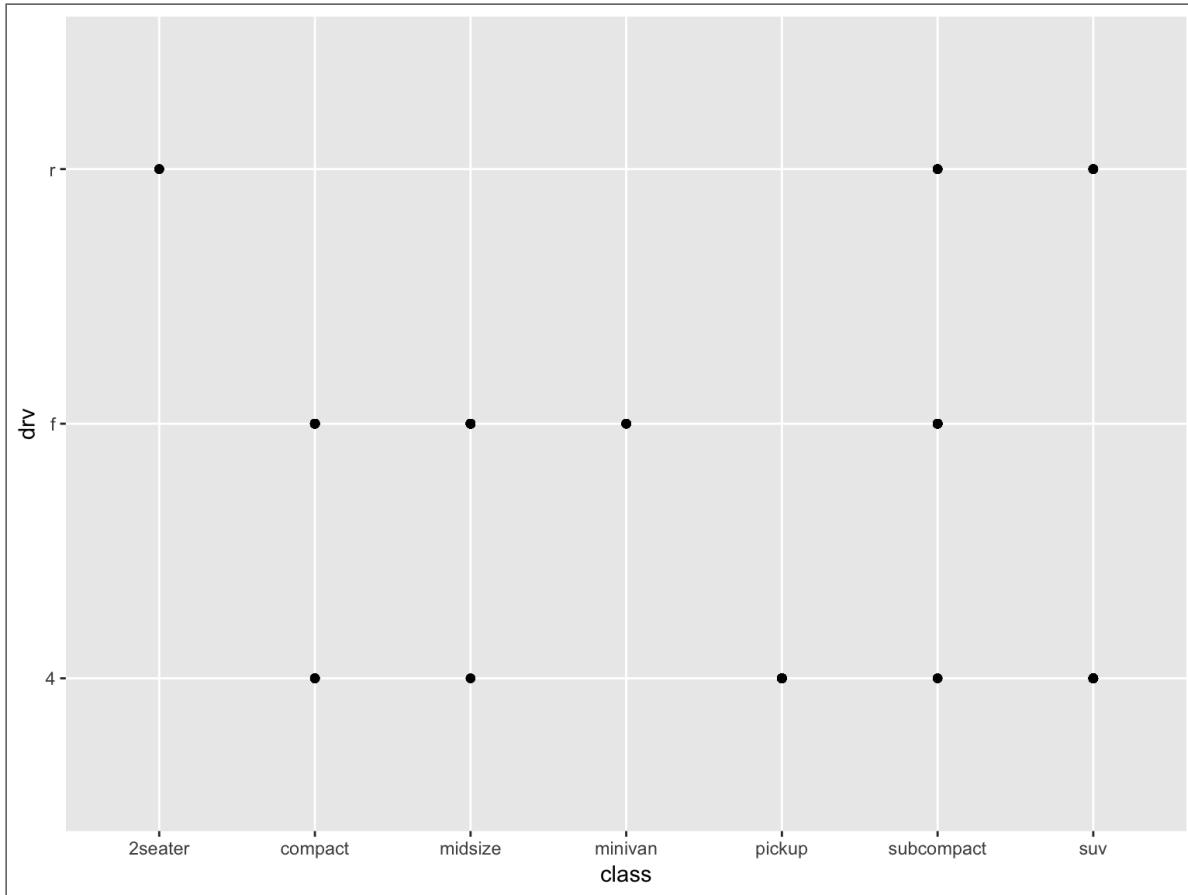
# Hwy vs cyl

```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = hwy, y = cyl))
```



# Class vs drv

```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = class, y = drv))
```

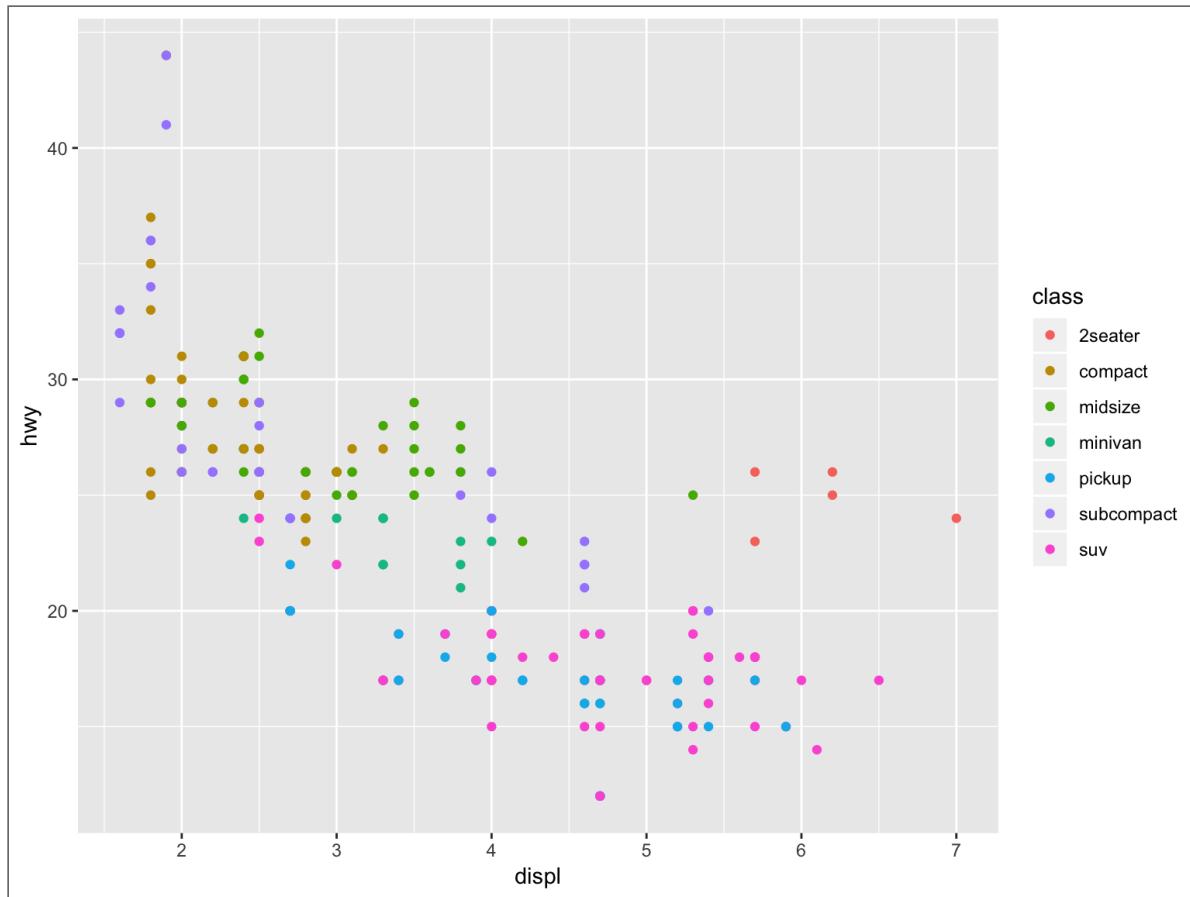


What's going on here? Is this useful? How might we make it more useful?

# Additional aesthetics

What about those outliers? Use the `color` aesthetic.

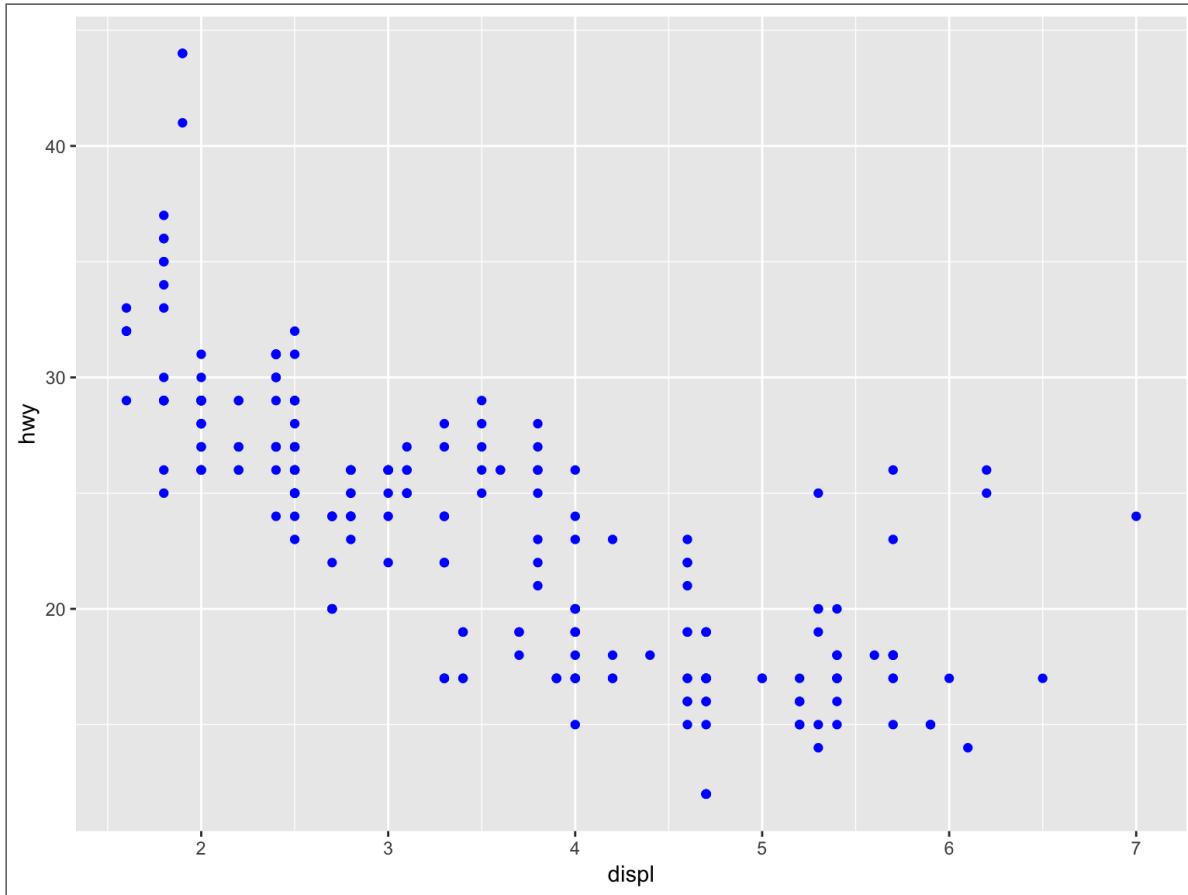
```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy, color = class))
```



# ‘Unmapped’ aesthetics

What’s happening here?

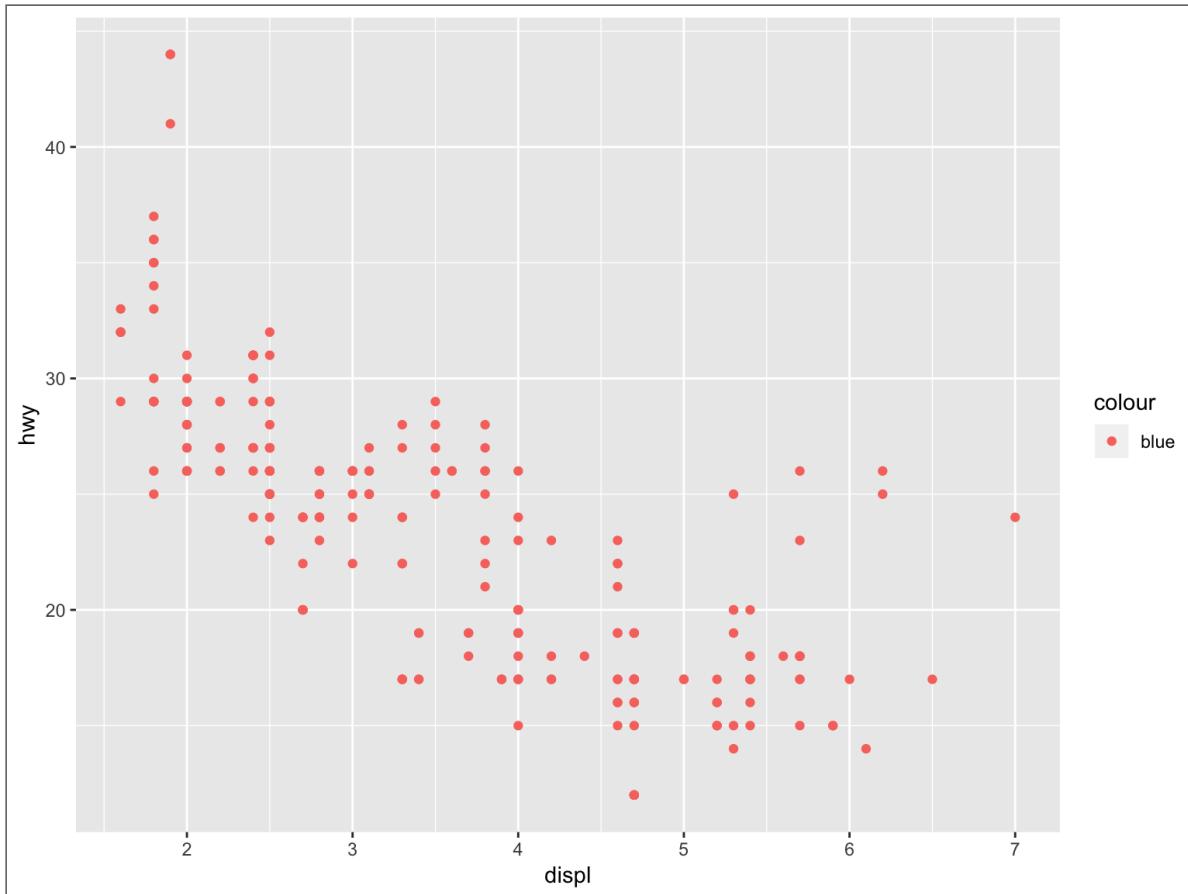
```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy), color = "blue")
```



# What's gone wrong?

What's happened here? What colour are the points?  
Why?

```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy, color = "blue"))
```

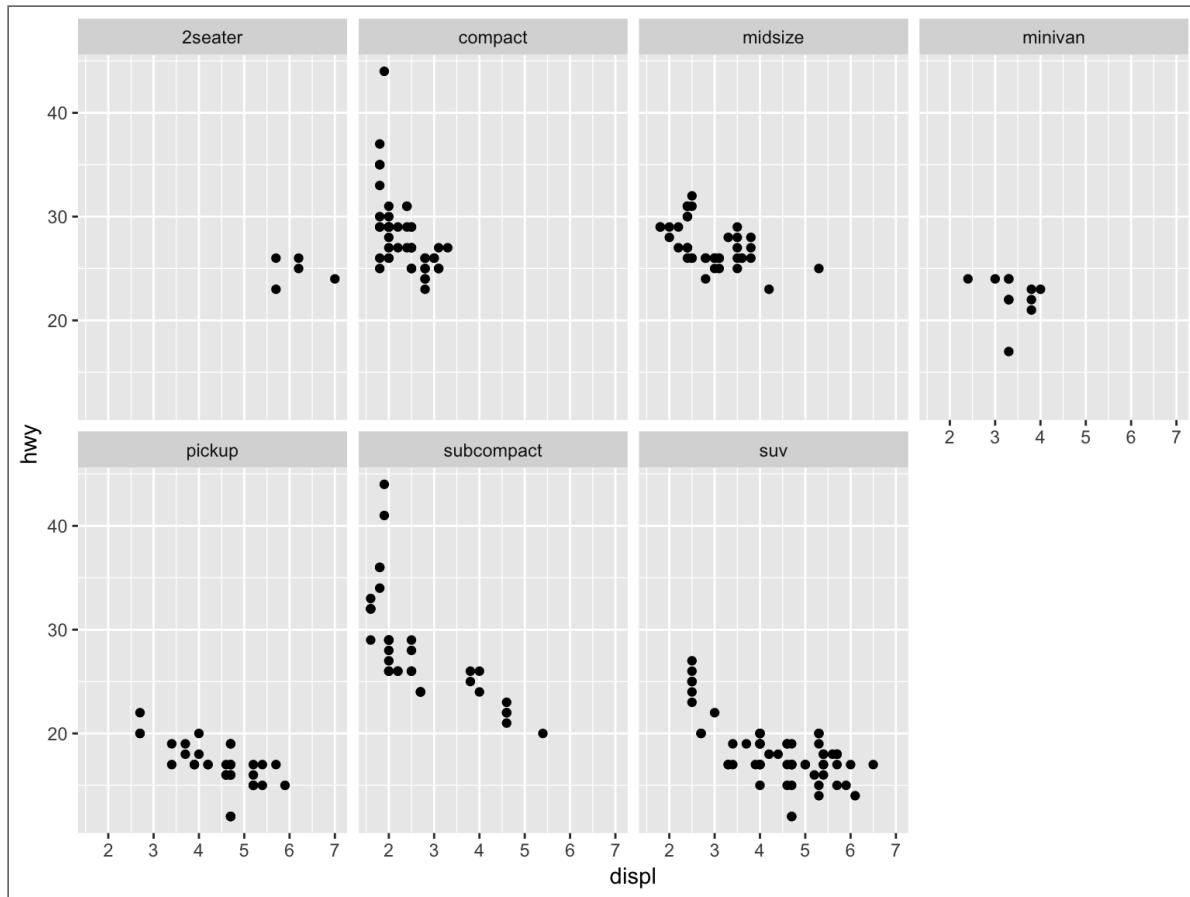


# Exercises

- mpg variable types: which are categorical, which are continuous?
- ?mpg
- Map a continuous variable to colour, size, and shape: how are these aesthetics different for categorical vs continuous?
- What happens if you map an aesthetic to something other than a variable name, like `aes(color = displ < 5)`?

# Facets: One Variable

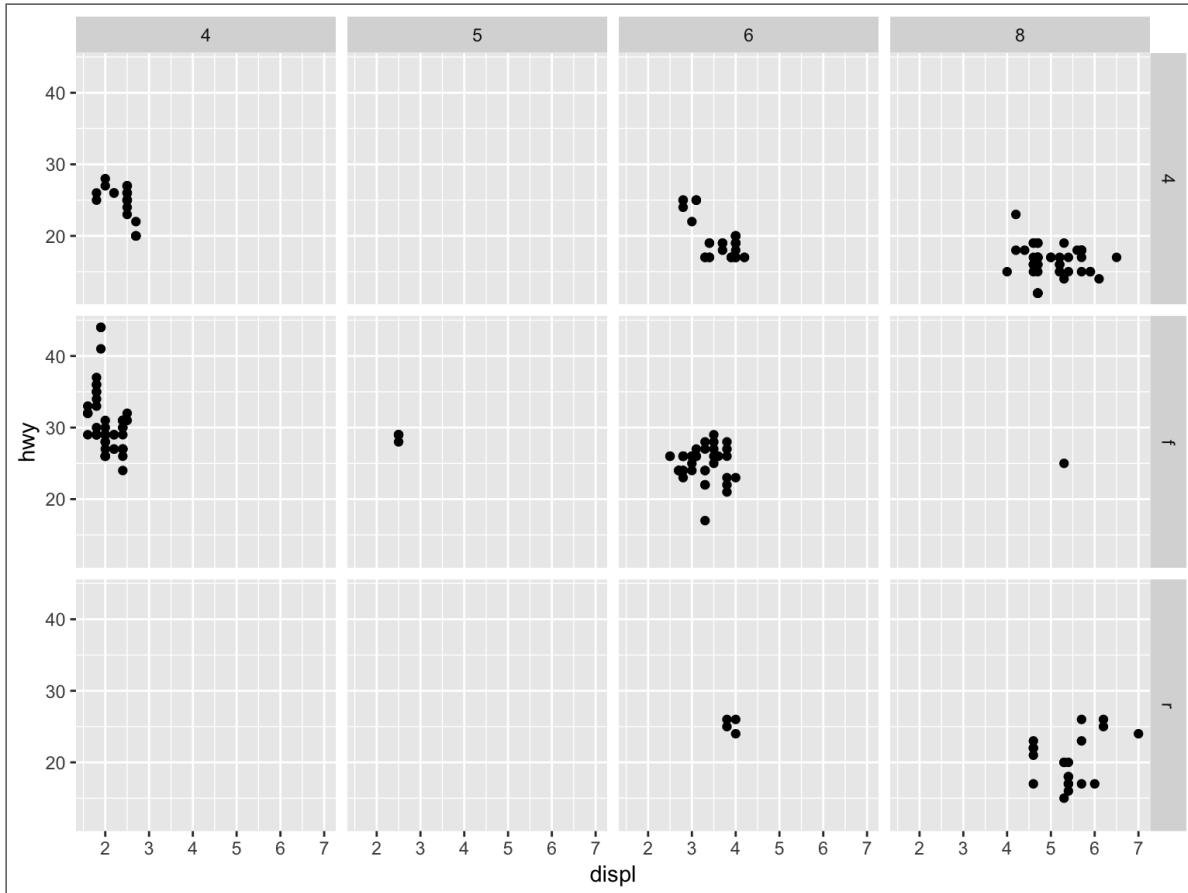
```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy)) +  
  facet_wrap(~ class, nrow = 2)
```



\*\*TODO: This might be a good point to talk about formulas - since we use the formula notation with facets.

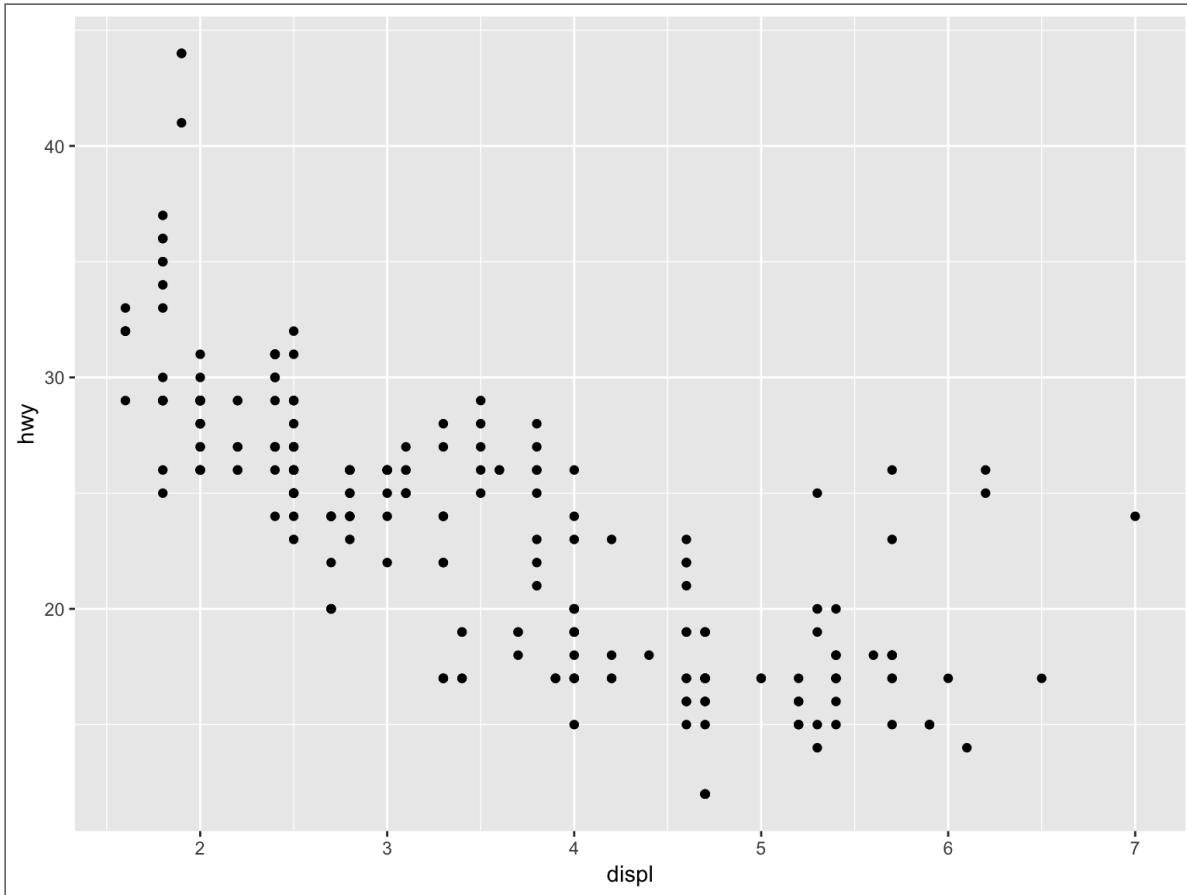
# Facets: Two Variables

```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy)) +  
  facet_grid(drv ~ cyl)
```



# Other “geoms” (geometries)

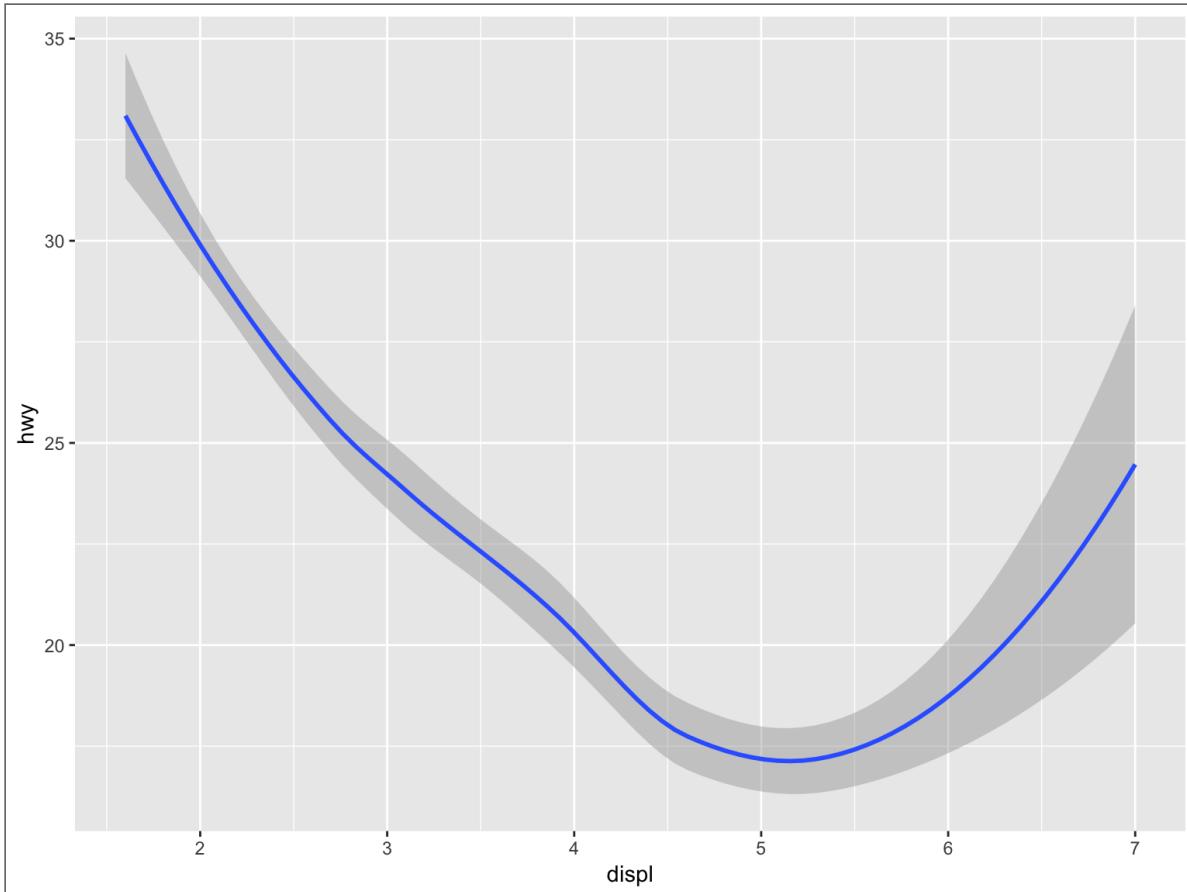
```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy))
```



# Smooth

```
ggplot(data = mpg) +  
  geom_smooth(mapping = aes(x = displ, y = hwy))
```

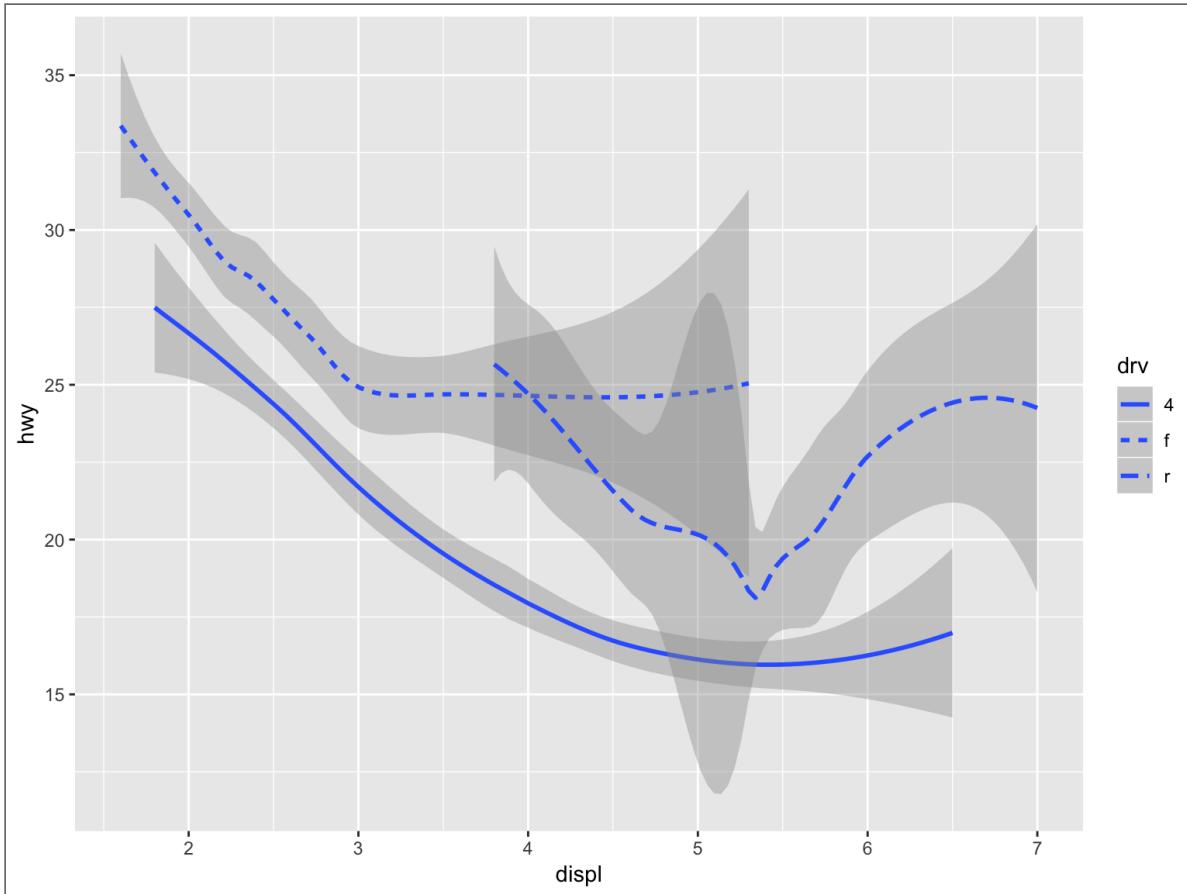
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



# Smooth aesthetics

```
ggplot(data = mpg) +  
  geom_smooth(mapping = aes(x = displ, y = hwy, linetype = drv))
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



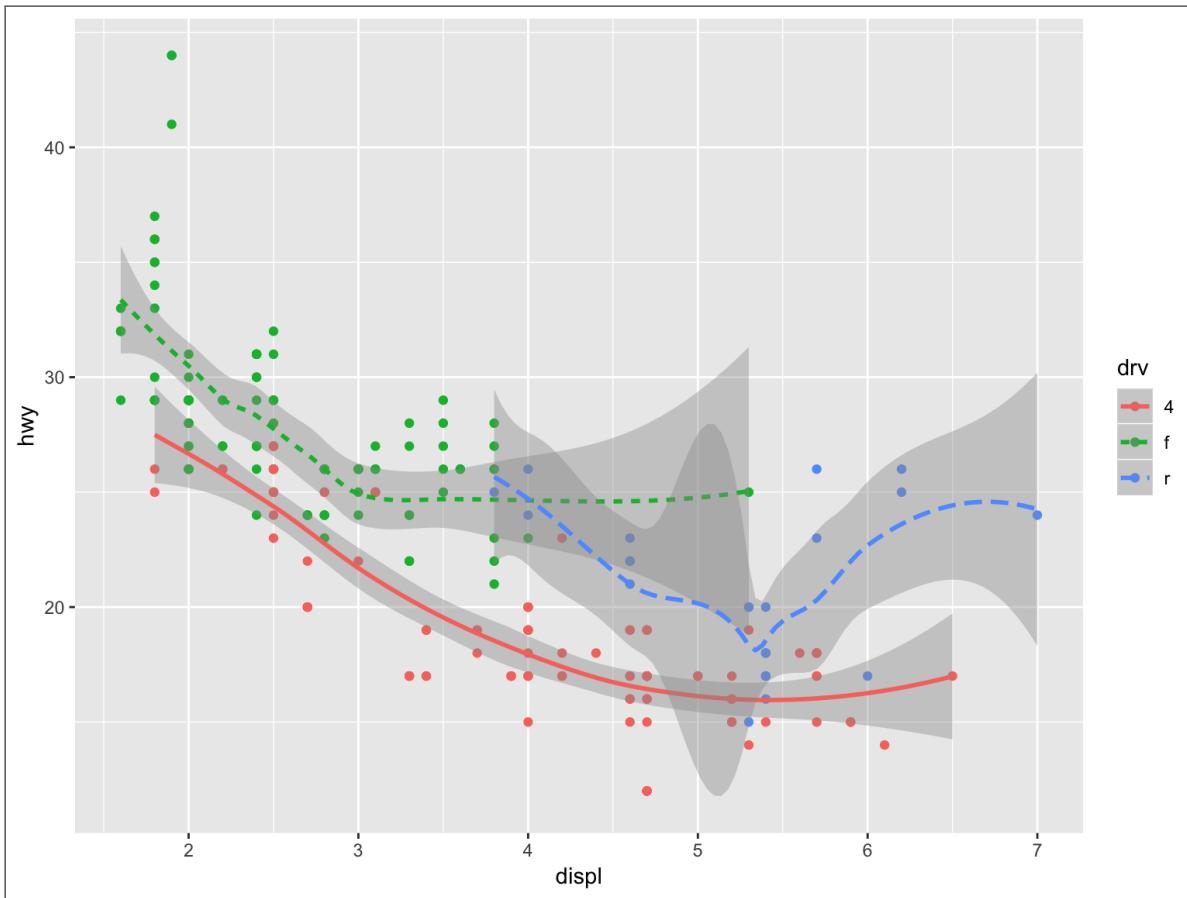
# Which aesthetics do geoms have?

?geom\_smooth

# Clearer?

```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy, color = drv)) +  
  geom_smooth(mapping = aes(x = displ, y = hwy, color = drv,  
    linetype = drv))
```

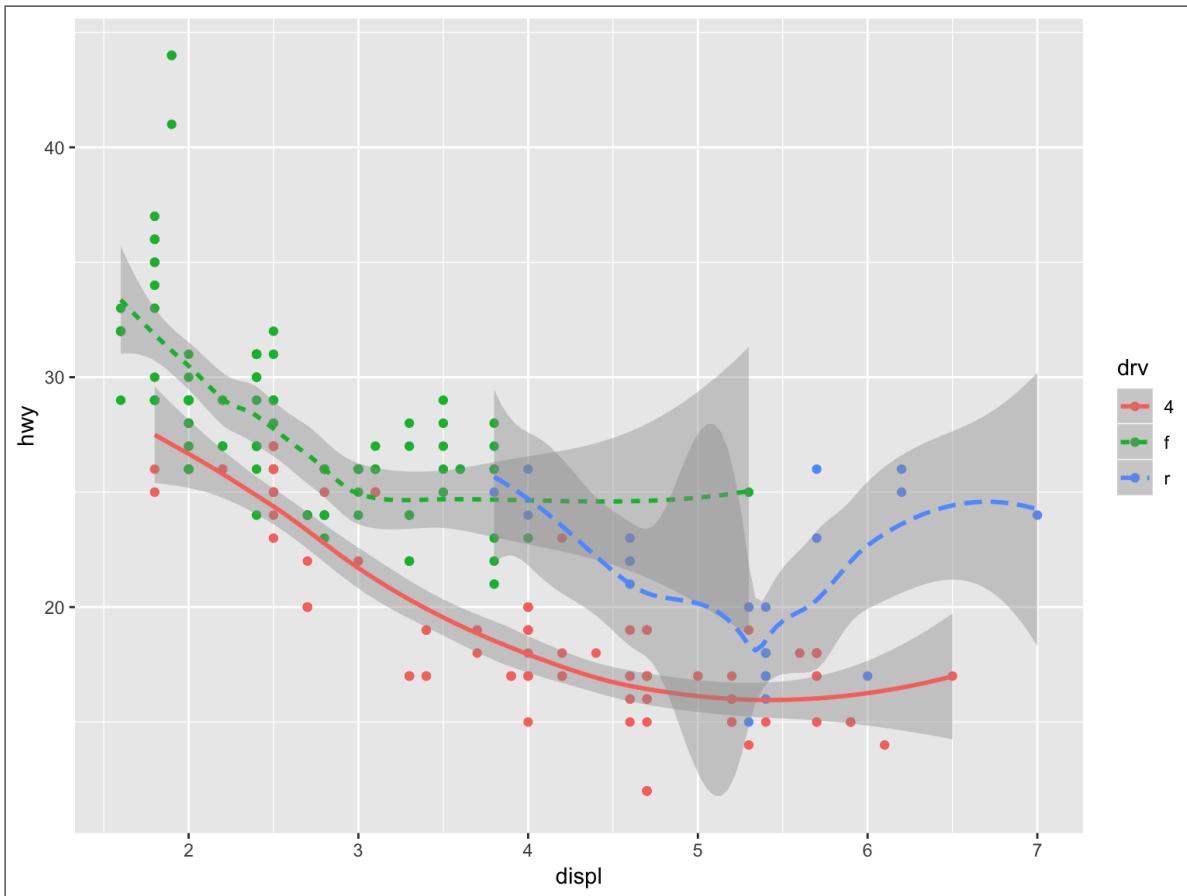
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



# Reducing Duplication

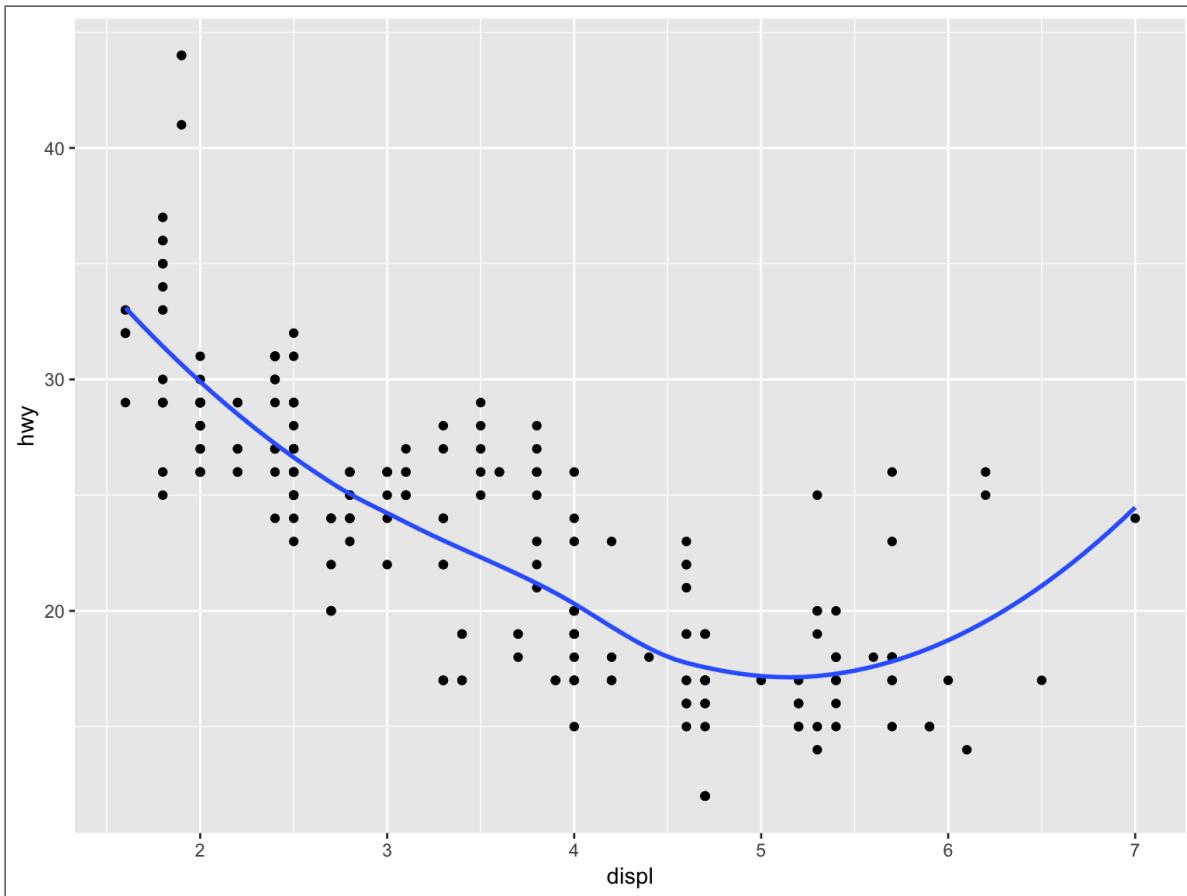
```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy, color =  
drv)) +  
geom_point() +  
geom_smooth(mapping = aes(linetype = drv))
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



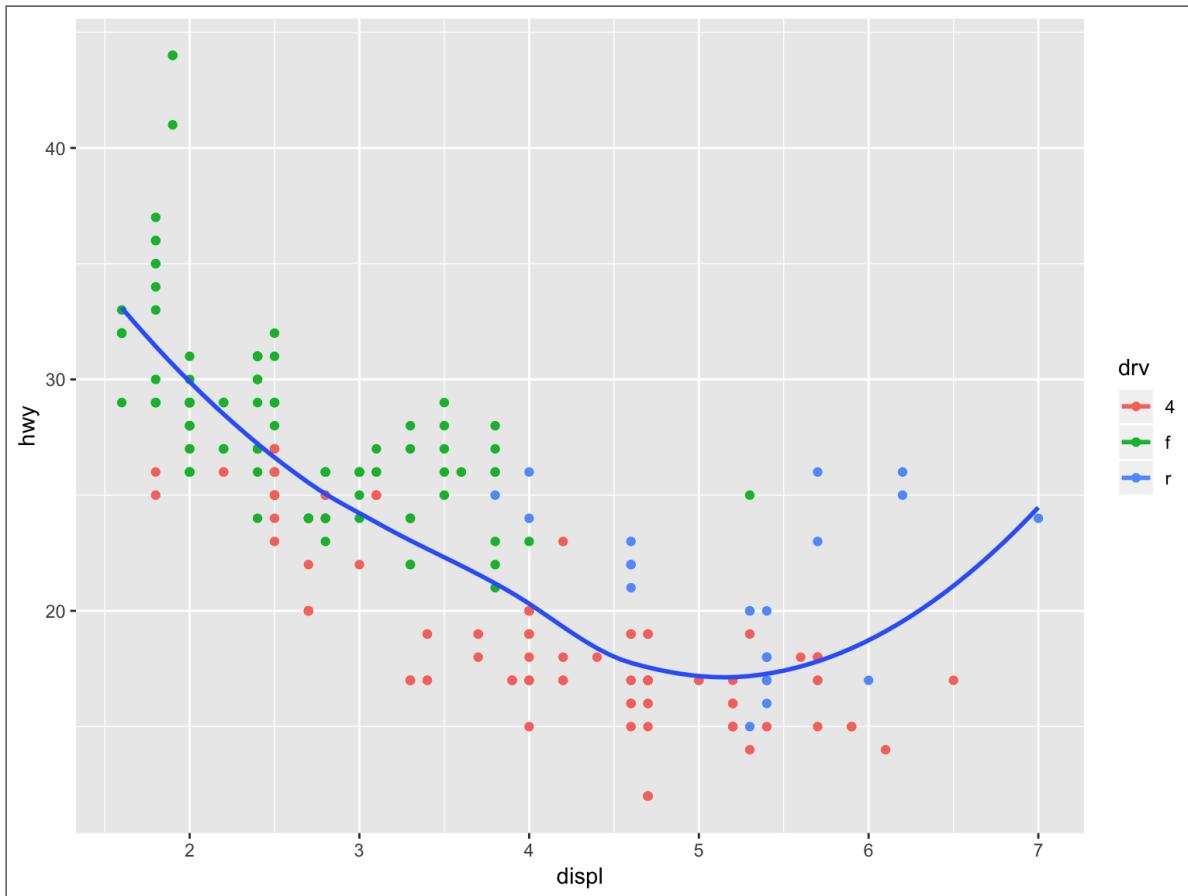
# Exercise: Recreate:

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



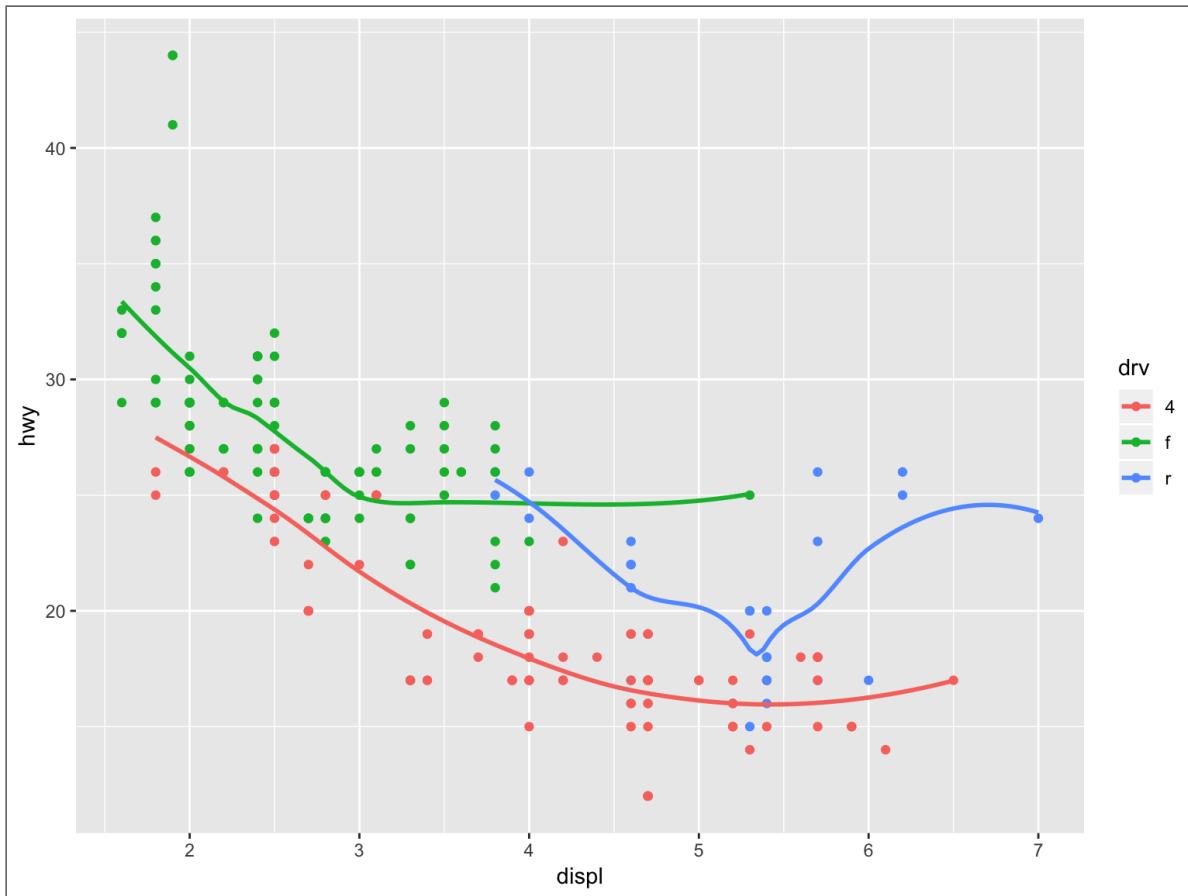
# Exercise: Recreate:

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



# Exercise: Recreate:

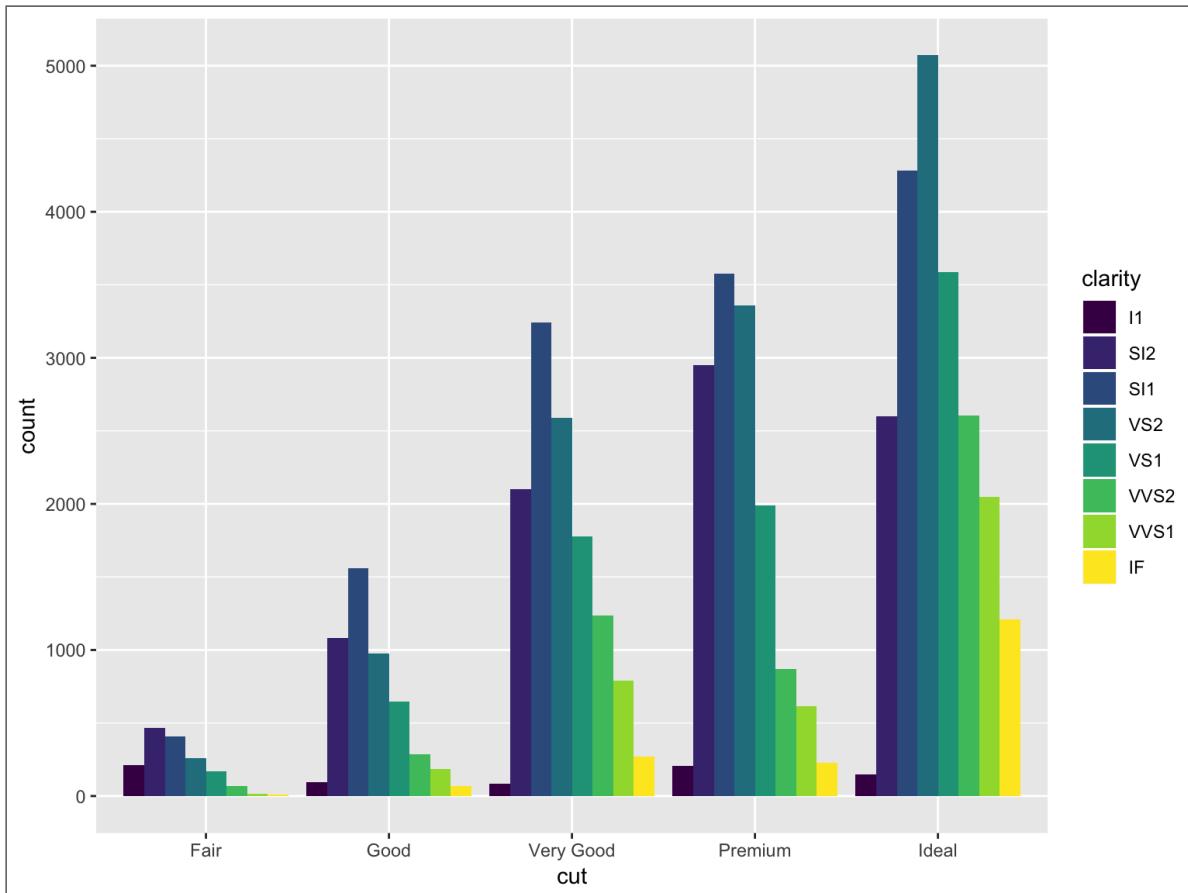
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



# One last visualization

A common scenario:

```
ggplot(data = diamonds) +  
  geom_bar(mapping = aes(x = cut, fill = clarity), position =  
  "dodge")
```



# Coding basics

# R as a calculator

```
1 / 200 * 30
```

```
## [1] 0.15
```

```
(59 + 73 + 2) / 3
```

```
## [1] 44.66667
```

```
sin(pi / 2)
```

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

# Assignments

```
x <- 3 * 4
```

# Calling functions

Calling R functions, in general:

```
function_name(arg1 = val1, arg2 = val2, ...)
```

An example function, `seq`:

```
seq(1, 10)
```

```
## [1] 1 2 3 4 5 6 7 8 9 10
```

# Data wrangling with `dplyr` and `tidyverse`

# Create RStudio project

1. File | New Project...
2. New Directory
3. Empty Project
4. Under “Directory name:” put in “CIRPA 2018”, under “Create project as a subdirectory of:”, pick somewhere you’ll remember.
5. Click “Create Project”, and RStudio will refresh, with your new project loaded.

## Download the data

We'll look at CANSIM Tables 0477-0058 and 0477-0059 which cover university revenues and expenditures, respectively.

To save time, I've already downloaded the data from CANSIM. You can get it here:

**<http://evancortens.com/cirpa/>**

# Load the data

We'll use `read_csv()` from `readr`, which I prefer to the base R function, as it has some more sensible defaults, and it's faster.

```
# http://www20.statcan.gc.ca/tables-
  tableaux/cansim/csv/04770058-eng.zip
# make sure the zip file is in the same directory as your
# RStudio project

revenue_raw <- read_csv('04770058-eng.zip')
```

```
## Parsed with column specification:
## cols(
##   Ref_Date = col_character(),
##   GEO = col_character(),
##   SCHOOL = col_character(),
##   REVENUE = col_character(),
##   FUND = col_character(),
##   Vector = col_character(),
##   Coordinate = col_character(),
##   Value = col_double()
## )
```

The `read_csv()` function tells us that it guessed the types of the various columns. In this situation, the default guesses are fine, but of course we can force it to treat columns certain ways if we wish.

# What does the data look like?

```
revenue_raw
```

```
## # A tibble: 122,240 x 8
##   Ref_Date GEO SCHOOL      REVENUE FUND      Vector Coordin...
##   <chr>     <chr> <chr>      <chr>    <chr>      <chr>  <chr>
## 1 2000/2001 Canada Total univ... Total r... Total f... v8070... 1.1.1.1
## 2 2001/2002 Canada Total univ... Total r... Total f... v8070... 1.1.1.1
## 3 2002/2003 Canada Total univ... Total r... Total f... v8070... 1.1.1.1
## 4 2003/2004 Canada Total univ... Total r... Total f... v8070... 1.1.1.1
## 5 2004/2005 Canada Total univ... Total r... Total f... v8070... 1.1.1.1
## 6 2005/2006 Canada Total univ... Total r... Total f... v8070... 1.1.1.1
## 7 2006/2007 Canada Total univ... Total r... Total f... v8070... 1.1.1.1
## 8 2007/2008 Canada Total univ... Total r... Total f... v8070... 1.1.1.1
## 9 2008/2009 Canada Total univ... Total r... Total f... v8070... 1.1.1.1
## 10 2009/2010 Canada Total univ... Total r... Total f... v8070... 1.1.1.1
## # ... with 122,230 more rows
```

We have 8 columns and 122,240 rows of data. `read_csv()` brings the data in as a `tibble`, which is just an R “data frame”, but with some handy defaults, some of which we’re seeing here. For instance, it gives us the size of the data frame in rows and columns, the types of the columns (e.g., “`<chr>`” for character) and only prints the first 10 rows, instead of overwhelming us with all of the data.

# What does the data look like?

```
View(revenue_raw) # in RStudio
```

Or click the icon in the Environment tab.

# What does the data look like?

```
head(revenue_raw, 1) # show just the first row
```

- *Ref\_Date*: fiscal year
- *GEO*: province or whole country
- *SCHOOL*: Direct participation in CAUBO survey, or via Statistics Canada, or combined (we'll just look at this)
- *REVENUE*: Type of income (e.g., tuition, provincial government, SSHRC, etc)
- *FUND*: Classification of income “in accordance with activities or objectives as specified by donors, ... regulations, restrictions, or limitations” (We’re just going to look at “Total funds (x 1,000)”, but the distinctions are important for full analysis)
- *Vector*: a CANSIM unique identifier
- *Coordinate*: ditto
- *Value*: the actual dollar value (x 1,000, per the FUND heading)

# What makes data “tidy”?

1. Each variable must have its own column.
2. Each observation must have its own row.
3. Each value must have its own cell.

(See Wickham, 2014 and *R for Data Science*, chapter 12)

Is the CANSIM data tidy?

- Nope! So let's tidy it.

# dplyr basics

- Pick observations by their values (`filter()`).
- Reorder the rows (`arrange()`).
- Pick variables by their names (`select()`).
- Create new variables with functions of existing variables (`mutate()`).
- Collapse many values down to a single summary (`summarise()`).
- All can be used in conjunction with `group_by()`

# dplyr cheat sheet

**<https://www.rstudio.com/resources/cheatsheets/>**  
(or Help | Cheatsheets in RStudio!)

## filter() the rows we want

```
filter(revenue_raw,  
       SCHOOL == 'Total universities and colleges',  
       FUND == 'Total funds (x 1,000)')
```

```
## # A tibble: 5,632 x 8  
##   Ref_Date   GEO     SCHOOL      REVENUE    FUND      Vector Coordin.  
##   <chr>      <chr>    <chr>      <chr>      <chr>      <chr>    <chr>  
## 1 2000/2001 Canada Total univ... Total r... Total f... v8070... 1.1.1.1  
## 2 2001/2002 Canada Total univ... Total r... Total f... v8070... 1.1.1.1  
## 3 2002/2003 Canada Total univ... Total r... Total f... v8070... 1.1.1.1  
## 4 2003/2004 Canada Total univ... Total r... Total f... v8070... 1.1.1.1  
## 5 2004/2005 Canada Total univ... Total r... Total f... v8070... 1.1.1.1  
## 6 2005/2006 Canada Total univ... Total r... Total f... v8070... 1.1.1.1  
## 7 2006/2007 Canada Total univ... Total r... Total f... v8070... 1.1.1.1  
## 8 2007/2008 Canada Total univ... Total r... Total f... v8070... 1.1.1.1  
## 9 2008/2009 Canada Total univ... Total r... Total f... v8070... 1.1.1.1  
## 10 2009/2010 Canada Total univ... Total r... Total f... v8070... 1.1.1.1  
## # ... with 5,622 more rows
```

`filter()` help tells us that everything after the first argument (i.e., the “...”) are: “Logical predicates defined in terms of the variables in `.data`. Multiple conditions are combined with `&`.”

# Comparison operators

```
> # greater than  
>= # greater than or equal to  
< # less than  
<= # less than or equal to  
!= # not equal  
== # equal  
  
%in% # not quite a comparison operator, but handy  
# x %in% y is true if the values of vector x are present in  
vector y
```

# Logical operators

Other operators we can use with `filter()`:

Complete set of boolean operations. x is the left-hand circle, y is the right-hand circle, and the shaded regions show which parts each operator selects.

## filter()

In other words, the previous command is equivalent to this:

```
filter(revenue_raw,  
       SCHOOL == 'Total universities and colleges' &  
       FUND == 'Total funds (x 1,000)')
```

```
## # A tibble: 5,632 x 8  
##   Ref_Date GEO    SCHOOL      REVENUE    FUND      Vector Coordin.  
##   <chr>     <chr>  <chr>      <chr>      <chr>      <chr>  <chr>  
## 1 2000/2001 Canada Total univ... Total r... Total f... v8070... 1.1.1.1  
## 2 2001/2002 Canada Total univ... Total r... Total f... v8070... 1.1.1.1  
## 3 2002/2003 Canada Total univ... Total r... Total f... v8070... 1.1.1.1  
## 4 2003/2004 Canada Total univ... Total r... Total f... v8070... 1.1.1.1  
## 5 2004/2005 Canada Total univ... Total r... Total f... v8070... 1.1.1.1  
## 6 2005/2006 Canada Total univ... Total r... Total f... v8070... 1.1.1.1  
## 7 2006/2007 Canada Total univ... Total r... Total f... v8070... 1.1.1.1  
## 8 2007/2008 Canada Total univ... Total r... Total f... v8070... 1.1.1.1  
## 9 2008/2009 Canada Total univ... Total r... Total f... v8070... 1.1.1.1  
## 10 2009/2010 Canada Total univ... Total r... Total f... v8070... 1.1.1.1  
## # ... with 5,622 more rows
```



I prefer this notation, as it's more explicit.

## filter()

But, one more thing: we need to assign the return value of the `filter()` function back to a variable:

```
revenue <- filter(revenue_raw,  
                  SCHOOL == 'Total universities and colleges' &  
                  FUND == 'Total funds (x 1,000)')
```

Shortcut for assignment operator: Alt-Hyphen in RStudio

# Combining operations

A new variable for each step gets cumbersome, so `dplyr` provides an operator, the pipe (`%>%`) that combines operations:

```
revenue_long <- revenue_raw %>%
  # only rows matching this
  filter(SCHOOL == 'Total universities and colleges' &
         FUND == 'Total funds (x 1,000)') %>%
  # remove these columns
  select(-SCHOOL, -FUND, -Vector, -Coordinate) %>%
  # fix up the date column
  mutate(Ref_Date = as.integer(stringr::str_sub(Ref_Date, 1,
                                             4)))
```

`x %>% f(y)` turns into `f(x, y)`, and  
`x %>% f(y) %>% g(z)` turns into `g(f(x, y), z)`  
etc.

Shortcut for pipe operator: Ctrl-Shift-M in RStudio

```
head(revenue_long, 1) # Looks good so far!
```

```
## # A tibble: 1 x 4
##   Ref_Date GEO      REVENUE        Value
##       <int> <chr>    <chr>        <dbl>
## 1     2000 Canada Total revenues 16224715
```

# Select helper functions

- There are a number of helper functions you can use within `select()`:
  - `starts_with("abc")`: matches names that begin with “abc”.
  - `ends_with("xyz")`: matches names that end with “xyz”.
  - `contains("ijk")`: matches names that contain “ijk”.
  - `matches("(.)\\1")`: selects variables that match a regular expression.
  - `num_range("x", 1:3)`: matches x1, x2 and x3.
  - `one_of(vector)`: columns whose names are in said vector.

# Tidy data with `tidyverse`

The two main `tidyverse` functions:

- `gather(data, key, value)`: Moves column names into a `key` column, with the values going into a single `value` column.
- `spread(data, key, value)`: Moves unique values of `key` column into column names, with values from the `value` column.

# tidyverse Cheatsheet

**<https://www.rstudio.com/resources/cheatsheets/>**  
(Data Import Cheatsheet)

# spread() CANSIM

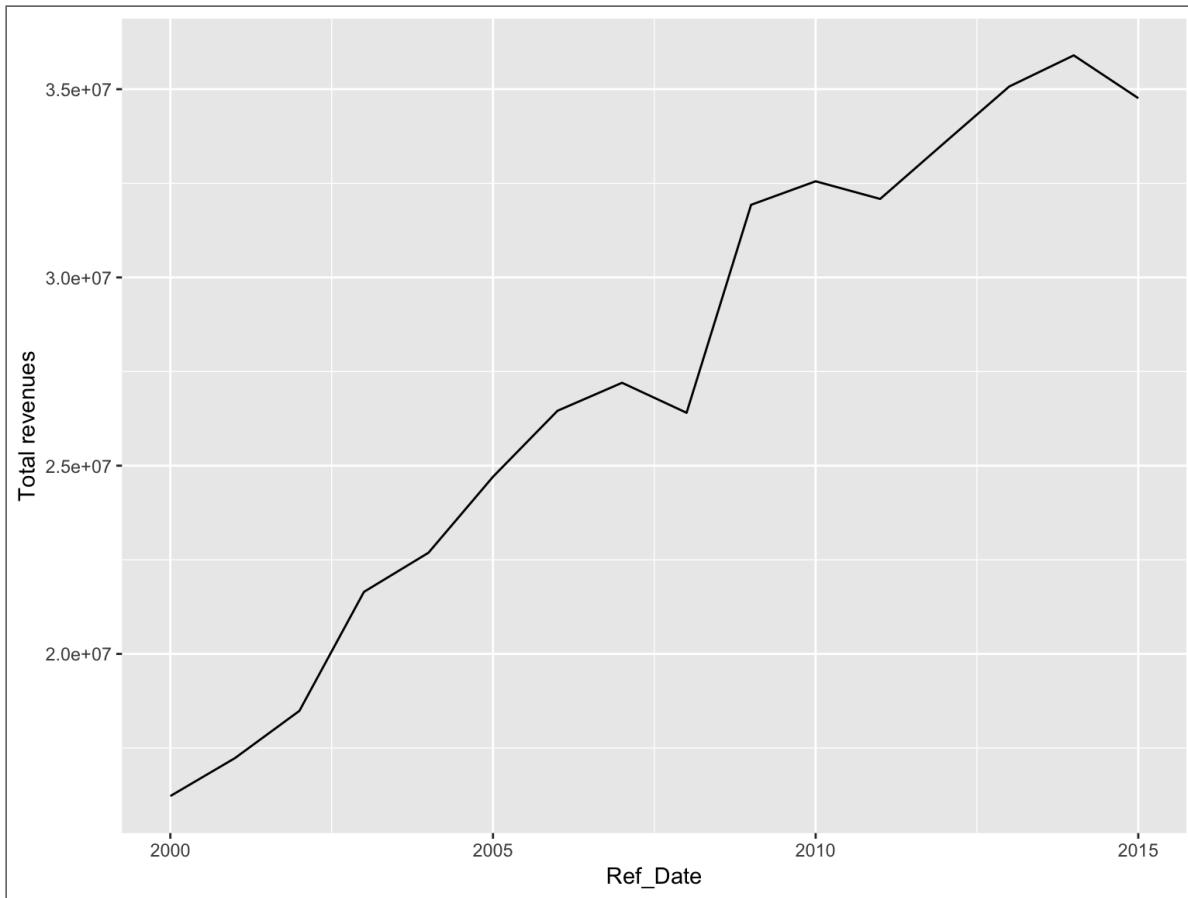
```
revenue <- revenue_long %>%  
  spread(REVENUE, Value)
```

```
revenue
```

```
## # A tibble: 176 x 34  
##   Ref_Date GEO `Canada Foundat... `Canada Researc... `Canadian  
##   <int> <chr>     <dbl>          <dbl>  
## 1 2000 Albe...      8208           1750  
## 2 2000 Brit...      5189           1525  
## 3 2000 Cana...    212418          19802  
## 4 2000 Mani...      7170            550  
## 5 2000 New ...     1745            300  
## 6 2000 Newf...     2799             0  
## 7 2000 Nova...     3234             0  
## 8 2000 Onta...    91066           8533  
## 9 2000 Prin...      355              0  
## 10 2000 Queb...    75988           7144  
## # ... with 166 more rows, and 29 more variables: `Credit cour  
## # tution` <dbl>, `Donations made by business enterprises` <  
## # `Donations made by individuals` <dbl>, `Donations made by  
## # not-for-profit organizations` <dbl>, Endowment <dbl>, Fed  
## # Foreign <dbl>, `Grants made by business enterprises` <dbl  
## # made by Individuals` <dbl>, `Grants made by not-for-profi  
## # organizations` <dbl>, `Health Canada` <dbl>, Investments  
## # Miscellaneous <dbl>, Municipal <dbl>, `Natural Sciences a  
## # Engineering Research Council` <dbl>, `Non-credit tution`  
## # `Non-federal` <dbl>, `Other federal` <dbl>, `Other fees`
```

# Visualizing

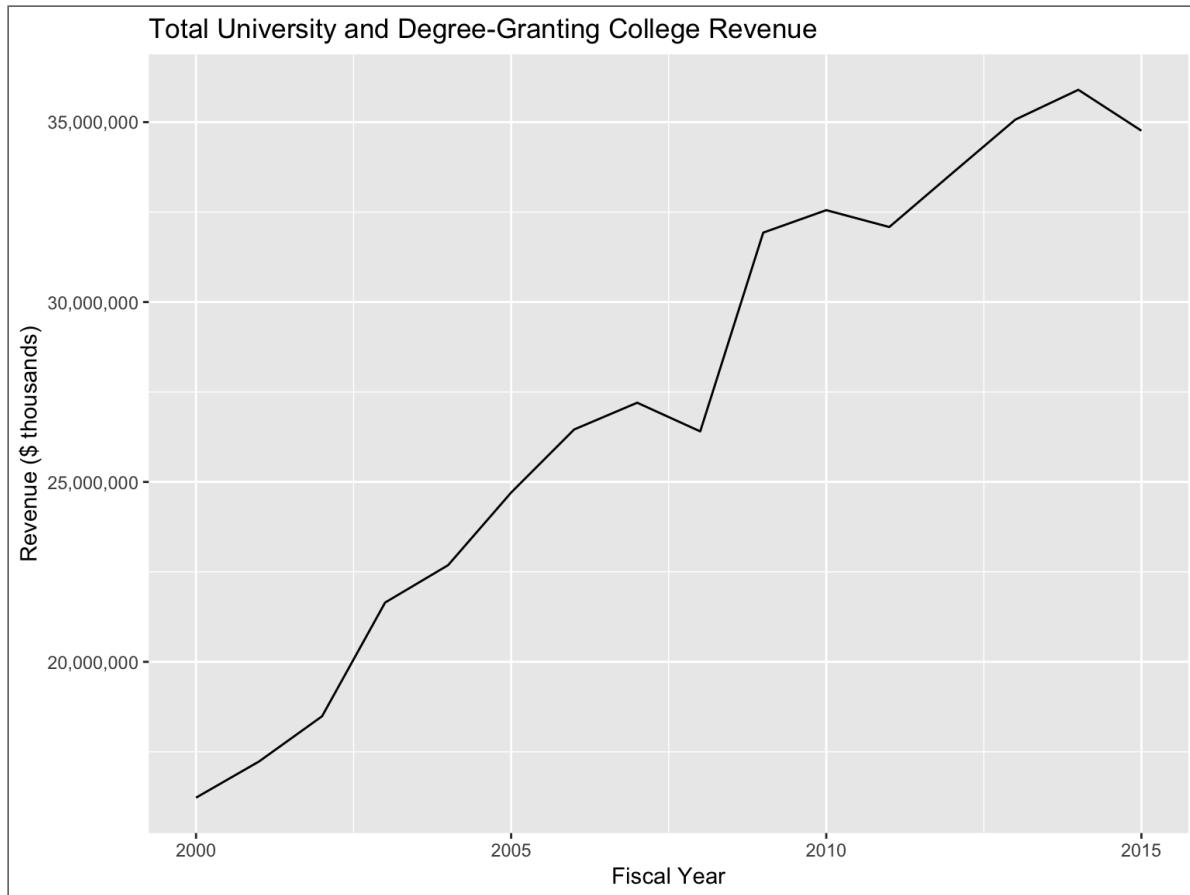
```
revenue %>%
  filter(GEO == 'Canada') %>%
  # pass the result to ggplot() as the first argument
  ggplot(aes(Ref_Date, `Total revenues`)) +
  # now it switches to + to combine, which is ggplot's way
  geom_line()
```



Not the prettiest, but it works!

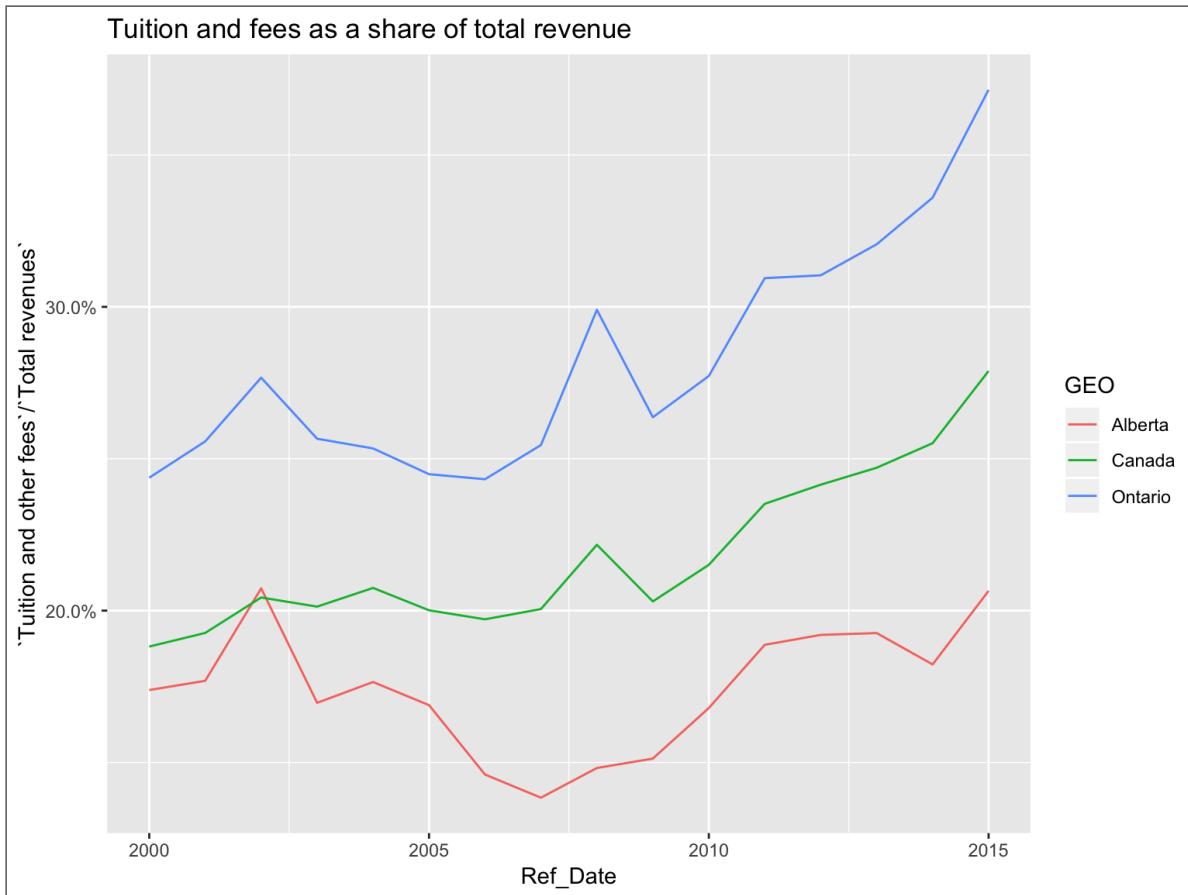
# Visualizing: Labels

```
revenue %>%
  filter(GEO == 'Canada') %>%
  ggplot(aes(Ref_Date, `Total revenues`)) +
  geom_line() +
  labs(title = 'Total University and Degree-Granting College
Revenue',
      x = 'Fiscal Year',
      y = 'Revenue ($ thousands') +
  scale_y_continuous(labels = scales::comma)
```



# Inline Calculation

```
revenue %>%
  filter(GEO %in% c('Canada', 'Alberta', 'Ontario')) %>%
  ggplot(aes(Ref_Date, `Tuition and other fees` / `Total
  revenues`, color = GEO)) +
  geom_line() +
  labs(title = 'Tuition and fees as a share of total revenue') +
  scale_y_continuous(labels = scales::percent)
```



# Exercises

1. Visualize another variable
2. Set some appropriate labels and scales
3. Graph a time subset (e.g., between 2005 and 2010)

## Other dplyr “verbs”

Which province took in the most tuition revenue in 2010?

```
revenue %>%
  filter(GEO != 'Canada', Ref_Date == 2010) %>%
  select(GEO, `Tuition and other fees`) %>%
  arrange(desc(`Tuition and other fees`))
```

```
## # A tibble: 10 x 2
##   GEO           `Tuition and other fees`
##   <chr>          <dbl>
## 1 Ontario        3563011
## 2 British Columbia 1009476
## 3 Quebec         825943
## 4 Alberta         711973
## 5 Nova Scotia    305597
## 6 Manitoba        176165
## 7 Saskatchewan   173600
## 8 New Brunswick   146213
## 9 Newfoundland and Labrador 61537
## 10 Prince Edward Island 27878
```

# Create a new variable

```
revenue %>%
  select(Ref_Date, GEO, `Tuition and other fees`, `Total
  revenues`) %>%
  mutate(tuition_share = `Tuition and other fees` / `Total
  revenues`)
```

```
## # A tibble: 176 x 5
##   Ref_Date GEO      `Tuition and othe... `Total revenues` tui...
##   <int> <chr>                <dbl>                  <dbl>
## 1 2000 Alberta            298464                1716783
## 2 2000 British Col...     321017                2020474
## 3 2000 Canada             3052960               16224715
## 4 2000 Manitoba           99932                 629163
## 5 2000 New Brunswi...      79153                 356899
## 6 2000 Newfoundlan...     49954                 268822
## 7 2000 Nova Scotia        173980                685133
## 8 2000 Ontario            1509442               6192454
## 9 2000 Prince Edwa...      13778                 63238
## 10 2000 Quebec             407458                3613399
## # ... with 166 more rows
```

Keeps the same number of rows

# Useful “mutations”

- Arithmetic: `+`, `-`, `*`, `/`, `^`
- Modular arithmetic: `%/%` (integer division), `%%` (remainder)
- Logs: `log()`, `log2()`, `log10()`
- Offsets: `lead()`, `lag()`
- Cumulative and rolling aggregates: `cumsum()`, `cumprod()`, `cummin()`, `cummax()`, `cummean()`
- Logical comparisons: `<`, `<=`, `>`, `>=`, `!=`, `==`
- Ranking: `min_rank()`, `row_number()`,  
`dense_rank()`, `percent_rank()`, `cume_dist()`,  
`ntile()`
- `ifelse()`
- `recode()`
- `case_when()`

# Summarize (aggregate)

```
revenue %>%
  group_by(GEO) %>%
  summarise(avg_endowment_revenue = mean(Endowment)) %>%
  arrange(-avg_endowment_revenue)
```

```
## # A tibble: 11 x 2
##   GEO           avg_endowment_revenue
##   <chr>          <dbl>
## 1 Canada        526071.
## 2 Ontario       211475.
## 3 Quebec        87323.
## 4 Alberta        80069
## 5 British Columbia 75752.
## 6 Nova Scotia    31148.
## 7 Saskatchewan   21256.
## 8 New Brunswick  13778.
## 9 Newfoundland and Labrador 2878.
## 10 Manitoba      1646.
## 11 Prince Edward Island 747.
```

# Useful summarising functions:

- `mean(x)`, `median(x)`
- `sd(x)`, `IQR(x)` (interquartile range), `mad(x)` (median absolute deviation)
- `min(x)`, `max(x)`, `quantile(x, 0.25)`
- `first(x)`, `nth(x, 2)`, `last(x)`
- `n(x)`, `n_distinct(x)`
- Counts and proportions of logical values:  
`sum(x > 10)`, `mean(y == 0)`
  - TRUE is converted to 1 and FALSE to 0

## Exercise:

Compute the total tuition revenue of the three western-most provinces in 2007.

```
revenue %>%
  filter(GEO %in% c('British Columbia', 'Alberta',
  'Saskatchewan'), Ref_Date == 2007) %>%
  summarise(sum(`Tuition and other fees`))
```

```
## # A tibble: 1 x 1
##   `sum(\`Tuition and other fees\``)`
##       <dbl>
## 1     1374509
```

# Group by new variable

These three provinces compared to all other provinces and national average.

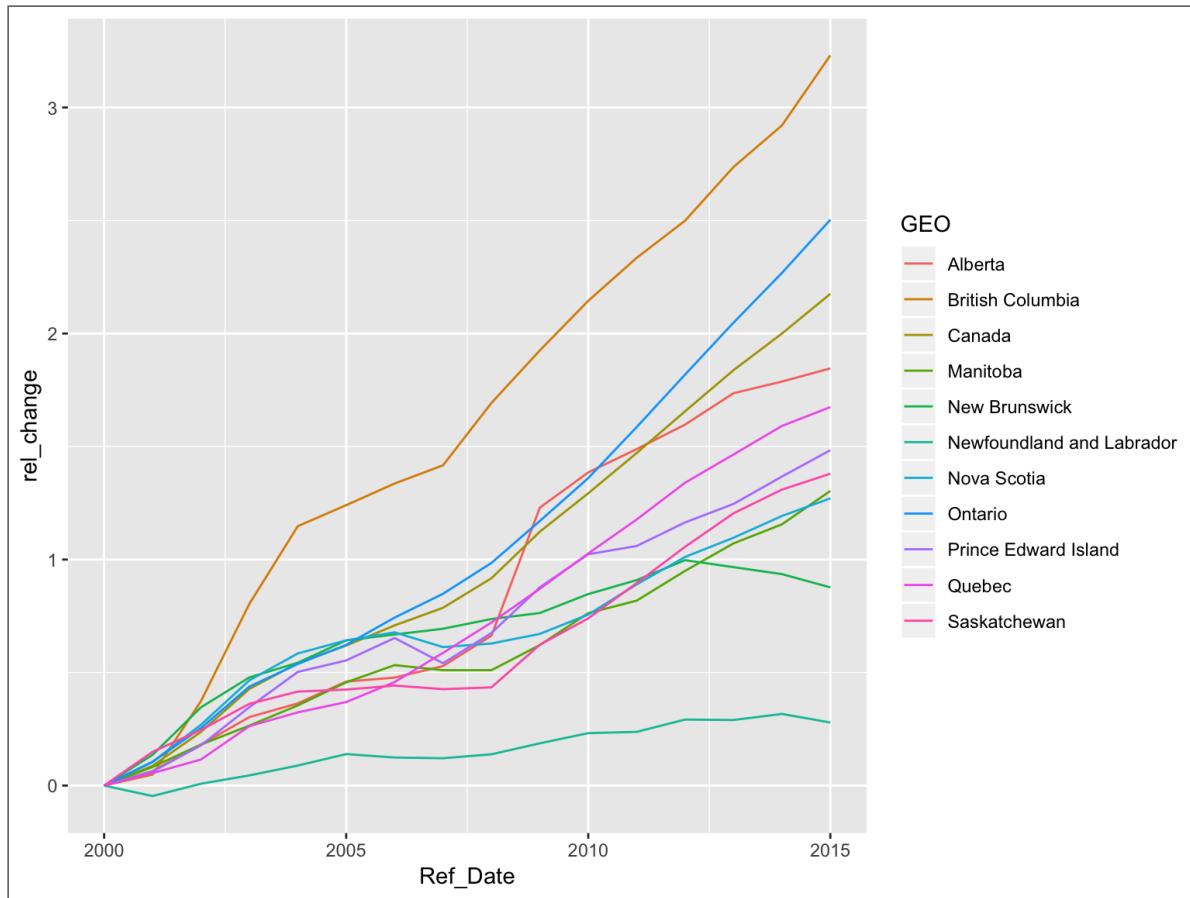
```
revenue %>%
  filter(Ref_Date == 2007) %>%
  mutate(category = case_when(
    GEO %in% c('British Columbia', 'Alberta', 'Saskatchewan') ~
    '3 Western Provinces',
    GEO == 'Canada' ~ GEO,
    TRUE ~ 'All Other Provinces'
  )) %>%
  group_by(category) %>%
  summarise(sum(`Tuition and other fees`))
```

```
## # A tibble: 3 x 2
##   category           `sum(`Tuition and other fees\``)`
##   <chr>                  <dbl>
## 1 3 Western Provinces     1374509
## 2 All Other Provinces     4079866
## 3 Canada                  5454375
```

# Change relative first year

```
tuition_relative_change <- revenue %>%
  arrange(Ref_Date, GEO) %>%
  group_by(GEO) %>%
  mutate(rel_change = (`Tuition and other fees` - first(`Tuition and other fees`)) / first(`Tuition and other fees`)) %>%
  select(Ref_Date, GEO, `Tuition and other fees`, rel_change)

ggplot(tuition_relative_change, aes(Ref_Date, rel_change, color = GEO)) +
  geom_line()
```



# Logical sums

Count years with positive endowment income by province

```
revenue %>%
  group_by(GEO) %>%
  summarise(num_pos_endow = sum(Endowment > 0))
```

```
## # A tibble: 11 x 2
##   GEO                  num_pos_endow
##   <chr>                <int>
## 1 Alberta                 13
## 2 British Columbia         13
## 3 Canada                  13
## 4 Manitoba                 14
## 5 New Brunswick            13
## 6 Newfoundland and Labrador 13
## 7 Nova Scotia               14
## 8 Ontario                   12
## 9 Prince Edward Island      13
## 10 Quebec                  16
## 11 Saskatchewan              12
```

# Exercises

- Compare the Maritimes to the rest of Canada on one or more variable
- Between 2005 and 2007, which provinces received more than \$100 M in Total grants?
- Which provinces, other than Nova Scotia and Ontario, had more than \$20 M in Endowment income in 2003?
- Which province was in “third place” in total revenue in 2012?

# purrr (quickly)

(There's a lot going on here, just in the slides for your quick reference.)

```
revenue %>%
  filter(GEO != 'Canada') %>%
  group_by(Ref_Date) %>%
  summarise(revenue_quantiles = list(qu quantile(`Total revenues`,
  c(0.25, 0.5, 0.75)))) %>%
  mutate(
    low_25 = map_dbl(revenue_quantiles, "25%"),
    mid = map_dbl(revenue_quantiles, '50%'),
    high_75 = map_dbl(revenue_quantiles, '75%')
  )
```

```
## # A tibble: 16 x 5
##   Ref_Date revenue_quantiles   low_25      mid  high_75
##   <int>     <list>          <dbl>     <dbl>     <dbl>
## 1 2000     <dbl [3]>       424965.  681742. 1944551.
## 2 2001     <dbl [3]>       417664.  700424. 2126550.
## 3 2002     <dbl [3]>       465112.  729434. 2378510.
## 4 2003     <dbl [3]>       480404.  804636. 2708008.
## 5 2004     <dbl [3]>       530995.  853132. 2818174.
## 6 2005     <dbl [3]>       560023.  911819. 3257599.
## 7 2006     <dbl [3]>       616412.  964536. 3320636.
## 8 2007     <dbl [3]>       592927.  944042. 3560364.
## 9 2008     <dbl [3]>       568461.  968932. 3404852.
## 10 2009    <dbl [3]>       741016. 1161066 4363439.
## 11 2010    <dbl [3]>       787082. 1213456 4436956.
## 12 2011    <dbl [3]>       775059. 1169906. 4177081
## 13 2012    <dbl [3]>       799797. 1240542 4296184.
## 14 2013    <dbl [3]>       791843. 1298845 4437751.
## 15 2014    <dbl [3]>       799297. 1316785 4623504.
## 16 2015    <dbl [3]>       770571. 1261936. 4451983.
```

# Vectors (and other types)

# Missing data

```
NA > 5
```

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

```
10 == NA
```

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

```
NA + 10
```

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

What about this?

```
NA / 2
```

# Filtering and NAs

```
x <- NA  
is.na(x)
```

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

`filter()` only includes rows where the condition is TRUE; it excludes both FALSE and NA values. If you want to preserve missing values, ask for them explicitly:

```
test_data <- tibble(x = c(1, NA, 3))  
filter(test_data, x > 1)
```

```
## # A tibble: 1 x 1  
##       x  
##   <dbl>  
## 1     3
```

```
filter(test_data, is.na(x) | x > 1)
```

```
## # A tibble: 2 x 1  
##       x  
##   <dbl>  
## 1     NA  
## 2     3
```



# Dealing with vectors...

```
c(1, 2, 3)
```

```
## [1] 1 2 3
```

```
1:3
```

```
## [1] 1 2 3
```

# Dealing with vectors...

```
1:3 + 1:9
```

```
## [1] 2 4 6 5 7 9 8 10 12
```

## What about...

```
1:3 + 1:10
```

# Dealing with vectors...

```
1:3 + 1:10
```

```
## Warning in 1:3 + 1:10: longer object length is not a multiple of  
## object length
```

```
## [1] 2 4 6 5 7 9 8 10 12 11
```

What happened?

# Missing values

```
mean(c(1,2,3))
```

```
## [1] 2
```

What do we expect here?

```
mean(c(1,NA,3))
```

# Missing values

```
mean(c(1,NA,3))
```

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

```
mean(c(1,NA,3), na.rm = TRUE)
```

```
## [1] 2
```

# Vectors

# Vectors

- Coercion (explicit and implicit)
  - Type hierarchy
- Atomic vectors (homogenous) vs lists (can nest/heterogeneous)
- Names
- Subsetting

# Vectors: Naming

```
c(x = 1, y = 2, z = 4)
```

```
## x y z  
## 1 2 4
```

```
set_names(1:3, c("a", "b", "c"))
```

```
## a b c  
## 1 2 3
```

# Vectors: Subsetting

```
# positive values
x <- c("one", "two", "three", "four", "five")
x[c(3, 2, 5)]
```

```
## [1] "three" "two"   "five"
```

```
# negative values
x[c(-1, -3, -5)]
```

```
## [1] "two"  "four"
```

```
#> [1] "two"  "four"

# subset a named vector w/character vector
x <- c(abc = 1, def = 2, xyz = 5)
x[c("xyz", "def")]
```

```
## xyz def
## 5    2
```

# Vectors: Logical subsetting

```
x <- c(10, 3, NA, 5, 8, 1, NA)  
  
# All non-missing values of x  
x[!is.na(x)]
```

```
## [1] 10  3   5   8   1
```

```
#> [1] 10  3   5   8   1  
  
# All even (or missing!) values of x  
x[x %% 2 == 0]
```

```
## [1] 10 NA  8 NA
```

```
#> [1] 10 NA  8 NA
```

# Lists

```
x <- list(1, 2, 3)  
x
```

```
## [[1]]  
## [1] 1  
##  
## [[2]]  
## [1] 2  
##  
## [[3]]  
## [1] 3
```

```
str(x) # structure
```

```
## List of 3  
## $ : num 1  
## $ : num 2  
## $ : num 3
```

# Lists: Can contain different types of objects:

```
y <- list("a", 1L, 1.5, TRUE)  
str(y)
```

```
## List of 4  
## $ : chr "a"  
## $ : int 1  
## $ : num 1.5  
## $ : logi TRUE
```

# Lists: Or other lists:

```
z <- list(list(1, 2), list(3, 4))
str(z)
```

```
## List of 2
## $ :List of 2
##   ..$ : num 1
##   ..$ : num 2
## $ :List of 2
##   ..$ : num 3
##   ..$ : num 4
```

# Modeling

# Types of Modeling

- define a family of models - expressed as an equation - you are trying to estimate the **parameters** of the question
- generate a fitted model that fills in the parameters

## The goal

You are trying to get the model from the family that best fits the data. That doesn't mean it's a "good" or "true" model.

# Simulated Data

The `modelr` package has a simulated dataset called `sim1`.

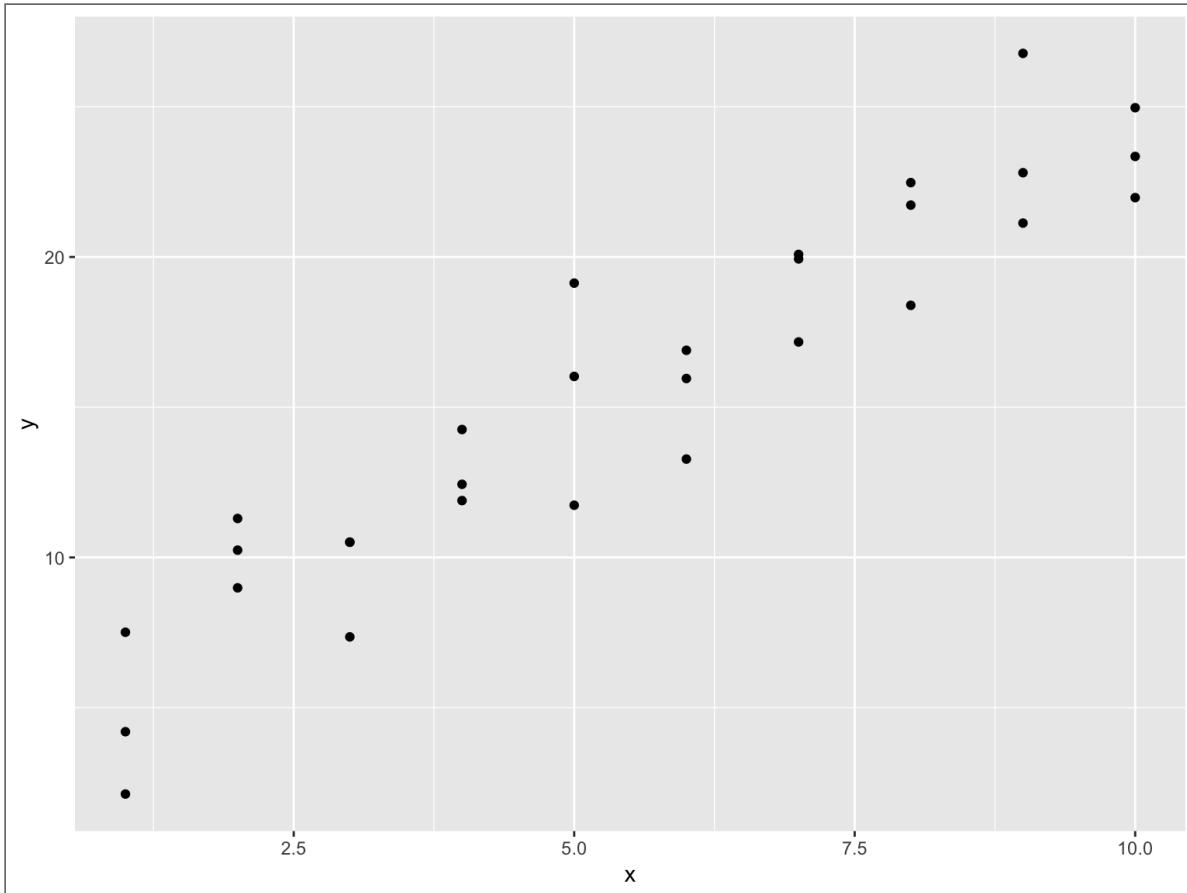
```
library(modelr)
sim1$x
```

```
## [1] 1 1 1 2 2 2 3 3 3 4 4 4 5 5 5 6 6 6 7 7
## [24] 8 9 9 9 10 10 10
```

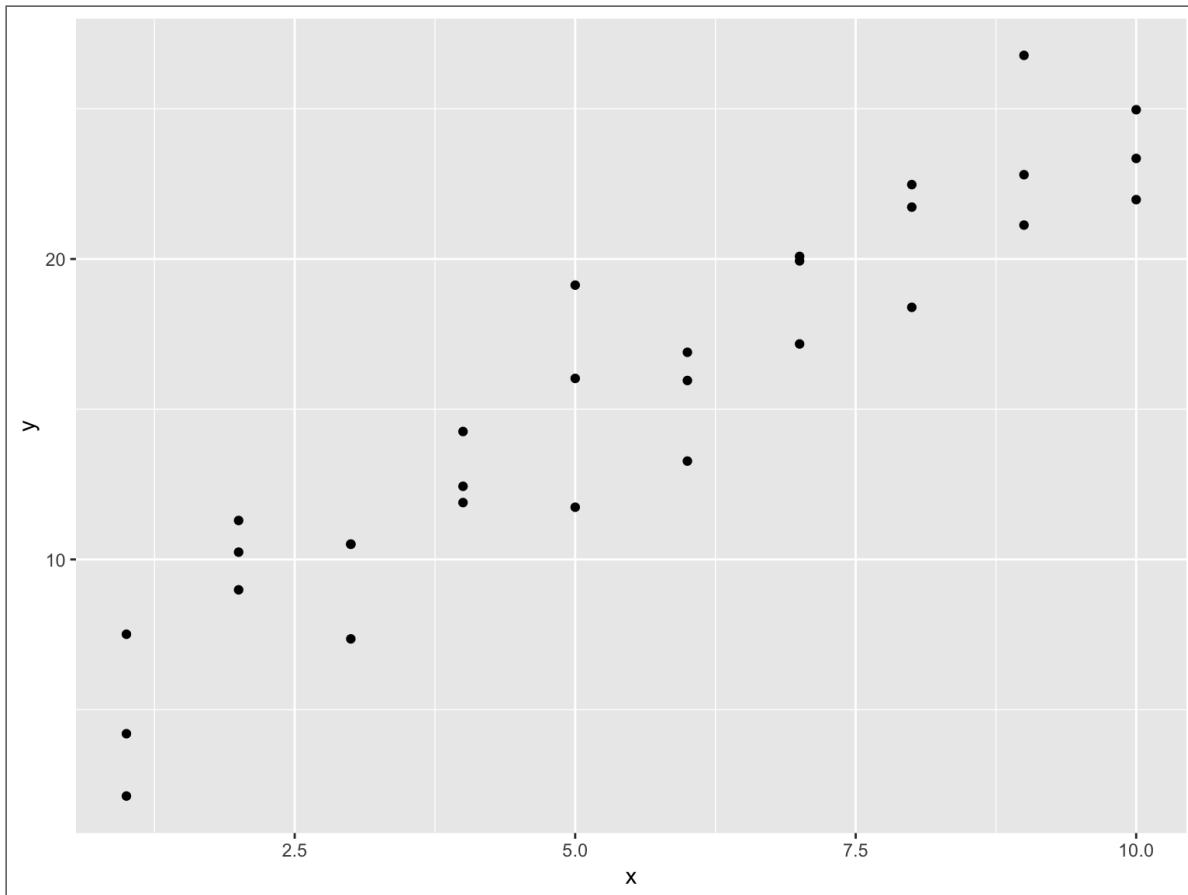


# Plotting sim1

```
ggplot(sim1, aes(x, y)) +  
  geom_point()
```



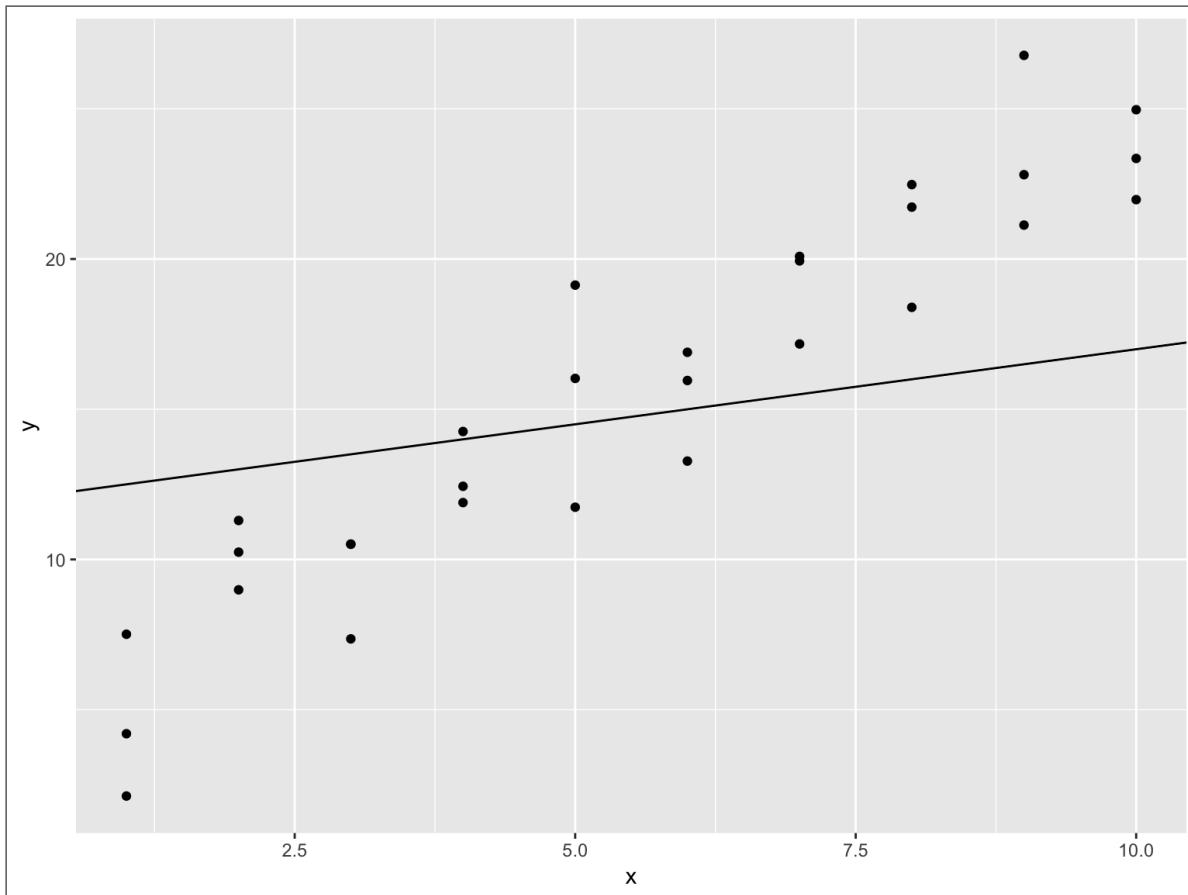
# Plotting sim1



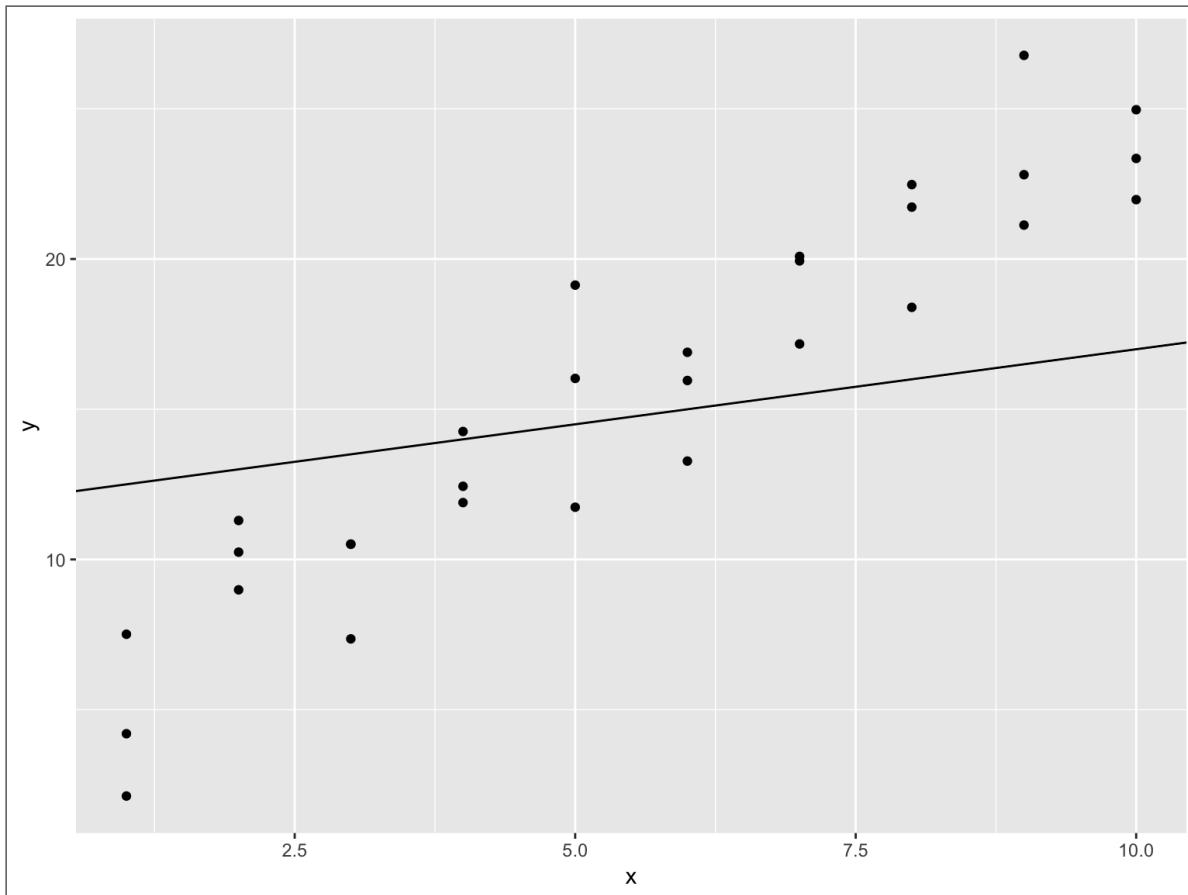
- we can see that there is a pattern from the scatter plot
- the pattern looks like a straight line - `geom_abline` plots straight line on the graph

# Using geom\_abline

```
ggplot(sim1, aes(x, y)) +  
  geom_point() +  
  geom_abline(aes(intercept = 12, slope = 0.5))
```



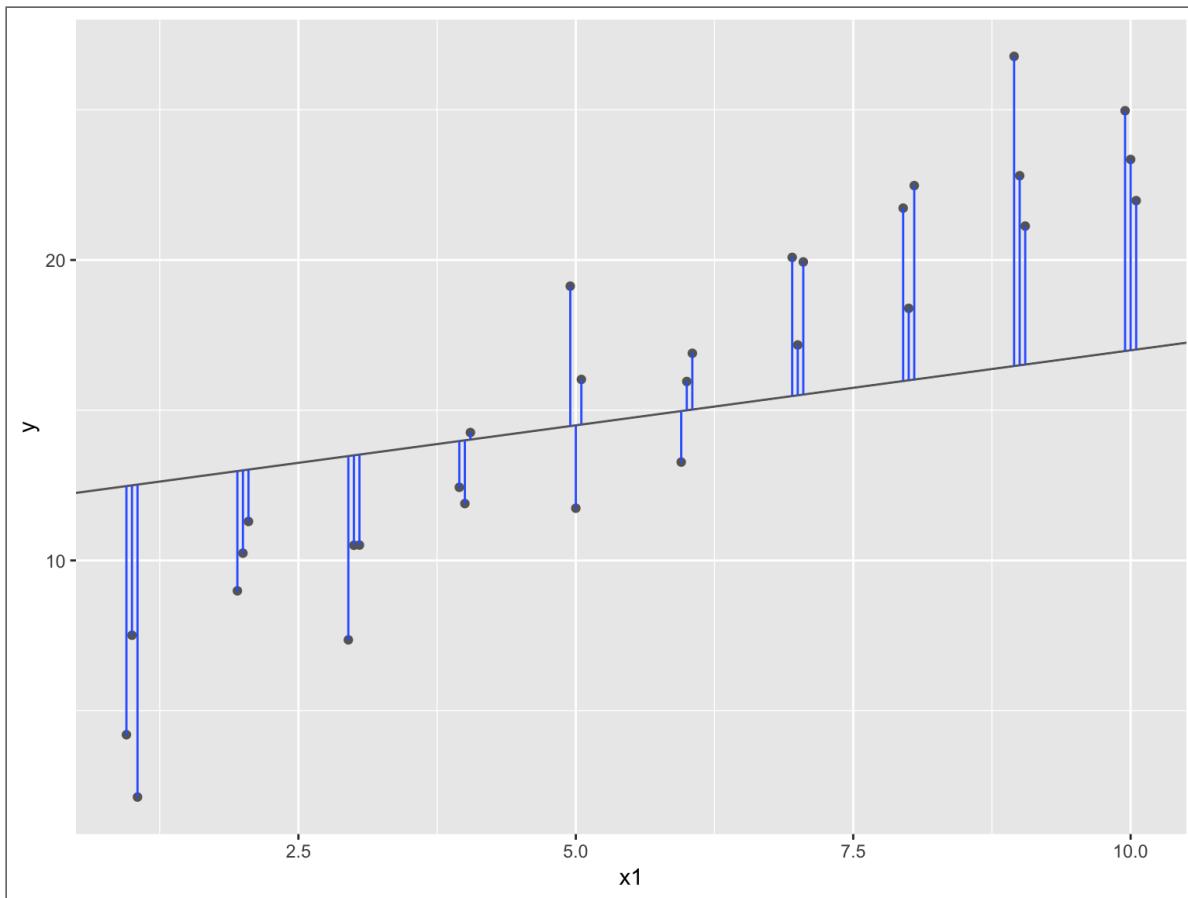
# Using geom\_abline



That's just one model - how can we tell if it's good or not?

# Evaluating Predictive Models

We look at the difference between the predictions and the actual data.



# Defining a model as an R function

We can find this for any model by creating an R function. We take the intercept and add the slope multiplied by x.

```
model1 <- function(data, a) {  
  a[1] + data$x * a[2]  
}  
  
model1(sim1, c(12, 0.5))
```

```
## [1] 12.5 12.5 12.5 13.0 13.0 13.0 13.5 13.5 13.5 14.0 14.0 14.0  
## [15] 14.5 15.0 15.0 15.0 15.5 15.5 15.5 16.0 16.0 16.0 16.5 16.5  
## [29] 17.0 17.0
```

# Difference between predicted and actual

Once we have the model's prediction, we can look at the difference between those and what the actual data is.

```
measure_distance <- function(data, mod) {  
  diff <- data$y - model1(data, mod)  
  diff  
}  
measure_distance(sim1, c(12, 0.5))
```

```
## [1] -8.3000870 -4.9893659 -10.3745272 -4.0111426 -2.7568946  
## [6] -1.7031768 -6.1436353 -2.9946506 -2.9883992 -1.5654109  
## [11] -2.1073988  0.2579641  4.6300498 -2.7619788  1.5248539  
## [16] -1.7260230  0.9559750  1.8947962  4.5859927  1.6718503  
## [21]  4.4363088  5.7259025  2.3909129  6.4755526 10.2770099  
## [26]  6.3051098  4.6283053  7.9680994  6.3464221  4.9752007
```



# Using squared distances

We want to both put all distances as positive numbers and more heavily penalize predictions that are far from the actual values - so we square the difference.

```
measure_distance <- function(data, mod) {  
  diff <- data$y - model1(data, mod)  
  diff ^ 2  
}  
measure_distance(sim1, c(12, 0.5))
```

```
## [1] 68.89144476 24.89377199 107.63081509 16.08926474 7.600  
## [6] 2.90081117 37.74425497 8.96793224 8.93052986 2.450  
## [11] 4.44112956 0.06654547 21.43736106 7.62852691 2.325  
## [16] 2.97915534 0.91388814 3.59025256 21.03132891 2.795  
## [21] 19.68083613 32.78595957 5.71646454 41.93278203 105.616  
## [26] 39.75440948 21.42120989 63.49060769 40.27707338 24.752
```



# Summary Measure of Distance

We want to summarize it - so we take the mean across all the predictions and take the square root to put the differences back in terms of the original units.

```
measure_distance <- function(data, mod) {  
  diff <- data$y - model1(data, mod)  
  sqrt(mean(diff ^ 2))  
}  
measure_distance(sim1, c(12, 0.5))
```

```
## [1] 4.995789
```

# Use this message to choose a model

So, to choose between models – we can just put in different values.

```
measure_distance(sim1, c(7, 1.5))
```

```
## [1] 2.665212
```

# Random Models Compared

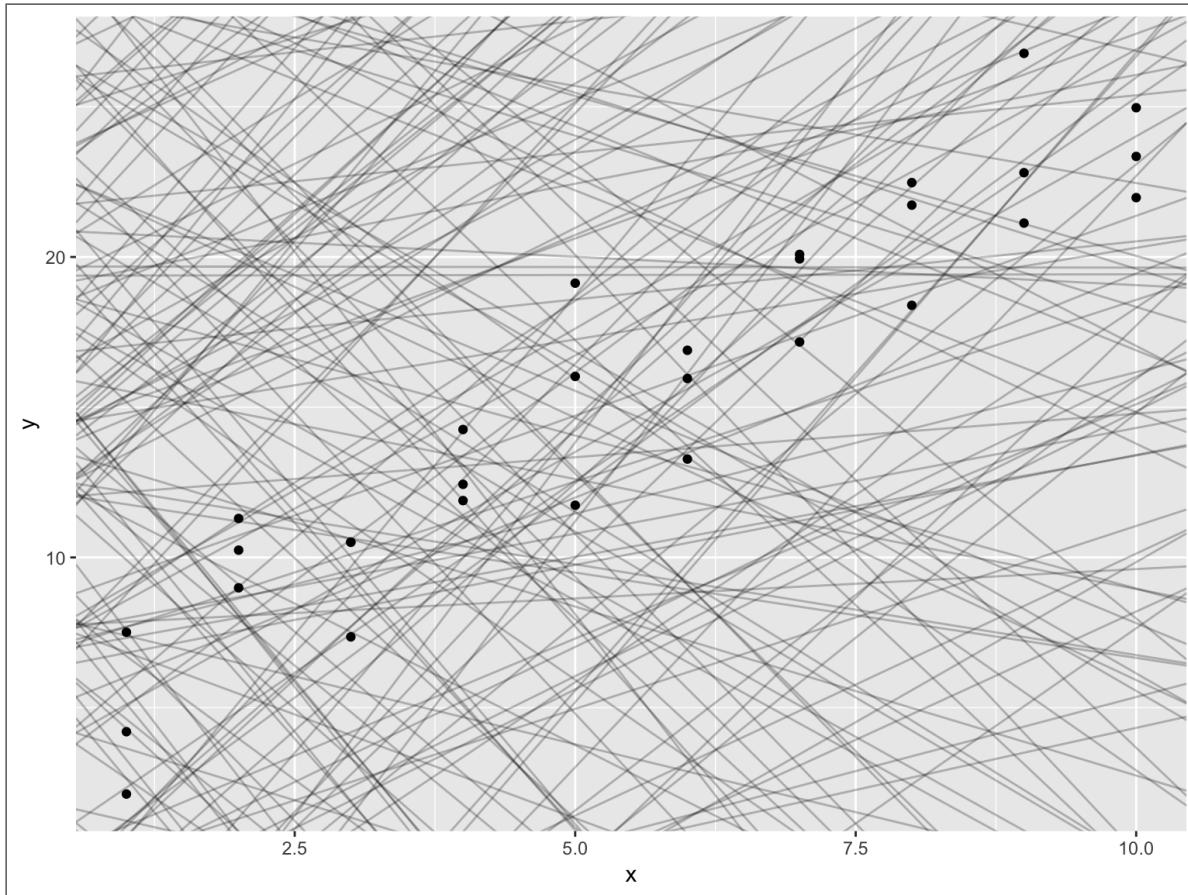
So, we can generate a lot of models and pick the one with the lowest error.

```
models <- tibble(  
  a1 = runif(250, -20, 40),  
  a2 = runif(250, -5, 5)  
)  
  
sim1_dist <- function(a1, a2) {  
  measure_distance(sim1, c(a1, a2))  
}  
  
models %>%  
  mutate(dist = purrr::map2_db1(a1, a2, sim1_dist)) -> models  
models
```

```
## # A tibble: 250 x 3  
##       a1     a2   dist  
##   <dbl> <dbl> <dbl>  
## 1 -3.97 -4.05 45.3  
## 2  8.23 -1.38 18.0  
## 3  9.46 -2.74 25.3  
## 4 -15.4 -3.61 53.4  
## 5 -0.415 -2.10 30.0  
## 6  6.09  0.733 6.92  
## 7 -11.8 -3.06 46.6  
## 8  9.89  3.46 14.2  
## 9 13.2 -1.88 17.1  
## 10 2.61 -3.08 33.3  
## # ... with 240 more rows
```

# Plot the Random Models

```
ggplot(sim1, aes(x, y)) +  
  geom_abline(aes(intercept = a1, slope = a2), data = models,  
  alpha = 1/4) +  
  geom_point()
```



# The 10 Best Models

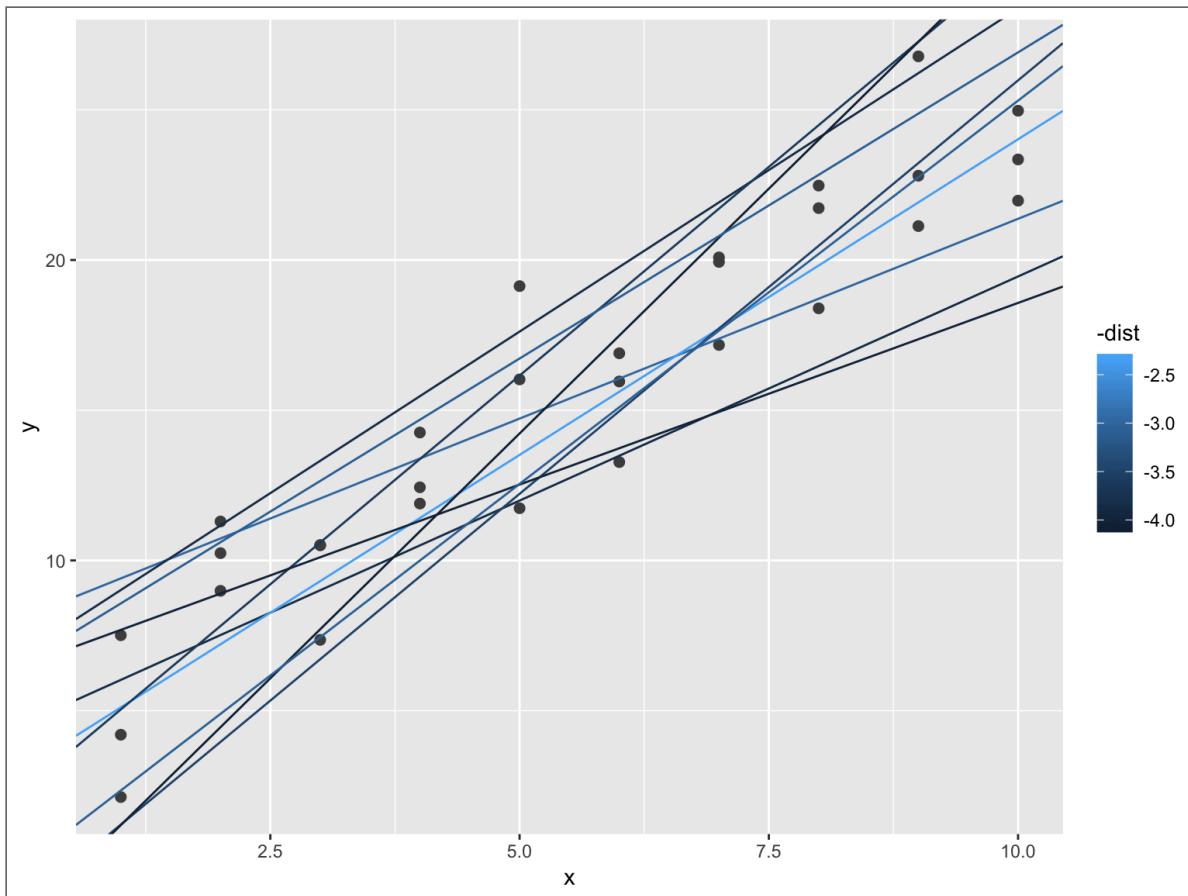
```
models %>%  
  filter(rank(dist) <= 10) %>%  
  arrange(dist)
```

```
## # A tibble: 10 x 3  
##       a1     a2   dist  
##   <dbl> <dbl> <dbl>  
## 1  3.01  2.10  2.33  
## 2  8.07  1.33  2.97  
## 3 -0.209  2.55  3.07  
## 4  6.53  2.04  3.09  
## 5 -1.54  2.75  3.50  
## 6  2.26  2.78  3.61  
## 7  4.53  1.49  3.84  
## 8  6.86  2.15  3.84  
## 9  6.48  1.21  4.00  
## 10 -2.07  3.26  4.08
```

# Plot all the random models.

Plot those models on the data. Lighter colored models are closer.

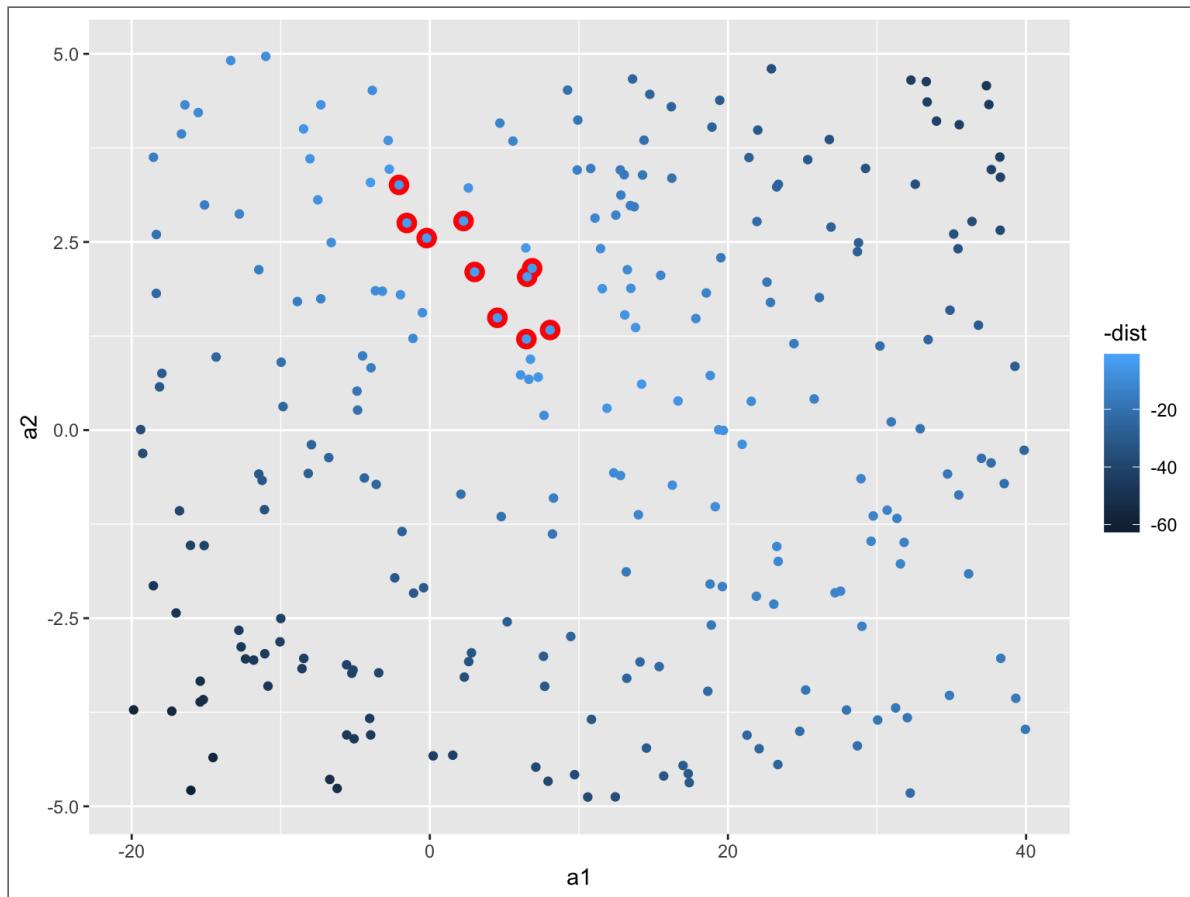
```
ggplot(sim1, aes(x, y)) +  
  geom_point(size = 2, color = "grey30") +  
  geom_abline(  
    aes(intercept = a1, slope = a2, color = -dist),  
    data = filter(models, rank(dist) <= 10)  
)
```



# Parameter Space

Now, we switch to parameter space... We plot every candidate model - showing the distance in color. We highlight our 10 best models in red.

```
ggplot(models, aes(a1, a2)) +  
  geom_point(data = filter(models, rank(dist) <= 10), size = 4,  
  colour = "red") +  
  geom_point(aes(colour = -dist))
```



## Zoom In

So, you can zoom in on the area where the best models tend to be. Note the axes and the legend.

```
grid <- expand.grid(  
  a1 = seq(-5, 20, length = 25),  
  a2 = seq(1, 3, length = 25)  
 ) %>%  
  mutate(dist = purrr::map2_dbl(a1, a2, sim1_dist))
```

```
fine_grid <- expand.grid(  
  a1 = seq(-5, 20, length = 50),  
  a2 = seq(1, 3, length = 50)  
 ) %>%  
  mutate(dist = purrr::map2_dbl(a1, a2, sim1_dist))
```

# The new best 10 models

What do we get as the new best 10 models?

```
grid %>%
  filter(rank(dist) <= 10) %>%
  arrange(dist)
```

```
##      a1      a2      dist
## 1 4.375000 2.000000 2.137234
## 2 4.375000 2.083333 2.155409
## 3 3.333333 2.166667 2.168677
## 4 5.416667 1.916667 2.210294
## 5 3.333333 2.250000 2.212637
## 6 5.416667 1.833333 2.218550
## 7 4.375000 1.916667 2.241533
## 8 3.333333 2.083333 2.246169
## 9 4.375000 2.166667 2.293149
## 10 2.291667 2.333333 2.308274
```

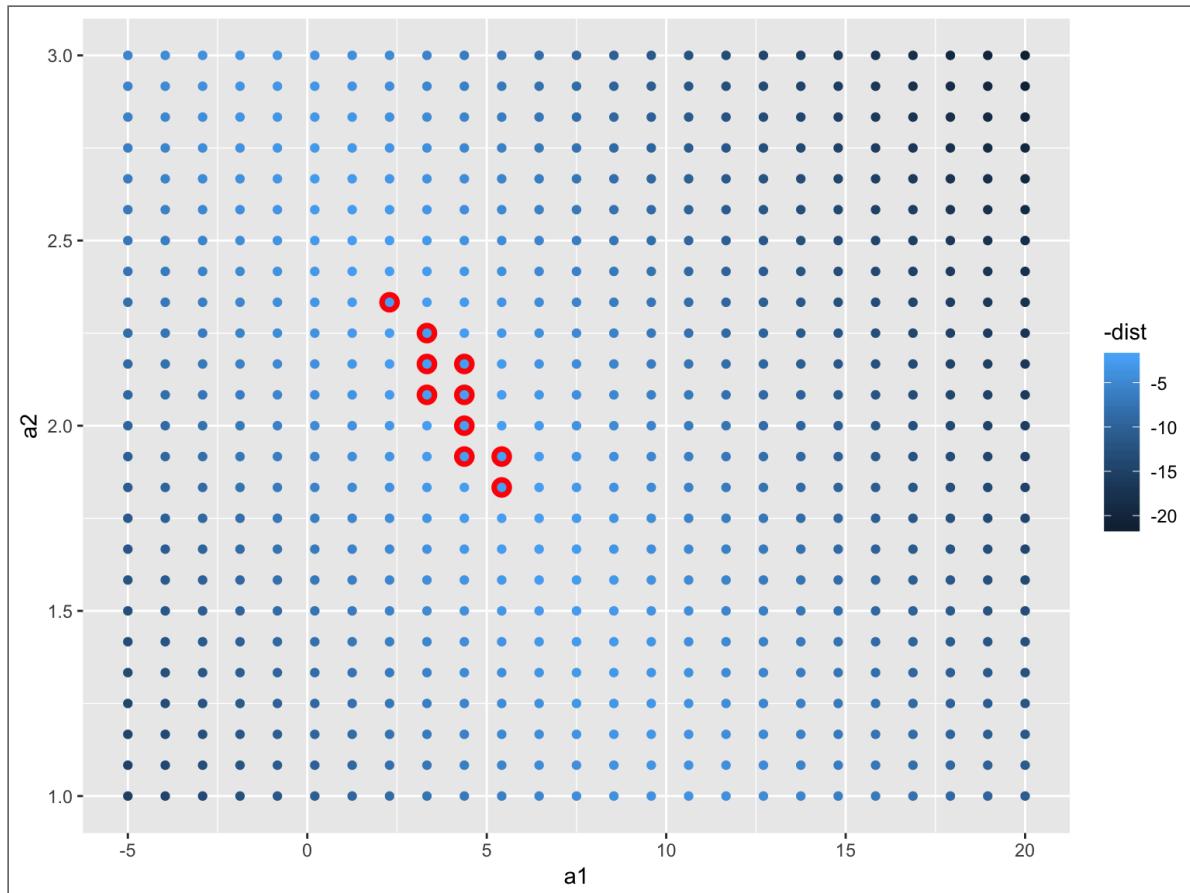
```
fine_grid %>%
  filter(rank(dist) <= 10) %>%
  arrange(dist)
```

```
##      a1      a2      dist
## 1 4.183673 2.061224 2.128424
## 2 4.693878 1.979592 2.139589
## 3 4.183673 2.020408 2.140222
## 4 3.673469 2.142857 2.144759
## 5 4.183673 2.102041 2.146651
## 6 3.673469 2.102041 2.150084
## 7 4.693878 2.020408 2.151341
## 8 4.693878 1.938776 2.157704
## 9 3.673469 2.183673 2.169193
## 10 5.204082 1.897959 2.177829
```



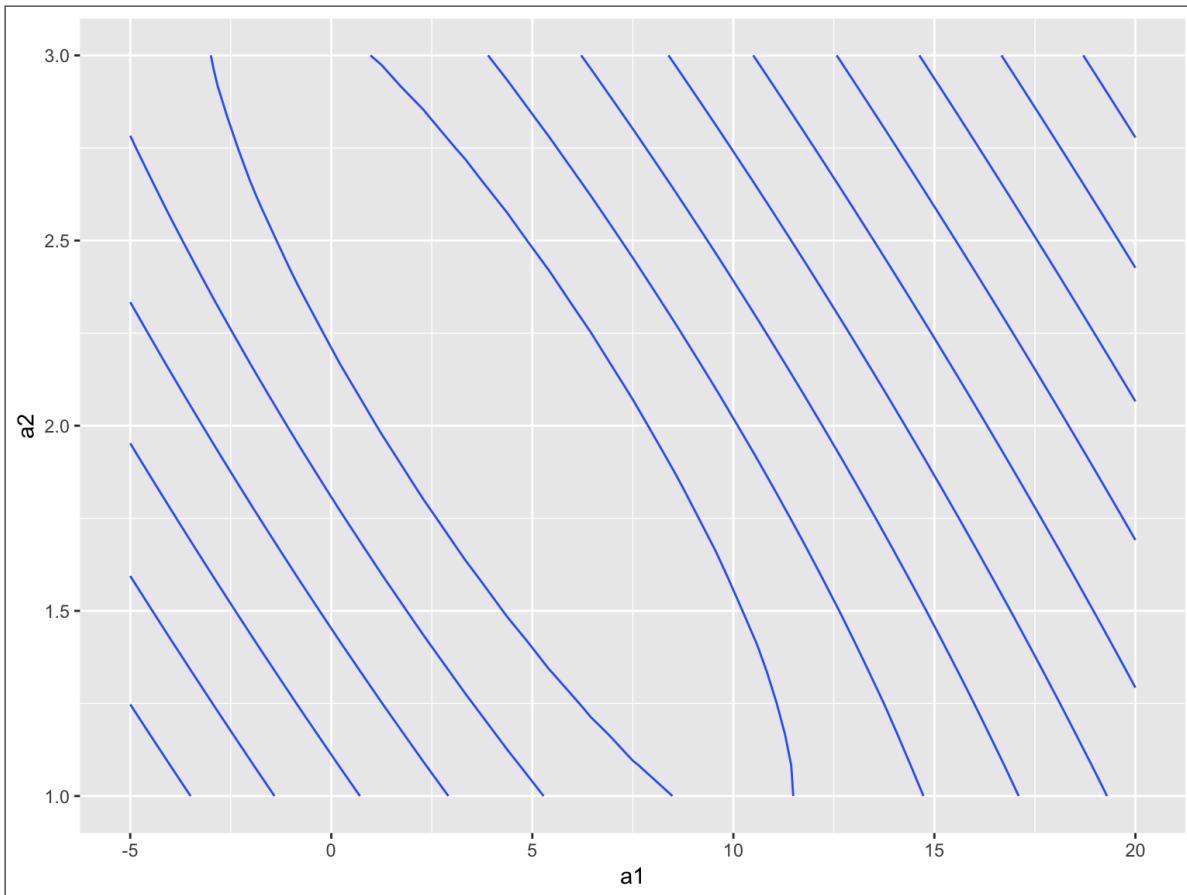
# Plot the new best 10 models

```
grid %>%
  ggplot(aes(a1, a2)) +
  geom_point(data = filter(grid, rank(dist) <= 10), size = 4,
  colour = "red") +
  geom_point(aes(colour = -dist))
```



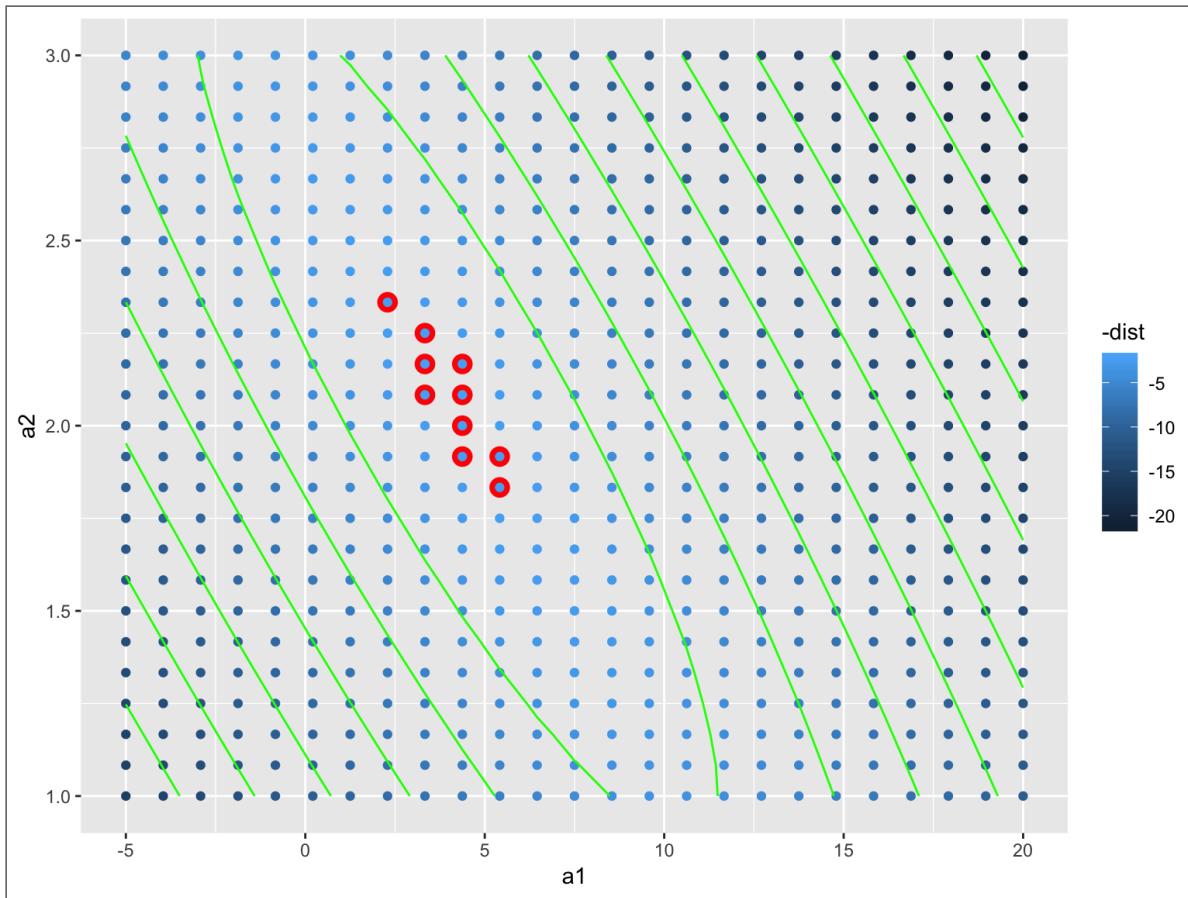
# Contour Map

```
grid %>%
  ggplot(aes(a1, a2, z=-dist)) +
  geom_contour()
```



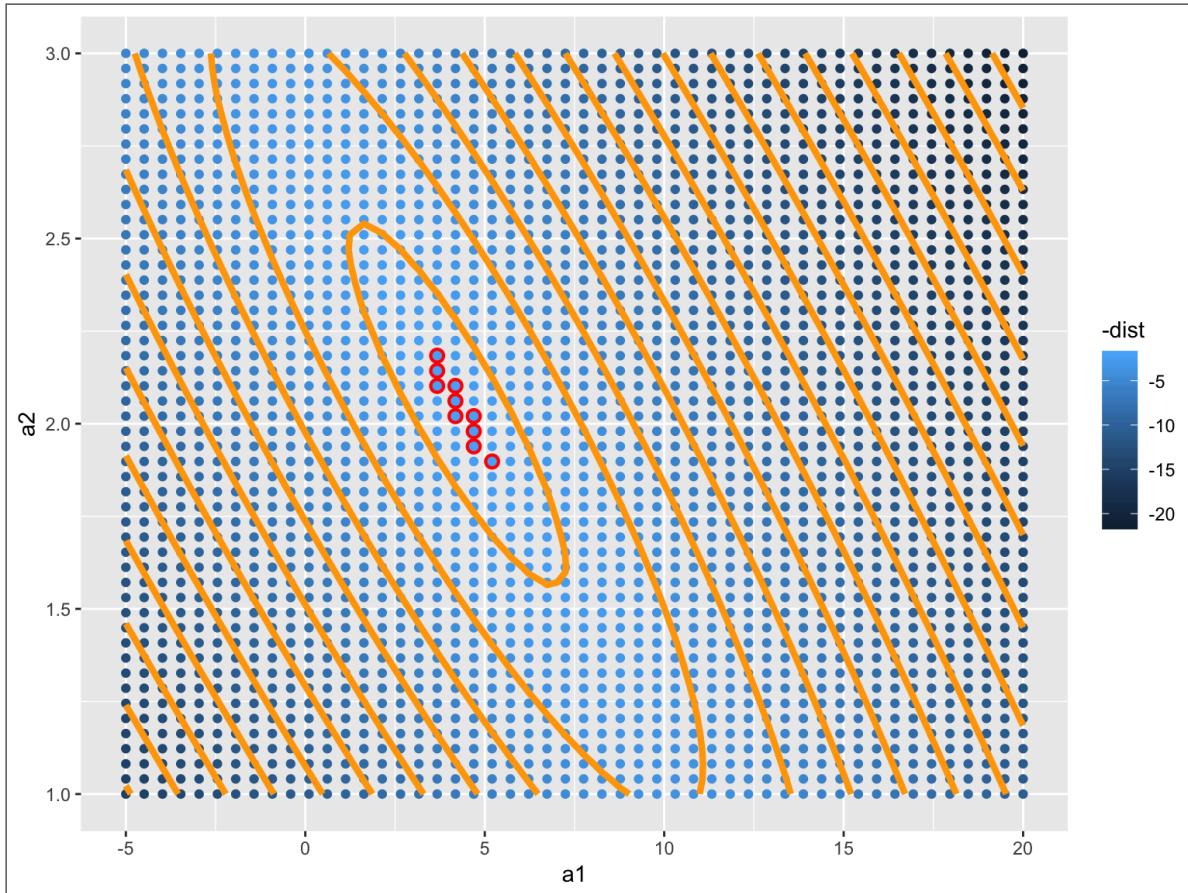
# Contour Map (2)

```
grid %>%
  ggplot(aes(a1, a2)) +
  geom_point(data = filter(grid, rank(dist) <= 10), size = 4,
             colour = "red") +
  geom_point(aes(colour = -dist)) +
  geom_contour(aes(z = -dist), colour = "green")
```



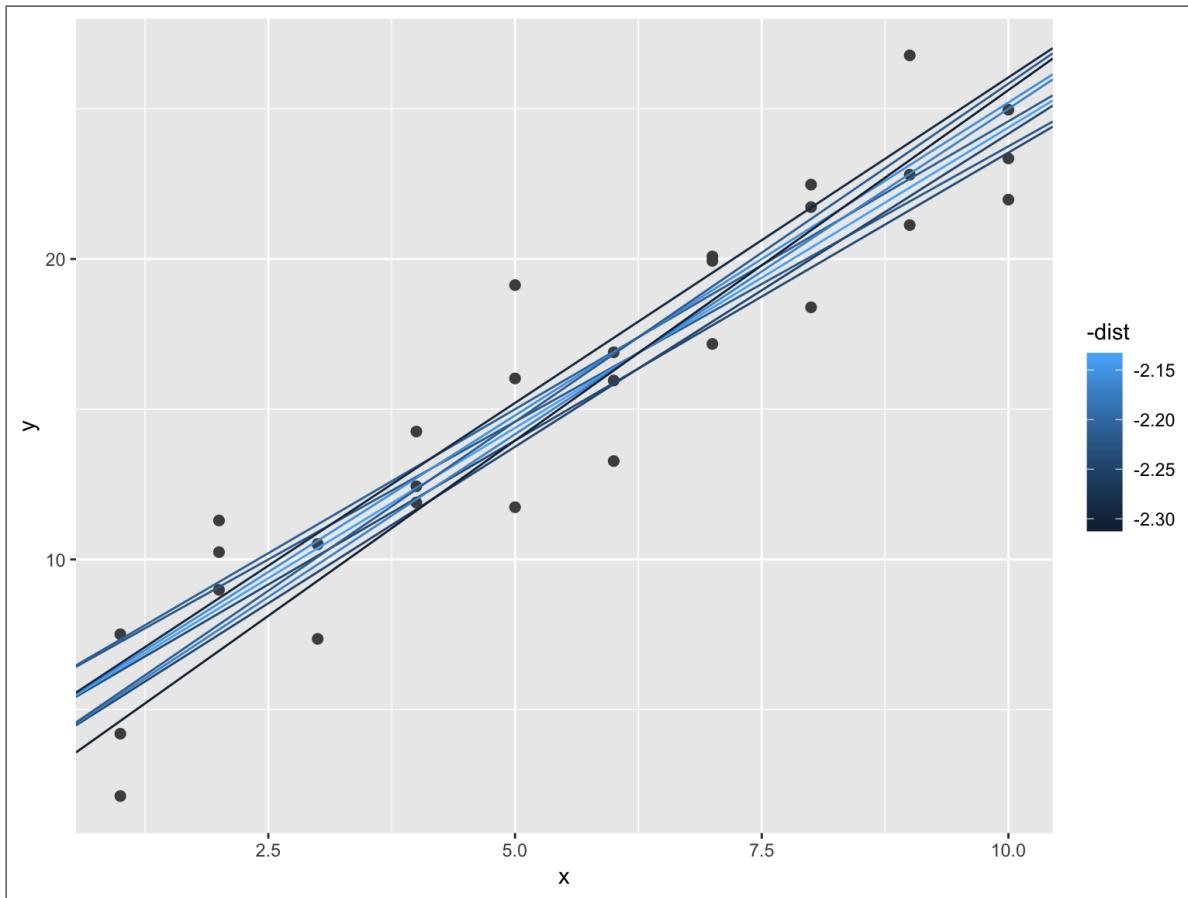
# A finer contour map

```
fine_grid %>%
  ggplot(aes(a1, a2)) +
  geom_point(data = filter(fine_grid, rank(dist) <= 10), size =
  3, colour = "red") +
  geom_point(aes(colour = -dist)) +
  geom_contour(aes(z = -dist), colour = "orange", bins = 15,
  size = 1.5)
```



# Overlay those 10 best models on the data.

```
ggplot(sim1, aes(x, y)) +  
  geom_point(size = 2, colour = "grey30") +  
  geom_abline(  
    aes(intercept = a1, slope = a2, colour = -dist),  
    data = filter(grid, rank(dist) <= 10)  
  )
```



# Optimize the models

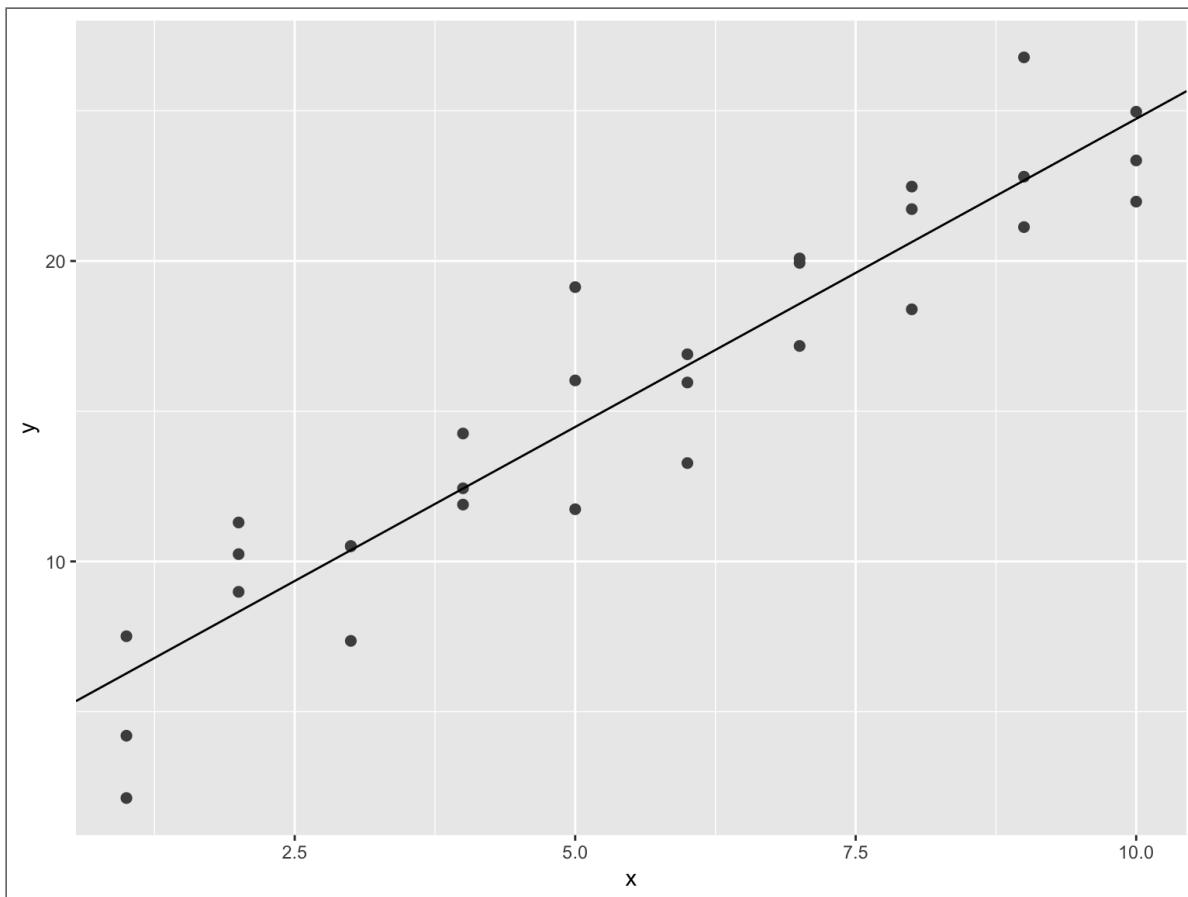
There is a tool to optimize the parameters so you get the smallest distance.

```
best <- optim(c(0, 0), measure_distance, data = sim1)
best$par
```

```
## [1] 4.222248 2.051204
```

# Plot the optimal model

```
ggplot(sim1, aes(x, y)) +  
  geom_point(size = 2, colour = "grey30") +  
  geom_abline(intercept = best$par[1], slope = best$par[2])
```



## R can do all of this in 1 command

Of course, R has a tool to do this whole process automatically. This is our standard OLS regression.

```
sim1_mod <- lm(y ~ x, data = sim1)
coef(sim1_mod)
```

```
## (Intercept)          x
##   4.220822    2.051533
```

# Generate Predictions

We can use a model object like `sim1_mod` to generate predictions. We can start with an empty grid.

```
grid <- sim1 %>%
  data_grid(x)
grid
```

```
## # A tibble: 10 x 1
##       x
##   <int>
## 1     1
## 2     2
## 3     3
## 4     4
## 5     5
## 6     6
## 7     7
## 8     8
## 9     9
## 10    10
```

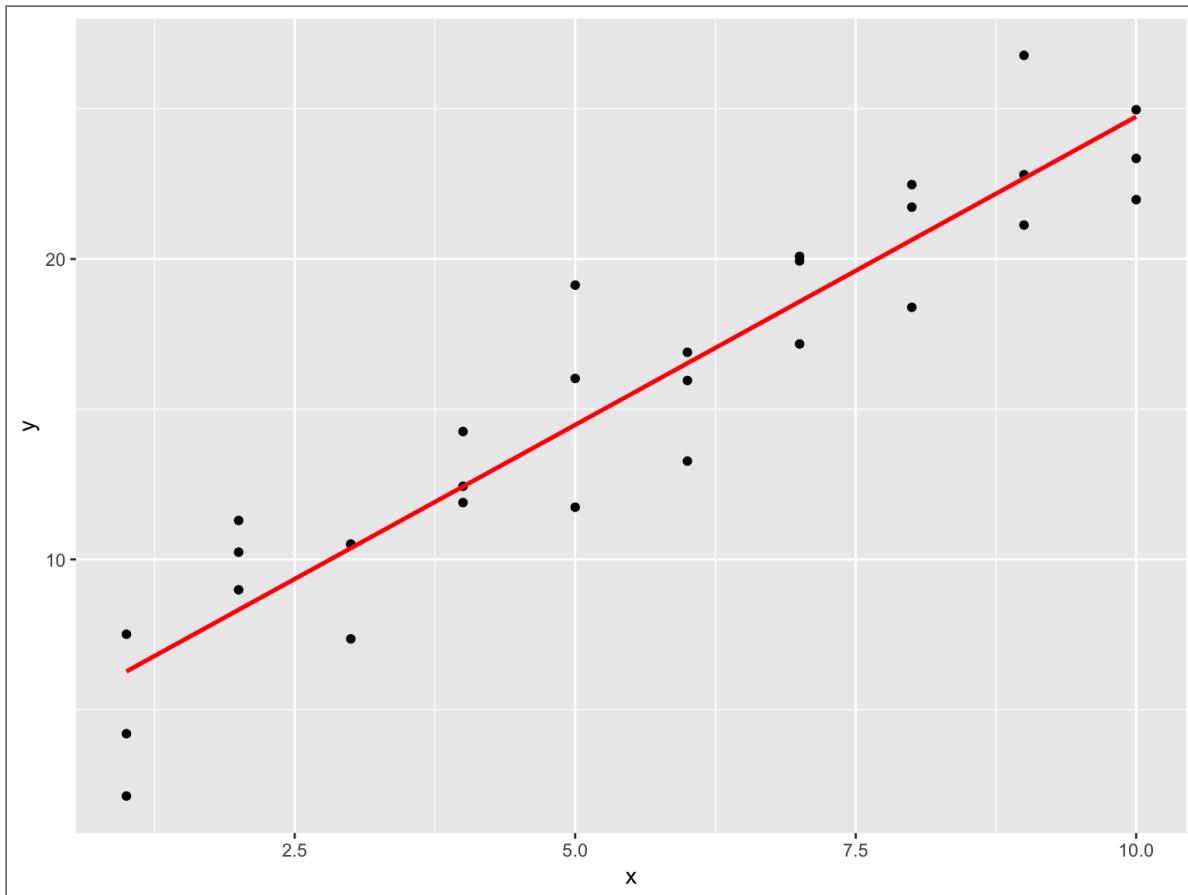
# Generate Predictions (2)

```
grid <- grid %>%
  add_predictions(sim1_mod)
grid
```

```
## # A tibble: 10 x 2
##       x     pred
##   <int>  <dbl>
## 1     1    6.27
## 2     2    8.32
## 3     3   10.4
## 4     4   12.4
## 5     5   14.5
## 6     6   16.5
## 7     7   18.6
## 8     8   20.6
## 9     9   22.7
## 10    10  24.7
```

# Plot those predictions using `geom_line`.

```
ggplot(sim1, aes(x)) +  
  geom_point(aes(y = y)) +  
  geom_line(aes(y = pred), data = grid, colour = "red", size =  
  1)
```



# Add residuals

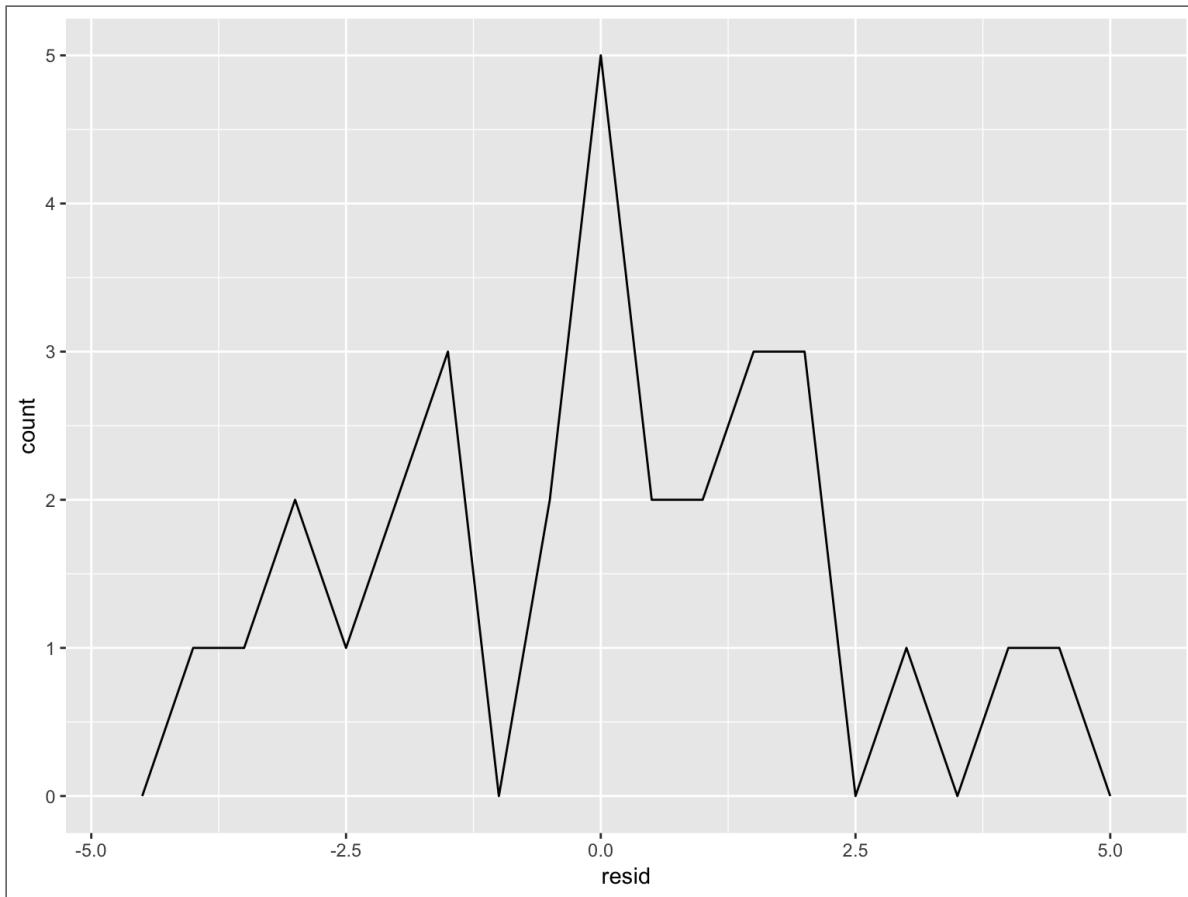
Add residuals. We need to use the original dataset.

```
sim1 <- sim1 %>%
  add_residuals(sim1_mod)
sim1
```

```
## # A tibble: 30 x 3
##       x     y   resid
##   <int> <dbl>   <dbl>
## 1     1  4.20 -2.07
## 2     1  7.51  1.24
## 3     1  2.13 -4.15
## 4     2  8.99  0.665
## 5     2 10.2   1.92
## 6     2 11.3   2.97
## 7     3  7.36 -3.02
## 8     3 10.5   0.130
## 9     3 10.5   0.136
## 10    4 12.4   0.00763
## # ... with 20 more rows
```

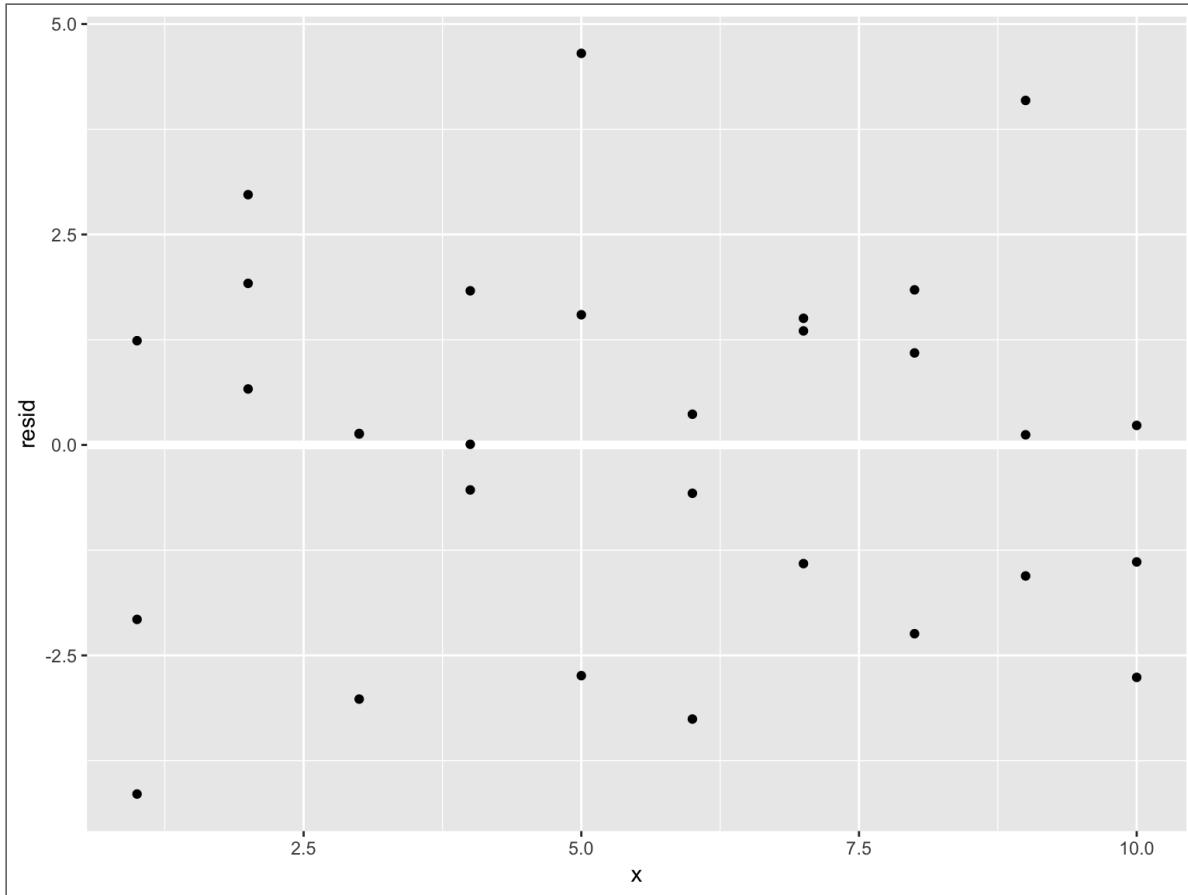
# Plot Residuals

```
ggplot(sim1, aes(resid)) +  
  geom_freqpoly(binwidth = 0.5)
```



# Plot Residuals

```
ggplot(sim1, aes(x, resid)) +  
  geom_ref_line(h = 0) +  
  geom_point()
```



## *t*-tests

```
t.test(disp ~ vs, data = mtcars)
```

```
##  
## Welch Two Sample t-test  
##  
## data: disp by vs  
## t = 5.9416, df = 26.977, p-value = 2.477e-06  
## alternative hypothesis: true difference in means is not equal to  
## 95 percent confidence interval:  
## 114.3628 235.0229  
## sample estimates:  
## mean in group 0 mean in group 1  
## 307.1500 132.4571
```



# Correlation

```
cor.test(mtcars$disp, mtcars$vs)
```

```
##  
## Pearson's product-moment correlation  
##  
## data: mtcars$disp and mtcars$vs  
## t = -5.5289, df = 30, p-value = 5.235e-06  
## alternative hypothesis: true correlation is not equal to 0  
## 95 percent confidence interval:  
## -0.8488377 -0.4808327  
## sample estimates:  
## cor  
## -0.7104159
```

# Chi Squared (1)

## Smoking and Exercise

```
library(MASS)
```

```
##  
## Attaching package: 'MASS'
```

```
## The following object is masked from 'package:dplyr':  
##  
##     select
```

```
tbl <- table(survey$Smoke, survey$Exer)  
tbl
```

```
##  
##          Freq  None Some  
##  Heavy      7     1    3  
##  Never     87    18   84  
##  Occas     12     3    4  
##  Regul      9     1    7
```

# Chi Squared (2)

```
chisq.test(tbl)
```

```
## Warning in chisq.test(tbl): Chi-squared approximation may be inc
```

```
##  
## Pearson's Chi-squared test  
##  
## data: tbl  
## X-squared = 5.4885, df = 6, p-value = 0.4828
```

# Capstone Exercise

# Thy these

Things to try:

1. Find some outliers and why they happened (try googling specific dates).
2. Estimate the effect that the seasons have on bike share usage? Day of the week?
3. Build a model that predicts bike share usage for a given day. Separate your data into three parts.

# Notes

TODO: Add more content here.

- The R formula language – using concepts from how the Tidyverse handles fields.
  - `data = .`
  - describe how to go from equations to R formulas - the `~` is the `=` we would write in an equation.  $y \sim x$  translates to  
 $y = a_1 + a_2 * x$
  - `model_matrix` function
  - categorical variables – does that mean we have to deal with factors?
- statistical testings `t.test` - use the formula language – come up with examples.
- modeling using `lm` - also use the formula language

# Additional Resources

# Online Resources

Start here:

- <http://r4ds.had.co.nz/>
- <http://stat545.com/>

Additional:

- <http://adv-r.had.co.nz/>
- <http://r-pkgs.had.co.nz/>
- <http://swcarpentry.github.io/r-novice-inflammation/>
- <https://rstudio-education.github.io/hopr/preface.html>

For help/community:

- <http://stackoverflow.com/questions/tagged/r>
- <https://twitter.com/search?q=%23rstats>
- <https://www.r-project.org/mail.html> (Though they're not terribly welcoming to newbies sometimes...)

## Print Books

- <https://us.sagepub.com/en-us/nam/discovering-statistics-using-r/book236067%20>
- <https://us.sagepub.com/en-us/nam/an-r-companion-to-applied-regression/book23389>
- <http://shop.oreilly.com/product/0636920028>

# References

- Wickham, Hadley (2014).  
**<https://www.jstatsoft.org/article/view/v059i01>**