Chapter 0: `ggplot2` and `tidyverse`

We will be using the `ggplot2` package for making graphics in this class.

The first time on your machine you'll need to install the package:

```
install.packages("ggplot2")
```

Whenever you first want to plot during an `R` session, we need to load the library.

```
library(ggplot2)
```

0.1 Why visualize?

The sole purpose of visualization is communication. Visualization offers an alternative way of communicating numbers than simply using tables. Often, we can get more information out of our numbers graphically than with numerical summaries alone. Through the use of exploratory data analysis, we can see what the data can tell us beyond the formal modeling or hypothesis testing task.

For example, let's look at the following dataset.

```
anscombe
```

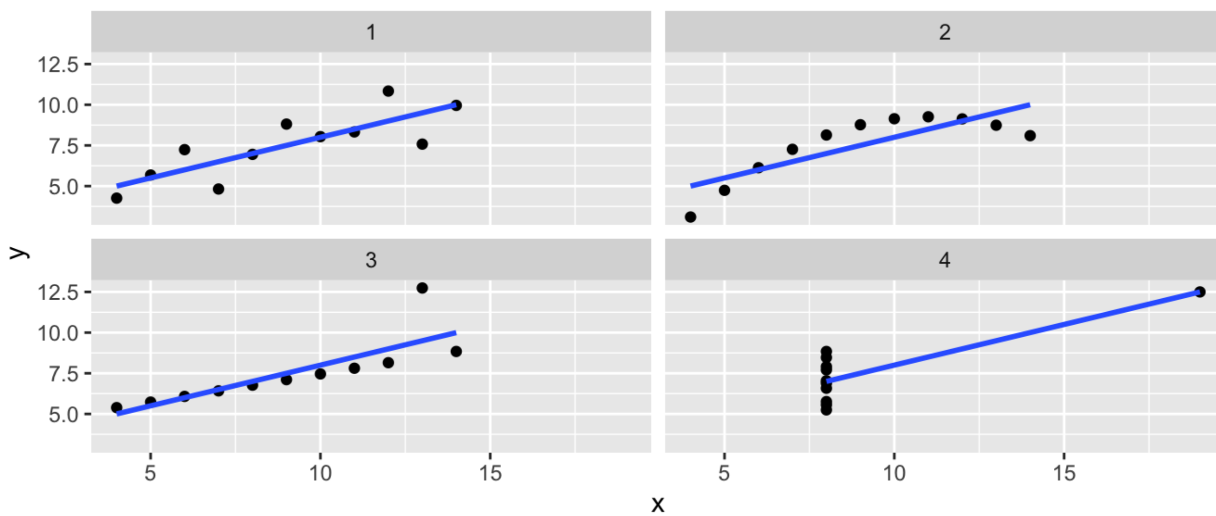
```
##    x1 x2 x3 x4    y1   y2    y3    y4
## 1  10 10 10  8  8.04 9.14  7.46  6.58
## 2   8  8  8  8  6.95 8.14  6.77  5.76
## 3  13 13 13  8  7.58 8.74 12.74  7.71
## 4   9  9  9  8  8.81 8.77  7.11  8.84
## 5  11 11 11  8  8.33 9.26  7.81  8.47
## 6  14 14 14  8  9.96 8.10  8.84  7.04
## 7   6  6  6  8  7.24 6.13  6.08  5.25
## 8   4  4  4 19  4.26 3.10  5.39 12.50
## 9  12 12 12  8 10.84 9.13  8.15  5.56
## 10  7  7  7  8  4.82 7.26  6.42  7.91
## 11  5  5  5  8  5.68 4.74  5.73  6.89
```

Anscombe's Quartet is comprised of 4 datasets that have nearly identical simple statistical properties. Each dataset contains 11 (x, y) points with the same mean, median, standard deviation, and correlation coefficient between x and y.

dataset	mean_x	sd_x	mean_y	sd_y	cor
1	9	3.316625	7.500909	2.031568	0.8164205
2	9	3.316625	7.500909	2.031657	0.8162365
3	9	3.316625	7.500000	2.030424	0.8162867
4	9	3.316625	7.500909	2.030578	0.8165214

But this doesn't tell the whole story. Let's look closer at these datasets.

```
## `geom_smooth()` using formula 'y ~ x'
```



Visualizations can aid communication and make the data easier to perceive. It can also show us things about our data that numerical summaries won't necessarily capture.

0.2 A Grammar of Graphics

The grammar of graphics was developed by Leland Wilkinson ([https://www.springer.-com/gp/book/9780387245447](https://www.springer.com/gp/book/9780387245447)). It is a set of grammatical rules for creating perceivable graphs. Rather than thinking about a limited set of graphs, we can think about graphical forms. This abstraction makes thinking, creating, and communicating graphics easier.

Statistical graphic specifications are expressed using the following components.

1. **data**: a set of data operations that create variables from datasets
2. **trans**: variable transformations
3. **scale**: scale transformations
4. **coord**: a coordinate system
5. **element**: graphs (points) and their aesthetic attributes (color)
6. **guide**: one or more guides (axes, legends, etc.)

`ggplot2` is a package written by Hadley Wickham (https://vita.had.co.nz/papers/layered-grammar.html) that implements the ideas in the grammar of graphics to create layered plots.

`ggplot2` uses the idea that you can build every graph with graphical components from three sources

1. the data, represented by `geoms`
2. the scales and coordinate system
3. the plot annotations

This works by mapping values in the data to visual properties of the geom (aesthetics) like size, color, and locations.

Let's build a graphic. We start with the data. We will use the `diamonds` dataset, and we want to explore the relationship between carat and price.

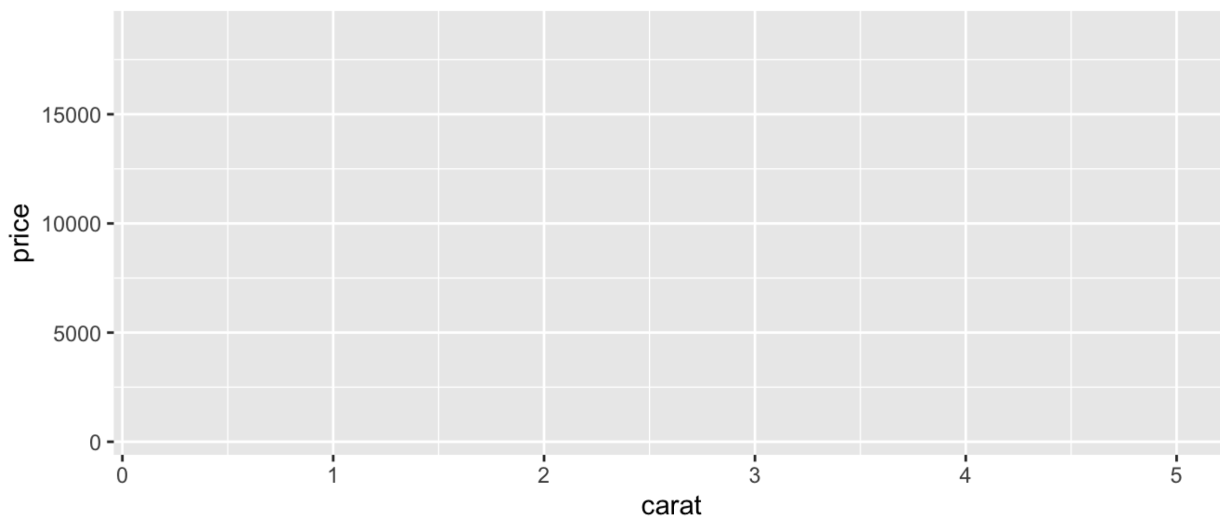
```
head(diamonds)
```

```
## # A tibble: 6 x 10
##   carat cut       color clarity depth table price     x     y     z
##   <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1 0.23  Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
## 2 0.21  Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
## 3 0.23  Good      E     VS1      56.9    65   327  4.05  4.07  2.31
## 4 0.290 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
## 5 0.31  Good      J     SI2      63.3    58   335  4.34  4.35  2.75
## 6 0.24  Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
```

```
ggplot(data = diamonds)
```

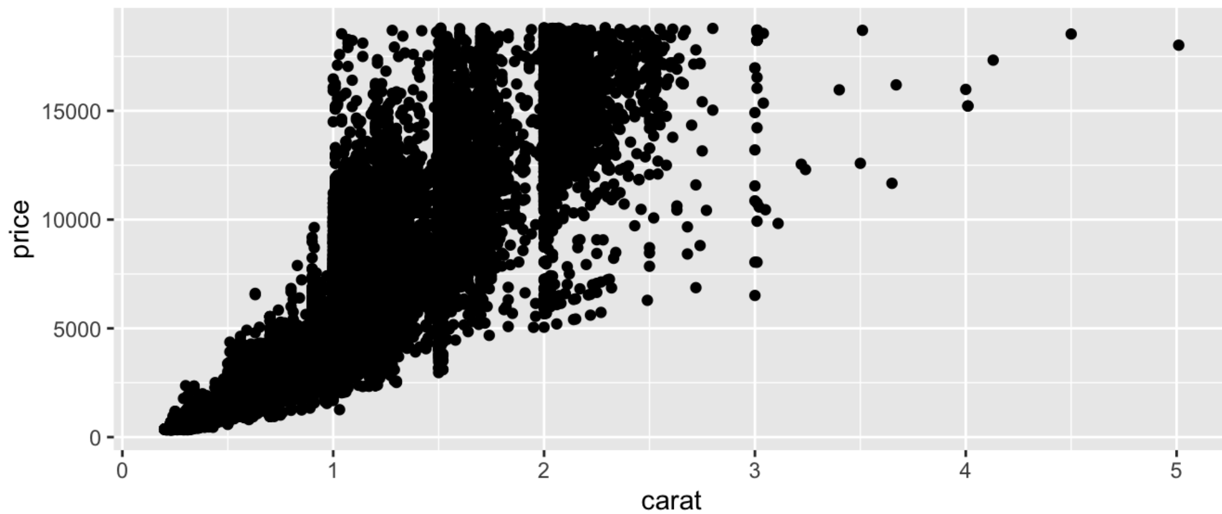
Next we need to specify the aesthetic (variable) mappings.

```r
ggplot(data = diamonds, mapping = aes(carat, price))
```



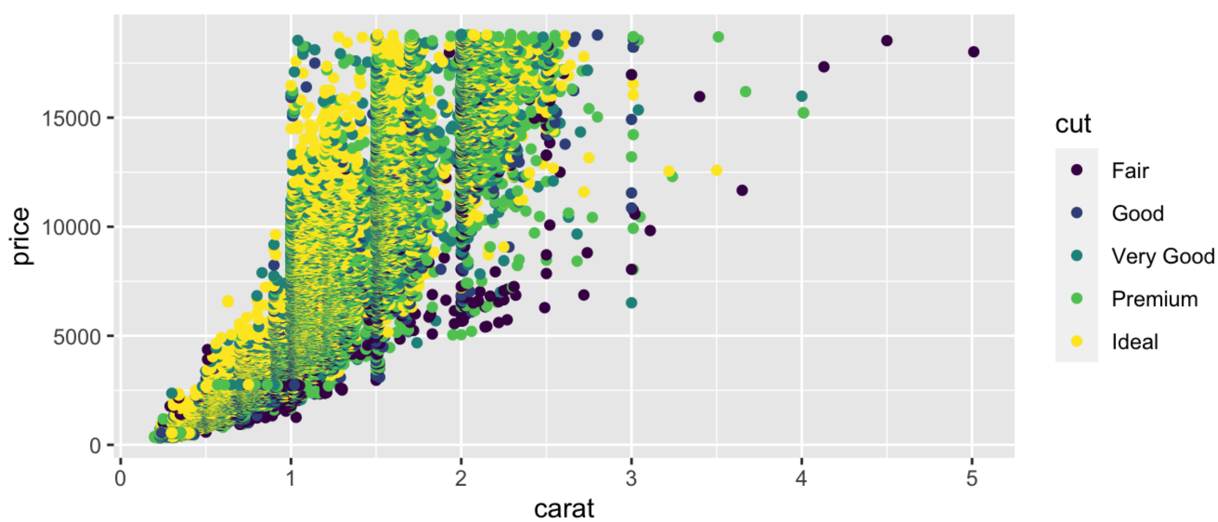
Now we choose a geom to display our data.

```r
ggplot(data = diamonds, mapping = aes(carat, price)) +
  geom_point()
```



And add an aesthetic to our plot.

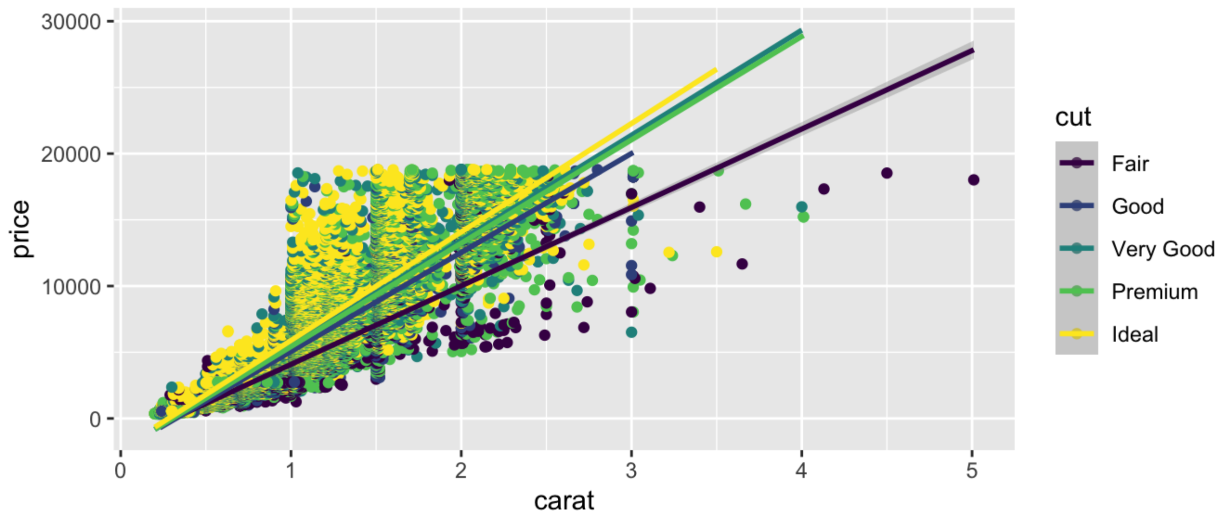
```
ggplot(data = diamonds, mapping = aes(carat, price)) +
  geom_point(aes(color = cut))
```



We could add another layer.

```
ggplot(data = diamonds, mapping = aes(carat, price)) +
  geom_point(aes(color = cut)) +
  geom_smooth(aes(color = cut), method = "lm")
```

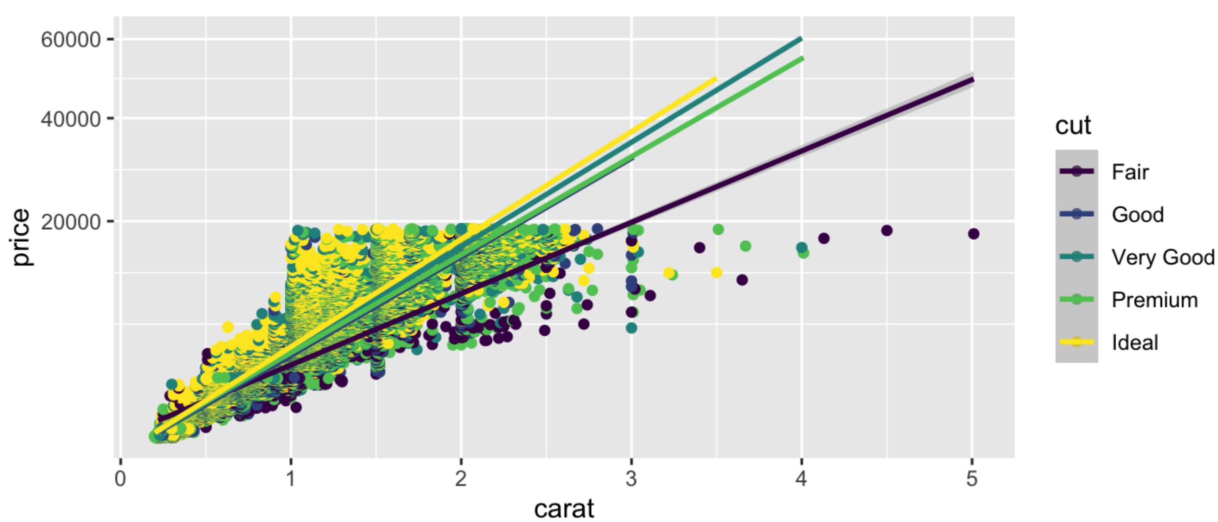
```
## `geom_smooth()` using formula 'y ~ x'
```



And finally, we can specify coordinate transformations.

```r
ggplot(data = diamonds, mapping = aes(carat, price)) +
  geom_point(aes(color = cut)) +
  geom_smooth(aes(color = cut), method = "lm") +
  scale_y_sqrt()
```

```
## `geom_smooth()` using formula 'y ~ x'
```



Notice we can add on to our plot in a layered fashion.

0.3 Graphical Summaries

There are some basic charts we will use in this class that cover a wide range of cases. For univariate data, we can use dotplots, histograms, and barcharts. For two dimensional data, we can look at scatterplots and boxplots.

0.3.1 Scatterplots

Scatterplots are used for investigating relationships between two numeric variables. To demonstrate some of the flexibility of scatterplots in `ggplot2`, let's answer the following question.

> Do cars with big engines use more fuel than cars with small engines?

We will use the `mpg` dataset in the `ggplot2` package to answer the question. This dataset contains observations collected by the US Environmental Protection Agency on 38 models of car.

```
dim(mpg)
```

```
## [1] 234  11
```

```
summary(mpg)
```

```
##  manufacturer          model               displ            year     
##  Length:234         Length:234         Min.   :1.600   Min.   :1999  
##  Class :character   Class :character   1st Qu.:2.400   1st Qu.:1999  
##  Mode  :character   Mode  :character   Median :3.300   Median :2004  
##                                        Mean   :3.472   Mean   :2004  
##                                        3rd Qu.:4.600   3rd Qu.:2008  
##                                        Max.   :7.000   Max.   :2008  
##       cyl           trans               drv                 cty       
##  Min.   :4.000   Length:234         Length:234         Min.   : 9.00-
```

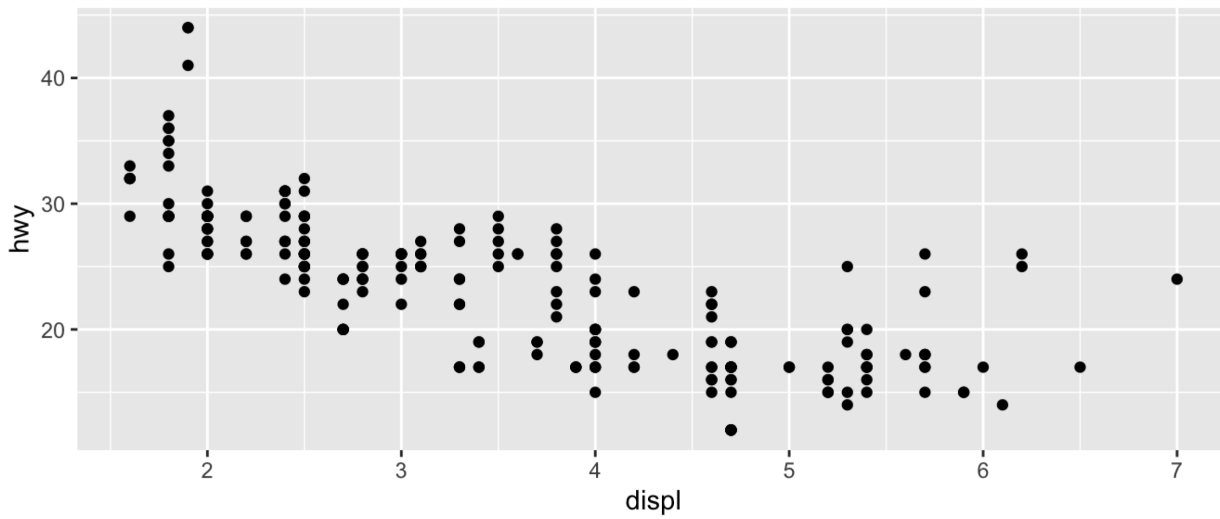
```
##  1st Qu.:4.000   Class :character   Class :character   1st Qu.:14.00  
##  Median :6.000   Mode  :character   Mode  :character   Median :17.00  
##  Mean   :5.889                                         Mean   :16.86  
##  3rd Qu.:8.000                                         3rd Qu.:19.00  
##  Max.   :8.000                                         Max.   :35.00  
##       hwy             fl               class          
##  Min.   :12.00   Length:234         Length:234        
##  1st Qu.:18.00   Class :character   Class :character  
##  Median :24.00   Mode  :character   Mode  :character  
##  Mean   :23.44                                        
##  3rd Qu.:27.00                                        
##  Max.   :44.00
```

```r
head(mpg)
```

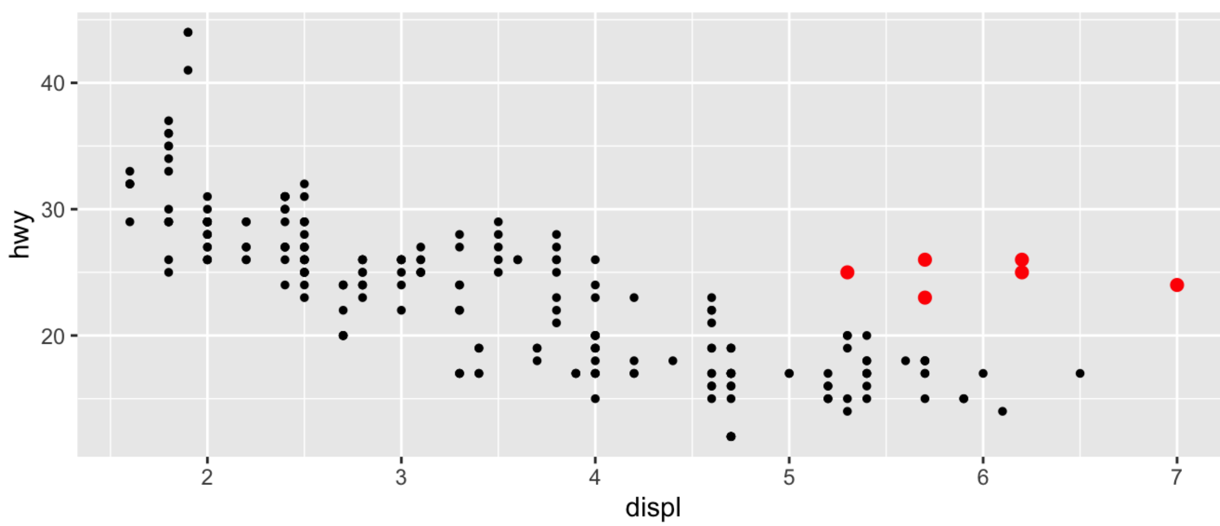
```
## # A tibble: 6 x 11
##   manufacturer model displ  year   cyl trans      drv     cty   hwy fl    class
##   <chr>        <chr> <dbl> <int> <int> <chr>      <chr> <int> <int> <chr> <chr>
## 1 audi         a4      1.8  1999     4 auto(l5)   f        18    29 p     compa
## 2 audi         a4      1.8  1999     4 manual(m5) f        21    29 p     compa
## 3 audi         a4      2    2008     4 manual(m6) f        20    31 p     compa
## 4 audi         a4      2    2008     4 auto(av)   f        21    30 p     compa
## 5 audi         a4      2.8  1999     6 auto(l5)   f        16    26 p     compa
## 6 audi         a4      2.8  1999     6 manual(m5) f        18    26 p     compa
```

`mpg` contains the following variables: `displ`, a car's engine size, in liters, and `hwy`, a car's fuel efficiency on the highway, in miles per gallon (mpg).

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

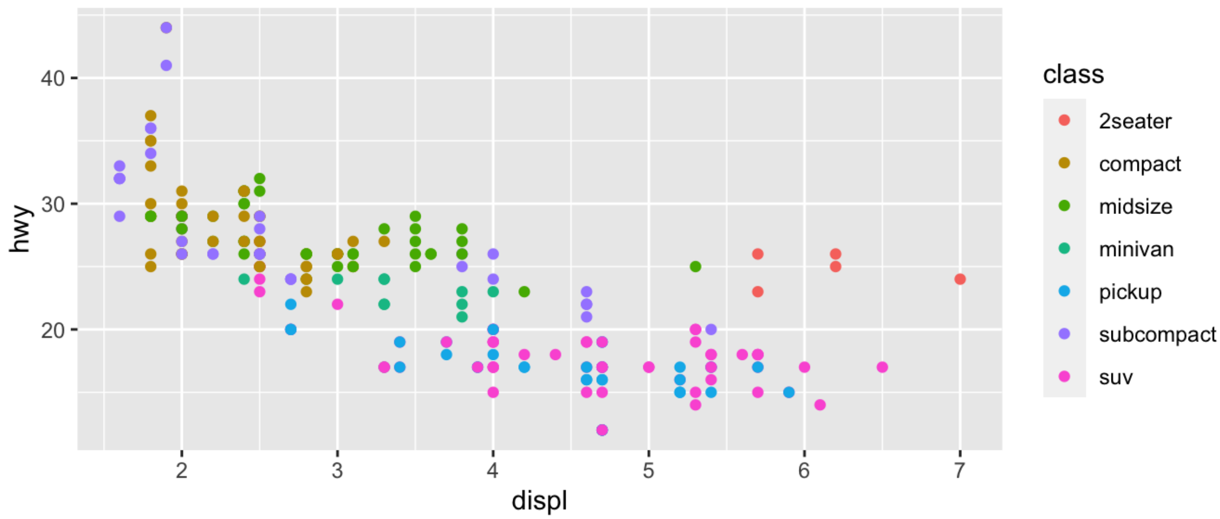


So we can say, yes, cars with larger engines have worse fuel efficiency. But there is more going on here.



The red points above seem to have higher `mpg` than they should based on engine size alone (outliers). Maybe there is a confounding variable we've missed. The `class` variable of the mpg dataset classifies cars into groups such as compact, midsize, and SUV.

```
ggplot(data = mpg) +
  geom_point(mapping = aes(displ, hwy, colour = class))
```

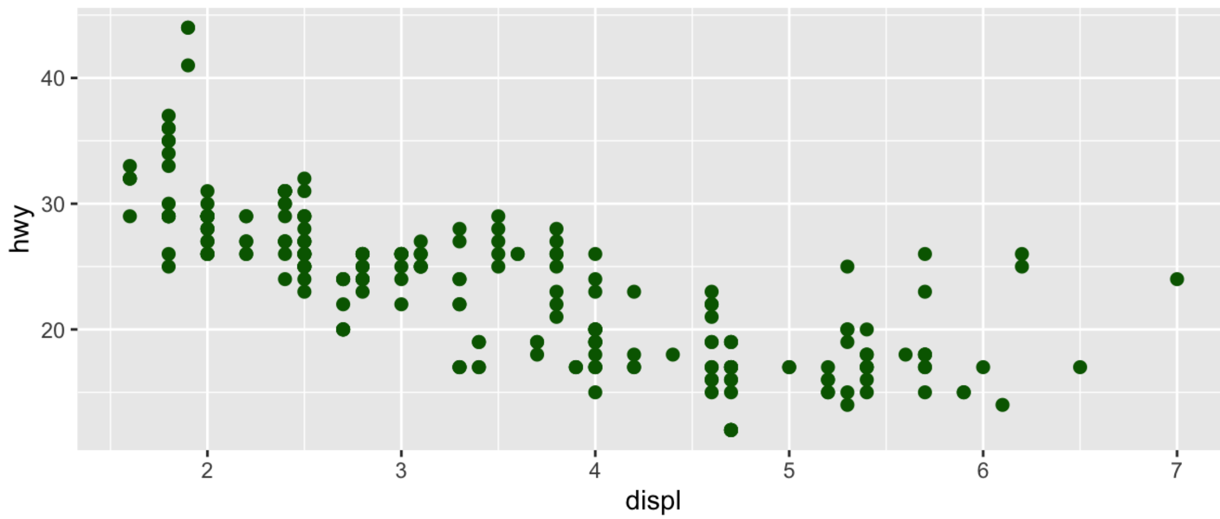


The colors show that many of the unusual points are two-seater cars, probably sports cars! Sports cars have large engines like SUVs and pickup trucks, but small bodies like midsize and compact cars, which improves their gas mileage.

Instead of color, we could also map a categorical variable (like `class`) to shape, size, and transparency (`alpha`).

So far we have mapped aesthetics to variables in our dataset. What happens if we just want to generally change the aesthetics of our plots, without tying that to data? We can specify general aesthetics as parameters of the `geom`, instead of specifying them as aesthetics (`aes`).

```
ggplot(data = mpg) +
  geom_point(mapping = aes(displ, hwy), colour = "darkgreen", size =
  2)
```

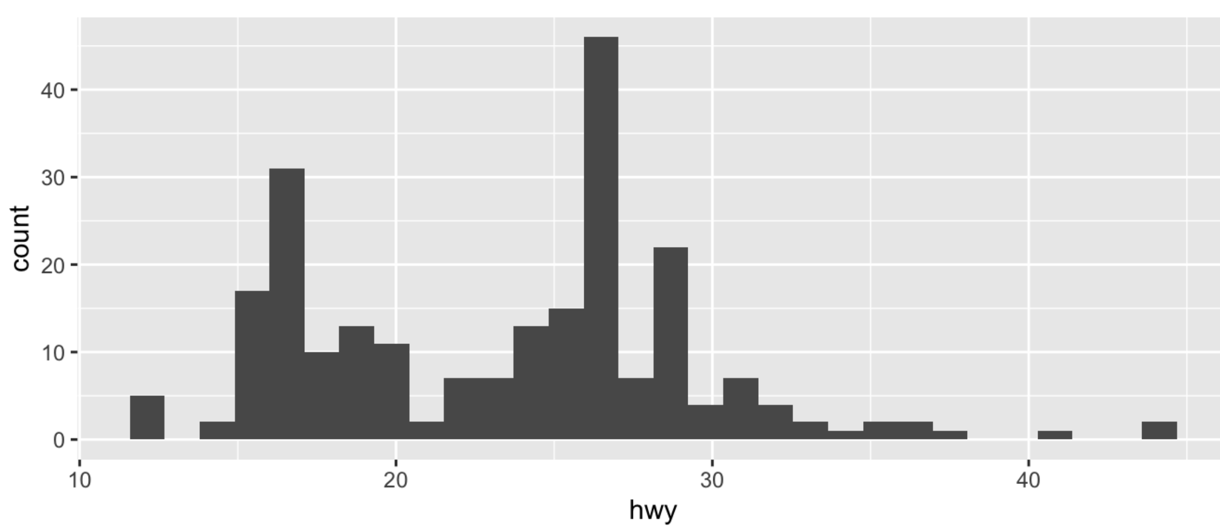


When interpreting a scatterplot we can look for big patterns in our data, as well as form, direction, and strength of relationships. Additionally, we can see small patterns and deviations from those patterns (outliers).

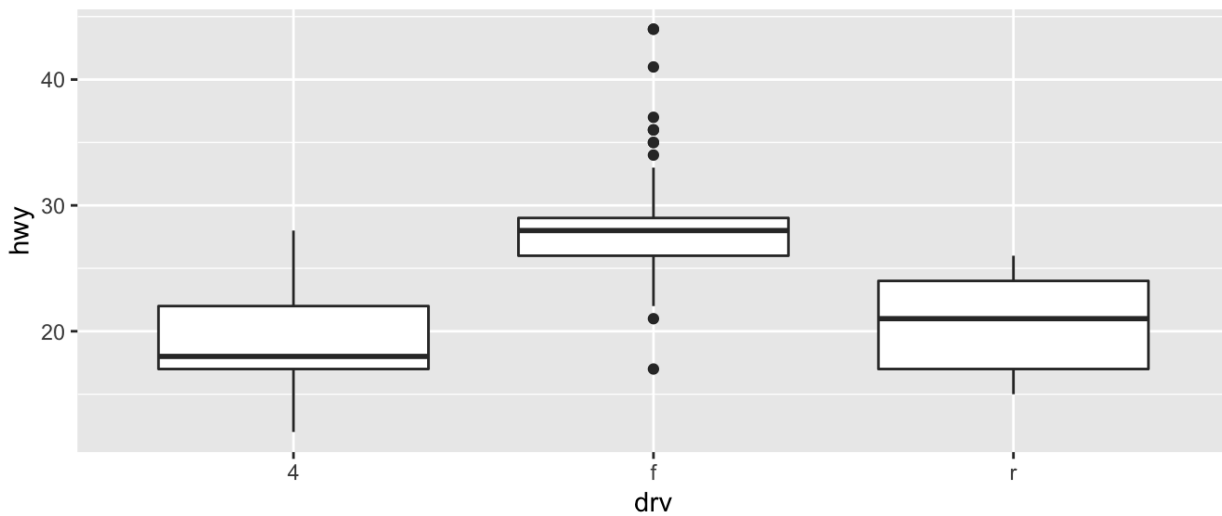
0.3.2 Histograms, Barcharts, and Boxplots

We can look at the distribution of continuous variables using **histograms** and **boxplots** and the distribution of discrete variables using **barcharts**.

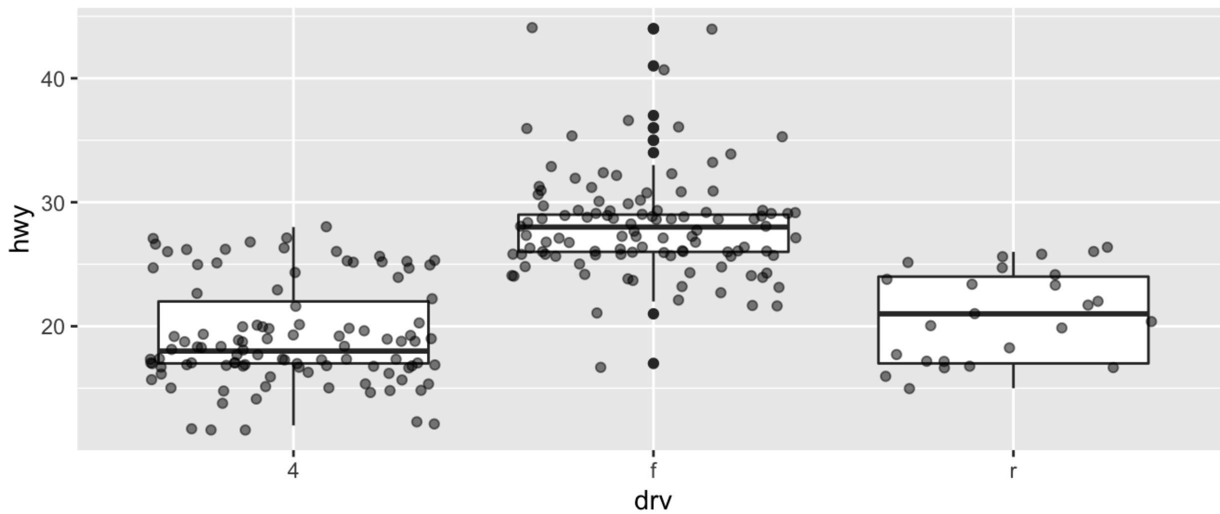
```
ggplot(data = mpg) +
  geom_histogram(mapping = aes(hwy), bins = 30)
```



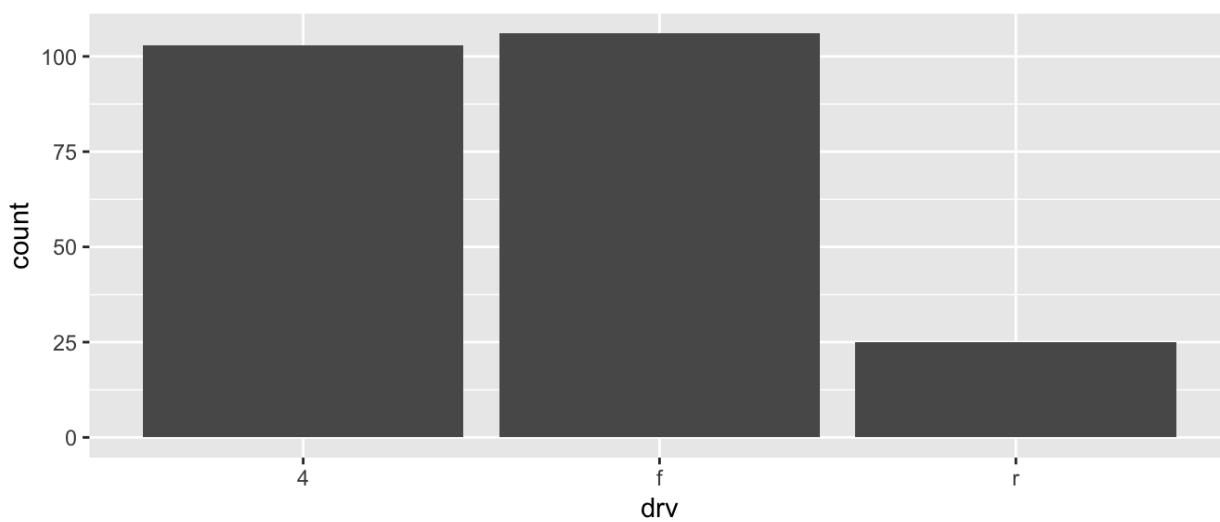
```r
## histograms will look very different sometimes with different
  binwidths

ggplot(data = mpg) +
  geom_boxplot(mapping = aes(drv, hwy))
```




```r
## boxplots allow us to see the distribution of a cts rv conditional
  on a discrete one
## we can also show the actual data at the same time
ggplot(data = mpg) +
  geom_boxplot(mapping = aes(drv, hwy)) +
  geom_jitter(mapping = aes(drv, hwy), alpha = .5)
```




```
ggplot(data = mpg) +
  geom_bar(mapping = aes(drv))
```




```
## shows us the distribution of a categorical variable
```

0.3.3 Facets

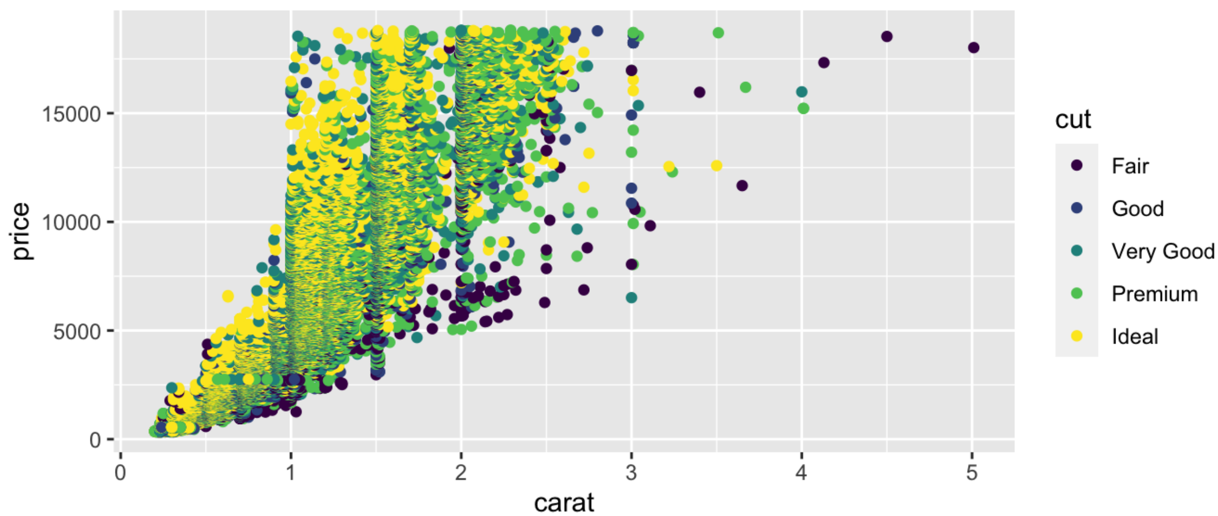
So far we've looked at

1. how one (or more) variables are distributed - barchart or histogram
2. how two variables are related - scatterplot, boxplot

3. how two variables are related, conditioned on other variables - color

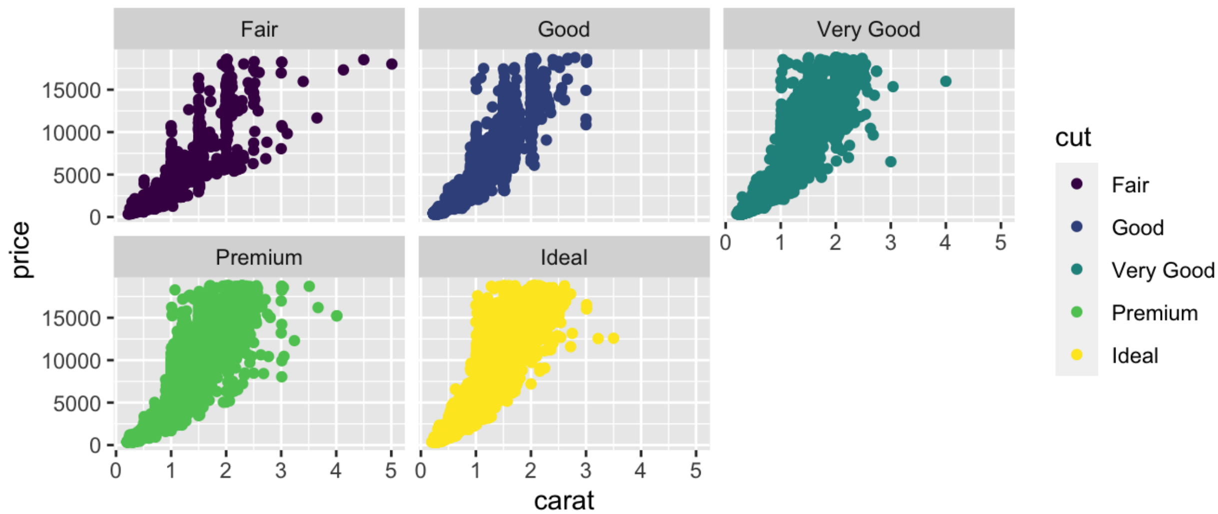
Sometimes color isn't enough to show conditioning because of crowded plots.

```
ggplot(data = diamonds, mapping = aes(carat, price)) +
  geom_point(aes(color = cut))
```



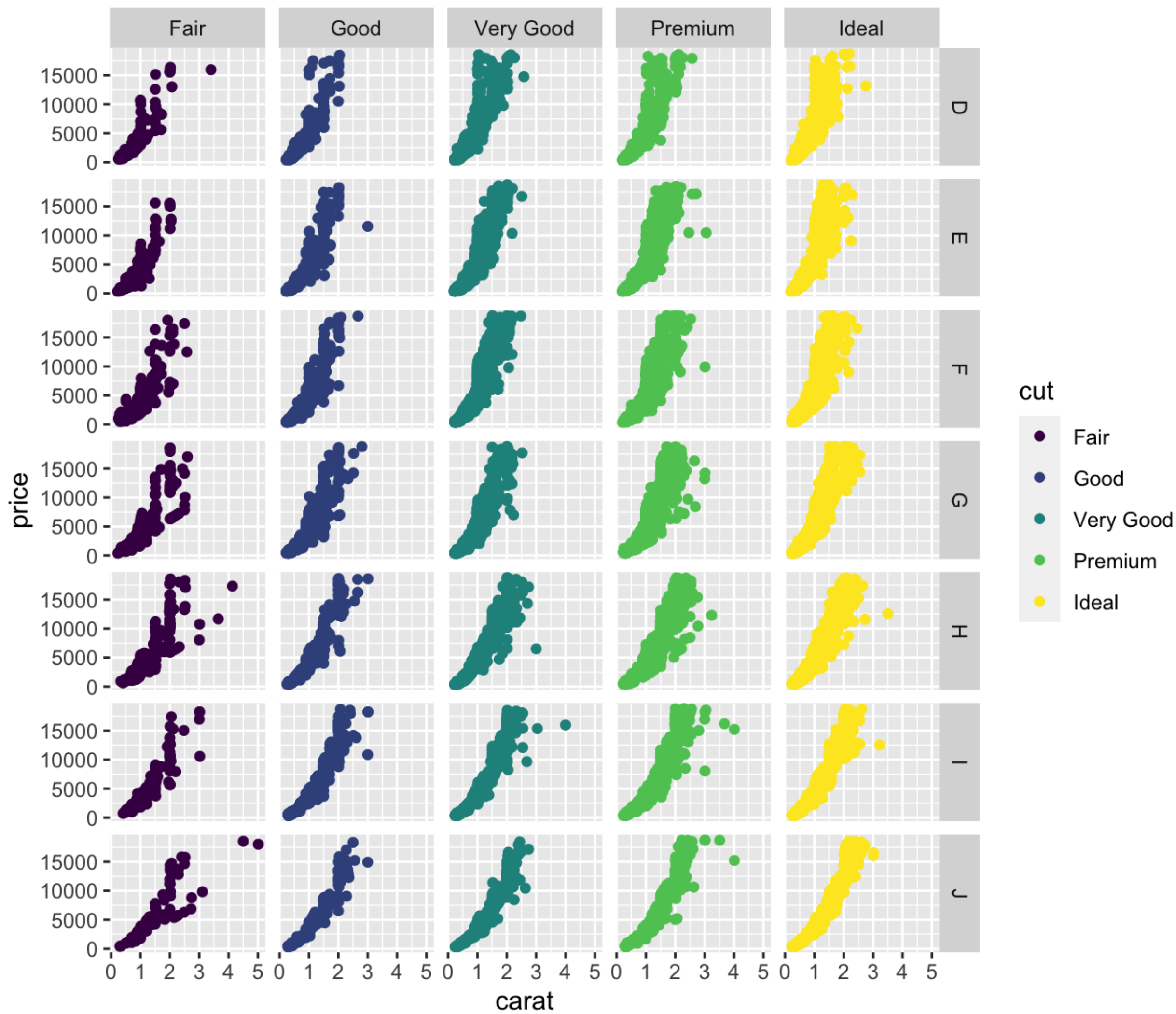
When this is the case, we can *facet* to display plots for different subsets. To do this, we specify row variables ~ column variables (or . for none).

```
ggplot(data = diamonds, mapping = aes(carat, price)) +
  geom_point(aes(color = cut)) +
  facet_wrap(. ~ cut)
```



If instead we have two variables we want to facet by, we can use `facet_grid()`.

```
ggplot(data = diamonds, mapping = aes(carat, price)) +
  geom_point(aes(color = cut)) +
  facet_grid(color ~ cut)
```



0.4 Additional resources

Documentation and cheat sheets (https://ggplot2.tidyverse.org)

Book website (http://had.co.nz/ggplot2/)

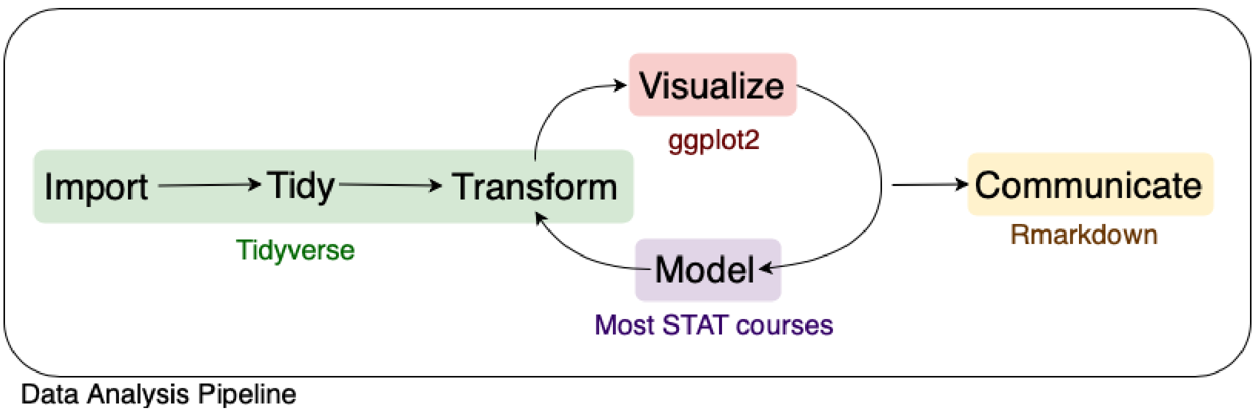
Ch. 3 of R4DS (https://r4ds.had.co.nz/data-visualisation.html)

1 tidyverse

The tidyverse is a suite of packages released by RStudio that work very well together ("verse") to make data analysis run smoothly ("tidy"). It's also a package in `R` that loads all the packages in the tidyverse at once.

```
library(tidyverse)
```

You actually already know one member of the tidyverse – `ggplot2`! We will highlight three more packages in the tidyverse for data analysis.



Adapted from R for Data Science, Wickham & Grolemund (2017)

1.1 `readr`

The first step in (almost) any data analysis task is reading data into `R`. Data can take many formats, but we will focus on text files.

> But what about `.xlsx`??

File extensions `.xls` and `.xlsx` are proprietary Excel formats/ These are binary files (meaning if you open one outside of Excel it will not be human readable). An alternable for rectangular data is a `.csv`.

`.csv` is an extension for *comma separated value* files. They are text files – directly readable – where each column is separated by a comma and each row a new line.

```
Rank,Major_code,Major,Total,Men,Women,Major_category,ShareWomen
1,2419,PETROLEUM ENGINEERING,2339,2057,282,Engineering,0.120564344
2,2416,MINING AND MINERAL ENGINEERING,756,679,77,Engineering,0.101851852
```

`.tsv` is an extension for *tab separated value* files. These are also text files, but the columns are separated by tabs instead of commas. Sometimes these will be `.txt` extension files.

```
Rank    Major_code    Major    Total    Men    Women    Major_category    ShareWom
1    2419    PETROLEUM ENGINEERING    2339    2057    282    Engineering    0.1205
2    2416    MINING AND MINERAL ENGINEERING    756    679    77    Engineering
```

The package `readr` provides a fast and friendly way to ready rectangular text data into `R`.

Here is an example csv file from fivethirtyeight.com on how to choose your college major (https://fivethirtyeight.com/features/the-economic-guide-to-picking-a-college-major/).

```
# load readr
library(readr)

# read a csv
recent_grads <- read_csv(file =
  "https://raw.githubusercontent.com/fivethirtyeight/data/master/college-
  majors/recent-grads.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   Major = col_character(),
##   Major_category = col_character()
## )
```

```
## See spec(...) for full column specifications.
```

`read_csv()` is just one way to read a file using the `readr` package.

- `read_delim()`: the most generic function. Use the delim argument to read a file with any type of delimiter
- `read_tsv()`: read tab separated files

- `read_lines()`: read a file into a vector that has one element per line of the file
- `read_file()`: read a file into a single character element
- `read_table()`: read a file separated by space

1.2 `dplyr`

We almost never will read in data and have it in exactly the right form for visualizing and modeling. Often we need to create variable or summaries.

To facilitate easy transformation of data, we're going to learn how to use the `dplyr` package. `dplyr` uses 6 main verbs, which correspond to some main tasks we may want to perform in an analysis.

We will do this with the `recent_grads` data from fivethiryeight.com we just read into `R` using `readr`.

1.2.1 `%>%`

Before we get into the verbs in `dplyr`, I want to introduce a new paradigm. All of the functions in the tidyverse are structured such that the first argument is a data frame and they also return a data frame. This allows for efficient use of the pipe operator `%>%` (pronounce this as "then").

```
a %>% b()
```

Taked the result on the left and passes it to the first argument on the right. This is equivalent to

```
b(a)
```

This is useful when we want to chain together many operations in an analysis.

1.2.2 `filter()`

`filter()` lets us subset observations based on their values. This is similar to using `[]` to subset a data frame, but simpler.

The first argument is the name of the data frame. The second and subsequent arguments are the expressions that filter the data frame.

Let's subset the `recent_grad` data set to focus on Statistics majors.

```r
recent_grads %>% filter(Major == "STATISTICS AND DECISION SCIENCE")
```

```
## # A tibble: 1 x 21
##    Rank Major_code Major Total   Men Women Major_category ShareWomen Sample_siz
##   <dbl>      <dbl> <chr> <dbl> <dbl> <dbl> <chr>               <dbl>       <dbl
## 1    47       3702 STAT…  6251  2960  3291 Computers & M…      0.526           3
## # … with 12 more variables: Employed <dbl>, Full_time <dbl>, Part_time <dbl>,
## #   Full_time_year_round <dbl>, Unemployed <dbl>, Unemployment_rate <dbl>,
## #   Median <dbl>, P25th <dbl>, P75th <dbl>, College_jobs <dbl>,
## #   Non_college_jobs <dbl>, Low_wage_jobs <dbl>
```

Alternatively, we could look at all Majors in the same category, "Computers & Mathematics", for comparison.

```r
recent_grads %>% filter(Major_category == "Computers & Mathematics")
```

```
## # A tibble: 11 x 21
##     Rank Major_code Major  Total   Men Women Major_category ShareWomen
##    <dbl>      <dbl> <chr>  <dbl> <dbl> <dbl> <chr>               <dbl>
##  1    21       2102 COMP… 128319 99743 28576 Computers & M…      0.223
##  2    42       3700 MATH…  72397 39956 32441 Computers & M…      0.448
##  3    43       2100 COMP…  36698 27392  9306 Computers & M…      0.254
##  4    46       2105 INFO…  11913  9005  2908 Computers & M…      0.244
##  5    47       3702 STAT…   6251  2960  3291 Computers & M…      0.526
##  6    48       3701 APPL…   4939  2794  2145 Computers & M…      0.434
##  7    53       4005 MATH…    609   500   109 Computers & M…      0.179
##  8    54       2101 COMP…   4168  3046  1122 Computers & M…      0.269
##  9    82       2106 COMP…   8066  6607  1459 Computers & M…      0.181
## 10    85       2107 COMP…   7613  5291  2322 Computers & M…      0.305
## 11   106       2001 COMM…  18035 11431  6604 Computers & M…      0.366
## # … with 13 more variables: Sample_size <dbl>, Employed <dbl>, Full_time <dbl>,
## #   Part_time <dbl>, Full_time_year_round <dbl>, Unemployed <dbl>,
## #   Unemployment_rate <dbl>, Median <dbl>, P25th <dbl>, P75th <dbl>,
## #   College_jobs <dbl>, Non_college_jobs <dbl>, Low_wage_jobs <dbl>
```

Notice we are using `%>%` to pass the data frame to the first argument in `filter()` and we do not need to use `recent_grads$Colum Name` to subset our data.

`dplyr` functions never modify their inputs, so if we need to save the result, we have to do it using `<-`.

```r
math_grads <- recent_grads %>% filter(Major_category == "Computers &
  Mathematics")
```

Everything we've already learned about logicals and comparisons comes in handy here, since the second argument of `filter()` is a comparitor expression telling `dplyr` what rows we care about.

1.2.3 `arrange()`

`arrange()` works similarly to `filter()` except that it changes the order of rows rather than subsetting. Again, the first parameter is a data frame and the additional parameters are a set of column names to order by.

```r
math_grads %>% arrange(ShareWomen)
```

```
## # A tibble: 11 x 21
##     Rank Major_code Major   Total   Men Women Major_category ShareWomen
##    <dbl>      <dbl> <chr>   <dbl> <dbl> <dbl> <chr>               <dbl>
##  1    53       4005 MATH…     609   500   109 Computers & M…      0.179
##  2    82       2106 COMP…    8066  6607  1459 Computers & M…      0.181
##  3    21       2102 COMP… 128319 99743 28576 Computers & M…      0.223
##  4    46       2105 INFO…   11913  9005  2908 Computers & M…      0.244
##  5    43       2100 COMP…   36698 27392  9306 Computers & M…      0.254
##  6    54       2101 COMP…    4168  3046  1122 Computers & M…      0.269
##  7    85       2107 COMP…    7613  5291  2322 Computers & M…      0.305
##  8   106       2001 COMM…   18035 11431  6604 Computers & M…      0.366
##  9    48       3701 APPL…    4939  2794  2145 Computers & M…      0.434
## 10    42       3700 MATH…   72397 39956 32441 Computers & M…      0.448
## 11    47       3702 STAT…    6251  2960  3291 Computers & M…      0.526
## # … with 13 more variables: Sample_size <dbl>, Employed <dbl>, Full_time <dbl>,
## #   Part_time <dbl>, Full_time_year_round <dbl>, Unemployed <dbl>,
## #   Unemployment_rate <dbl>, Median <dbl>, P25th <dbl>, P75th <dbl>,
## #   College_jobs <dbl>, Non_college_jobs <dbl>, Low_wage_jobs <dbl>
```

If we provide more than one column name, each additional column will be used to break ties in the values of preceding columns.

We can use `desc()` to re-order by a column in descending order.

```r
math_grads %>% arrange(desc(ShareWomen))
```

```
## # A tibble: 11 x 21
##     Rank Major_code Major  Total   Men Women Major_category ShareWomen
##    <dbl>      <dbl> <chr>  <dbl> <dbl> <dbl> <chr>               <dbl>
##  1    47       3702 STAT…   6251  2960  3291 Computers & M…      0.526
##  2    42       3700 MATH…  72397 39956 32441 Computers & M…      0.448
##  3    48       3701 APPL…   4939  2794  2145 Computers & M…      0.434
##  4   106       2001 COMM…  18035 11431  6604 Computers & M…      0.366
##  5    85       2107 COMP…   7613  5291  2322 Computers & M…      0.305
##  6    54       2101 COMP…   4168  3046  1122 Computers & M…      0.269
##  7    43       2100 COMP…  36698 27392  9306 Computers & M…      0.254
##  8    46       2105 INFO…  11913  9005  2908 Computers & M…      0.244
##  9    21       2102 COMP… 128319 99743 28576 Computers & M…      0.223
## 10    82       2106 COMP…   8066  6607  1459 Computers & M…      0.181
## 11    53       4005 MATH…    609   500   109 Computers & M…      0.179
## # … with 13 more variables: Sample_size <dbl>, Employed <dbl>, Full_time <dbl>,
## #   Part_time <dbl>, Full_time_year_round <dbl>, Unemployed <dbl>,
## #   Unemployment_rate <dbl>, Median <dbl>, P25th <dbl>, P75th <dbl>,
## #   College_jobs <dbl>, Non_college_jobs <dbl>, Low_wage_jobs <dbl>
```

1.2.4 `select()`

Sometimes we have data sets with a ton of variables and often we want to narrow down the ones that we actually care about. `select()` allows us to do this based on the names of the variables.

```r
math_grads %>% select(Major, ShareWomen, Total, Full_time, P75th)
```

```
## # A tibble: 11 x 5
##    Major                                       ShareWomen  Total Full_time P75t
##    <chr>                                            <dbl>  <dbl>     <dbl> <dbl
##  1 COMPUTER SCIENCE                                 0.223 128319     91485 7000
##  2 MATHEMATICS                                      0.448  72397     46399 6000
##  3 COMPUTER AND INFORMATION SYSTEMS                 0.254  36698     26348 6000
##  4 INFORMATION SCIENCES                             0.244  11913      9105 5800
##  5 STATISTICS AND DECISION SCIENCE                  0.526   6251      3190 6000
##  6 APPLIED MATHEMATICS                              0.434   4939      3465 6300
##  7 MATHEMATICS AND COMPUTER SCIENCE                 0.179    609       584 7800
##  8 COMPUTER PROGRAMMING AND DATA PROCESSING         0.269   4168      3204 4600
##  9 COMPUTER ADMINISTRATION MANAGEMENT AND SEC…      0.181   8066      6289 5000
## 10 COMPUTER NETWORKING AND TELECOMMUNICATIONS       0.305   7613      5495 4900
## 11 COMMUNICATION TECHNOLOGIES                       0.366  18035     11981 4500
```

We can also use

- `:` to select all columns between two columns
- `-` to select all columns except those specified
- `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"
- `everything()` mathes all columns

```
math_grads %>% select(Major, College_jobs:Low_wage_jobs)
```

```
## # A tibble: 11 x 4
##    Major                                College_jobs Non_college_jobs Low_wage_job
##    <chr>                                       <dbl>            <dbl>         <dbl
##  1 COMPUTER SCIENCE                            68622            25667          514
##  2 MATHEMATICS                                 34800            14829          456
##  3 COMPUTER AND INFORMATION SYSTEMS            13344            11783          167
##  4 INFORMATION SCIENCES                         4390             4102           60
##  5 STATISTICS AND DECISION SCIENCE              2298             1200           34
##  6 APPLIED MATHEMATICS                          2437              803           35
##  7 MATHEMATICS AND COMPUTER SCIENCE              452               67            2
##  8 COMPUTER PROGRAMMING AND DATA PR…            2024             1033           26
##  9 COMPUTER ADMINISTRATION MANAGEME…            2354             3244           30
## 10 COMPUTER NETWORKING AND TELECOMM…            2593             2941           35
## 11 COMMUNICATION TECHNOLOGIES                   4545             8794          249
```

`rename()` is a function that will rename an existing column and select all columns.

```
math_grads %>% rename(Code_major = Major_code)
```

```
## # A tibble: 11 x 21
##     Rank Code_major Major  Total   Men Women Major_category ShareWom-
```

```
en
##    <dbl>      <dbl> <chr>  <dbl> <dbl> <dbl> <chr>               <dbl>
##  1    21       2102 COMP… 128319 99743 28576 Computers & M…      0.223
##  2    42       3700 MATH…  72397 39956 32441 Computers & M…      0.448
##  3    43       2100 COMP…  36698 27392  9306 Computers & M…      0.254
##  4    46       2105 INFO…  11913  9005  2908 Computers & M…      0.244
##  5    47       3702 STAT…   6251  2960  3291 Computers & M…      0.526
##  6    48       3701 APPL…   4939  2794  2145 Computers & M…      0.434
##  7    53       4005 MATH…    609   500   109 Computers & M…      0.179
##  8    54       2101 COMP…   4168  3046  1122 Computers & M…      0.269
##  9    82       2106 COMP…   8066  6607  1459 Computers & M…      0.181
## 10    85       2107 COMP…   7613  5291  2322 Computers & M…      0.305
## 11   106       2001 COMM…  18035 11431  6604 Computers & M…      0.366
## # … with 13 more variables: Sample_size <dbl>, Employed <dbl>, Full_time <dbl>,
## #   Part_time <dbl>, Full_time_year_round <dbl>, Unemployed <dbl>,
## #   Unemployment_rate <dbl>, Median <dbl>, P25th <dbl>, P75th <dbl>,
## #   College_jobs <dbl>, Non_college_jobs <dbl>, Low_wage_jobs <dbl>
```

1.2.5 `mutate()`

Besides selecting sets of existing columns, we can also add new columns that are functions of existing columns with `mutate()`. `mutate()` always adds new columns at the end of the data frame.

```r
math_grads %>% mutate(Full_time_rate = Full_time_year_round/Total)
```

```
## # A tibble: 11 x 22
##     Rank Major_code Major  Total   Men Women Major_category ShareWom-
```

```
en
##     <dbl>      <dbl> <chr>  <dbl> <dbl> <dbl> <chr>             <dbl>
##  1    21       2102 COMP… 128319 99743 28576 Computers & M…      0.223
##  2    42       3700 MATH…  72397 39956 32441 Computers & M…      0.448
##  3    43       2100 COMP…  36698 27392  9306 Computers & M…      0.254
##  4    46       2105 INFO…  11913  9005  2908 Computers & M…      0.244
##  5    47       3702 STAT…   6251  2960  3291 Computers & M…      0.526
##  6    48       3701 APPL…   4939  2794  2145 Computers & M…      0.434
##  7    53       4005 MATH…    609   500   109 Computers & M…      0.179
##  8    54       2101 COMP…   4168  3046  1122 Computers & M…      0.269
##  9    82       2106 COMP…   8066  6607  1459 Computers & M…      0.181
## 10    85       2107 COMP…   7613  5291  2322 Computers & M…      0.305
## 11   106       2001 COMM…  18035 11431  6604 Computers & M…      0.366
## # … with 14 more variables: Sample_size <dbl>, Employed <dbl>, Full_time <dbl>,
## #   Part_time <dbl>, Full_time_year_round <dbl>, Unemployed <dbl>,
## #   Unemployment_rate <dbl>, Median <dbl>, P25th <dbl>, P75th <dbl>,
## #   College_jobs <dbl>, Non_college_jobs <dbl>, Low_wage_jobs <dbl>,
## #   Full_time_rate <dbl>
```

```
# we can't see everything
math_grads %>%
  mutate(Full_time_rate = Full_time_year_round/Total) %>%
  select(Major, ShareWomen, Full_time_rate)
```

```
## # A tibble: 11 x 3
##    Major                                      ShareWomen Full_time_rate
##    <chr>                                           <dbl>          <dbl>
##  1 COMPUTER SCIENCE                                0.223          0.553
##  2 MATHEMATICS                                     0.448          0.466
##  3 COMPUTER AND INFORMATION SYSTEMS                0.254          0.576
##  4 INFORMATION SCIENCES                            0.244          0.619
##  5 STATISTICS AND DECISION SCIENCE                 0.526          0.344
##  6 APPLIED MATHEMATICS                             0.434          0.525
##  7 MATHEMATICS AND COMPUTER SCIENCE                0.179          0.642
##  8 COMPUTER PROGRAMMING AND DATA PROCESSING        0.269          0.589
##  9 COMPUTER ADMINISTRATION MANAGEMENT AND SECURITY 0.181          0.612
## 10 COMPUTER NETWORKING AND TELECOMMUNICATIONS      0.305          0.574
## 11 COMMUNICATION TECHNOLOGIES                      0.366          0.504
```

1.2.6 summarise()

The last major verb is summarise(). It collapses a data frame to a single row based on a summary function.

```
math_grads %>% summarise(mean_major_size = mean(Total))
```

```
## # A tibble: 1 x 1
##   mean_major_size
##             <dbl>
## 1          27183.
```

A useful summary function is a count (`n()`), or a count of non-missing values (`sum(!is.na())`).

```
math_grads %>% summarise(mean_major_size = mean(Total), num_majors =
  n())
```

```
## # A tibble: 1 x 2
##   mean_major_size num_majors
##             <dbl>      <int>
## 1          27183.         11
```

1.2.7 group_by()

`summarise()` is not super useful unless we pair it with `group_by()`. This changes the unit of analysis from the complete dataset to individual groups. Then, when we use the `dplyr` verbs on a grouped data frame they'll be automatically applied "by group".

```
recent_grads %>%
  group_by(Major_category) %>%
  summarise(mean_major_size = mean(Total, na.rm = TRUE)) %>%
  arrange(desc(mean_major_size))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 16 x 2
##    Major_category                        mean_major_size
##    <chr>                                           <dbl>
##  1 Business                                      100183.
##  2 Communications & Journalism                    98150.
##  3 Social Science                                 58885.
##  4 Psychology & Social Work                       53445.
##  5 Humanities & Liberal Arts                      47565.
##  6 Arts                                           44641.
##  7 Health                                         38602.
##  8 Law & Public Policy                            35821.
##  9 Education                                      34946.
## 10 Industrial Arts & Consumer Services            32827.
## 11 Biology & Life Science                         32419.
## 12 Computers & Mathematics                        27183.
## 13 Physical Sciences                              18548.
## 14 Engineering                                    18537.
## 15 Interdisciplinary                              12296
## 16 Agriculture & Natural Resources                 8402.
```

We can group by multiple variables and if we need to remove grouping, and return to operations on ungrouped data, we use `ungroup()`.

Grouping is also useful for `arrange()` and `mutate()` within groups.

1.3 `tidyr`

> "Happy families are all alike; every unhappy family is unhappy in its own way." –– Leo Tolstoy

> "Tidy datasets are all alike, but every messy dataset is messy in its own way." –– Hadley Wickham

Tidy data is an organization strategy for data that makes it easier to work with, analyze, and visualize. `tidyr` is a package that can help us tidy our data in a less painful way.

The following all contain the same data, but show different levels of "tidiness".

```
table1
```

```
## # A tibble: 6 x 4
##   country      year  cases population
##   <chr>       <int>  <int>      <int>
## 1 Afghanistan  1999    745   19987071
## 2 Afghanistan  2000   2666   20595360
## 3 Brazil       1999  37737  172006362
## 4 Brazil       2000  80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

```
table2
```

```
## # A tibble: 12 x 4
##    country      year type            count
##    <chr>       <int> <chr>           <int>
##  1 Afghanistan  1999 cases             745
##  2 Afghanistan  1999 population   19987071
##  3 Afghanistan  2000 cases            2666
##  4 Afghanistan  2000 population   20595360
##  5 Brazil       1999 cases           37737
##  6 Brazil       1999 population  172006362
##  7 Brazil       2000 cases           80488
##  8 Brazil       2000 population  174504898
##  9 China        1999 cases          212258
## 10 China        1999 population 1272915272
## 11 China        2000 cases          213766
## 12 China        2000 population 1280428583
```

```
table3
```

```
## # A tibble: 6 x 3
##   country      year rate
## * <chr>       <int> <chr>
## 1 Afghanistan  1999 745/19987071
## 2 Afghanistan  2000 2666/20595360
## 3 Brazil       1999 37737/172006362
## 4 Brazil       2000 80488/174504898
## 5 China        1999 212258/1272915272
## 6 China        2000 213766/1280428583
```

```r
# spread across two data frames
table4a
```

```
## # A tibble: 3 x 3
##   country     `1999` `2000`
## * <chr>        <int>  <int>
## 1 Afghanistan    745   2666
## 2 Brazil       37737  80488
## 3 China       212258 213766
```

```r
table4b
```

```
## # A tibble: 3 x 3
##   country         `1999`     `2000`
## * <chr>            <int>      <int>
## 1 Afghanistan   19987071   20595360
## 2 Brazil       172006362  174504898
## 3 China       1272915272 1280428583
```

While these are all representations of the same underlying data, they are not equally easy to use.

There are three interrelated rules which make a dataset 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.

In the above example,

`table2` isn't tidy because each variable doesn't have its own column.

`table3` isn't tidy because each value doesn't have its own cell.

`table4a` and `table4b` aren't tidy because each observation doesn't have its own row.

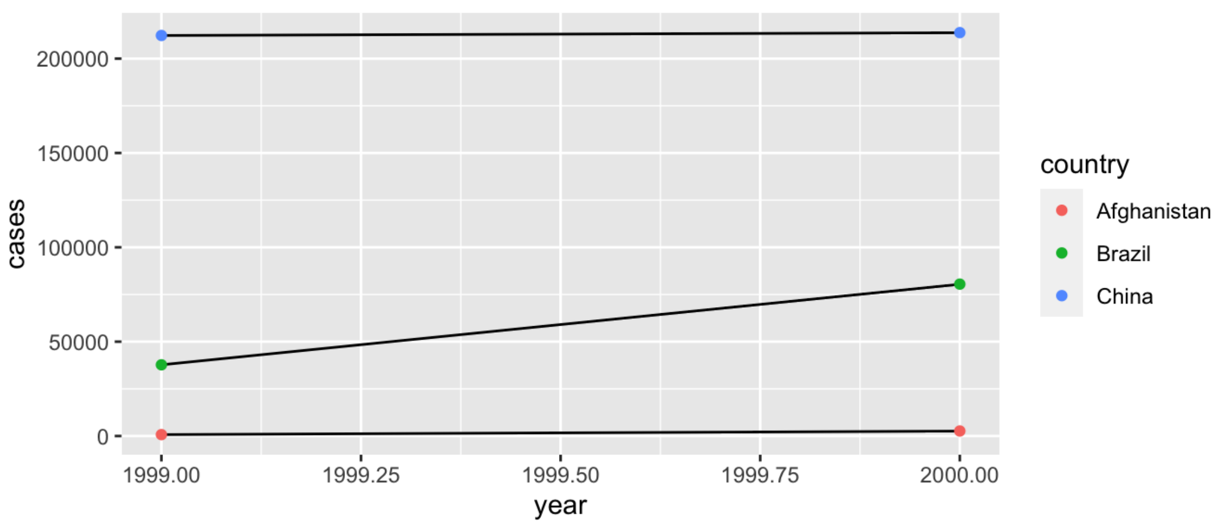
`table1` is tidy!

Being tidy with our data is useful because it's a consistent set of rules to follow for working with data and because it allows `R` to be efficient.

```
# Compute rate per 10,000
table1 %>%
  mutate(rate = cases / population * 10000)
```

```
## # A tibble: 6 x 5
##   country      year  cases population  rate
##   <chr>       <int>  <int>      <int> <dbl>
## 1 Afghanistan  1999    745   19987071 0.373
## 2 Afghanistan  2000   2666   20595360 1.29
## 3 Brazil       1999  37737  172006362 2.19
## 4 Brazil       2000  80488  174504898 4.61
## 5 China        1999 212258 1272915272 1.67
## 6 China        2000 213766 1280428583 1.67
```

```
# Visualize cases over time
library(ggplot2)
ggplot(table1, aes(year, cases)) +
  geom_line(aes(group = country)) +
  geom_point(aes(colour = country))
```



1.3.1 Spread and Gather

Unfortunately, most of the data you will find in the "wild" is not tidy. So, we need tools to help us tidy unruly data.

The main tools in `tidyr` are the ideas of `spread()` and `gather()`. `gather()` "lengthens" our data, increasing the number of rows and decreasing the number of columns. `spread()` does the opposite, increasing the number of columns and decreasing the number of rows.

These two functions resolve one of two common problems:

1. One variable might be spread across multiple columns. (`gather()`)
2. One observation might be scattered across multiple rows. (`spread()`)

A common issue with data is when values are used as column names.

```
table4a
```

```
## # A tibble: 3 x 3
##   country     `1999` `2000`
## * <chr>        <int>  <int>
## 1 Afghanistan    745   2666
## 2 Brazil       37737  80488
## 3 China       212258 213766
```

We can fix this using `gather()`.

```
table4a %>%
  gather(-country, key = "year", value = "cases")
```

```
## # A tibble: 6 x 3
##   country     year   cases
##   <chr>       <chr>  <int>
## 1 Afghanistan 1999     745
## 2 Brazil      1999   37737
## 3 China       1999  212258
## 4 Afghanistan 2000    2666
## 5 Brazil      2000   80488
## 6 China       2000  213766
```

Notice we specified with columns we wanted to consolidate by telling the function the column we *didn't* want to change (`-country`). We can use the `dplyr::select()` syntax here for specifying the columns to pivot.

We can do the same thing with `table4b` and then **join** the databases together by specifying unique identifying attributes.

```
table4a %>%
  gather(-country, key = "year", value = "cases") %>%
  left_join(table4b %>% gather(-country, key = "year", value =
  "population"))
```

```
## Joining, by = c("country", "year")
```

```
## # A tibble: 6 x 4
##   country     year  cases population
##   <chr>       <chr> <int>      <int>
## 1 Afghanistan 1999    745   19987071
## 2 Brazil      1999  37737  172006362
## 3 China       1999 212258 1272915272
## 4 Afghanistan 2000   2666   20595360
## 5 Brazil      2000  80488  174504898
## 6 China       2000 213766 1280428583
```

If, instead, variables don't have their own column, we can `spread()`.

```
table2
```

```
## # A tibble: 12 x 4
##    country      year type            count
##    <chr>       <int> <chr>           <int>
##  1 Afghanistan  1999 cases             745
##  2 Afghanistan  1999 population   19987071
##  3 Afghanistan  2000 cases            2666
##  4 Afghanistan  2000 population   20595360
##  5 Brazil       1999 cases           37737
##  6 Brazil       1999 population  172006362
##  7 Brazil       2000 cases           80488
##  8 Brazil       2000 population  174504898
##  9 China        1999 cases          212258
## 10 China        1999 population 1272915272
## 11 China        2000 cases          213766
## 12 China        2000 population 1280428583
```

```r
table2 %>%
  spread(key = type, value = count)
```

```
## # A tibble: 6 x 4
##   country      year  cases population
##   <chr>       <int>  <int>      <int>
## 1 Afghanistan  1999    745   19987071
## 2 Afghanistan  2000   2666   20595360
## 3 Brazil       1999  37737  172006362
## 4 Brazil       2000  80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

1.3.2 Separating and Uniting

So far we have tidied `table2` and `table4a` and `table4b`, but what about `table3`?

```r
table3
```

```
## # A tibble: 6 x 3
##   country      year rate
## * <chr>       <int> <chr>
## 1 Afghanistan  1999 745/19987071
## 2 Afghanistan  2000 2666/20595360
## 3 Brazil       1999 37737/172006362
## 4 Brazil       2000 80488/174504898
## 5 China        1999 212258/1272915272
## 6 China        2000 213766/1280428583
```

We need to split the `rate` column into the cases and population columns so that each value has its own cell. The function we will use is `separate()`. We need to specify the column, the value to split on ("/"), and the names of the new coumns.

```r
table3 %>%
  separate(rate, into = c("cases", "population"), sep = "/")
```

```
## # A tibble: 6 x 4
##   country      year cases  population
```

```
##   <chr>        <int> <chr>  <chr>
## 1 Afghanistan  1999 745    19987071
## 2 Afghanistan  2000 2666   20595360
## 3 Brazil       1999 37737  172006362
## 4 Brazil       2000 80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

By default, `separate()` will split values wherever it sees a character that isn't a number or letter.

`unite()` is the opposite of `separate()` – it combines multiple columns into a single column.

1.4 Additional resources

`readr` (https://readr.tidyverse.org)

`dplyr` (https://dplyr.tidyverse.org)

`tidyr` (https://tidyr.tidyverse.org)